UNIVERSITAT DE BARCELONA

On-line norm synthesis for open Multi-Agent Systems

Javier Morales

On-line norm synthesis for open Multi-Agent Systems

by

Javier Morales

A dissertation presented in partial fulfilment
for the award of the degree in
Doctor of Philosophy in Mathematics and Computer Science.

Advisors:
Dr. Maite López-Sánchez
Dr. Juan A. Rodríguez-Aguilar

Facultat de Matemàtiques
Departament de Matemàtica Aplicada i Anàlisi
October 2015

In memoriam of Dr. Marc Esteva Vivanco.

A mis padres, a quienes debo la vida,
lo que tengo y lo que soy.

Abstract

Multi-Agent Systems (MAS) are computerised systems composed of autonomous software agents that interact to solve complex problems. Within a MAS, agents require some mechanism to coordinate their activities. In the MAS literature, norms have been widely used to coordinate agents' activities. Thus, given a MAS, a major research challenge is how to synthesise a normative system, namely a collection of norms, which supports its agents' coordination.

This dissertation focuses on the automated synthesis of norms for *open* Multi-Agent Systems. In an open MAS, the agent population may change along time, agents may be developed by third parties and their behaviours are not known beforehand. These particular conditions make specially challenging to synthesise a normative system to govern an open MAS.

The MAS literature has mainly investigated two general approaches to norm synthesis: *off-line design*, and *on-line synthesis*. The first approach aims at synthesising a normative system at design time. With this aim, it assumes that the MAS state space is known at design time and does not change at runtime. This goes against the nature of open MAS, and thus off-line design is not appropriate to synthesise their norms. Alternatively, on-line norm synthesis considers that norms are synthesised at runtime. Most on-line synthesis research has focused on norm emergence, which considers that agents synthesise their own norms, thus assuming that they have norm synthesis capabilities. Again, this cannot be assumed in open MAS.

Against this background, this dissertation introduces a whole computational framework to perform on-line norm synthesis for open Multi-Agent Systems. Firstly, this framework provides a computational model to synthesise norms for a MAS at runtime. Such computational model requires neither knowledge about agents' behaviours beforehand nor their participation in the norm synthesis process. Instead, it considers a regulatory entity that observes agents' interactions at runtime, identifying situations that are undesirable for coordination to subsequently synthesise norms that regulate these situations. Our computational model has been conceived to be of general purpose so that it can be employed to synthesise norms in a wide range of application domains by providing little domain-dependent information. Secondly, our framework provides an abstract architecture to implement such regulatory entity (the so-called *Norm Synthesis Machine*), which observes a MAS and executes a synthesis strategy to synthe-

sise norms. Thirdly, our framework encompasses a family of norm synthesis strategies intended to be executed by the Norm Synthesis Machine. Overall, this family of strategies supports multi-objective on-line norm synthesis.

Our first synthesis strategy, the so-called BASE, aims at synthesising *effective* normative systems that successfully avoid situations that are undesirable for a MAS' coordination. Then, two further strategies (called IRON and SIMON) go beyond effectiveness and also consider *compactness* as a norm synthesis goal. IRON and SIMON take alternative approaches to synthesise compact normative systems that, in addition to effectively achieve coordination, are as synthetic as possible. This allows them to reduce agents' computational efforts when reasoning about norms. A fourth strategy, the so-called LION, goes beyond effectiveness and compactness to also consider *liberality* as a synthesis goal. LION aims at synthesising normative systems that are effective and compact while preserving agents' freedom to the greatest possible extent. Our final strategy is DESMON, which is capable of synthesising norms by considering different degrees of reactivity. DESMON allows to adjust the amount of information that is required to decide whether a norm must be included in a normative system or not. Thus, DESMON can synthesise norms either by being reactive (i.e., by considering little information), or by being more deliberative (by employing more information).

We provide empirical evaluations of our norm synthesis strategies in two application domains: a road traffic domain, and an on-line community domain. In this former domain, we employ these strategies to synthesise effective, compact and liberal normative systems that successfully avoid collisions between cars. In the latter domain, our strategies synthesise normative systems based on users' complaints about inappropriate contents. In this way, our strategies implement a regulatory approach that synthesises norms when there is enough user consensus about the need for norms.

Overall, this thesis advances in the state of the art in norm synthesis by providing a novel computational model, an abstract architecture and a family of strategies for on-line norm synthesis for open Multi-Agent Systems.

Resum

Els sistemes Multi-Agent (MAS) són sistemes computeritzats composats d'agents autònoms que interaccionen per resoldre problemes complexos. A un MAS, els agents requereixen algun mecanisme per coordinar les seves activitats. A la literatura en Sistemes Multi-Agent, les normes han sigut àmpliament utilitzades per coordinar les activitats dels agents. Per tant, donat un MAS, un dels majors reptes d'investigació és el de sintetizar el sistema normatiu, és a dir la col·lecció de normes, que suporti la coordinació dels agents.

Aquesta tesi es centra en la síntesi automàtica de normes per sistemes Multi-Agent *oberts*. A un MAS obert, la població d'agents pot canviar amb el temps, els agents poden ésser desenvolupats per terceres parts, i els comportaments dels agents són desconeguts per endavant. Aquestes condicions particulars fan especialment complicat sintetizar el sistema normatiu que reguli un sistema Multi-Agent obert.

En general, la literatura en Sistemes Multi-Agent ha investigat dúes aproximacions a la síntesi de normes: *disseny off-line*, i *síntesi on-line*. La primera aproximació consisteix en sintetizar un sistema normatiu en temps de disseny. Amb aquest propòsit, aquesta aproximació assumeix que l'espai d'estats d'un MAS és conegut en temps de disseny i no canvia en temps d'execució. Això va contra la natura dels sistemes Multi-Agent oberts, i per tant el disseny off-line no és apropiat per sintetitzar les seves normes. Com a alternativa, la síntesi on-line considera que les normes són sintetizades en temps d'execució. La majoria de recerca en síntesi on-line s'ha centrat en la emergència de normes, que considera que els agents sintetizen les seves pròpies normes, per tant assumint que tenen la capacitat de sintetitzar-les. Aquestes condicions tampoc es poden assumir en un MAS obert.

Donat això, aquesta tesi introdueix un marc computacional per la síntesi on-line de normes en sistemes Multi-Agent oberts. Primer, aquest marc proveeix un model computacional per sintetizar normes per un MAS en temps d'execució. Aquest model computacional no requereix ni coneixement sobre els comportaments dels agents per endavant ni la seva participació en la síntesi de normes. En canvi, considera que una entitat reguladora observa les interaccions dels agents en temps d'execució, identificant situacions indesitjades per la coordinació i sintetizant normes que regulen aquestes situacions. El nostre model computacional ha sigut dissenyat per ésser de propòsit general per tal que pugui ser utilitzat

en la síntesi de normes en un ampli ventall de dominis d'aplicació proporcionant només informació clau sobre el domini. Segon, el nostre marc proveeix una arquitectura abstracta per implementar aquesta entitat reguladora, anomenada *Màquina de Síntesi*, que observa un MAS en temps d'execució i executa una estratègia de síntesi que s'encarrega de sintetizar normes. Tercer, el nostre marc incorpora una familia d'estratègies de síntesi destinades a ésser executades per una màquina de síntesi. En general, aquesta familia d'estratègies soporta la síntesi multi-objectiu i on-line de normes.

La nostra primera estratègia, anomenada BASE, està dissenyada per sintetitzar sistemes normatius *eficaços* que evitin de manera satisfactòria situacions indesitjades per la coordinació d'un sistema Multi-Agent. Després, dúes estratègies de síntesi, anomenades IRON i SIMON, van més enllà de la eficàcia i també consideren la *compacitat* com a objectiu de síntesi. IRON i SIMON prenen aproximacions alternatives a la síntesi de sistemes normatius compactes que, a més d'aconseguir la coordinació de manera efectiva, siguin tant sintètics com possible. Això permet a aquestes estratègies reduir els esforços computacionals dels agents a l'hora de raonar sobre les normes. Una quarta estratègia, anomenada LION, va més enllà de la eficàcia i la compacitat per considerar també la *liberalitat* com a objectiu de síntesi. LION sintetitza sistemes normatius que són eficaços i compactes mentre preserven la llibertat dels agents tant com possible. La nostra última estratègia és DESMON, que és capaç de sintetizar normes considerant diferents graus de *reactivitat*. DESMON permet ajustar la quantitat d'informació necessària per decidir si una norma ha d'ésser o no inclosa a un sistema normatiu. DESMON pot sintetizar normes éssent reactiu (considerant poca informació), o éssent més deliberatiu (considerant més informació).

En aquesta tesi presentem avaluacions empíriques de les nostres estratègies de síntesi en dos dominis d'aplicació: el domini del tràfic, i el domini de les comunitats on-line. En aquest primer domini, utilitzem les nostres estratègies per sintetizar sistemes normatius eficaços, compactes i liberals que eviten colisions entre cotxes. Al segon domini, les nostres estratègies sintetizen sistemes normatius basant-se en les queixes dels usuaris de la comunitat sobre continguts inapropiats. D'aquesta manera, les nostres estretègies implementen un mecanisme de regulació que sintetiza normes quan hi ha suficient consens entre els usuaris sobre la necessitat de normes.

Aquesta tesi avança en l'estat de l'art en síntesi de normes al proporcionar un novedós model computacional, una arquitectura abstracta i una familia d'estratègies per la síntesi on-line de normes per sistemes Multi-Agent oberts.

Acknowledgements

I am so grateful to so many people that I do not know where to start by. During the years this adventure lasted, I had the chance to work side by side with brilliant minds and to know wonderful people. Somehow, all of them have left an indelible mark on me.

First, and foremost, I want to express my most sincere gratitude to my advisors. Because thanks to you I could learn from the best. To Maite López-Sánchez, for giving me the chance to start this journey. For carrying me by the hand from the beginning and teaching me to do research. For teaching me to be rigorous and the value of hard work. For stimulating me both in the good and the difficult moments. I will never forget these years. To Marc Esteva, for always believing in me and for being an incredible advisor in every respect. For your infinite patience, and for the years I could enjoy your company. I will always keep you in my heart. To Juan A. Rodríguez-Aguilar, for so many things. I have learned so many things from you that I will never cease thanking you. Thank you for teaching me to think abstractly. For teaching me how to explain things clearly, and to be rigorous. Thank you for inspiring me, for your always open door, for your kindness, and the infinite help you gave me. I hope to share many more years with you.

My immense gratitude to Mike Wooldridge, for giving me the chance of collaborating with one of the most brilliant minds I have ever known. For your kindness, for believing in my potential even when I did not believe in myself, and for giving me the chance to continue this journey. I hope to work again with you soon. To Wamberto Vasconcelos, for all the help you gave me. Your kindness is only surpassed by your intelligence.

Thanks to Jesús Cerquides, for opening for me the doors of research. Actually, this all started when I met you. You inspired me from the beginning.

To Iosu Mendizábal and David Sánchez-Pinsach, who collaborated in the work developed during this thesis. Iosu, thank you for all those hours analysing lines of code, and for your immense dedication. You are brilliant, man! David, thank you for so much you have contributed with in this work. I am sure there is a great future waiting for you out there.

Special thanks to all the people at the Artificial Intelligence Research Institute (IIIA-CSIC). There, I had the opportunity to know wonderful people, and to develop this thesis in a superb working environment. I really had great years

there. Thanks to all my fellow colleagues in this adventure. To Marc Pujol, because you always inspired me. To Toni Penya, because your happiness enlightened the lab. To Sergio Manzano, because in you I have a friend forever. To Pablo Almajano, for being a friend instead of a teammate. We have a pending party! To Arturo Ribes, Dani Abril, Albert Vilamala, Isaac Cano, for every precious moment we shared together. To Ana Peleteiro, because, thanks to you, I was not the only "sports freak" in the lab for some time. And to all others whom I failed to mention, sorry mates.

My gratitude to all the people at the University of Barcelona (UB), where I had the opportunity to develop myself as a researcher and as a teacher. Thanks to Jordi Campos, Maria Salamó, Inma Rodríguez, and Anna Puig, for opening me the doors of the Applied Mathematics and Analysis department (MAiA). Thank you for making me feel at home.

I also want to thank all those who have supported me throughout this long journey. To my coach and friend, Iván Herruzo. For so many hours "fixing the world" while riding a bike together. For inspiring me, motivating me, and helping me always and with everything you could. You definitely changed me forever.

To my RAYOTEAM mates. Thank you for all the hours we have spent swimming, cycling and running together. I would change those times for nothing. I hope this journey lasts forever! Vamos rayos!

To my childhood friends, who have always been there. David, Pedro, Raul, Carlos, Rudy, Ivan. I hope our friendship lasts at least twenty more years.

A mi familia, porque sin ellos no soy nada. A mi madre, por ser la increíble persona y madre que eres. Por quererme tanto y hacer que te quiera tanto. Por haber luchado toda tu vida para que pudiera llegar hasta aquí. No sería nada sin ti. A mi padre, por querernos tanto y dejarse la piel para que pudiéramos tener un futuro. Sin ti esto nunca habría sido posible. Te quiero, papa. A mi hermana, por una infancia llena de cálidos recuerdos juntos y por estar ahí siempre que la he necesitado. Te quiero, tata.

Above all, thanks to my soulmate Silvia. Thank you for showing me so much love, and picking me up when I was down. Thank you for always being there, through thick and thin. Thank you for existing, and for being by my side. I love you, and I will do forever. Someday, our dreams will become true!

Javier Morales

This work has been funded by the Spanish government through the grant "Formación del Profesorado Universitario" (FPU), with number AP-2010-2468, and the projects Engineering Self-* Virtually Embedded systems (TIN2009-14702-C02-01 / TIN2009-14702-C02-02), Agreement Technologies (CONSOLIDER CSD2007-0022, INGENIO 2010), RObust Collaborations (TIN2012-38876-C02-01), and Support to Consolidated Research Groups (2009-SGR-1434). Also, the Catalan Association for Artificial Intelligence (ACIA), funded my assistance to the Catalan Conference for Artificial Intelligence (CCIA).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A norm is a behavioural pattern that specifies how the individuals of a society are expected to behave [Bicchieri, 2006]. Typically, norms impose restrictions on the behaviour of individuals. These behavioural constraints play two important roles. At the individual (micro) level, they ease decision making by ruling out various courses of action, thus reducing the decision space of alternatives that need to be considered. At the social (macro) level, they provide means whereby individuals can coordinate their activities using only local decision-making [Binmore, 2005]. Norms have thus played an essential role in facilitating cooperation [Elster, 1989b] and coordination [Bicchieri, 2006] in human societies. For instance, humans employ traffic rules to coordinate drivers' activities and hence ensure traffic safety and fluidity [Åberg, 1998].

During the last three decades, the rise of technology has promoted the advent of Multi-Agent Systems (MAS), these computerised systems in which autonomous agents meet and interact to solve complex problems. Within a MAS, agents are typically assumed to have preferences or goals that may not be aligned with each other. To successfully interact, they require some mechanism whereby they can *coordinate* their activities. Inspired on human societies, researchers have widely used norms to coordinate agents' activities in Multi-Agent Systems [Dignum, 1999a, Boella et al., 2006, Shoham and Leyton-Brown, 2009]. Coordination in this sense is usually understood as regulating agents' actions in such a way that (1) each agent can successfully achieve its goals without preventing other agents from achieving theirs; and (2) some global, system-level goals can be achieved. Since the seminal work of Shoham et al. [Shoham and Tennenholtz, 1995] the problem of synthesising norms to achieve coordination in a Multi-Agent System has attracted considerable attention. Solving this *norm synthesis problem* is specially challenging in the case of *open* Multi-Agent Systems, which are typically characterised by their *uncertainty*. Briefly, in an open MAS, the behaviours of its participants are unknown, which makes highly difficult to determine what type of norms should govern the system to achieve coordination.

Against this background, this dissertation introduces a computational frame-

work to synthesise norms for open Multi-Agent Systems. This framework is composed of: (1) a computational model to synthesise norms for a MAS without requiring previous knowledge about the agents' behaviours and their potential interactions; (2) an abstract architecture to support the implementation of this norm synthesis model; and (3) a family of synthesis strategies that implement this norm synthesis model and focus on different synthesis goals. Overall, these strategies allow to synthesise normative systems (i.e., sets of norms) that are effective to coordinate a MAS, while reducing the agents' efforts when reasoning about norms and preserving their freedom as much as possible.

The remainder of this chapter is organised as follows. Section 1.1 introduces the reasons that motivated this dissertation. Then, Section 1.2 presents the research questions this dissertation aims at answering and settles some research goals. Next, Section 1.3 introduces two case studies in which norms can be employed to achieve coordination. Thereafter, Section 1.4 introduces the contributions of this thesis and the publications produced as a result of it, and Section 1.6 outlines the publications derived from this thesis.

1.1 Motivation

Open Multi-Agent Systems are those whose population is *open*, namely in which the agents can freely join and leave the system at any time. Examples of open MAS include agent-mediated electronic commerce [Tsvetovatyy et al., 1997, Guttman et al., 1998, Dellarocas and Klein, 2000], search and rescue operations [Kitano et al., 1997, Jennings et al., 1997, Kitano et al., 1999], and virtual supply chains [Fischer et al., 1996, Fox et al., 2000]. One of the most relevant characteristics of open MAS is the heterogeneity of their participants. In such systems, agents can be designed independently (maybe by third parties) and thus there is no possible access to their internal machinery. Therefore, as previously detailed, no assumptions can be made about their behaviours *a priori*. These particular conditions make specially challenging to determine the norms that should govern such kind of systems. The uncertainty about the situations that will arise in a MAS make challenging to assess what and how to regulate.

In the literature, we can differentiate two strands of work regarding the synthesis of norms for MAS: *off-line design* of norms, and *on-line synthesis* of norms. On the one hand, the off-line design of norms aims at computing the norms of a MAS at design time, namely before the agents start to interact. On the other hand, on-line norm synthesis mainly focuses on how developing techniques whereby norms can emerge at runtime within a MAS. The subsequent sections analyse the strengths and weaknesses of both approaches. Furthermore, they motivate the need for a norm synthesis framework such as the one introduced in this dissertation.

1.1.1 Off-line norms design

When facing the problem of computing the norms that will coordinate a MAS, one may naturally think that a good approach is to design the norms while designing the MAS. This is the case of off-line norms design [Goldman and Rosenschein, 1993, Conte and Castelfranchi, 1995, Shoham and Tennenholtz, 1995, Castelfranchi et al., 1998, Saam and Harrer, 1999, Fitoussi and Tennenholtz, 2000, Schelling, 2006]. Briefly, off-line design consists in: (i) characterising all the possible states of a MAS at design time; (ii) identifying those states that must be regulated to achieve coordination; (iii) designing norms to regulate such states; and (iv) hard-wiring norms into the agent machinery (logic). In essence, this *top-down* regulation approach equals to designing a software system. An example is the seminal work by Shoham and Tennenholtz [Shoham and Tennenholtz, 1995], in which norms are designed off-line to coordinate a group of robots to prevent them from colliding while moving through a grid.

The main advantage of off-line design is that it allows a greater degree control of a MAS from its very initial stages, since norms govern it from the beginning. Nevertheless, the off-line design problem has been proven to be highly complex (NP-complete) [Shoham and Tennenholtz, 1995]. Briefly, going trough the entire system state space is infeasible in systems where the size of the state space is large. These complexity issues have prompted research into the development of methods for managing the size of the system state space [Christelis and Rovatsos, 2009], but even so the state space is typically unmanageable. Therefore off-line design can be applied only whenever the system state space can be computed in a *reasonable* amount of time. Unfortunately, even if the state space were manageable, off-line design cannot be used to synthesise norms for open MAS for the following reasons. First, it assumes that the agents' behaviours and the population composition are known at design time and do not change at runtime. Second, it assumes that norms can be hard-wired into the agents. As previously discussed, these conditions cannot be guaranteed in an open MAS, where the agents' behaviours are unpredictable and their internal architectures are inaccessible.

1.1.2 On-line norm synthesis

Against the disadvantages of off-line design, an alternative approach to norm synthesis is computing norms at runtime (i.e., on-line). Most previous research in on-line norm synthesis has focused on norm emergence (e.g., [Shoham and Tennenholtz, 1992a, Kittock, 1995, Walker and Wooldridge, 1995, Sen and Airiau, 2007a, Sen and Sen, 2010, Villatoro et al., 2011]), which investigates how norms can emerge *bottom-up* at runtime within the agent population. As stated in [Turner and Killian, 1957], typically norm emergence can be employed whenever change and *uncertainty* are characteristics of a MAS. The main advantage of norm emergence is that it does not require any global state representation or centralised control. Instead, agents employ some computational mechanism whereby they *synthesise* their own norms at runtime based on local information.

Norm emergence is based on the agents' capability to converge to a solution from a space of alternative solutions, and it assumes that the agents will choose to cooperate in the emergence process. A norm is considered to have emerged when a significant proportion of the agent population has adhered to some common behaviour. For instance, in a traffic scenario with autonomous cars (agents), two alternative norms may emerge: driving on the left, or driving on the right. When most agents drive on the same side of the road, it is said that a norm has emerged. A paradigmatic example of norm emergence is the *tee-shirt game* [Shoham and Tennenholtz, 1992a], in which a group of agents choose wearing either a blue or a red t-shirt. Initially, each agent wears either a blue or a red t-shirt randomly. The game is played in rounds. At each round agents are grouped in pairs and each agent is able to *see* the colour of its partner's t-shirt. Afterwards, each agent decides what colour to wear based on its memory about encountered agents. After some rounds, the agents are expected to end up wearing the same coloured t-shirt. It is worth noticing that agents only employ local information to decide which colour to wear – they are never able to *see* the colour of all the agents in the MAS simultaneously.

Approaches based on norm emergence have the advantage that they do not suffer from the computational complexity issues of off-line approaches, since there is no need to compute the system state space at design time. Another advantage is that on-line synthesised norms are more likely to be aligned with the agents' behaviour at runtime, and thus to effectively achieve coordination. However, norm emergence techniques make assumptions that hinder their use to synthesise norms for open MAS. First, they assume that the agents' machinery is endowed with the computational capability to participate in norm emergence, and that the agents will choose to collaborate in the emergence process. Second, with some exceptions (such as [Salazar et al., 2010b]), most norm emergence approaches rely on the fact that the agents know at design time an initial set of alternative norms (e.g., driving on the left and driving on the right). In this sense, norm emergence consists in choosing to adhere to one (or more) of the initially available norms. Therefore, it requires previous knowledge about the domain and the possible norms that may govern the system. Again, all these conditions cannot be guaranteed in an open MAS – not only the agents' computational capabilities cannot be guaranteed, but also their willingness to collaborate in norm emergence. In fact, the agents within an open MAS are self-interested, and may be buggy, or even malicious [Klein et al., 2003].

1.1.3 Analysis

At this point, we have the necessary background to conclude that synthesising norms that will help coordinate an open MAS remains an open problem. On the one hand, off-line design approaches are appropriate to synthesise norms for *closed* MAS whose state space can be computed in a *reasonable* amount of time, and whose agents' internal architectures are accessible. On the other hand, existing on-line approaches mainly focus on norm emergence, which assumes that agents are endowed with the computational capabilities to collaborate in

the emergence process, and that they will choose to collaborate.

In the literature, most off-line and on-line approaches have focused on how to synthesise norms that effectively coordinate a MAS. However, there are further objectives that can be considered when synthesising norms. As an example, [Fitoussi and Tennenholtz, 2000] aimed at designing *minimal* and *simple* norms at design time. On the one hand, minimality is concerned with minimising the number of constraints that are imposed on agents. The intuition is that, ideally, the agents should be provided with norms that constrain their behaviour just the necessary to achieve coordination. In this way, minimality can be seen as attempting to compute *liberal* normative systems (i.e., sets of norms) [Ågotnes et al., 2007] that avoid over-regulation, namely that regulate agents' behaviours while preserving their freedom as much as possible. On the other hand, simplicity is concerned with synthesising norms that are *easy* to reason about by agents. Thus, simplicity can be seen as an attempt to synthesise *compact* normative systems that reduce agents' computational efforts when reasoning about norms. However, in the literature there seems to be no previous work that has considered *compactness* and *liberality* as objectives to synthesise norms on-line.

Moreover, there is a criterion that is worth considering during norm synthesis: the *reactivity* of the norm synthesis mechanism. Reactivity is directly related to the reaction time of a system. As stated in [Knight, 1993], reactive systems move in real time and require little information, but cannot guarantee efficient solutions (norms in our case). By contrast, being more *deliberative* is bound to generate more efficient solutions, but it also requires more information and may incur in increasing time and computation costs. Therefore, reactivity is also closely related with the amount of information required to take a decision (e.g., creating a norm). A key point is then to adjust the degree of reactivity in order to generate efficient solutions (norms) without requiring too much time and computation costs. Reactivity and deliberation are two criteria that have been widely considered in *multi-agent planning* [Georgeff and Lansky, 1987, Knight, 1993, Stoytchev and Arkin, 2001, Iocchi et al., 2001, Mavromichalis and Vouros, 2001]. However, they have not been considered in the on-line synthesis of norms.

1.2 Research questions

As previously introduced, there is no computational approach in the literature to synthesise norms at runtime for open MAS and without involving the agents within a MAS' domain (namely, *domain agents*) in norm synthesis. Ideally, in an open MAS, norms should be synthesised by some computational mechanism that observes agents' activities, and creates norms to coordinate their interactions. These norms should be then published so that agents become aware of them. Such mechanism should ensure that norms provided to domain agents are effective in coordinating their activities so that they can successfully interact. Also, it should be designed as generally as possible so that it can be applied to a wide extent of domains by providing little domain-dependent information. From this discussion, a fundamental question of this dissertation is:

Research Question R1. *Can we develop computational means for the on-line synthesis of norms in Multi-Agent Systems without involving domain agents in norm synthesis?*

From [Fitoussi and Tennenholtz, 2000], Fitoussi et al. derive that, in addition to effectively avoid conflicts, a normative system should be as compact and liberal as possible. On the one hand, a normative system should be compact to reduce agents' computational efforts when reasoning about norms. On the other hand, a normative system should be liberal to preserve agents' freedom as much as possible. Therefore the two next research questions are:

Research Question R2. *Can we synthesise compact normative systems?*

Research Question R3. *Can we synthesise liberal normative systems?*

As detailed in Section 1.1.3, it is worth being able to adjust the amount of information that is required to create norms. In other words, to adjust the degree of reactivity of the norm synthesis mechanism. Therefore, the final research question of this dissertation is:

Research Question R4. *Can we synthesise normative systems by considering different degrees of reactivity?*

1.3 Case studies

As detailed in previous section, one of the desired properties of the aforementioned norm synthesis mechanism is domain independence. This mechanism should be able to successfully synthesise norms regardless of the scenario it operates in. This section introduces two different scenarios that can be modelled as a Multi-Agent System and in which norms can be employed to coordinate the activities of their participants. These two scenarios not only will help illustrate the norm synthesis mechanism introduced in this dissertation, but also will be employed to empirically evaluate it in order to prove its generality. First, a road traffic scenario is introduced in which autonomous cars travel through an intersection to reach their destinations. Thereafter, an on-line community scenario is described in which a population of users interact by exchanging contents.

1.3.1 The road traffic scenario

Road traffic is a highly complex problem that has attracted considerable attention in Multi-Agent Systems research [Burmeister et al., 1997, Wiering et al., 2000, France et al., 2003, Adler et al., 2005]. Traffic networks are open systems whose participants have different goals and motivations, and their behaviours are unexpected. Also, the traffic environment is highly dynamic. The weather conditions, the lighting, and even the topology of the network may change along time. For instance, a car accident may oblige to cut lanes, hence forcing other drivers to re-plan their trajectories, which eventually may lead to traffic jams

and even other accidents. These conditions make particularly challenging to determine the norms that will effectively achieve coordination in a traffic network. In an attempt to coordinate traffic activities, humans have developed traffic rules, which describe general situations in which drivers are permitted, obliged and prohibited to perform certain actions. Generally, traffic rules have proven to effectively coordinate drivers' activities. However, such rules must be continuously adapted to the changing environment – e.g., bad weather conditions force the traffic authorities to decrease speed limit.

One of the main reasons why coordination is particularly important in road traffic is the preservation of human lives. Briefly, the lack of coordination between drivers often leads to collisions, which occasionally lead to casualties. This is why, for example, when a traffic light does not work properly, the traffic authorities rapidly send officers to regulate the situation. In this sense we could say that the traffic authorities are highly *reactive* to conflictive traffic situations because it is *essential* to guarantee traffic safety and fluidity.

The traffic scenario used in this dissertation is a simplified version of the widely-used traffic intersection scenario [Wiering et al., 2000, Dresner and Stone, 2008, Doniec et al., 2008]. Here, the traffic scenario is modelled as a Multi-Agent System to be used throughout this dissertation. It is composed of two orthogonal roads represented by a 21×21 grid. Figure 1.1a shows a 11×11 sub-grid that corresponds to the centre of the junction. Each agent is a car that travels along the grid at one cell per tick by following a randomly chosen trajectory. A car enters the scenario from four possible start points (light/green arrows in Figure 1.1a), and travels towards randomly chosen destinations (exit points, depicted in dark/red in Figure 1.1a). Cars' behaviours and their destinations are unknown, aiming at simulating the conditions of an open MAS. Each car has its own local view of the road, which corresponds to the perception of cars in its vicinity. As Figure 1.1b depicts, each car is able to perceive the three cells in front of it (i.e., the left, front and right cells). Likewise in actual-world traffic, the main objective in this scenario is that drivers reach their destinations safely. Therefore, norms are aimed at coordinating the drivers' activities to prevent them from colliding.

1.3.2 The on-line community scenario

During the last two decades, the diffusion of the Internet and the advancement in information technologies has promoted the advent of on-line communities as communication channels (e.g., Facebook, Reddit, Twitter). On-line communities are open, virtual systems in which users with different preferences, goals and beliefs continuously interact by exchanging contents. These continuous interactions may occasionally lead to *frictions* between users (e.g., arguments), which cause discomfort to the users and complicate information exchange. The designers of on-line communities have widely used norms to avoid frictions between users. In this case, norms come typically in the form of pre-defined *terms and conditions* that describe what is acceptable and what is not. The idea is that norms can help achieve *healthy* on-line communities [Hinds and Lee, 2011], in which frictions rarely occur and contents are fluidly exchanged.

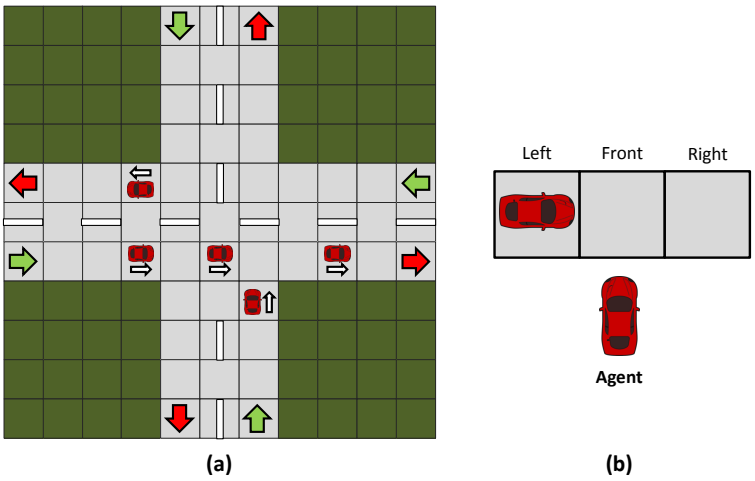

Figure 1.1: a) Traffic intersection scenario. b) A car's perception of the scenario.

The particular features of an on-line community make norms design specially challenging. Within an on-line community, frictions will arise or not depending on what *its users* consider that is (not) acceptable, which may differ from the designers' opinion. Therefore, the norms should be aligned with the users' preferences to effectively achieve a healthy on-line community. Nevertheless, the preferences of the participants are unknown *beforehand*, which makes challenging to design norms that will effectively regulate their behaviours. An alternative solution could be *monitoring* their interactions in real time to determine their preferences, and then synthesise norms based on these preferences. Unlike the traffic case, in this scenario being more *deliberative* may allow synthesise *better* norms. Briefly, the more evidences gathered about the users' preferences before creating a norm, the more likely the norm will reflect the preferences of the overall community, instead of the preferences of a reduced group of users (or even a single user, in the most reactive case).

This scenario has been modelled as an on-line community with a population of users that exchange contents in different sections.Users are allowed to (1) *upload* contents to the different sections, (2) *view* uploaded contents, and (3) *complain* about those viewed contents that they consider as inappropriate. Uploaded contents may have a *type* out of {*correct, spam, porn, violent, insult*}. On the one hand, *correct* contents are those that the users feel comfortable with, and hence do not trigger users' complaints. On the other hand, *inappropriate* contents (i.e., *spam, porn, violent, insult*) may cause discomfort between the users, hence triggering complaints. Reported complaints can be thus seen as user feedback that can be used to determine their preferences, and hence to synthesise norms.

1.4 Contributions

This dissertation contributes to the research questions posed in Section 1.2 by introducing a whole computational framework to synthesise norms for open Multi-Agent Systems. This framework is composed of:

1. A *computational model* to perform on-line norm synthesis without involving domain agents in norm synthesis. Such model does not require previous knowledge of agents' behaviours and their potential interactions, which makes it appropriate for open MAS. Moreover, it is described generally enough to be applied in different application domains by providing little domain-dependent information.

2. An *abstract architecture* to support the implementation of such norm synthesis model.

3. A *family of synthesis strategies* that implement the computational norm synthesis model, and focus on achieving different synthesis objectives. Overall, this family of strategies can be employed to synthesise effective, compact, and liberal normative systems at runtime for open MAS.

Subsequent sections detail these contributions.

Contribution C1: a computational model for on-line norm synthesis

Aiming at answering Research question R1, Chapter 3 introduces an abstract, domain-independent computational model to synthesise norms at runtime for a MAS without requiring the agents' participation in norm synthesis. Such model considers that norm synthesis is done by a regulatory entity that iteratively observes a MAS at runtime, detecting situations that are undesirable for coordination, and synthesising norms to regulate such situations. This norm synthesis model is highly relevant and original, since it is the first computational model to synthesise norms:

- While the agents interact within a MAS at runtime.
- Without considering an initial set of norms.
- Without requiring previous knowledge about the agents' behaviours.
- Without involving domain agents in norm synthesis.

Moreover, it is general enough to be applicable in multiple domains by providing little domain-dependent information. All this together make this on-line norm synthesis model to be appropriate to synthesise norms for open MAS.

To support the implementation of the aforementioned model, the chapter also introduces an abstract architecture to implement such regulatory entity – the so-called *Norm Synthesis Machine* – that iteratively observes a MAS, and executes a *synthesis strategy*, which implements the norm synthesis model, and

is in charge of performing norm synthesis. A Norm Synthesis Machine is based on three core components: (1) a *data structure*, to keep track of synthesised norms; (2) a collection of *operators* to manage such data structure; and (3) a synthesis strategy, which decides *when* and *how* to add or remove norms to such data structure, and *which* norms to make public to the agents.

Contribution C2: a strategy to synthesise effective normative systems

Chapter 4 introduces BASE, a synthesis strategy intended to be executed by a Norm Synthesis Machine to perform norm synthesis. BASE implements the aforementioned norm synthesis model, and produces normative systems that are effective in avoiding conflicts as long as agents comply with their norms. To prove both BASE's performance and the validity of our norm synthesis model, BASE is empirically evaluated in agent-based simulations of the road traffic scenario introduced in Section 1.3.1. Initial results prove that BASE can synthesise norms that successfully avoid collisions between cars as long as they comply with norms.

Contribution C3: strategies to synthesise compact normative systems

Chapters 5 and 6 aim at answering research question R2 by introducing two strategies to synthesise compact normative systems. With this aim, *norm generalisations* are introduced in the norm synthesis process. Briefly, norm generalisations allow to represent several norms by means of a single norm, hence compacting a normative system. As an example, in a traffic scenario, a normative system with one single norm to give way to *emergency* vehicles is more compact than another one with a norm for *each type of* emergency vehicle (e.g., *police, ambulance, fire brigade, etc.*).

Firstly, Chapter 5 introduces IRON (*I*ntelligent *R*obust *O*n-line *N*orm synthesis), which takes a *conservative* approach to norm generalisation. Briefly, IRON generalises a norm only whenever there is enough evidence, to ensure generalisation. Secondly, Chapter 6 introduces SIMON (*S*imple *M*inimal *O*n-line *N*orm synthesis). SIMON takes an *optimistic* approach to norm generalisation: it generalises norms when there is a minimum amount of evidence to performs generalisations. Thus, SIMON manages to synthesise normative systems that are more compact than those synthesised by IRON. SIMON's generalisation is inspired in the anti-unification of terms [Armengol and Plaza, 2000], which consists in generalising a set of feature terms to their *least common subsumer* or *most specific generalisation*. Both strategies are evaluated to analyse how they perform in synthesising norms in the traffic scenario and the on-line community scenario described in Sections 1.3.1 and 1.3.2, respectively. Finally, they are compared in terms of the compactness of the normative systems they synthesise.

From the literature reviewed in the related work, we observe that IRON and SIMON are the first computational on-line norm synthesis algorithms that consider the goal of synthesising *compact* normative systems. By using these strategies, one can provide the agents within an open MAS with compact normative

systems that, in addition to effectively coordinate their activities, reduce their computational efforts when reasoning about norms.

Contribution C4: a strategy to synthesise liberal normative systems

Chapter 7 aims at answering research question R3 by introducing LION (*L*iberal *O*n-line *N*orm synthesis), a strategy to synthesise liberal normative systems. Building on SIMON, LION detects *synergies* between norms to detect and disregard norms that are dispensable to regulate a MAS. In this way, LION reduces the amount of constraints that are imposed on agents, thus increasing their freedom. LION's performance is evaluated and compared with SIMON's in terms of the liberality of the normative systems they synthesise.

To the best of our knowledge, LION is the first on-line computational on-line norm synthesis algorithm that considers liberality as a synthesis objective. LION can be employed to synthesise compact and liberal normative systems that effectively coordinate agents' interactions within open MAS. In the literature, no other computational approach had been presented before that could detect and exploit synergies between norms at runtime. Thus, LION not only contributes to the synthesis of liberal normative systems, but also paves the way for future research in on-line detection and exploitation of semantic relationships between norms.

Contribution C5: a strategy that can operate with different degrees of reactivity

Chapter 8 aims at answering research question R4 in Section 1.2 by introducing DESMON (*DE*liberative *S*imple *M*inimal *O*n-line *N*orm synthesis). DESMON extends SIMON to synthesise norms with different degrees of reactivity. In this way, DESMON successfully synthesises effective, compact and liberal normative systems for MAS in which it is convenient to be reactive (e.g., the traffic scenario), and for MAS in which being more deliberative is more convenient (e.g., the on-line community scenario). To the best of our knowledge, DESMON is the first computational approach that can consider different degrees of reactivity during on-line norm synthesis, namely that can consider different amounts of information to synthesise norms.

1.5 Dissertation outline

The remainder of this dissertation is organised as follows.

Chapter 2 contextualises this work within the state-of-the-art, and discusses different existing approaches to off-line and on-line norm synthesis.

Chapter 3 characterises the norm synthesis problem and introduces a domain-independent model along with an architecture to synthesise norms for open MAS without involving domain agents in norm synthesis.

Chapter 4 introduces BASE, a synthesis strategy that implements the computational norm synthesis model introduced in Chapter 3. BASE's synthesis is driven by the goal of synthesising effective normative systems that successfully coordinate a MAS.

Chapter 5 addresses the synthesis of compact normative systems by introducing IRON (*I*ntelligent *R*obust *O*n-line *N*orm synthesis), a synthesis strategy that extends BASE to include norm generalisations in the synthesis process.

Chapter 6 introduces an alternative strategy to synthesise compact normative systems. It introduces SIMON (*S*imple *M*inimal *O*n-line *N*orm synthesis), which takes an optimistic approach to norm generalisation.

Chapter 7 introduces LION (*L*iberal *O*n-line *N*orm synthesis), a strategy to synthesise liberal normative systems. LION extends SIMON by incorporating the detection and treatment of negative synergies between norms. This allows LION to remove dispensable norms, hence reducing the amount of constraints that are imposed on the agents and increasing their freedom.

Chapter 8 introduces the criterion of *reactivity* in norm synthesis by introducing DESMON (*DE*liberative *S*imple *M*inimal *O*n-line *N*orm synthesis). DESMON is capable of synthesising norms with different degrees of reactivity, and thus by considering different quantities of information.

Chapter 9 draws the conclusions derived from this work, and outlines possible directions for future research.

1.6 Publications derived from this thesis

Below are detailed the publications derived from the work developed during this thesis. Regarding Chapter 3, the following publications introduce and evaluate our computational norm synthesis model, and the BASE strategy. BASE is empirically evaluated in the road traffic scenario introduced in Section 1.3.1.

- Koeppen, J., Lopez-Sanchez, M., Morales, J., and Esteva, M. (2011). Learning from experience to generate new regulations. In *Coordination, Organizations, Institutions, and Norms in Agent Systems VI*, volume 6541 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 337–356. Springer.

- Morales, J., López-Sánchez, M., and Esteva, M. (2011b). Using experience to generate new regulations. In *IJCAI '11: Proceedings of the 22nd International Joint Conference in Artificial Intelligence*, pages 307–312, Barcelona, Spain. AAAI Press, USA.

- Morales, J., López-Sánchez, M., and Esteva, M. (2011a). Evaluation of an automated mechanism for generating new regulations. In *Proceedings of the 14th international conference on Advances in artificial intelligence:*

spanish association for artificial intelligence, CAEPIA'11, pages 12–21, Berlin, Heidelberg. Springer-Verlag.

As for Chapter 5, the following publications introduce IRON's synthesis strategy and empirically evaluate it in the road traffic scenario:

- Morales, J., Lopez-Sanchez, M., Rodriguez-Aguilar, J. A., Wooldridge, M., and Vasconcelos, W. (2013a). Automated synthesis of normative systems. In *AAMAS '13: Proceedings of the 12th international conference on autonomous agents and multiagent systems*, pages 483–490. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).

- Morales, J., López-Sánchez, M., Rodríguez-Aguilar, J. A., Wooldridge, M., and Vasconcelos, W. W. (2013b). Iron: A machine for the automated synthesis of normative systems (demonstration). In *AAMAS '13: Proceedings of the 12th international conference on autonomous agents and multiagent systems*, pages 1389–1390, Saint Paul, Minnesota, USA. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).

- Morales, J., Mendizabal, I., Sánchez-Pinsach, D., López-Sánchez, M., and Rodríguez-Aguilar, J. A. (2015e). Using iron to build frictionless on-line communities. *AI Communications*, 28:55–71.

- Morales, J., López-Sánchez, M., Rodríguez-Aguilar, J. A., Vasconcelos, W., and Wooldridge, M. (2015a). On-line automated synthesis of compact normative systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 10(1):2:1–2:33.

Additionally, the publication below illustrates an empirical evaluation of IRON in the on-line community scenario described in Section 1.3.2:

- Morales, J., Mendizabal, I., Sánchez-Pinsach, D., López-Sánchez, M., and Rodríguez-Aguilar, J. A. (2015e). Using iron to build frictionless on-line communities. *AI Communications*, 28:55–71.

With regard to chapter 6, the following publications introduce and empirically evaluate SIMON in the aforementioned road traffic scenario:

- Morales, J., Lopez-Sanchez, M., Rodriguez-Aguilar, J. A., Wooldridge, M., and Vasconcelos, W. (2014a). Minimality and simplicity in the on-line automated synthesis of normative systems. In *AAMAS '14: Proceedings of the 13th international conference on autonomous agents and multiagent systems*, pages 109–116, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).

- Morales, J., Mendizabal, I., Sanchez-Pinsach, D., Lopez-Sanchez, M., Wooldridge, M., and Vasconcelos, W. (2014b). Normlab: A framework to support research on norm synthesis. In *AAMAS '14: Proceedings of the*

13th international conference on autonomous agents and multiagent systems, pages 1697–1698. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).

Regarding chapter 7, the following publications introduce and empirically evaluate LION in the road traffic scenario:

- Morales, J., Lopez-Sanchez, M., Rodriguez-Aguilar, J. A., Wooldridge, M., and Vasconcelos, W. (2015c). Synthesising liberal normative systems. In *AAMAS '15: Proceedings of the 14th international conference on autonomous agents and multiagent systems*, pages 433–441. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).

- Morales, J., Lopez-Sanchez, M., Rodriguez-Aguilar, J. A., Wooldridge, M., and Vasconcelos, W. (2015b). Extending normlab to spur research on norm synthesis (demonstration). In Printing, S., editor, *AAMAS '15: Proceedings of the 14th international conference on autonomous agents and multiagent systems*, pages 1931–1932, Istanbul, Turkey. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).

The source code implemented during this thesis is publicly available via Github. There, two projects are available:

1. A project called *Norm Synthesis Machine*, which implements the architecture for on-line norm synthesis introduced in this dissertation, along with software implementations of all the synthesis strategies introduced in this thesis: BASE, IRON, SIMON, LION, and DESMON.

2. A project with two Multi-Agent simulators of the road traffic scenario and the on-line community scenario described in Section 1.3.

- Morales, J. (2010–2015). The norm synthesis machine, a library to synthesise norms for multi-agent systems. `https://github.com/NormSynthesis/NormSynthesisMachine`.

- Morales, J., Mendizabal, I., and Sánchez-Pinsach, D. (2010–2015d). Normlab, a framework to support research on norm synthesis. `https://github.com/NormSynthesis/NormLabSimulators`.

Chapter 2

State of the art

2.1 Introduction

Norms are a fundamental technique for coordinating activities [Lewis, 1969]. The role of norms in societies has been studied from several disciplines including sociology [Gibbs, 1965, Elster, 1989a, Coleman and Coleman, 1994], philosophy [Von Wright, 1951, von Wright, 1963, Kanger, 1971, Hilpinen et al., 1971, Bicchieri, 2006], economics [North, 1990, Epstein, 2001], and computer science [Shoham and Tennenholtz, 1992a, Shoham and Tennenholtz, 1995]. Within the field of computer science, particularly in the area of Multi-Agent Systems (MAS), norms have been a subject of interest for many researchers. The community in MAS research has widely studied different problems related to norms and Normative Multi-Agent Systems (NMAS). One of the most relevant problems is how to synthesise norms to effectively coordinate a given MAS. This chapter provides a general overview of the different problems that have been studied in the literature in NMAS research, and contextualises the norm synthesis problem within this general frame. Thereafter, it provides a review and a classification of the most relevant norm synthesis mechanisms for MAS, and also identifies open problems and research opportunities.

This chapter is structured as follows. Section 2.2 gives a brief introduction to norms and Normative Multi-Agent Systems. Section 2.3 provides a critical review of the research on norm synthesis, and classifies the most relevant contributions based on how they tackle the norm synthesis problem and the synthesis techniques they employ. Thereafter, Section 2.4 analyses the literature in norms research and identifies the problems that still remain unsolved, and Section 2.5 draws some conclusions.

2.2 Norms in Multi-Agent Systems

Norms are rules that represent the expected behaviour towards specific situations. [Bicchieri, 2006]. In the literature in Multi-Agent Systems, the concept

of *norm* has been presented in different manners, such as convention [Jennings, 1993, Walker and Wooldridge, 1995] and social law [Lewis, 1969]. Researchers in NMAS have widely discussed about the difference between conventions and social laws. However, it is generally understood that conventions are established patterns of behaviour that are not associated with sanctions (e.g., style of dress, or forming a queue when waiting for the bus); and social laws carry with them some authority [Shoham and Leyton-Brown, 2009].

In an attempt to understand norms, the NMAS community has classified norms into different categories. Next, a review of some of these classifications is provided in order to characterise the type of norms that are of interest in this dissertation.

2.2.1 Classification of norms

For the purpose of this dissertation, two norm classifications are specially interesting: Coleman's classification ([Coleman and Coleman, 1994]), and Boella's classification ([Boella and van der Torre, 2004]). Coleman classifies norms into two main categories: *conventions* and *essential* norms.

- *Conventions* are self-enforced norms that emerge naturally within an agents' population. Conventions solve coordination problems when there is no conflict between the individual goals and the collective goals. In his seminal 1969 work, David Lewis [Lewis, 1969] argued that conventions are regularities in behaviour such that: everyone conforms; everyone expects everyone else to conform; and everyone prefers to conform, on the assumption that everyone else does. Lewis explicitly argued that the role of conventions in a society is to solve coordination problems.

- *Essential norms* are aimed at solving coordination problems when there is conflict between the individual goals and the collective goals. Essential norms *mandate* behaviour that is beneficial to others but costly to the individual [Piskorski and Gorbatai, 2013], and *prohibit* behaviour harmful to others but gratifying for the individual [Hechter, 1988]. A paradigmatic example of essential norm is the norm not to pollute urban streets, which is beneficial to the society but it requires the individual to carry her trash until she finds a dustbin.

According to Coleman's classification, the type of norms that are considered in this dissertation are *essential* norms. Our aim is to synthesise norms that regulate agents' behaviours, even though sometimes they may imply a certain cost for them. For instance, consider the road traffic scenario introduced in Section 1.3.1. There, avoiding car collisions may require to force some cars to give way to other cars, or to limit their speed under certain situations.

According to their purpose, Boella and van der Torre classified norms into three different categories: *regulative* norms, *constitutive* norms, and *procedural* norms.

- *Regulative norms.* Norms of this type describe the expected behaviour of the agents in certain situations by means of deontic statements (i.e., obligations, prohibitions, and permissions). These norms regulate activities that can exist independently of the norm. An example of a regulative norm could be a *"Give way to emergency vehicles"* traffic norm.

- *Constitutive norms.* This type of norms are aimed at creating institutional facts, such as the rules of a game (e.g., chess). Constitutive norms give an abstract meaning to facts of the environment (e.g., *property* or *marriage*).

- *Procedural norms.* Also known as *enforcement norms*, procedural norms define a practical connection between a regulation and its consequences. In other words, they define how rewards and punishments are allocated within a normative system. An example of procedural norm is one like *"Those drivers that do not give way to emergency vehicles will be punished with a traffic ticket"*.

Against this background, this dissertation focuses on the synthesis of *regulative* norms. Thus, a norm is assumed to be a formal specification of a deontic statement that imposes prohibitions, obligations, or permissions on the behaviour of agents.

2.2.2 Normative Multi-Agent Systems

A Normative Multi-Agent System is the result of introducing norms in a Multi-Agent System to regulate agents' behaviours [Boella et al., 2006]. Recently, Boella and van der Torre [Boella et al., 2008a] defined a Normative Multi-Agent System (NMAS) as *"a multi-agent system organised by means of mechanisms to represent, communicate, distribute, detect, create, modify and enforce norms, and mechanisms to deliberate about norms and detect norm violation and fulfilment"*.

In a later work, Boella et al. [Boella et al., 2008b] proposed five levels in the development of NMAS. At level 1 (off-line design), norms are designed off-line and hard-wired into the internal machinery of the agents. In this way, norms are automatically enforced and agents are not free to choose to comply or not with norms. At level 2 (norm representation), norms are explicitly represented and made available to the agents, who can choose whether to comply with norms or not. Thus, at this level *norm-compliance* becomes a key issue and norm enforcement mechanisms become necessary. At level 3 (norm manipulation), agents can manipulate norms (i.e., to add and remove norms) following the rules of the normative system. Thus, norms can be dynamically adapted in response to changes of the system. Nowadays, the NMAS community is moving to level 4 (social reality), in which the norms are synthesised based on agents' interactions. At this level, norms are emergent patterns of behaviour that emerge within the agent population. Further ahead, level 5 is concerned with ethical issues, and norms create a moral reality.

Against this background, the scope of this dissertation can be located at level 4 of NMAS (social reality), since here we focus on the synthesis of norms based on the observation of agents' interactions. Notice therefore that this dissertation is framed within the current state of the art of NMAS research. Next, a review of the research in Normative Multi-Agent Systems is provided.

2.3 Research on Normative Multi-Agent Systems

The NMAS community has studied a wide extent of problems related to the theory, design and implementation of Normative Multi-Agent Systems. Hereafter, some of the problems that have been studied are presented and described.

1. *Normative architectures.* Research has focused on the design and development of architectures to build normative agents [Castelfranchi et al., 1999, Dignum et al., 2000, Broersen et al., 2001, Governatori and Rotolo, 2007, Criado et al., 2010] and Normative Multi-Agent Systems [Omicini, 2001, Esteva et al., 2001, Dignum et al., 2004]. Such architectures can be employed to endow agents with capabilities to recognise and to process norms, and to build Multi-Agent Systems wherein the agents' behaviours are governed by norms, respectively.

2. *Norm representation.* In a NMAS, norms must be presented in such a way that they can be understood and processed by the agents. The literature has employed different means to represent norms, such as deontic logic [Boella and van der Torre, 2006, Sadri et al., 2006], binary variables[Epstein, 2001, Flentge et al., 2001, Chalub et al., 2006, Sen and Airiau, 2007b], and game-theoretical strategies [Axelrod, 1986, Shoham and Tennenholtz, 1992b, Kittock, 1993].

3. *Norm reasoning.* Once the agents know the norms of a NMAS, they must be able to think about these norms to incorporate them in their decision making process. Norm reasoning studies mechanisms whereby the agents can assess norm applicability and decide on norm compliance [Conte et al., 1998, Castelfranchi et al., 1999, Dignum et al., 2000, Boella and Lesmo, 2001].

4. *Norm enforcement.* Some works, such as [Axelrod, 1986, Kittock, 1993, Walker and Wooldridge, 1995, Epstein, 2001], regard norms as *soft constraints* that agents are free to fulfil or infringe. Consequently, the problem of *non-compliance* (also called *deviance*) is a concern in NMAS research. The literature has studied different approaches to discourage deviant agents through some form of sanctioning, such as punishments [Axelrod, 1986, López et al., 2002], and reputation [Castelfranchi et al., 1998, Huynh et al., 2006].

5. *Norm synthesis.* Within the NMAS community, a problem that has attracted considerable attention is that of determining what is the set of norms that will effectively coordinate a NMAS. Since the seminal work of Shoham and Tennenholtz [Shoham and Tennenholtz, 1992b], several approaches have been proposed by which norms can come to exist in a NMAS [Kittock, 1993, Walker and Wooldridge, 1995, Epstein, 2001, Sen and Airiau, 2007b, Salazar et al., 2010a].

The remainder of this section analyses some relevant contributions in the literature in the research areas identified above. The analysis below is not meant to be extensive, but to identify the issues that have been dealt and are being dealt by the state of the art. Due to the special relevance of the norm synthesis problem in this dissertation, norm synthesis research is analysed in more detail in Section 2.4.

2.3.1 Architectures for Normative Multi-Agent Systems

The literature offers several architectures to develop normative agents and normative multi-agent systems. Most researchers have focused on providing architectures to develop normative agents, generally by extending the BDI architecture [Rao and Georgeff, 1991]. Castellfranchi et al. [Castelfranchi et al., 1999] introduced a BDI-based generic architecture for deliberative normative agents whereby norms can be communicated, adopted and used as meta-goals on the agent's own processes. Deliberative normative agents are agents that have explicit knowledge about the enacted norms in a multi-agent environment and can make a choice whether to obey the norms or not in specific cases. Dignum et al. [Dignum et al., 2000] proposed an extension of the classic BDI architecture to consider norms. They study how to explicitly represent norms to infer the agents' intentions, and they modify the original BDI algorithm to consider the existence of normative beliefs. Additionally, Kollingbaum et al. [Kollingbaum, 2005] introduced a BDI-based architecture (the so-called NoA architecture) for developing normative agents that can detect norm applicability and determine which norms are relevant to them at a given moment. They employed predicate logic as a representation language. In this work, agents do not have internal motivations, though, and they are considered to always try to fulfil norms.

In a later work, Broersen et al. [Broersen et al., 2001] introduced the BOID architecture, which incorporates obligations into the BDI architecture [Rao et al., 1995]. The BOID architecture contains feedback loops to consider all effects of actions before committing to them, and mechanisms to resolve the conflicts of its four components (i.e., beliefs, obligations, intentions and desires). Based on the BOID architecture, Governatori and Rotolo [Governatori and Rotolo, 2007] introduced the BIO architecture (Beliefs, Intentions, and Obligations), in which agents and agent types are described in *defeasible logic*. In such architecture, the system develops a positive account of those modalities that match to mental states and obligations. Thus, norms are aimed at devising appropriate logical conditions to introduce modalities. Criado et al. [Criado et al., 2010] extended

the BDI architecture with a recognition model and a normative context in order to allow agents to acquire norms from their environment and consider norms in their decision making processes.

Alternatively, other contributions have not considered the BDI architecture as a basis. An example is the work by Sadri et al. [Sadri et al., 2006], which presented a framework that extends the logical model of agency known as the KGP model (Knowledge, Goals and Plans) to support agents with normative concepts (e.g., obligations and prohibitions). They illustrated how these concepts can be combined with the existing capabilities of KGP agents in order to plan for their goals, react to changes in the environment, and interact with other agents.

As for architectures for Normative Multi-Agent Systems, significant contributions have been made. Boella and van der Torre [Boella and van der Torre, 2006] introduced an architecture for a Normative System, containing separate subsystems for institutional rules, obligations and permissions. Their architecture contains a norm database in which the three kinds of norms are stored, and uses a coordination model to describe the relations among the normative components. Andrea Omicini [Omicini, 2001] introduced *SODA*, an extension to *GAIA* [Wooldridge et al., 2000] that enables open societies to design and enforce norms to coordinate individuals' interactions. In this work, normative aspects reduce to the notion of permission to access infrastructure services. Alternatively, Esteva et al. [Esteva et al., 2001] introduced a model for the formal specification of Electronic Institutions (EI). Briefly, an Electronic Institution is an electronic implementation of a classical organisation of the human society (e.g., an auction house). This work presented a specification formalism for Electronic Institutions that founded their design, analysis and development. Dignum et al. [Dignum et al., 2004] introduced the *OperA* model to build normative organizations where individuals can collaborate, while prescribing a formal structure for organisational processes.

2.3.2 Norm representation

We identify several approaches in the literature to represent norms, such as deontic logic, variables, and game-theoretical strategies.

Deontic logic

Deontic logic [Von Wright, 1951] is the result of extending modal logic to deal with obligations, prohibitions, and permissions [Meyer and Wieringa, 1993]. In NMAS research, some works have employed deontic logics to define and represent norms. Mainly, deontic logic has been used as a norm representation language for developing architectures for Normative Multi-Agent Systems. An example is given by Boella and van der Torre [Boella and van der Torre, 2006], which introduced an architecture for Normative Multi-Agent Systems that considered obligations and permissions, represented as deontic statements. García et al. [García-Camino et al., 2006] included an explicit representation of permissions, prohibitions and obligations in Electronic Institutions. Further ahead, Sadri et

al. [Sadri et al., 2006] used deontic logic to represent the normative concepts of obligation and prohibition in their KGP architecture.

However, most computational approaches have not considered deontic logic to represent norms. Instead, they have represented norms with simple data types, mainly to simplify the model and to reduce the computation required.

Variables

Some works have employed a simpler representation of norms as binary variables [Epstein, 2001, Flentge et al., 2001, Chalub et al., 2006, Sen and Airiau, 2007b]. Briefly, a binary variable is a string composed of ones and zeros digits, where the digit one represents the occurrence of a norm, and digit zero represents the absence of a norm. This model is often used in research that examines the transmission, evolution and emergence of norms in a population. Representing norms as variables has the advantage that norms are quite simple and easy to process by agents. However, in complex, open MAS, the representation of norms needs to be flexible and dynamic. For example, at a given point it may be necessary to regulate situations that had not been considered before, and thus creating norms that had never been explored. Therefore, a richer model of norm representation is required.

Game theoretical strategies

Other works have represented norms as game-theoretical strategies [Axelrod, 1986, Kittock, 1993, Kittock, 1995, Axelrod, 1997]. In these works, norms are represented by the strategies that the agents use to make their decisions. Briefly, an agent's decisions yield to a particular pay-off matrix. The idea is that an agent decides its actions based on what it thinks its opponents will do, with the aim of maximising its pay-off. After a certain time, it is said that a norm emerges when a significant number of agents end up playing the very same strategy. This representation model has the disadvantage that norms are not explicitly represented, but they are implicit in the strategies played by the agents. Furthermore, likewise the binary-variable-based representation, this model is not flexible enough to be employed in open MAS.

2.3.3 Norm reasoning

Norm reasoning is the process of thinking about norms to make decisions [Van Der Torre and Tan, 1999]. From the agents perspective, norm reasoning is a *decision making process* whereby an agent decides, at each moment, what actions to perform, considering its goals and the restrictions that norms impose on it. This implies that the agents are capable of recognising the norms in force and are capable of managing normative beliefs. Mainly, the norm reasoning mechanisms provided in the literature are logic-based. Conte et al. [Conte et al., 1998] described what information should be considered for the recognition of norms

and the determination of norm compliance. With this aim, they used a logic-based language defined in [Dignum, 1999b]. However, this work does not describe how this information can be managed and considered by an agent. Along these lines, Castelfranchi et al. [Castelfranchi et al., 1999] proposed a model to extend an agent architecture with capabilities to represent norms explicitly as deontic statements, and to deliberatively decide whether to comply or not with them. In [Dignum et al., 2000], Dignum et al. proposed an architecture to develop BDI agents with norm reasoning capabilities, in which norms are represented by means of PDL (Prohairetic Deontic Logic) [van der Torre and Tan, 1998]. In this work, agents can detect norm applicability, namely when norms hold for them, and to decide what action to perform based on applicable norms. However, agents cannot consider their own goals to decide norm compliance. Instead, their behaviour is completely determined by norms.

Kollingbaum et al. [Kollingbaum, 2005] employed predicate logic as a representation language in his NoA architecture for normative agents. Alternatively, Lopez et al. [López et al., 2006] proposed a model to develop agents that can decide norm compliance by considering both norms and their own goals. There, the norm representation language was the Z language [Spivey and Abrial, 1992], which is based on set-theory and first order logic. Lopez et al. developed strategies to allow agents to make decisions about norm compliance. They assumed that there is an enforcement mechanism that rewards compliant agents and punishes deviant agents.

In this thesis, agents are assumed to have their own goals and motivations. Moreover, they are assumed to be able to interpret norms and detect norm applicability, while they are free to decide norm compliance. Since we represent norms as deontic statements, agents are required to have the computational capability to process and reason about deontic notions. The problem of how the agents decide norm compliance is out of the scope of this dissertation. The norm synthesis mechanisms proposed in this dissertation reasons about what happens when agents fulfil and infringe norms, but it does not reason about the agents' motivations to take these decisions.

2.3.4 Norm enforcement

Norms can be used to pursue a desired collective behaviour in a given society. However, if the agents are free to choose whether to abide by norms or not, this ideal behaviour cannot be guaranteed. Norm enforcement studies how deviant agents can be discouraged through some form of sanctioning. Typically, an enforcement agent is given the task of implementing and applying sanctions in order to discourage other agents from infringing norms. A widely used mechanism for sanctioning is the *punishment* of deviant behaviour. A paradigmatic example is the well-known work of Axelrod [Axelrod, 1986], which introduced the use of *meta-norms* to punish deviant agents. In this work, a stable norm emerged within the agent population only when the punishment mechanism was employed. Later, Lopez et al. [López et al., 2002] argued that punishments can be used as an enforcement mechanism only if they affect an agent's goals.

They proposed to use the motivation of an agent as the main driving force behind norm compliance. They also considered punishments and rewards in their model. However, these works do not explicitly consider the cost of sanctioning on the part of the agent imposing this punishment. They assume that all punishments and rewards are applied by someone else in a society.

Alternatively, reputation mechanisms have also been used to enforce norms in an agent society. Reputation studies how the (positive or negative) opinion about an agent can be used to discourage deviant agents, for instance by making agents not being willing to interact with an agent that does not abide by norms. A relevant work in reputation is that of Castelfranchi et al. [Castelfranchi et al., 1998]. In this work, agents are endowed with a mechanism to recognise whether an agent is norm-compliant – namely, it abides by norms (good reputation) – or it is deviant (bad reputation). This allows norm-compliant agents to adapt their behaviour when interacting with deviant agents in order to better achieve their goals. Sabater et al. [Sabater and Sierra, 2002] focus on the computation of reputation, and propose a reputation system that employs social relations between agents to calculate an agent's reputation in large multi-agent systems. Other works have proposed alternative approaches to enforce norms. The work in [de Pinninck et al., 2007] proposed a distributed mechanism to enforce norms by ostracising agents that do not abide by them. This work showed that the ostracism mechanism substantially reduces the number of norm infringements in a society.

In this thesis, no enforcement mechanism is considered. Our aim is not to enforce norms to guarantee a desired overall behaviour, but to synthesise norms that allow the agents to successfully interact as long as they comply with them.

2.4 Research on norm synthesis

As detailed in Section 1.1.3, a key, open problem in NMAS research is the process whereby norms come to exist in a Normative Multi-Agent System. This *norm synthesis problem*, which in fact is the problem this dissertation focuses on, has been studied from different perspectives. This section provides a review of the research on norm synthesis. First, Section 2.4.1 establishes some criteria to analyse the different contributions in the literature, and classifies these contributions based on the identified criteria, pointing out their strengths and weaknesses. Thereafter, Section 2.4.2 analyses the problems that still remain unsolved in the literature.

2.4.1 Classifying research on norm synthesis

Since the work by Shoham et al. [Shoham and Tennenholtz, 1995], the norm synthesis problem has attracted considerable attention within the MAS community. Several works have made significant contributions that have shed light on how to synthesise the norms that will effectively coordinate a MAS. These works have been classified in several manners. For instance, Savarimuthu and

Cranefield [Savarimuthu et al., 2009] classified norm synthesis approaches into *top-down* and *bottom-up*. Briefly, top-down approaches (also known as prescriptive approaches) are those in which an institutional entity (e.g., a designer, or a regulation mechanism) specifies how agents should behave; and bottom-up approaches consider that norms emerge within the agent population (i.e., norm emergence). Alternatively, Walker and Wooldridge [Walker and Wooldridge, 1995] classified norm synthesis into *off-line design* and *on-line emergence*. On the one hand, off-line design approaches aim at designing the norms of a MAS *before* its operation. In this way, the MAS starts running with norms that regulate its agents' behaviours from the very beginning. On the other hand, on-line emergence corresponds to the bottom-up approaches described by Savarimuthu, in which a norm emerges within the agent society.

This section aims at providing a richer classification to characterise the different norm synthesis contributions in the literature along several dimensions. Additionally, it aims at contextualising the norm synthesis mechanism provided in this dissertation within this general frame. We consider that a norm synthesis mechanism can be analysed based on *five* different dimensions: (1) the *type of Multi-Agent System* for which to synthesise norms; (2) the *synthesis time*, namely *when* norms are synthesised; (3) the *actor(s)* running the synthesis process; (4) the *synthesis scope*, namely if the search of normative systems is reduced to a sub-area of the total space of possible normative systems, or the full space is considered; and (5) the *objectives* of a norm synthesis process. Subsequent sections describe in detail these dimensions and classifies the different contributions in the literature in terms of these dimensions.

Multi-Agent System type

Firstly, the different norm synthesis approaches in the literature may be classified by the type of MAS to synthesise norms for. In short, a MAS may be either *closed* or *open* [Hewitt, 1986]. Closed systems are those for which the agents population is known at design time, and does not change at runtime. In closed systems, the system state space is known beforehand and the agents' internal machinery is accessible. For these reasons, closed systems can take benefit of off-line design approaches, such as [Shoham and Tennenholtz, 1992b, Shoham and Tennenholtz, 1995, Fitoussi and Tennenholtz, 2000, Conte and Castelfranchi, 1995, Agotnes and Wooldridge, 2010]. However, not only off-line design applies to closed systems. Other on-line synthesis approaches such as [Axelrod, 1986, Shoham and Tennenholtz, 1992a, Walker and Wooldridge, 1995] have studied mechanisms whereby norms can *emerge* within a closed agent society.

By contrast, in open systems the agent population is open, which means that (i) the agents may be designed by third parties, and (ii) the agents can freely join and leave the system at runtime [Dignum, 2009]. This implies that the agents' behaviours cannot be predicted at design time, and thus off-line design is not appropriate. Instead, norms should be synthesised at runtime, namely on-line synthesis is required. Some examples of open systems include agent-mediated electronic commerce [Tsvetovatyy et al., 1997, Guttman et al.,

1998, Dellarocas and Klein, 2000], search and rescue operations [Kitano et al., 1997, Jennings et al., 1997, Kitano et al., 1999], and virtual supply chains [Fischer et al., 1996, Fox et al., 2000].

Synthesis time

According to *when* norms are synthesised, we identify to main approaches in the literature: synthesising norms at *design time* (off-line design), and synthesising norms at *run time* (on-line synthesis). As detailed above, in off-line design norms are designed before the system starts running, and cannot be changed at runtime. By contrast, on-line synthesis approaches consider that the agents within a MAS synthesise their own norms at runtime, and thus they can adapt norms while the system is running.

– **Off-line design**. The first work tackling off-line norms design was [Shoham and Tennenholtz, 1992b] (later extended in [Shoham and Tennenholtz, 1995]), which introduced the *useful social law* problem. There, a social law was implemented as a set of restrictions on the agents' behaviour that constrain them so that they will not interfere with each other, while allowing them enough freedom to achieve their goals. Later, Goldman and Rosenschein [Goldman and Rosenschein, 1993] presented a way to characterise domains as multi-agent deterministic finite automata. They characterised norms for cooperation as transformations of these automata; and proved that agents were capable of coordinating by abiding to norms, without any explicit coordination mechanism. Conte and Castellfranchi [Conte and Castelfranchi, 1995] designed norms for a group of agents finding food in a simulated grid environment. Norms were simple rules for movement and food collection, which were proved to effectively allow the agents to reach their goals (i.e., gaining strength by consuming food).

More recently, Bulling et al. [Bulling and Dastani, 2011] applied methods from mechanism design to verify whether some norm restrictions imposed on a MAS agree with the behaviour the designer expects. They introduced normative behaviour functions to represent the "ideal" behaviour of MAS with respect to different sets of agents' preferences. Then they applied concepts from game theory to identify agents' rational behaviour. These formal ideas can be used to verify whether a set of norms and sanctions is sufficient to motivate agents to act in such a way that the behaviour described by the normative behaviour function is met.

Generally, off-line design approaches are often intractable in the general case or not easily applicable in real world (open) systems, due to their computational complexity. These complexity issues have prompted research into the problem of managing the size of the system state space. For instance, in [Onn and Tennenholtz, 1997] the off-line design problem is reduced to a graph routing problem and computed efficiently using existing graph theoretic methods. However, that approach is very domain-specific, and alternative problems can be difficultly expressed by employing that representation. Agotness et al. [Agotnes and Wooldridge, 2010] modelled the problem of implementing a social law as an

optimisation problem, and characterised the computational complexity of the problem. They showed how the optimisation problem can be solved using integer linear programming, an approach that has been successfully and widely used to tackle computationally hard problems in other domains. Later, Christelis et al. [Christelis and Rovatsos, 2009] proposed to design norms by performing local search around declarative specifications of states using AI planning methods. Using their approach, norms can be synthesised over incomplete state specifications.

However, even if off-line design approaches were computationally feasible, they assume that norms must be hard-wired into an agent's reasoning mechanism. Moreover, they assume that agents always abide by norms. These assumptions make off-line design inappropriate to synthesise norms for dynamic, open MAS.

– **On-line synthesis**. The literature provides a wide extent of approaches to synthesise norms at runtime. Mostly, research on on-line synthesis focuses on norm emergence, yet other approaches have been considered, such as *norm negotiation*, *norm agreement*, and *norm learning*. Particularly, norm emergence [Shoham and Tennenholtz, 1992a, Kittock, 1993, Kittock, 1995, Griffiths and Luck, 2010, Epstein, 2001] studies how norms can emerge within an agent society. A norm is considered to have emerged when a significant proportion of the agent population has adhered to some common behaviour. Researchers have employed different techniques to develop norm emergence. Of these, maybe the simplest has been *imitation*, which is based on the principle *"When in Rome, do as the Romans"*. Imitation considers that agents learn how to behave (that is, which norms to follow) by imitating the most commonly followed pattern of behaviour. This implies that the agents are endowed with capabilities to recognise and to reason about other agents' behaviours. In this sense, imitation can be regarded as a *spreading* mechanism whereby a norm can be propagated through an agent society.

More sophisticated spread-based techniques have employed Learning. In this approach, agents are endowed with the computational capability to *learn* which norms are better to achieve their goals. The key issue is to design a *strategy update function* that represents an agent's decision-making process. This function considers what the agent has observed so far to decide its actions. A paradigmatic example is the work by Shoham and Tennenholtz [Shoham and Tennenholtz, 1992a], which employed a *reinforcement learning* algorithm to allow agents to reach an agreement on norms. This work inspired later works like [Walker and Wooldridge, 1995], which empirically evaluated sixteen different strategies for the emergence of norms. In a more recent work, Pujol et al. [Pujol, 2006] employed Shoham's algorithm as a mechanism for the emergence of social conventions.

Tag-based cooperation [Riolo et al., 2001, Traulsen and Schuster, 2003, Axelrod et al., 2004] has also been employed to spread norms through an agent society. Briefly, in this approach each agent cooperates with other agents that

are sufficiently similar to itself according to some arbitrary characteristic, or *tag*. Along these lines, Griffiths et al. [Griffiths and Luck, 2010, Griffiths and Luck, 2011] investigated on the issue of group recognition using tag-based cooperation as the interaction model. In this work, norms are in fact tags. The main drawback of this approach is that only the agents that have a certain tag possess the corresponding norm. Moreover, it does not allow multiple norms to co-exist in a group.

The main disadvantage of spread-based mechanisms is that they limit themselves to analysing how a *single* norm can be spread through an agent population. In an open MAS, though, typically there will be a wide space of possible norms that may be spread, and thus the agents should be able to adopt several norms. With this problem in mind, Salazar et al. [Salazar et al., 2010a] proposed an emergence mechanism whereby agents could explore new norms at runtime, and several norms can be spread through an agent population. Nevertheless, what is common to all emergence approaches is that: (1) they require the agents to be endowed with the computational capability to synthesise norms; and (2) they assume that the agents will collaborate in norm synthesis. As argued in Section 1.1.2, these assumptions cannot be taken in open MAS, whose agents' computational capabilities and behaviours cannot be predicted.

As an alternative to norm emergence, other works have studied how norms can be synthesised at runtime through an explicit *negotiation* process [Rosenschein, 1986, Sycara, 1988, Rosenschein and Zlotkin, 1994, Boella and Van Der Torre, 2007]. In norm negotiation, agents negotiate on how to coordinate their activities. Negotiation can be thus seen as a process whereby the agents agree on a (maybe non-explicitly represented) norm that coordinates their interactions to achieve their goals. Alternatively, Artikis et al. [Artikis et al., 2009] investigated *norm agreement*, in which empowered members use a meta-level *argumentation* protocol to modify norms at run-time. Nevertheless, both negotiation and agreement approaches have two common drawbacks: (1) they assume that agents are enriched with negotiation/agreement capabilities, and (2) they assume that agents will cooperate in a negotiation/agreement process. Again, these assumptions are against the nature of open MAS.

Finally, further research has focused on *norm learning*. Bou et al. [Bou et al., 2007a, Bou et al., 2006, Bou et al., 2007b] studied how norms may be adapted at runtime by a regulative institution. They provided mechanisms whereby the computational environment hosting the agents (an Electronic Institution) monitors the agents' behaviour at runtime, and synthesises norms in order to achieve some institutional goals. They employed learning techniques – such as Genetic Algorithms (GA) – to *learn* how to synthesise norms. Along these lines, Campos et al. [Campos et al., 2008, Campos et al., 2009, Campos et al., 2011] studied how to adapt norms' parameters to the changes of agents' behaviours at runtime. They proposed to endow a MAS organisation (e.g., Electronic Institutions) with capabilities to adapt norms to agents, instead of having agents adapt to norms. They employed Case-Based Reasoning (CBR) as the learning technique. However, neither of these approaches considered the creation of *new norms* at

runtime, but they simply limit themselves to adapt an initial set of norms at runtime.

Synthesis making

Research on norm synthesis can be also classified based on *the actors* involved in the synthesis making, namely based on who synthesises norms. We here consider that norm synthesis can be either *endogenous* or *exogenous*. Endogenous synthesis considers that norms are internalised by agents, who are in charge of synthesising their own norms. This approach mainly applies to on-line synthesis techniques such as norm emergence [Shoham and Tennenholtz, 1992a, Kittock, 1993, Kittock, 1995, Griffiths and Luck, 2010, Villatoro et al., 2011], norm negotiation [Boella and Van Der Torre, 2007], and norm agreement [Artikis et al., 2009]. Endogenous approaches require the agents within a MAS' domain to be endowed with computational capabilities to synthesise norms. Nevertheless, this cannot be assumed in open MAS, and hence endogenous synthesis is not appropriate for open MAS.

Alternatively, exogenous synthesis is agnostic about norm internalisation, and considers that norms are synthesised by some regulatory entity, which may be an external regulatory mechanism, or even a group of regulatory agents. Thus, in exogenous synthesis, agents within a MAS' domain are not assumed to be capable of synthesising their own norms. In the literature, exogenous synthesis mainly maps to off-line design approaches, in which either a human designer or a computational mechanism defines norms at design time. However, as argued previously, off-line approaches are not appropriate for open MAS. The only on-line synthesis approaches that have used exogenous synthesis are those in Bou et al [Bou et al., 2007a, Bou et al., 2007b] and Campos et al. [Campos et al., 2008, Campos et al., 2011]. There, norms are synthesised by either a computational mechanism ([Bou et al., 2007a]), or a group of special agents that do not belong to the population to regulate, and who are in charge of synthesising norms [Campos et al., 2008, Campos et al., 2011].

Synthesis scope

In short, we consider that a synthesis scope may be either *constrained* or *unconstrained*. A synthesis scope is constrained whenever norm synthesis limits itself to exploiting an initial set of alternative norms. In that case, the amount of normative systems that can be explored is limited to all possible combinations between those norms. Norm emergence is an illustrative example. With the exception of [Salazar et al., 2010a], norm emergence techniques typically consider an initial set of alternative norms, and the emergence process limits itself to exploit this initial set. Eventually, the system may converge either to a norm, or a combination of these norms. However, norm emergence techniques cannot explore alternative norms, and thus the space of normative systems they can explore is constrained. This may be a problem if the normative system that would successfully regulate a MAS is *out of scope*, namely it considers norms

that are not available in the initial set of norms.

Alternatively, an unconstrained synthesis scope implies that there are no restrictions, and any normative system may be synthesised. In the literature, this approach mainly applies to off-line design techniques. Briefly, such techniques consider a designer that may synthesise any possible normative system before the MAS starts operating. However, it is worth to notice that once the system has started running, the normative system cannot be changed, and hence no alternative normative system can be explored. Again, this is not appropriate for open MAS. To summarise, in the literature no on-line synthesis mechanism has been proposed that can explore any possible normative system at runtime, namely that has an unconstrained scope.

Synthesis objectives

A key issue that has been scarcely studied in norm synthesis is the different objectives that a norm synthesis process may consider. Typically, the literature has considered the goal of synthesising *effective* norms, which is crucial to assess to what extent norms are successful in helping a MAS coordinate. Shoham and Tennenholtz [Shoham and Tennenholtz, 1992b, Shoham and Tennenholtz, 1995] proposed the design of social laws that effectively coordinate an agent society. However, they also considered the objective of allowing agents enough freedom to achieve their goals. In other words, the goal of not to over-restrict agents' behaviours. Thus, this work was maybe the first one considering the *liberality* of the agents as a synthesis objective. Briefly, liberality is concerned with preserving the agents' freedom to the greatest possible extent while regulating their behaviour to achieve MAS coordination. Later, Fitoussi and Tennenholtz [Fitoussi and Tennenholtz, 2000] introduced *minimality* and *simplicity* as objectives for the off-line design of norms. While minimality attempts to reduce agents' computational efforts when reasoning about norms, simplicity is concerned with preserving their freedom to the greatest possible extent. Therefore, minimality can be understood as an attempt to synthesise *compact* normative systems, and simplicity can be understood as a means to synthesise *liberal* normative systems.

In this dissertation, effectiveness, compactness, and liberality are considered as crucial synthesis objectives. These three objectives have been considered in the literature that focus on off-line design, but no contribution has been made that considers them together as on-line norm synthesis objectives.

2.4.2 Analysing research on norm synthesis

Table 2.1 provides an overview of selected norm synthesis contributions, and classifies them based on the different dimensions analysed in Section 2.4.1. By observing this table, we draw the following conclusions:

1. *Off-line design approaches are only appropriate for closed systems.* Research in off-line design has mainly focused on *closed* systems [Shoham and Tennenholtz, 1995, Fitoussi and Tennenholtz, 2000, Goldman and Rosenschein, 1993, van Der Hoek et al., 2007]. The main reason of this is that

	System type		**Synthesis time**		**Synthesis making**		**Synthesis scope**		**Synthesis objectives**		
	Closed	Open	Design time	Run time	Endogenous	Exogenous	Constrained	Unconstrained	Effectiveness	Compactness	Liberality
[Shoham and Tennenholtz, 1995]	✓		✓			✓		✓	✓		✓
[Fitoussi and Tennenholtz, 2000]	✓		✓			✓		✓		✓	✓
[Goldman and Rosenschein, 1993]	✓		✓			✓		✓	✓		
[van Der Hoek et al., 2007]	✓		✓			✓		✓	✓		
[Shoham and Tennenholtz, 1992a]	✓			✓	✓		✓		✓		
[Walker and Wooldridge, 1995]	✓			✓	✓		✓		✓		
[Kittock, 1995]	✓			✓	✓		✓		✓		
[Epstein, 2001]	✓			✓	✓		✓		✓		
[Pujol, 2006]	✓			✓	✓		✓		✓		
[Boella and Van Der Torre, 2007]	✓			✓	✓		✓		✓		
[Sen and Airiau, 2007b]		✓		✓	✓		✓		✓		
[Griffiths and Luck, 2010]		✓		✓	✓		✓		✓		
[Artikis et al., 2009]		✓		✓	✓		✓		✓		
[Salazar et al., 2010a]		✓		✓	✓			✓	✓		
[Bou et al., 2007a]		✓		✓		✓	✓		✓		
[Campos et al., 2011]		✓		✓		✓	✓		✓		

Table 2.1: Classification of selected works on norm synthesis based on the different dimensions analysed in Section 2.4.1.

off-line design is not appropriate for synthesising norms for open MAS. Approaches based on off-line design are computationally infeasible, and make assumptions that go against the nature of open MAS – that the norms can be hard-wired into the agents' machinery; and that the agents' behaviours are predictable at design time.

2. *Existing on-line synthesis mechanisms are mainly endogenous.* Most on-line synthesis mechanisms in the literature assume that the agents within a MAS synthesise their own norms, either by means of norm emergence [Shoham and Tennenholtz, 1992a, Kittock, 1995, Walker and Wooldridge, 1995, Epstein, 2001, Pujol, 2006, Sen and Airiau, 2007b, Griffiths and Luck, 2010], through norm negotiation [Boella and Van Der Torre, 2007] or by means of norm agreement [Artikis et al., 2009]. The only exogenous norm synthesis approaches in the literature are those on-line in [Bou et al., 2007a, Campos et al., 2011]. Nevertheless, as detailed above, these approaches have a constrained synthesis scope.

3. *Existing on-line synthesis approaches have a constrained synthesis scope.* Generally, the literature provides on-line synthesis approaches with constrained scopes. Existing approaches for norm emergence, norm negotiation, norm agreement, and norm learning consider an initial set of norms that they exploit to converge to either a single norm, or a normative system. One exception is Salazar et al. [Salazar et al., 2010a], which can explore new norms at runtime. Alternatively, all off-line design approaches have unconstrained scopes, yet they are not appropriate for open MAS.

4. *No on-line work has considered multi-objective synthesis.* With the exception of [Shoham and Tennenholtz, 1992b], which considered effectiveness and liberality as synthesis objectives, and [Fitoussi and Tennenholtz, 2000], which considered the synthesis of compact and liberal normative systems, the literature has mainly considered effectiveness as a driving for norm synthesis. Furthermore, no approach has been proposed that considers together effectiveness, compactness and liberality as synthesis objectives.

2.5 Conclusions

In the literature, off-line design approaches have mainly focused on norm synthesis for closed systems. As for open systems, on-line synthesis is the only feasible approach. Existing on-line synthesis approaches are mainly endogenous, that is, they require the agents within a MAS to synthesise their own norms. This implies assuming that agents are capable of synthesising norms, and that they are benevolent and will choose to cooperate in norm synthesis. These assumptions go against the nature of open MAS, which are populated by heterogeneous, self-interested agents. Against this background, here we argue that norm synthesis in open MAS should be performed exogenously.

Moreover, existing on-line synthesis approaches have constrained synthesis scopes, namely can only explore normative systems that are combinations of an initial set of norms. This represents a disadvantage in the sense that they may never *find* the normative system that effectively will coordinate a given MAS if it is out of scope. Finally, no on-line synthesis approach has considered the three key objectives of effectiveness, compactness, and liberality to drive norm synthesis. We consider that these three synthesis objectives are crucial and they should be all considered together during norm synthesis.

At this point, the necessary background has been provided to argue that no computational mechanism has been proposed to synthesise norms for open MAS while having the following characteristics:

1. The capability to synthesise norms at runtime, namely **on-line**.
2. The capability to synthesise norms **exogenously**, that is, without requiring domain agents to synthesise their own norms.
3. An **unconstrained** scope that allows it to explore any possible normative system.
4. A **multi-objective** approach to norm synthesis that considers the key objectives of synthesising effective, compact, and liberal normative systems.

Therefore, the remainder of this dissertation is focused on introducing such mechanism, answering each research question proposed in Section 1.2.

Chapter 3

Synthesising norms for open Multi-Agent Systems

3.1 Introduction

In the previous chapter we argued that the *norm synthesis problem*, namely the problem of synthesising the norms that will coordinate agents' activities within an open MAS, still remains an open problem. Particularly, we argued that the literature does not provide any computational approach to perform *multi-objective* norm synthesis for open MAS at *runtime*, and without involving the agents in the norm synthesis, namely *exogenously*. Moreover, existing on-line norm synthesis approaches have *constrained* synthesis scopes. That is, they limit themselves to exploit an initial set of norms, thus only being able to synthesise normative systems that are combinations of those norms.

Against this background, this chapter introduces a computational model along with an architecture to perform *exogenous*, *on-line* and *multi-objective* norm synthesis for open MAS. In this model, the synthesis scope is *unconstrained*, since new norms can be synthesised (explored) at runtime. The agents within a MAS are not required to synthesise their own norms. Instead, a synthesis machine, the so-called *Norm Synthesis Machine* (NSM), observes the agents' interactions at runtime, and iteratively executes a *synthesis strategy* to decide (i) *when* it is necessary to synthesise *new* norms and to discard synthesised norms; (ii) *how* to create these norms; and (iii) *which* normative system (i.e., set of norms) must be provided to the agents at each time. A NSM is aimed at producing norms that effectively coordinate agents' interactions as long as the agents comply with them. Furthermore, a NSM is designed to be *domain-independent* in order to be applicable to different application domains by providing little domain-dependent information. With all this, this chapter aims at answering research question R1, introduced in Section 1.2.

The remainder of this chapter is organised as follows. Section 3.2 introduces the general framework that will underpin the rest of this work and provides

some definitions to better characterise the norm synthesis problem at hand. Section 3.3 introduces an abstract and domain-independent computational model to perform exogenous, on-line and multi-objective norm synthesis in open MAS. Finally, Section 3.4 provides an abstract architecture to support the implementation of such norm synthesis model.

3.2 Basic definitions and problem statement

This section is structured as follows. Section 3.2.1 introduces a formal framework to be used throughout this dissertation, and provides some basic definitions. Thereafter, Section 3.2.2 provides the formal definition of the norm synthesis problem at hand.

3.2.1 Basic definitions

Let us consider a Multi-Agent System composed of a finite set of agents $Ag = \{ag_1, \ldots, ag_m\}$ with a shared finite set of actions $Ac = \{ac_1, \ldots, ac_n\}$ that these agents can perform. Let S be the set of all possible *states* of the MAS. Let $\mathbb{T} : S \times Ac^{|Ag|} \to S$ be a *state transition function* that leads the MAS to a state s' from a state s after the agents perform a set of actions $A \in Ac^{|Ag|}$. The differences between these states capture the changes that occurred when the MAS evolved from s to s'. For convenience, Ac includes a special action *nil* that stands for not performing any action.

To describe the states of a MAS, let us consider two different languages. On the one hand, a *MAS language* $\mathcal{L}_{MAS}$ describes the states of a MAS from a global, external observer's perspective. On the other hand, an *agent language* $\mathcal{L}_{Ag}$ describes MAS states from an agent's local, individual perspective. Both languages are first-order languages with grounded terms containing the standard classical connectives, and a notion of consequence defined for it via a relation $\models$. Hereafter, given a state $s \in S$, d_s will denote an expression of $\mathcal{L}_{MAS}$ that describes a MAS from a global perspective. In particular, a MAS state description (d_s) is a set of first-order predicates of the form $p(\tau_1, \ldots, \tau_k)$, with p being a predicate symbol and $\tau_1, \ldots, \tau_k$ being terms of $\mathcal{L}_{MAS}$.

Now, let us introduce a running example to be used throughout the remainder of this dissertation. With this aim, we assume a further simplification of the traffic scenario described in Section 1.3.1. Let us consider a traffic junction with two lanes, modelled as a 3×3 grid. Figure 3.1a shows a graphical representation of this scenario, with two cars (car a and car b) going through an intersection. The actions available to the cars are $Ac = \{go, stop\}$. In particular, action *go* means that a car moves forward towards its target direction at a constant speed of 1 cell per move (in particular it may include turns); and *stop* means that a car remains stopped at its position. Figure 3.1a shows a state of the junction (state s). Its description (d_s) would be composed of the predicates describing *each cell in the grid* from a external observer's perspective. These predicates are described in terms of a language $\mathcal{L}_{MAS}$ with one unary predicate symbol

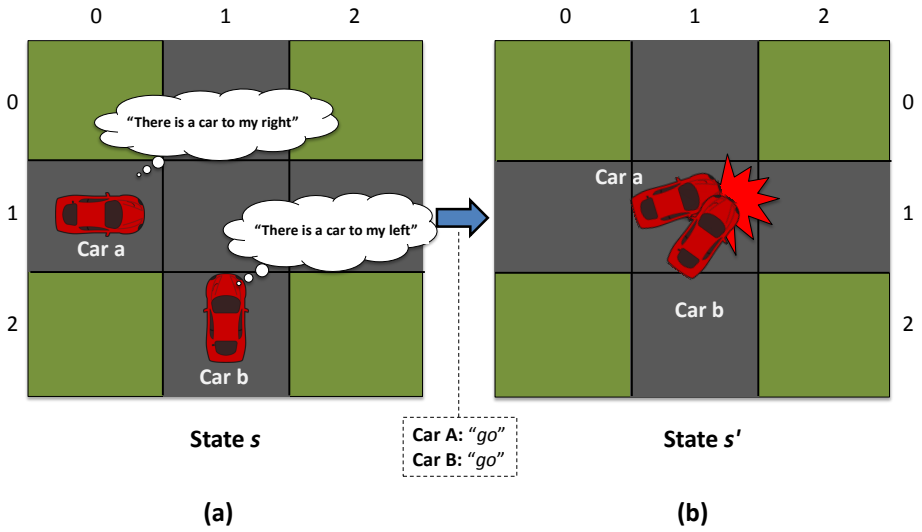

Figure 3.1: Collision between two cars in a traffic junction. In state s, car a has context *"there is a car to my right"*, and car b has context *"there is a car to my left"*.

"cell", used to describe the contents in each particular cell of the scenario. This predicate may have the following terms:

- Two numerical values representing the coordinates (x, y) of a cell.
- One term out of a set of terms {*car-heading-north, car-heading-east, car-heading-south, car-heading-west, wall, nil*}, which represent either a car heading towards different cardinal points, a wall, or nothing.
- A numerical value that represents the *identifier* of a car.

With this $\mathcal{L}_{MAS}$ language, state s in Figure 3.1a could be described as follows:

$$
\begin{aligned}
d_s = \{ \quad & \textit{cell(0,0,wall)}, && \textit{cell(0,1,nil)}, && \textit{cell(0,2,wall)}, \\
& \textit{cell(1,0,car-heading-east,a)}, && \textit{cell(1,1,nil)}, && \textit{cell(1,2,nil)}, \\
& \textit{cell(2,0,wall)}, && \textit{cell(2,1,car-heading-north,b)}, && \textit{cell(2,2,wall)} \quad \}
\end{aligned}
$$

The notion of *partial view* of some state s can be captured as a sub-expression (subset) of d_s. For instance, given expression d_s above, an example of partial view of s could be a subset of d_s describing a subgroup of cells in s. Hereafter, ν_i^s will stand for the i-th partial view of state s. Thus, a given state s could be described as a set of partial views $\{\nu_1^s, \ldots, \nu_j^s\}$ such that $\nu_i^s \cap \nu_j^s = \emptyset$ and $d_s = \{\bigcup_{i=1}^{j} \nu_i^s\}$.

Each agent has her own local view of the state of the system that she is part of. For instance, car a located in the junction in Figure 3.1a will have its own local perception of the junction, which corresponds to the perception of cars in its vicinity, without including other cars further away in the road. In this model, an *agent's context* is an expression of $\mathcal{L}_{Ag}$ that represents the agent's representation of a MAS from her local perspective. Formally, the local representation (context) of an agent is defined as:

Definition 1 (Agent context)**.** *Given a state $s \in S$ and an agent $ag \in Ag$ that is part of s, function context : $Ag \times S \rightarrow \mathcal{P}(\mathcal{L}_{Ag})$ yields the context of ag in s, namely a description of state s from ag's local perspective.*

An agent's context is a set of first-order predicates of the form $p(\tau_1, \ldots, \tau_k)$, with p being a predicate symbol and $\tau_1, \ldots, \tau_k$ being terms of $\mathcal{L}_{Ag}$. In our running example, language $\mathcal{L}_{Ag}$ contains three unary predicate symbols $\{left, front, right\}$ employed to represent what occupies the three road positions (i.e., cells) in front of a car from its perspective. Each predicate has a single term from {*car-heading-left, car-heading-right, car-opposite-heading, car-same-heading, wall, nil*}, representing either a *car* heading to different directions from the perspective of a reference car, a wall, or nothing. With this language, the context of car a in state s in Figure 3.1a can be described as follows:

$$\textit{context(a,s)} = \{\textit{left(nil), front(nil), right(car-heading-left)}\}$$

At each moment, agents can perform actions that change the state of a MAS. For instance, in Figure 3.1, if cars a and b perform action *go*, this leads to a different state s' (Figure 3.1b) in which the two cars have changed their positions. Assuming that agents' actions are *observable*, it is possible to infer the action that an agent performed during the transition between a state and its subsequent state by computing the differences between those states. Next, we define a function that yields the action that an agent performed during a state transition.

Definition 2 (Agent action)**.** *Given an agent ag, and a state transition $\langle s, A, s' \rangle$, function action : $Ag \times S \times S \rightarrow Ac$ yields the action that agent ag performed in the transition from s to s'.*

As an example, by observing the differences between state s and state s' in Figure 3.1, we can infer that both cars performed action *go*, since during the state transition each car advanced one cell following its direction.

Let $C \subseteq S$ be a set of *undesirable states* of a MAS. In particular, we say that a MAS state is undesirable if it is detrimental to achieve coordination, namely if it contains *conflicts*. Considering our running example, and the goal of avoiding car collisions, the set of undesirable states would correspond to those states containing collisions, and each particular collision in a given state would be regarded as a particular conflict. For instance, state s' in Figure 3.1b is undesirable, since it contains a collision between cars a and b. The notion of conflict in some state can be captured as a sub-expression of its representation that describes a

particular conflict (e.g., a collision). Considering that $d_{s'}$ describes state s', a particular conflict in s' would be a sub-expression of $d_{s'}$ that describes the cars colliding in cell $(1, 1)$.

Definition 3 (Conflict)**.** *Let $s \in C$ be an undesirable MAS state, and d_s an expression that describes s in terms of $\mathcal{L}_{MAS}$. A conflict $c \subseteq d_s$ is a sub-expression of d_s describing an situation in s that is detrimental to achieve coordination.*

Notice therefore that an undesirable state must at least contain a conflict. Hereafter, c_i^s will stand for the i-th conflict of state s. Considering undesirable state s' in Figure 3.1b, the conflict in such state could be described as follows:

$$c_1^{s'} = \{\ \textit{cell(1,1,car-heading-east,a)},\ \textit{cell(1,1,car-heading-north,b)}\ \}$$

which describes cars a and b occupying the same cell of the junction (i.e., a collision between the two cars). Next, we define a function that we can use to detect conflicts in a given state.

Definition 4 (Conflict detection)**.** *Given an undesirable state $s \in C$ with representation d_s, and a partial view ν_i^s, the conflict detection function conflict : $C \times \mathcal{P}(\mathcal{L}_{MAS}) \rightarrow \{true, false\}$ indicates whether the partial view contains a conflict. If so, $conflict(s, \nu_i^s) = true$ and we say that ν_i^s contains a conflict in s.*

Given an undesirable state, and a particular conflict of that state, we say that the agents described in the conflict are *conflicting agents*. Formally:

Definition 5 (Conflicting agents)**.** *Given an undesirable state $s \in C$, and a partial view c_i^s representing a conflict, we note the set of agents involved in the conflict as $agents(c_i^s)$.*

As an example, if we consider conflict $c_1^{s'}$ introduced above, its set of conflicting agents is $agents(c_1^{s'}) = \{a, b\}$.

Next, our notion of norm is introduced. As detailed in Section 2.3, this dissertation focuses on the synthesis of *regulative norms*. Thus, a norm is a deontic statement that imposes restrictions (i.e., obligations, permissions and/or prohibitions) on agents' behaviours whenever some pre-conditions are fulfilled. Also, as argued in Section 2.3.2, the norms of a NMAS must be presented in such a way that the agents are able to interpret and reason about them. Therefore, norms are described from the agents' perspective, and thus expressed as formulae of $\mathcal{L}_{Ag}$.

Definition 6 (Norm)**.** *A* norm *is a pair $\langle \varphi, \theta(ac) \rangle$ where $\varphi \in \mathcal{L}_{Ag}$ stands for the precondition of the norm, $ac \in Ac$ is an action available to the agents, and $\theta \in \{obl, perm, prh\}$ is an atomic deontic operator: obl indicates an* obligation*, perm indicates a* permission*, and prh indicates a* prohibition*.*

More concretely, we assume that the precondition φ is a set of first-order predicates $p(\tau_1, \ldots, \tau_k)$, with p being a predicate symbol and $\tau_1, \ldots, \tau_k$ being

terms of $\mathcal{L}_{Ag}$ (the set of predicates represents their conjunction); and $\theta(ac)$ is an atomic deontic formula. Next, we provide a formal definition of a *normative system* as a set of norms.

Definition 7. *A normative system is a possibly empty set of norms* $\Omega = \{\langle\varphi_1, \theta_1(ac_1)\rangle, \ldots, \langle\varphi_l, \theta_l(ac_l)\rangle\}$. *In particular,* $\Omega = \emptyset$ *stands for a normative system with no norms.*

In particular, we regard a norm as a *soft constraint* on the behaviours of agents. Thus, agents are free to decide whether to comply or not with the norms that *apply to* them. An agent $ag \in Ag$ evaluates whether a norm $n = \langle\varphi, \theta(ac)\rangle$ applies to her in a state s by checking if her context in state s satisfies the precondition of norm n. In this case, norm n applies to agent ag and the deontic expression $\theta(ac)$ holds for her. Formally:

Definition 8 (Norm applicability)**.** *Let* s *be a MAS state,* ag *an agent that is part of* s*, and* $n = \langle\varphi, \theta(ac)\rangle$ *a norm. We say that* n *applies to* ag *in state* s *iff* $context(ag, s) \models \varphi$.

Using our running example, a norm could be defined to prohibit car a in Figure 3.1a to go forward if it observes a car to its right (hence giving way to it), and nothing to its front and right positions as follows:

$$n : \langle\{left(nil), front(nil), right(car\text{-}heading\text{-}left)\}, prh(go)\rangle$$

The precondition φ contains two predicates $left(nil)$ and $front(nil)$ that are *true* when the positions to the left and to the front of the reference car are empty; and a predicate $right(car\text{-}heading\text{-}left)$ that is *true* if there is a car to the right of the reference car which is heading to its left, from the perspective of the reference car. So, if the reference car's context is $\{left(nil), front(nil), right(car\text{-}heading\text{-}left)\}$, then, $context(ag, s) \models \varphi$ holds, and $prh(go)$ applies to the agent. Note that this norm is similar to the *"Give way to your right"* norm employed in human societies.

Now that we know the concepts of conflict and norm applicability, let us introduce the concept of *unregulated conflict*. We say that a conflict is unregulated iff, for each agent involved in the conflict, no norms applied to the agent in the state previous to the state containing the conflict. Formally:

Definition 9 (Unregulated conflict)**.** *Let* $\langle s, A, s'\rangle$ *be a transition state such that* s' *is undesirable,* $c^{s'}$ *a conflict in state* s'*, and* $\Omega = \{\langle\varphi_1, \theta_1(ac_1)\rangle, \ldots, \langle\varphi_j, \theta_j(ac_j)\rangle\}$ *a normative system. We say that* $c^{s'}$ *is unregulated iff for each agent* $ag \in agents(c^{s'})$*,* $context(ag, s) \not\models \varphi_i$ *for each norm* $\langle\varphi_i, \theta_i(ac_i)\rangle \in \Omega$.

Analogously, we say that a conflict is *regulated* iff, for some agent involved in the conflict, some norm applied to the agent in the state previous to the state containing the conflict. Formally:

Definition 10 (Regulated conflict)**.** *Let* $\langle s, A, s' \rangle$ *be a transition state such that* s' *is undesirable,* $c^{s'}$ *a conflict in state* s'*, and* $\Omega = \{\langle \varphi_1, \theta_1(ac_1) \rangle, \ldots, \langle \varphi_j, \theta_j(ac_j) \rangle\}$ *a normative system. We say that* $c^{s'}$ *is regulated iff there is at least an agent* $ag \in agents(c^{s'})$ *such that* $context(ag, s) \models \varphi_i$ *for some norm* $\langle \varphi_i, \theta_i(ac_i) \rangle \in \Omega$*.*

Now, let us formally define *norm compliance*. Given a state transition $\langle s, A, s' \rangle$, and a norm n that applies to an agent ag in s, the agent is considered to have fulfilled n during the state transition iff (i) she carried out an action that is obliged by n; or (ii) did not perform an action prohibited by n.

Definition 11 (Norm fulfilment)**.** *Given a state transition* $\langle s, A, s' \rangle$*, an agent* ag*, and a norm* $\langle \varphi, \theta(ac) \rangle$ *applicable to* ag *in* s*, we say that* ag *fulfilled the norm during the transition from* s *to* s' *iff (i)* $\theta = prh$ *and* $action(ag, s, s') \neq ac$*; or (ii)* $\theta = obl$ *and* $action(ag, s, s') = ac$*.*

Analogously, the agent is considered to have infringed the norm iff, during the state transition, (i) she performed an action that is prohibited by the norm; or (ii) she did not perform an action mandated by the norm.

Definition 12 (Norm infringement)**.** *Given a state transition* $\langle s, A, s' \rangle$*, an agent* ag*, and a norm* $\langle \varphi, \theta(ac) \rangle$ *applicable to* ag *in* s*, we say that* ag *infringed the norm during the transition from* s *to* s' *iff (i)* $\theta = prh$ *and* $action(ag, s, s') = ac$*; or (ii)* $\theta = obl$ *and* $action(ag, s, s') \neq ac$*.*

Given a state transition $\langle s, A, s' \rangle$, and an agent ag that fulfilled a norm n in the transition from s to s', we say that the fulfilment of norm n is:

- *Successful* if agent ag *is not part of* a conflict in state s'.
- *Harmful* if agent ag *is part of* a conflict in state s'.

For instance, in the road traffic scenario, a fulfilment would be successful if a car avoided colliding by fulfilling a norm, and it would be harmful if the car collided after the norm fulfilment. Formally:

Definition 13 (Successful fulfilment)**.** *Let* $\langle s, A, s' \rangle$ *be a state transition,* $d_{s'}$ *a description of* s' *in terms of language* $\mathcal{L}_{MAS}$*,* ag *an agent, and* $n = \langle \varphi, \theta(ac) \rangle$ *a norm such that* ag *fulfilled norm* n *in the transition from* s *to* s'*. We say that the fulfilment of norm* n *was successful if either (i)* $s \notin C$*, or (ii)* $s \in C$ *and, for each* c_i^s *in* s*,* $ag \notin agents(c_i^s)$*.*

Definition 14 (Harmful fulfilment)**.** *Let* $\langle s, A, s' \rangle$ *be a state transition,* $d_{s'}$ *a description of* s' *in terms of language* $\mathcal{L}_{MAS}$*,* ag *an agent, and* $n = \langle \varphi, \theta(ac) \rangle$ *a norm such that* ag *fulfilled norm* n *in the transition from* s *to* s'*. We say that the fulfilment of norm* n *was harmful iff* $s \in C$ *and there is a conflict* c_i^s *such that* $ag \in agents(c_i^s)$*.*

Analogously, given a state transition $\langle s, A, s' \rangle$, and an agent ag that infringed a norm n in the transition from s to s', we say that the infringement of norm n is:

- *Successful* if agent *ag is not part of* a conflict in state s'.
- *Harmful* if agent *ag is part of* a conflict in state s'.

In our road traffic example, a successful infringement can be regarded as a car that infringes a norm and does not collide with other cars, and a harmful infringement may be a car that infringes a norm and collides. Formally:

Definition 15 (Successful infringement)**.** *Let $\langle s, A, s' \rangle$ be a state transition, $d_{s'}$ a description of s' in terms of language $\mathcal{L}_{MAS}$, ag an agent, and $n = \langle \varphi, \theta(ac) \rangle$ a norm such that ag infringed norm n in the transition from s to s'. We say that the infringement of norm n was successful if either (i) $s \notin C$, or (ii) $s \in C$ and, for each c_i^s in s, $ag \notin agents(c_i^s)$.*

Definition 16 (Harmful infringement)**.** *Let $\langle s, A, s' \rangle$ be a state transition, $d_{s'}$ a description of s' in terms of language $\mathcal{L}_{MAS}$, ag an agent, and $n = \langle \varphi, \theta(ac) \rangle$ a norm such that ag infringed norm n in the transition from s to s'. We say that the infringement of norm n was harmful iff $s \in C$ and there is a conflict c_i^s such that $ag \in agents(c_i^s)$.*

3.2.2 Research problem

This dissertation focuses on Normative Multi-Agent Systems, particularly those whose population is *open*. As described in Section 2.2.2, a NMAS is a Multi-Agent System whose agents have their actions regulated by some normative system they are aware of. Moreover, the system itself can assess whether and to whom norms in the normative system apply. The particular notion of NMAS that we adopt in this dissertation is the one that follows.

Definition 17 (Normative MAS)**.** *A Normative Multi-agent System (NMAS) is a tuple $\langle Ag, Ac, \Omega, \mathcal{L}_{Ag}, S, s_0, s_c \rangle$, where: (i) Ag is a set of agents; (ii) Ac is a set of actions; (iii) Ω is a normative system, whose norms are expressed in terms of the agent language $\mathcal{L}_{Ag}$; (iv) S is a set of the states; (v) s_0 is the initial state; and (vi) s_c is the current state at a given moment.*

Hereafter, for the sake of simplicity, we will refer to a Normative Multi-Agent System (NMAS) as a MAS.

As detailed In Section 1.1, the norm synthesis problem is that of computing the norms that will allow the agents in a MAS to effectively coordinate their interactions. Therefore, Section 1.1.3 introduced the goal of synthesising *effective* normative systems. In this dissertation, a normative system is considered to be effective if the norms it contains are effective to avoid *conflicts* (c.f. Definition 3). For instance, in the traffic scenario, a normative system will be effective if its norms avoid collisions. Note therefore that the synthesis objective of effectiveness can be achieved by synthesising norms that are *effective*. Thus, we consider a *norm evaluation criterion* – the so-called *effectiveness* – to evaluate to what extent a norm is successful in avoiding conflicts. This norm evaluation criterion will be employed to pursue the synthesis of effective normative systems.

Also, Section 1.1.3 introduced the goal of synthesising *liberal* normative systems that give to agents enough freedom to interact. In this dissertation, liberality is approached as the avoidance of *over-regulation*, namely to avoid to impose on the agents more restrictions than strictly necessary to avoid conflicts. That is, the less *unnecessary* norms a normative system contains, the more liberal it is. Note therefore that the synthesis objective of liberality can be achieved by synthesising norms that are *necessary*. Thus, we consider a second norm evaluation criterion – the so-called *necessity* – to evaluate to what extent a norm is necessary to avoid conflicts. Such norm evaluation criterion will be employed to pursue the synthesis of liberal normative systems.

However, a third synthesis objective, the so-called *compactness*, was also introduced in Section 1.1.3. In short, compactness is regarded with reducing the norms' reasoning costs. Thus, the more compact a normative system, the *easier* to reason about its norms. In this thesis, the compactness of a normative system is not pursued through the individual evaluation of its norms. In other words, we do not consider a norm evaluation criterion to pursue compactness. Instead, we aim at achieving compactness by reducing the amount of norms that a normative system contains, as well as the amount of predicates and terms that norms contain. Overall, the norm evaluation criteria of effectiveness and necessity allow to synthesise effective and liberal normative systems, and reducing the amount of norms in a normative system and simplifying norms' representations allow to achieve compactness.

At this point, enough background has been provided to characterise the norm synthesis problem in the context of (normative) open MAS. As detailed in Chapters 1 and 2, within an open MAS, agents' behaviours are unpredictable and can change at runtime. Therefore, a normative system should be adapted at runtime to these changes, searching for a normative system that effectively regulates conflicts. In this thesis, the norm synthesis problem does not consist in finding a particular (target) normative system, but in finding a normative system that is *successful enough*. In particular, a normative system is considered as successful enough if, for a long enough period of time:

1. It remains *stable* (unchanged).
2. The norms it contains are successful enough in terms of some norm evaluation criteria (e.g., effectiveness, necessity).

Formally, an evaluation criterion is a pair $\langle \mu_i, \alpha_i \rangle$, where:

- $\mu_i : \Omega \times \mathbb{N} \times C \rightarrow [0, 1]$ is a function that, given a norm of a normative system Ω, a time step t, and a set C of conflicting states, returns the *utility* of the norm at time t in terms of the i-th norm evaluation criterion (e.g., effectiveness).
- α_i stands for a threshold that sets a satisfaction degree for the i-th norm evaluation criterion. Such threshold sets the minimum utility a norm must have in terms of such criterion to be part of the normative system.

Thus, we consider that a *norm's performance* is evaluated in terms of a non-empty set of norm evaluation criteria $\mathcal{EC} = \{\langle \mu_1, \alpha_1 \rangle \ldots, \langle \mu_r, \alpha_r \rangle\}$. Formally, the norm synthesis problem is:

Definition 18 (Norm Synthesis Problem)**.** *Let $\langle Ag, Ac, \Omega, \mathcal{L}_{Ag}, S, s_0, s_c \rangle$, be a MAS, $C \subseteq S$ a set of undesirable (conflicting) states, ψ a function that returns the normative system at each given time t, t_0 the time at which the MAS starts operating, and $T = [t_{begin}, \ldots, t_{end}]$ a time interval. The norm synthesis problem (NSP) amounts to finding a normative system $\bar{\Omega}$ such that for some $t_{begin} \geq t_0$, and for all t in T, the following conditions hold: (1) for each norm $n \in \psi(t)$, $\mu_i(n, t, C) > \alpha_i$ for all $\langle \mu_i, \alpha_i \rangle \in \mathcal{EC}$; and (2) $\psi(t) = \bar{\Omega}$.*

3.3 A computational model for runtime exogenous norm synthesis

This section introduces an abstract and domain-independent computational model aimed at solving the norm synthesis problem formalised by Definition 18. In particular, this model is intended to synthesise normative systems at runtime, and exogenously. In this model, a regulatory entity observes agents' interactions at runtime, and synthesises norms to avoid conflicts when necessary. Figure 3.2 illustrates an abstract view of this model. On top of a MAS, there is a *Norm Synthesis Machine* (NSM) that monitors its evolution. Based on the perceptions collected from the MAS, the NSM synthesises a normative system that eventually makes public for the agents. With this aim, it carries out a norm synthesis process implemented by a *synthesis strategy*. This strategy decides *when* and *how* to change the current normative system, and *which* normative system publish for the agents.

As detailed in Section 3.2.2, in this dissertation norms are aimed to avoid conflicts. In particular, this model is not intended to avoid any possible conflict from the very beginning. That would be infeasible in an open MAS, since it would require to know *all* the possible conflicts that may arise. Instead, our model detects when conflicts arise at runtime. Then, it synthesises norms aimed at avoiding such conflicts in the future. A NSM can operate by considering an initially empty normative system, or by considering an initial normative system with norms. Therefore, given a MAS, a NSM can be employed either to synthesise its norms from scratch, or to adapt an initial normative system as the MAS evolves. To carry out the norm synthesis process described above, a NSM requires to perform *four* essential tasks:

1. *Detect conflicts.* A NSM creates new norms when it detects new conflicts. For this purpose, it is endowed with the computational capabilities to detect conflicts in a given MAS.

2. *Create new norms.* For each new conflict a NSM detects, it creates a new norm aimed at avoiding that particular conflict in the future. Therefore, a

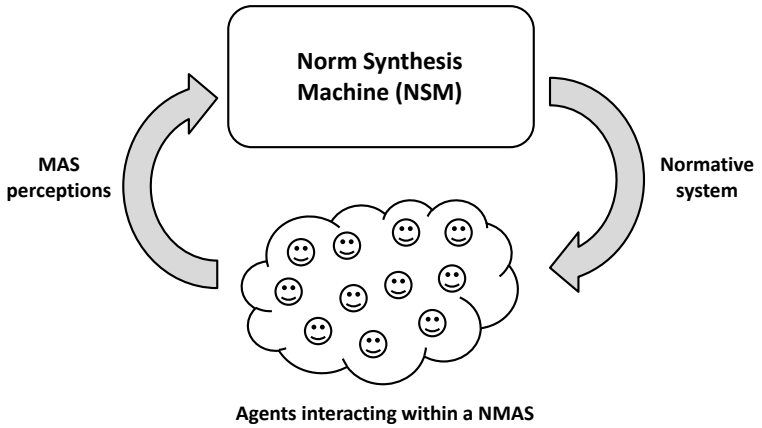

Figure 3.2: Abstract view of our model for runtime exogenous norm synthesis.

NSM is also endowed with the computational capabilities to create norms based on detected conflicts.

3. *Evaluate created norms.* Once a NSM has created a norm, it does not have any evidence about its performance in regulating conflicts. For this reason, a NSM evaluates along time how each synthesised norm performs in regulating conflicts in terms of some norm evaluation criteria.

4. *Refine the normative system.* A NSM exploits norms' performances to eventually decide whether to preserve or discard each synthesised norm. The purpose is to preserve norms that are successful enough, and discard those that are not.

Note that this norm synthesis model is *conflict-driven*. Norms are created based on detected conflicts, and they are then continuously evaluated based on the conflicts that arise once agents comply (or not) with them. Each one of the processes described above are detailed in subsequent sections.

3.3.1 Detecting conflicts within a MAS

The starting point of the norm synthesis process is the perception of the MAS. A NSM permanently observes a MAS through a set of *sensors* that have a partial view of the MAS. Each sensor monitors agents' activities and generates descriptions of what it perceives in terms of language $\mathcal{L}_{MAS}$ (described in Section 3.2). Particularly, given a MAS state $s \in S$, each sensor encodes its partial view of the state in the form of a view ν_i^s. A NSM employs partial views to generate an overall description d_s of state s.

Note that perceiving the state of a specific MAS requires domain-dependent information. That is, it requires to identify *"what is going on"* in the state,

and how to describe it. As an example, as discussed in Section 3.2, perceiving the state of a traffic junction requires the identification and description of cars' positions and headings. Therefore, a sensor considers a domain-dependent function to perceive a MAS whose signature is $perceive : S \rightarrow \mathcal{P}(\mathcal{L}_{MAS})$. It inputs a state $s \in S$, and returns a description of s in terms of $\mathcal{L}_{MAS}$. In particular, each sensor has a particular implementation of function *perceive* that encodes its partial perception of the MAS in the form of a view.

After perceiving a MAS, a NSM proceeds to detect conflicts. Likewise perception, conflict detection requires domain information to assess what types of situations are undesirable in a specific scenario. Therefore, a NSM considers a domain-dependent function to detect conflicts whose signature is $getConflicts : \mathcal{P}(\mathcal{L}_{MAS}) \rightarrow \mathcal{P}(\mathcal{L}_{MAS})$. It inputs the description of a state s (that is, d_s), and returns a (possibly empty) set of conflicts $\{c_1^s, \ldots, c_q^s\}$ such that $c_i^s \subset d_s$. Such function is an implementation of function *conflict*, formally described in Definition 4 (Section 3.2). Obviously, if the set of conflicts of a state s is empty, then s is not undesirable. As an example, state s' in Figure 3.1b has a single conflict (a collision between cars a a b), which can be described as follows:

$$\{cell(1,1,car\text{-}heading\text{-}east,a), cell(1,1,car\text{-}heading\text{-}north,b)\}$$

3.3.2 Creating new norms when necessary

A NSM creates new norms when it considers that it is necessary to avoid conflicts. Particularly, conflicts are not avoided before they arise for the first time. Instead, once a conflict has arisen, a NSM creates a new norm aimed at avoiding the conflict in the future. Given a detected conflict, a NSM proceeds to create a norm by first guessing the *source* of the conflict, and then creating a norm that regulates that source. In particular, a NSM considers that a conflict's source is an agent that performed an action during the transition to the state containing the conflict. Therefore, to guess a conflict's source, a NSM analyses the state previous to the state containing the conflict, and then it *blames* the conflict on an agent whose action the NSM considers that caused it. Finally, it creates a norm aimed at prohibiting the action the agent performed in the state previous to the state containing the conflict. The rationale is that prohibiting one of the actions that the agents performed during the transition from a state s to an undesirable state s' (i.e., a state with a conflict) may avoid state s' from arising in the future, and thus, the conflict. There are three main assumptions implicit in this model:

1. A conflict is *caused* by the actions of the agents involved in the conflict. In other words, a NSM considers that one of the agents involved in a conflict is the agent responsible for the conflict. As an example, in Figure 3.1b (Section 3.2), the car responsible for the collision in state s' is either car a or car b.

2. The source of a conflict is an *observable* action. In Figure 3.1, action *go* performed by car a and car b can be inferred by observing the differences

between states s and s'. That makes possible to infer a single action *go* to be prohibited in order to avoid the state transition. Therefore, non-observable actions, which cannot be inferred, cannot be employed to create norms.

3. The consequences of performing an action in a state arise in the *next* state (i.e., the state resulting from a state transition). In Figure 3.1, a collision arises in s' *immediately* after car a and car b perform respective actions *go* during the transition from state s to s'. If the collision between both cars arose several states after they performed action *go*, a NSM could not yield this action by analysing the state previous to the state containing the collision. Therefore, it could never find the conflict's source.

In what follows, we detail how a NSM guesses the source of a given conflict, and how it creates a norm to regulate the source of a conflict.

Guessing a conflict's source

Given a state transition $\langle s, A, s' \rangle$ such that s' is undesirable, and a particular conflict $c^{s'}$ in s', a NSM guesses the conflict's source by employing a function $getSource : \mathcal{P}(\mathcal{L}_{MAS}) \times \mathcal{P}(\mathcal{L}_{MAS}) \rightarrow Ag$. Such function inputs a conflict $c^{s'}$ in state s', and a description d_s of the state s previous to the conflict; and returns an agent that is part of $c^{s'}$ and is considered as responsible for the conflict. This function is *domain-dependent* and must be implemented for each particular NSM and each particular application domain. For example, a naive NSM may use an implementation of such function that, given a conflict, chooses an agent randomly. This particular strategy may work when a conflict may be caused for any agent involved in it. An example can be illustrated with the collision in Figure 3.1b, whose responsible agent may be either car a or car b, indistinctly. However, in more complex situations it may be necessary to make more informed decisions. Consider an alternative traffic situation in which a car performs a dangerous overtaking, and collides with another car. In this case, there is only one agent responsible for the conflict – the car that performed the overtaking. Therefore, the *"choose randomly"* strategy is not appropriate. Instead, some domain knowledge should be considered to choose the right agent.

Regulating a conflict's source

Given a state transition $\langle s, A, s' \rangle$, a conflict $c^{s'}$ of s', and an agent ag that is considered as responsible for conflict $c^{s'}$, a NSM aims at preventing conflict $c^{s'}$ in the future by creating a norm. Such norm prohibits the action that agent ag performed during the state transition for any agent encountering the same context that agent ag had in state s. Note that, likewise in the case of MAS perception, yielding an agent's context, as well as the action an agent performed during a state transition, requires domain-dependent information. Therefore, to create norms, a NSM considers the following domain-dependent functions:

- Function $getContext(ag, s)$, which yields the local context of an agent (ag) in a given state (s). Such function is an implementation of function *context*, formally defined in Definition 1 from Section 3.2.

- Function $getAction(ag, s, s')$, which yields the action an agent (ag) performed during the transition from one state s to another state s'. This function is an implementation of function *action*, formally defined in Definition 2 from Section 3.2.

Let us illustrate an example with car a in Figure 3.1. By using function *getContext*, a NSM can yield the local context of car a in state s, which is

$$\{left(nil),\ front(nil),\ right(car\text{-}heading\text{-}left)\}$$

Also, by means of function *getAction*, a NSM can obtain the observable action that car a performed during the transition from state s to state s' (i.e., action *go*). Finally, a NSM can create a new norm composed of the car's context as its precondition, and a prohibition of the observed action as its postcondition (cf. Definition 6). With this aim, it employs a grammar $\mathcal{G}$ that a NSM can employ to create norms of the form $\langle \varphi, \theta(ac) \rangle$. This grammar has been adapted from [García-Camino et al., 2009], using as building blocks *atomic formulae* of the form $p^n(\tau_1, \ldots, \tau_k)$, p being an n-ary predicate symbol and $\tau_1, \ldots, \tau_k$ terms of $\mathcal{L}_{Ag}$. Such grammar is as follows:

$$\begin{array}{lll} \text{Norm} & ::= & \langle \{LHS\}, RHS \rangle \\ \text{LHS} & ::= & LHS, LHS \mid \rho \\ \text{RHS} & ::= & prh(Ac) \\ Ac & ::= & ac_1 \mid \ldots \mid ac_n \\ \rho & ::= & p^n(\tau_1, \ldots, \tau_k) \\ \tau & ::= & tr_1 | \ldots | tr_j \end{array}$$

where ac_i and tr_j are constants. The set of all norms which comply with the grammar above are represented as $\mathbf{N}$. Considering the running example introduced in Section 3.2, the generic grammar above could be instantiated as:

$$\begin{array}{lll} \text{Norm} & ::= & \langle \{LHS\}, RHS \rangle \\ \text{LHS} & ::= & LHS, LHS \mid \rho \\ \text{RHS} & ::= & prh(Ac) \\ Ac & ::= & go \mid stop \\ \rho & ::= & left(\tau) \mid front(\tau) \mid right(\tau) \\ \tau & ::= & car\text{-}heading\text{-}left \mid car\text{-}heading\text{-}right \\ & & car\text{-}opposite\text{-}heading \mid car\text{-}same\text{-}heading \\ & & wall \mid nil \end{array}$$

In our example, the resulting norm would correspond to norm n described in Section 3.2, which is described below again for convenience:

$$n : \langle \{left(nil), front(nil), right(car\text{-}heading\text{-}left)\}, prh(go) \rangle$$

Particularly, a NSM creates new norms only for those conflicts that are *unregulated* (cf. Definition 9 in Section 3.2). The reason of this is the following. Consider that a NSM is monitoring a MAS, and a conflict arises. Then, the NSM creates a new norm to avoid the conflict in the future. Thereafter, the NSM considers the conflict as *regulated* (cf. Definition 10 in Section 3.2). Consider that the created norm is ineffective, and thus unable to avoid the conflict. Therefore, after some time the conflict arises again. Initially, one may think that the norm should be immediately discarded, and an alternative norm should be created. However, as previously detailed, a NSM evaluates norms along time, and thus may take some time to detect whether a norm performs poorly. During this time, the same conflict may arise several times, but a NSM cannot assess if the conflict arises because the norm is ineffective, or for another reason. Therefore, the NSM will not create new norms to regulate the conflict until it has detected that the original norm is ineffective, and thus removes it from the normative system. Thereafter, the conflict will be unregulated again, and the NSM will be able to create a new, alternative norm when the conflicts arises again.

Note that implicit within this model is a reduction of deontic operators to only prohibitions. It is natural to ask whether this represents a problematic restriction. But, notice that obligations and permissions are frequently and naturally interpreted as dual notions: an action is obligatory if it is not permitted to refrain from performing the action; and an action is permissible if it is not obligatory to refrain from performing the action. In this case, obligations can be reduced to prohibitions: to make an action obligatory, prohibit everything else. With this in mind, this approach of focussing on prohibitions can also be seen as encompassing obligations. Of course, richer deontic operators (e.g., conditional obligations) have also been considered in the literature, and we do not claim that our approach encompasses a full range of these: but with respect to core operators, this approach is sufficient. Moreover, this approach can also handle permissions. Observe that the interpretation of permissions can vary. For instance, there may be normative systems whereby all actions are prohibited unless they are explicitly permitted. Alternatively, there may be systems whereby all actions are permitted unless explicitly prohibited, and permissions are ways of encouraging certain behaviours. When dealing with obligations and permissions, the undesirable states are *idealised* situations which did not occur, and the context of an agent is used to establish an obligation or permission on a missing action (which would have reached the idealised situation).

3.3.3 Evaluating norms' performances

Once a NSM creates a new norm, it has no guarantee that the norm will succeed in regulating conflicts. Therefore, a NSM needs to *evaluate* each created norm to assess *how it performs* in regulating conflicts. Particularly, a NSM iteratively evaluates a norm along time by:

1. Gathering empirical evidences about *what happens in the MAS* after agents fulfil and infringe the norm.

2. Exploiting these empirical evidences to compute the norm's utility in terms of some norm evaluation criteria.

These norm evaluation criteria, already introduced in Section 3.2.2, are crucial to determine to what extent synthesised norms perform well at regulating conflicts.

In Section 3.2.2, two essential norm evaluation criteria were introduced – the so-called *effectiveness* and *necessity* – to pursue the synthesis of effective and liberal normative systems, respectively. These two criteria can be computed based on the conflicts that arise in a MAS after agents fulfil/infringe norms, namely based on *norms' compliance outcomes*. On the one hand, a NSM can compute a norm's effectiveness based on the conflicts that arise after agents fulfil the norm, specifically as its ratio of *successful fulfilments* (i.e., fulfilments that did not lead to conflicts, see Definition 13 from Section 3.2). Briefly, the more conflicts a norm avoids after being fulfilled, the more *effective*. Analogously, a norm's necessity can be computed based on the conflicts that arise after agents infringe the norm, specifically as its ratio of *harmful infringements* (i.e., infringements that lead to conflicts, see Definition 16 from Section 3.2). The intuition is that, if no conflicts arise after agents infringe a norm, maybe the norm is *unnecessary*.

Note the are three main assumptions implicit in this norm evaluation model:

1. A NSM can detect *norm applicability*. Namely, it can detect when a given norm holds for an agent.

2. A NSM can assess *norm compliance*. That is, it can assess whether an agent has fulfilled or infringed a given norm that held for her.

3. A NSM can detect the conflicts that arise after an agent fulfils or infringes a norm.

The concepts of norm applicability and norm compliance were already introduced in Section 3.2.1 (Definitions 8, 11, and 12), and conflict detection was detailed in Section 3.3.1. Briefly, to detect norm applicability a NSM needs to yield the local context of each agent in a given state s to detect if any norm applies to her context. With this aim, it employs function *getContext* introduced in Section 3.3.2. To assess norm compliance, a NSM must be able to detect when a norm has been fulfilled or infringed by an agent. With this aim, it employs function *getAction* introduced in Section 3.3.2 to yield the action that an agent performed during the transition from a state s to another state s'. This will allow the NSM to assess if the agent has complied with the norm or not. Finally, it employs function *getConflicts* introduced in Section 3.3.1 to detect conflicts in a given state s.

Next, we describe an information model whereby a NSM can keep track of norms' compliance outcomes.

Gathering norms' compliance outcomes

For each norm a NSM synthesises, it creates a tuple of finite series

$$\langle \mathcal{SF}^n, \mathcal{HF}^n, \mathcal{SI}^n, \mathcal{HI}^n \rangle$$

that accumulate, respectively, its (i) *successful fulfilments* ($\mathcal{SF}^n$) along time; (ii) *harmful fulfilments* ($\mathcal{HF}^n$) along time; (iii) *successful infringements* ($\mathcal{SI}^n$) along time; and (iv) *harmful infringements* ($\mathcal{HI}^n$) along time. Formally, these series are:

$$\mathcal{SF}^n = \langle sf_1^n, \ldots, sf_m^n \rangle \in \mathbb{N}^m \tag{3.1}$$

$$\mathcal{HF}^n = \langle hf_1^n, \ldots, hf_m^n \rangle \in \mathbb{N}^m \tag{3.2}$$

$$\mathcal{SI}^n = \langle si_1^n, \ldots, si_m^n \rangle \in \mathbb{N}^m \tag{3.3}$$

$$\mathcal{HI}^n = \langle hi_1^n, \ldots, hi_m^n \rangle \in \mathbb{N}^m \tag{3.4}$$

where $sf_t^n \in \mathcal{SF}^n$, $hf_t^n \in \mathcal{HF}^n$, $si_t^n \in \mathcal{SI}^n$ and $hi_t^n \in \mathcal{HI}^n$ stand for the number of successful fulfilments, harmful fulfilments, successful infringements, and harmful infringements of norm n at time t, respectively. As an example, if at a given time t a norm n is fulfilled by an agent, and it leads to a conflict (harmful fulfilment), we will set $hf_t^n = 1$ and $sf_t^n = 0$. Analogously, if norm n is infringed by two agents at time t, and neither of the infringements lead to conflicts (successful infringements), we will set $si_t^n = 2$ and $hi_t^n = 0$.

A NSM can exploit norm compliance outcomes to assess how a given normative system performs at regulating conflicts after agents fulfil and infringe its norms.

3.3.4 Refining the normative system

A NSM exploits computed norms' performances to *refine* the normative system, both by discarding those norms that do not perform well, and by preserving those norms that perform properly. Particularly, a NSM decides that a norm does not perform well whenever its utility in terms of some norm evaluation criterion is *below* a corresponding threshold. For instance, let us consider the effectiveness norm evaluation criteria introduced in Section 3.2.2 It could be formally described as $\langle \mu_{\mathit{eff}}, \alpha_{\mathit{eff}} \rangle$. If, at a given moment, the effectiveness (μ_{eff}) of a norm is below threshold α_{eff}, then the norm will be considered as *ineffective*, and will be discarded from the normative system.

3.4 An abstract architecture for runtime exogenous norm synthesis

Next, an abstract architecture is provided to support the implementation of the computational model described in Section 3.3. Figure 3.3 illustrates the architecture of our NSM, which is composed of the following core elements:

1. An abstract *synthesis strategy* in charge of norm synthesis.
2. A data structure – the so-called *normative network* – to store the norms synthesised (explored) so far, along with their relationships.
3. A collection of *operators* to apply changes to the normative network.

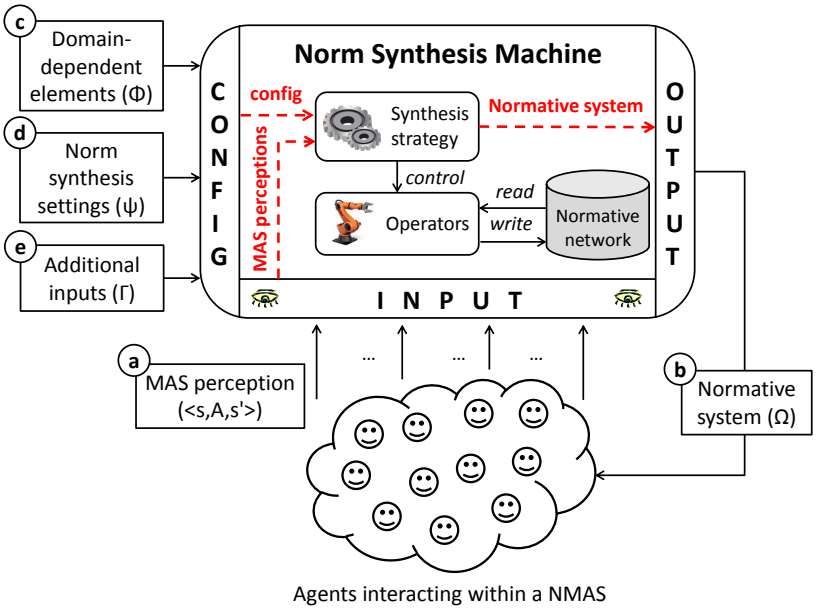

Figure 3.3: Abstract architecture to support the implementation of our norm synthesis model.

A NSM continuously perceives a target MAS by means of state transitions (box a in Figure 3.3), which describe both the current state of the MAS and the previous state (s' and s, respectively). After that, it executes a synthesis strategy that synthesises norms based on the perceptions collected from the MAS. This strategy implements the computational model described in Section 3.3. Thus, it is in charge of detecting conflicts within the MAS perceptions, and synthesising norms to avoid these conflicts. In this way, it aims at synthesising a normative system that solves the norm synthesis problem (cf. Definition 18 in Section 3.2.2). During this synthesis process, the strategy employs a set of operators to manipulate a normative network, adding/removing norms when necessary, and establishing relationships between these norms.

After executing a synthesis strategy, a NSM publishes the resulting normative system Ω (box b in Figure 3.3) so that the agents become aware of it. Once the new normative system is deployed, the NSM keeps on perceiving the MAS and executing its strategy. This cyclic process continues until the strategy deploys a normative system that, for a long enough period of time, remains stable, and its norms are successful enough in terms of some evaluation criteria, hence solving the norm synthesis problem.

To perform norm synthesis, a NSM receives as an input a set of *domain-dependent elements* (Φ in box c in Figure 3.3) that must be implemented for the particular domain for which norms are synthesised. These elements are:

1. Function *perceive* to describe a MAS state from a global, external observer's perspective (introduced in Section 3.3.1).
2. Function *getConflicts* to detect conflicts within a MAS state (see Section 3.3.1).
3. Function *getContext* to yield an agent's context in a MAS state (see Section 3.3.2).
4. Function *getAction* to infer an agent's action in a MAS state transition (see Section 3.3.2).
5. A grammar $\mathcal{G}$ to define norms (introduced in Section 3.3.2).
6. Function *getSource* to guess a conflict's source, namely the agent responsible for a given conflict (introduced in Section 3.3.2).

Moreover, a NSM receives as an input a set of domain-independent *norm synthesis settings* (Ψ in box d in Figure 3.3) to evaluate norms' performances and to determine the convergence to a normative system. In particular, it contains:

1. A set $\mathcal{EC}$ of norm evaluation criteria (cf. Section 3.2.2), where each evaluation criterion is a pair $\langle \mu_i, \alpha_i \rangle \in \mathcal{EC}$.
2. A time interval T to assess the convergence to a normative system that solves the NSP (cf. Definition 18 in Section 3.2.2).

Moreover, a NSM receives as an input a set of *additional inputs* (Γ in box e in Figure 3.3) that a particular synthesis strategy may require to perform norm synthesis. Subsequent sections describe the three core elements that compose a NSM, namely the normative network, the operators, and the synthesis strategy.

3.4.1 A normative network to keep track of norms

A NSM employs a graph-based data structure – the so-called *normative network* – to keep track of the norms it explores during the synthesis process. A normative network is a *graph* whose nodes stand for norms and whose edges stand for *relationships* between norms. In a normative network, a norm may have different *states*. We assume that a NSM can represent the current normative system at a given time based on the norms the normative network contains at that time. For instance, a normative system may be computed as *all* the norms of the normative network, or it may be computed as those norms that satisfy certain conditions (such as being in a certain state). Formally, a normative network is:

Definition 19 (Normative network)**.** *A normative network (NN) is a tuple* $\langle \mathcal{N}, \mathcal{R}, \Delta, \delta \rangle$ *where: (i)* $(\mathcal{N}, \mathcal{R})$ *is a graph such that:* $\mathcal{N}$ *is a set of norms that correspond to the vertices in the graph, and* $\mathcal{R} \subseteq \mathcal{N} \times \mathcal{N}$ *is a set of directed edges representing relationships between norms; (ii)* Δ *is a set of possible states of a norm in* $\mathcal{N}$*; and (iii)* $\delta : \mathcal{N} \to \Delta$ *returns a norm's state in the normative network.*

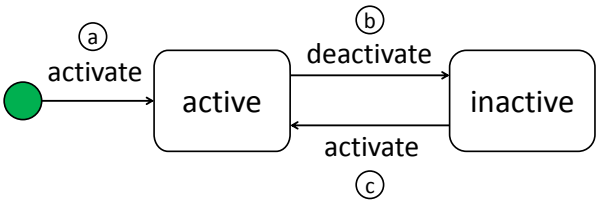

Figure 3.4: Possible states of a norm in our normative network, along with their transitions.

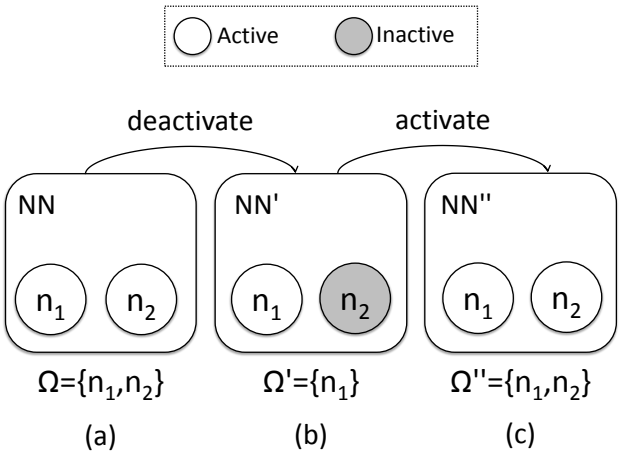

Figure 3.5: Evolution of a normative network (and its corresponding normative system) by changing norms' states.

We now identify a basic set of norm states that we consider as essential to perform norm synthesis. This set of states will be employed (and extended) in the remainder of this dissertation. In short, a norm in a network may be either *active* or *inactive*. Formally, the set of possible states is $\Delta = \{active, inactive\}$. Moreover, we consider that a normative network represents a normative system as its *active* norms. Formally, $\Omega = \{n \,|\, n \in \mathcal{N}, \delta(n) = active\}$. Thus, the normative system can be changed both: (1) by changing the set of norms of the normative network ($\mathcal{N}$); and (2) by changing norms' states in the network (δ). Figure 3.4 depicts the states in which a norm may be in a network, along with the transitions between these states. Initially, a norm may be *activated* by setting its state to active (transition labelled with *"a"* in Figure 3.4). In this way, the norm would be included in the normative system. Thereafter, the norm may be *deactivated*, by setting its state to inactive (transition b in Figure 3.4), hence removing it from the normative system. Finally, an inactive norm may be activated again (transition c in Figure 3.4), and thus re-included to the normative system.

Figure 3.5 illustrates how to change a normative system by changing norms' states in a normative network. Initially, the normative network *NN* contains two *active* norms n_1, n_2 (represented as white circles), hence representing normative system $\Omega = \{n_1, n_2\}$ (Figure 3.5a). Then, norm n_2 is *deactivated* in *NN* (represented as a gray circle), yielding $NN' = \{n_1, n_2\}$ and $\Omega' = \{n_1\}$ (Figure 3.5b). Finally, norm n_2 is activated in NN', giving rise to $NN'' = \{n_1, n_2\}$ and $\Omega'' = \{n_1, n_2\}$ (Figure 3.5a). In general, a NSM performs norm synthesis by continuously iterating over (applying changes to) the normative network until it finds a normative system that solves the norm synthesis problem.

3.4.2 Operators for normative networks

A NSM transforms a normative network through a collection of *operators*. Each operator transforms a normative network $\langle \mathcal{N}, \mathcal{R}, \Delta, \delta \rangle$ into another one $\langle \mathcal{N}', \mathcal{R}', \Delta, \delta' \rangle$, consequently leading from one normative system to another. We identify three basic types of operations that a synthesis strategy may require to transform a normative network:

- *Modifying the set of norms* in a normative network. The basic operations that a NSM performs in a network are the addition and removal of norms to/from the normative network, hence changing $\mathcal{N}$.
- *Changing norms' states* in a normative network. Since the norms in a network may be at a certain state, a NSM requires operators to change these states. Examples of this are depicted in Figure 3.5.
- *Establishing relationships* between norms in the normative network. As detailed above, norms in a network may have relationships between them. Therefore, a NSM should provide operators to establish and to remove these relationships. These type of operations will be illustrated in subsequent chapters (from Chapter 5 onwards), which exploit relationships between norms.

We now introduce a basic set of normative network operators to be used (and extended) in the remainder of this dissertation. These operators, which are described in Table 3.1, can be employed to:

1. *Add* a norm to the normative network. The `add` operator extends the normative network with norm n ($\mathcal{N}' = \mathcal{N} \cup \{n\}$).
2. *Activate* a norm in the normative network. The implementation of this `activate` operator sets the state of a given norm to active ($\delta'(n) = \mathit{active}$), which consequently becomes part of the normative system.
3. *Deactivate* a norm in the normative network. This `deactivate` operator sets the state of a given norm to *inactive* ($\delta'(n) = \mathit{inactive}$). Hence, although the norm remains in the normative network, it is no longer part of the normative system.

Operator	Specification
$add(n, NN)$	$\mathcal{N}' \leftarrow \mathcal{N} \cup \{n\}$ $NN' \leftarrow \langle \mathcal{N}', R, \Delta, \delta \rangle$
$activate(n, NN)$	$\delta'(n) \leftarrow active$ $NN' \leftarrow \langle \mathcal{N}, R, \Delta, \delta' \rangle$
$deactivate(n, NN)$	$\delta'(n) \leftarrow inactive$ $NN' \leftarrow \langle \mathcal{N}, R, \Delta, \delta' \rangle$

Table 3.1: Basic operators for a NSM.

Note that, while operator *add* is employed to modify the set of norms of a normative network, operators *activate* and *deactivate* are employed to change the states of its norms. In our particular case, synthesised norms are never removed from the normative network, and thus no *remove* operator has been included. Instead, a norm can be activated/deactivated to add it to/discard it from the normative system, respectively.

3.4.3 A strategy to synthesise normative systems

As previously discussed, a synthesis strategy implements the norm synthesis computational model described in Section 3.3, which consists in:

1. *Detecting conflicts* as described in Section 3.3.1.
2. *Creating norms* to prevent conflicts as described in Section 3.3.2.
3. *Evaluating norms* as described in Section 3.3.3.
4. *Refining the normative system* as described in Section 3.3.4.

By performing these four steps, a synthesis strategy should be able to find, after some time, a normative system that effectively achieves coordination, avoiding conflicts as long as agents comply with norms.

3.5 Conclusions

This chapter has contributed to answer research question R1 described in Section 1.2 by introducing:

1. An abstract and domain-independent computational model to perform *on-line, exogenous* and *multi-objective* norm synthesis for open Multi-Agent Systems. In this model, a Norm Synthesis Machine (NSM) monitors agents' activities at runtime, and synthesises normative systems based on the perceptions collected from the MAS. With this aim, a NSM iteratively executes a *synthesis strategy* that is in charge of performing norm synthesis. In particular, a synthesis strategy implements such norm synthesis model and performs the four essential

tasks of (1) conflict detection, (2) norm creation, (3) norm evaluation, and (4) norm refinement. The introduced computational model has been described in an abstract manner so that it can be applied to different application domains by providing key domain-dependent information.

2. An abstract architecture to support the implementation of the norm synthesis computational model described above. Our abstract architecture is composed of

- A synthesis strategy in charge of directing the norm synthesis process.
- A normative network whereby the synthesis strategy can keep track of synthesised norms.
- A collection of operators to apply changes to the normative network.

Such architecture considers a set of domain-dependent elements to perform norm synthesis in a given application domain, a set of domain-independent settings to configure the norm synthesis strategy, and a set of additional inputs that a particular synthesis strategy may require to perform norm synthesis.

One can use this computational model to perform norm synthesis on different domains by implementing relatively simple domain-dependent functions such as conflict detection and retrieval of agents' context and actions. We thus claim that this innovative approach can be highly relevant for open MAS. To the best of our knowledge, and as argued by our analysis of the state of the art in Chapter 2, the literature does not provide any general approach to perform on-line, exogenous, and multi-objective norm synthesis for open MAS.

Hereafter, subsequent chapters introduce a family of synthesis strategies intended to be executed by a NSM to perform norm synthesis. These synthesis strategies implement the computational model introduced in this chapter, each focusing on one of the key synthesis objectives introduced in Section 1.1.3. Thus, Chapter 4 introduces a strategy to synthesise *effective* normative systems. Then, Chapters 5 and 6 present two strategies to synthesise *compact* normative systems, and Chapter 7 introduces a strategy to synthesise *liberal* normative systems. Finally, Chapter 8 introduces a synthesis strategy that can consider different degrees of *reactivity* during norm synthesis.

Chapter 4

Synthesising effective normative systems

4.1 Introduction

Aiming at answering research question R1 (introduced in Section 1.2), the previous chapter introduced a computational model to synthesise norms for a MAS at runtime and without requiring the agents' participation in norm synthesis. This computational model considers a regulatory entity – the so-called Norm Synthesis Machine (NSM) – that observes agents' activities, and executes a *synthesis strategy* in charge of performing norm synthesis.

Against this background, this chapter contributes to answer research question R1 (Section 1.2) by introducing BASE, a synthesis strategy intended to be executed by a NSM to perform norm synthesis. BASE implements the computational model introduced in Chapter 3 to synthesise norms for open MAS, and achieves the synthesis of *effective* normative systems (see Section 1.1.3). Thus, BASE is intended to be executed by a NSM to synthesise normative systems that successfully avoid conflicts within a MAS. BASE divides norm synthesis into three stages: *norm generation*, *norm evaluation*, and *norm refinement*. During norm generation, it detects conflicts within a MAS as detailed in Section 3.3.1, and creates norms for each conflict it detects as detailed in Section 3.3.2. Thereafter, during norm evaluation, BASE evaluates norms based on their outcomes in the MAS as detailed in Section 3.3.3. Finally, during norm refinement, BASE refines the normative system by discarding those norms that are not successful enough (i.e., *under-perform*) in terms of some norm evaluation criteria. It performs this operation as detailed in Section 3.3.4.

BASE aims at creating effective norms by *learning from experience*. In short, each time BASE creates a norm from a conflict, it keeps detailed information about the conflict, and how it *solved* it. Thereafter, if the norm is successful in avoiding conflicts, it employs this information to solve similar conflicts, thus creating similar norms. We empirically evaluate BASE in agent-based simulations of

the road traffic scenario introduced in Section 1.3.1. In this way, we demonstrate not only BASE's performance, but also the validity of the computational norm synthesis model introduced in Chapter 3.

The remainder of this chapter is organised as follows. Section 4.2 describes the information model employed by BASE to perform norm synthesis. Section 4.3 describes how BASE creates norms by learning from experience. Then, Section 4.4 introduces BASE's approach to evaluate norms' performances. Section 4.5 details how BASE refines the normative system. Section 4.6 introduces BASE's synthesis strategy. Section 4.7 provides an analysis of the BASE's computational complexity. Next, Section 4.8 provides an empirical evaluation of BASE in the road traffic scenario. Finally, Section 4.9 draws some conclusions and identifies the strengths and weaknesses of BASE.

4.2 Information model

BASE considers the formal model and the basic definitions introduced in Section 3.2, with a MAS that has a set of states S, a set of conflicting states $C \subseteq S$, and a set of agents Ag that can perform actions out of a set of actions Ac. A state transition function $\mathbb{T} : S \times Ac^{|Ag|} \rightarrow S$ leads the MAS to a state s' from a state s after the agents perform a set of actions $A \in Ac^{|Ag|}$. There are two languages to describe the states of the MAS. On the one hand, a MAS language ($\mathcal{L}_{MAS}$) can be employed to describe the states of the MAS from a external observer's perspective. On the other hand, an agent's language ($\mathcal{L}_{Ag}$) can be employed to describe a MAS from the agents' perspective. Norms are of the form $\langle \varphi, \theta(ac) \rangle$, being $\varphi \in \mathcal{L}_{Ag}$ the precondition of the norm, and $\theta(ac)$ its post-condition.

To keep track of synthesised norms, BASE employs the normative network described in Section 3.4.1. Moreover, BASE considers that a norm in a normative network may be in a state out of a set of states $\Delta = \{active, inactive\}$, and it represents a normative system as the set of norms that are *active* in the normative network. To apply changes to the normative network, BASE incorporates the normative network operators employed by described in Table 3.1 (Section 3.4.2). These operators allow BASE to: (i) *add* norms to the normative network; (ii) *activate* norms so that they become part of the normative system; and (iii) *deactivate* so that they no longer belong to the normative system.

4.3 Creating norms by learning from experience

BASE starts norm synthesis by detecting conflicts within a MAS as detailed in Section 3.3.1. Thereafter, for each detected conflict, it creates a norm aimed at avoiding the conflict in the future. Given a conflict, BASE creates a norm by:

1. *Guessing* the conflict's source. It retrieves the agents involved in the conflict, and then *blames* one of these agents for the conflict. With this aim, it implements function *getSource* (introduced in Section 3.3.2), which given

a conflict, guesses its source. That is, it returns an *agent* considered responsible for the conflict.

2. *Regulating* the conflict's source. It creates a norm that prohibits the action the chosen agent performed during the transition to the state containing the conflict. With this aim, it employs two domain-dependent functions *getContext* and *getAction* (introduced in Section 3.3.2) to retrieve: (i) the context that the agent had in the state previous to the state containing the conflict; and (ii) the action the agent performed during the transition to the state containing the conflict, respectively. Then, it creates a norm composed of the agents' context as its precondition, and a prohibition of the agent's action as its post-condition.

Further details of these two operations above are provided in Section 3.3.2.

Once BASE has created a norm n, it keeps track of it by including it in the normative network. With this aim, it employs operator $add(n, NN)$ described in Table 3.1 in Section 3.4.2. Finally, it adds the norm to the normative system by activating it in the normative network. With this aim, it employs operator $activate(n, NN)$ described in Table 3.1 in Section 3.4.2.

Note that a key step during the creation of a norm is the process of *guessing* a conflict's source. During such operation, BASE "chooses" an agent as responsible for a conflict, thus indirectly choosing the norm that will be created to avoid the conflict. Therefore, given a conflict, choosing the incorrect agent may lead to create an ineffective norm. As an example, consider a traffic situation in which there is a car stopped in a jam, and a second car collides with it from behind. In this case, this second car is the only responsible for the collision. Hence, choosing the stopped car as responsible for the collision may lead to create a norm that prohibits to remain stopped in a jam. This norm would be ineffective in avoiding collisions.

Against this background, it is natural to see that, the more informed the process of guessing a conflict's source, the more likely to synthesise effective norms. Following this rationale, BASE incorporates a computational mechanism that learns from experience to guess the correct source of a conflict. Briefly, once BASE determines a conflict's source, it keeps information about the conflict, its source, and the norm created to regulate the conflict. Thereafter, if the norm is successful in regulating conflicts, BASE exploits this information to guess the source of similar conflicts in the future.

Note that this rationale is very similar to the way in which humans learn how to solve problems. Briefly, humans usually solve new problems by *analogy*, namely by considering how similar problems were solved in the past. The literature provides a Machine Learning technique that is based on this rationale, the so-called *Case-Based Reasoning* (CBR). CBR [Riesbeck and Schank, 1989, Aamodt and Plaza, 1994] is a *supervised* Machine Learning technique that solves new problems (i.e., cases) by retrieving similar problems from a knowledge base (i.e., a *case base*) and adapting their *solutions*. However, during this process CBR requires the supervision of a human expert, which cannot be assumed

in our norm synthesis model. Therefore, we have implemented an *unsupervised* approach of classical CBR. Our unsupervised CBR can operate with an initially empty case base (i.e., with no previous knowledge), and does not require an expert to evaluate generated solutions. Instead, cases and their solutions are elicited at runtime, and case solutions are evaluated experimentally in an unsupervised manner during the norm evaluation phase.

Particularly, BASE keeps track of arisen conflicts and their sources in the form of *cases* and their *solutions* in a case base. Subsequent sections detail how BASE represents cases, and how it exploits cases to effectively choose the correct source of a conflict.

4.3.1 Cases and solutions

In classic CBR, a case contains the description of a problem, and a single solution for that problem. Formally, a case in classic CBR is a tuple $\langle desc, sol \rangle$, being $desc$ the description of a problem, and sol its solution. However, BASE employs an unsupervised approach to CBR. This means that, given a problem, it may need to explore alternative solutions to solve it. Therefore, a BASE's case is a tuple $\langle desc, \{\langle sol_1, u_1\rangle, \ldots, \langle sol_j, u_j\rangle\}\rangle$, being $desc$ the problem description, sol_i a solution proposed for the case, and u_i the *utility* of the i-th solution. This utility allows BASE to, given a problem, choose its best solution.

In the case of BASE, a case description is a pair containing a *conflict* and a description of the state previous to the state containing the conflict; and a solution contains an *agent* proposed as responsible for the conflict. Formally, given a state transition $\langle s, A, s'\rangle$ such that s' is undesirable, and a conflict $c^{s'}$ in s', a case description is a tuple $\langle d_s, c^{s'}\rangle$, being d_s a description of state s in terms of language $\mathcal{L}_{MAS}$, and $c^{s'}$ the conflict in state s'. The case solution is an agent proposed as responsible for conflict $c^{s'}$. Note therefore that a BASE's case is a tuple $\langle \langle d_s, c^{s'}\rangle, \{\langle ag_1, u_1\rangle, \ldots, \langle ag_j, u_j\rangle\}\rangle$.

Let us illustrate an example in the road traffic scenario in Figure 3.1 from Section 3.2.1. Figure 3.1a shows two cars (car a and car b) in state s, perceiving one another in a traffic junction. Figure 3.1b shows a collision between these cars in the central cell of the junction in a state s'. Given this problem (i.e., the collision), BASE can create a *case* whose description describes state s and the collision in state s'. Figure 4.1 shows a graphical representation of this case. Its corresponding description is $desc = \langle d_s, c^{s'}\rangle$, where d_s and $c^{s'}$ are as follows:

$d_s = \{$ *cell(0,0,wall)*, *cell(0,1,nil)*, *cell(0,2,wall)*,
cell(1,0,car-heading-east,a), *cell(1,1,nil)*, *cell(1,2,nil)*,
cell(2,0,wall), *cell(2,1,car-heading-north,b)*, *cell(2,2,wall)* $\}$

$c_1^{s'} = \{$ *cell(1,1,car-heading-east,a)*, *cell(1,1,car-heading-north,b)* $\}$

In particular, d_s describes each cell of state s in terms of language $\mathcal{L}_{MAS}$, and $c^{s'}$ describes the central cell of state s', containing the colliding cars. BASE can employ this case description to choose an agent responsible for the conflict in s' (either car a, or car b). As an example, consider BASE blames the collision

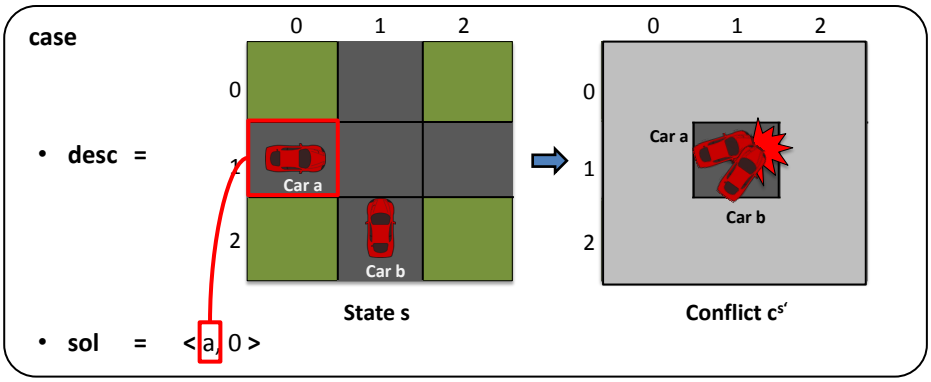

Figure 4.1: A case describing a conflict (a collision) between cars a and b ($desc_a$), with a solution (sol_a) proposing car a as responsible for the collision.

on car a. The corresponding case solution is $\langle a, 0\rangle$, being a the identifier of the car, and 0 the initial utility of the solution. From this solution, BASE can create a norm by retrieving the context of car a in state s, and the action car a performed during the state transition. Such norm is as follows:

$$n : \langle \{left(nil), front(nil), right(car\text{-}heading\text{-}left)\}, prh(go)\rangle$$

Thereafter, if norm n performs well in avoiding conflicts, it will mean that the proposed solution (i.e., blaming car a for the collision) is useful. To keep track of the utility of this solution, BASE employs the utility of the norm that was created from it (i.e., norm n's utility). That is, each time the utility of norm n is updated, BASE updates the utility of its corresponding case solution ($\langle a, 0\rangle$) described above. More details about this process are provided in Section 4.6.2.

4.3.2 Using experience to guess a conflict's source

Given a conflict, BASE guesses its source by learning from experience. For this purpose, it creates a new case describing the conflict as described in Section 4.3.1. Then, it searches in the case base for a similar case. Finally, it retrieves the solution that best solved the case (the conflict), and exploits it to guess the source of the current conflict to solve. BASE performs these operations by carrying out a complete execution of the unsupervised CBR cycle, which is composed of 4 different phases: *retrieve, reuse, revise and retain*. Hereafter, we describe each one of the 4 phases of the unsupervised CBR cycle, and we provide an algorithm that implements these phases.

Retrieving similar cases

Given a a state transition $\langle s, A, s\rangle$ such that s' is undesirable, a description d_s of state s, and a conflict $c^{s'}$ in s', BASE retrieves a conflict similar to $c^{s'}$ as follows.

First, it creates a new case whose description is a tuple $\langle d_s, c^{s'} \rangle$. Then, it searches in the case base for the case whose description is most similar. With this aim, BASE employs a *domain-dependent* function $getSimilarity(desc, desc') \in [0, 1]$ that returns the similarity between two case descriptions $(desc, desc')$. Specifically, two case descriptions are considered as similar iff their similarity is above a threshold α_{sim}. This amounts to satisfying the following condition:

$$getSimilarity(desc, desc') > \alpha_{sim} \tag{4.1}$$

In this sense, retrieving a similar case equals to finding a conflict that is similar to the conflict BASE aims at regulating. Note that, since function *getSimilarity* is domain-dependent, it must be implemented for each particular domain in which BASE is executed. As an example, in the road traffic scenario, the similarity between two cases may be computed as the number of cells that are *equal* in two states. Figure 4.2 illustrates an example. On its top, the figure depicts a case describing a conflict BASE aims at solving ($case$). On its bottom, it depicts a case that contains a similar conflict ($case_{sim}$). Such case corresponds to the case depicted in Figure 4.1. As we will see next, BASE can exploit the solution of $case_{sim}$ to guess a source for the conflict in the case to solve ($case$).

Reusing a case

If BASE has retrieved a similar case (i.e., a similar conflict), it reuses its solution to propose a solution for the conflict at hand. In particular, once a conflict has been retrieved, there may be two possible situations:

– *The retrieved case is similar to the current case.* This means that BASE has found a conflict that is similar to the conflict to solve. In this case, BASE retrieves its solution ($\langle ag_i, u_i \rangle$) with the greatest utility (u_i). Thereafter, it exploits the solution to create a norm that solves the conflict to solve ($c^{s'}$). With this aim, it employs a domain-dependent function $exploitSolution(\langle d_s, c^{s'} \rangle, \langle ag'_i, u'_i \rangle))$, which inputs the description of the current conflict ($\langle d_s, c^{s'} \rangle$), and the solution of the similar conflict ($\langle ag'_i, u'_i \rangle$), and returns an agent that is considered as responsible for the current conflict ($c^{s'}$).

Let us illustrate an example with Figure 4.2, which contains a case to solve ($case$), and a similar, solved case ($case_{sim}$). In particular, $case_{sim}$ contains a solution with high utility ($\langle a, 0.9 \rangle$). This solution, which proposes car a as responsible for the collision in $case_{sim}$, yielded the creation of norm n introduced in Section 4.3.1. BASE can adapt $case_{sim}$'s solution to solve $case$, thus creating a similar norm. With this aim, it proposes car a in state s in the case to solve ($case$) as responsible for conflict $c^{s'}$. This will lead BASE to create a norm like

$$n' : \langle \{left(nil), front(\textit{car-same-heading}), right(\textit{car-heading-left})\}, prh(go) \rangle$$

which is very similar to norm n solving $case_{sim}$ (see Section 4.3.1).

– *The retrieved case is equal to the current case.* This may happen whenever a norm that was created to avoid a particular conflict is eventually considered

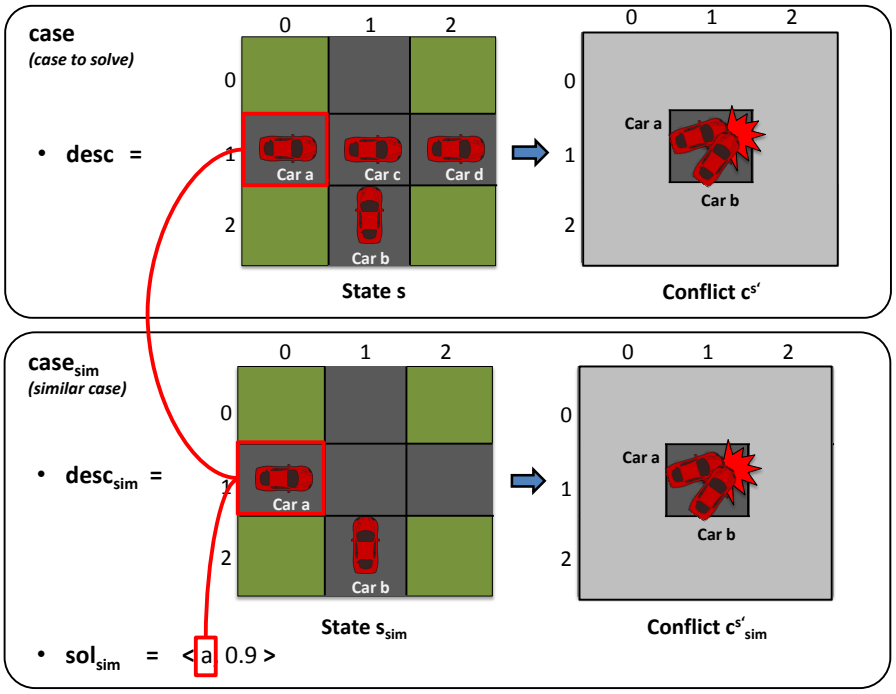

Figure 4.2: A traffic case to solve (*case*), and a similar case (*simCase*).

as unsuccessful, and then deactivated. In that case, the conflict would arise again, and the original case (the case from which the norm was created) may be retrieved. When that happens, BASE avoids creating the same norm again by choosing an *alternative*, randomly chosen agent as the one responsible for the conflict.

Finally, if BASE did not retrieve a similar case, this means that there are no similar conflicts, and thus there is no experience available that can help in choosing the source of conflict $c^{s'}$. Therefore, BASE generates a *random* solution, namely it randomly chooses an agent to be responsible for the conflict.

Revising generated solutions

In classic CBR, case solutions are evaluated by means of a human expert. However, our unsupervised CBR approach cannot consider a human evaluator. Instead, it revises the utility of each generated solution by continuously updating it with the utility of the norm created from that solution. For instance, in Figure 4.2, the utility of solution sol_{sim} corresponds to the utility of norm n described in Section 4.3.1, which was created from sol_{sim}. Particularly, norms' utilities are empirically evaluated along time during BASE's norm evaluation stage (see Section 4.6.2).

Retaining new cases

During this phase, BASE keeps track of the resulting experience in the case base. For this purpose, it stores the new case in the case base. Eventually, BASE may employ the new case to solve similar conflicts in the future

An algorithm to guess a conflict's source

We now provide an algorithm that implements the unsupervised CBR phases described above. Algorithm 1 illustrates the implementation of function *getSource*, which BASE employs to guess the source of a given conflict. It takes as input a description (d_s) of the state s previous to a conflict, and a description of a conflict ($c^{s'}$) in a state s'. It starts by carrying out *case retrieval*. It creates a new case description containing d_s and $c^{s'}$ (line 2). Then, it retrieves the case whose description is most similar to the recently created description (line 3). With this aim, it employs function *getSimilarity* described in Equation 4.1 above. Thereafter, it *reuses* the case. If a similar case was retrieved, then it retrieves its description (line 5). If the descriptions of the current case and the similar case are different, then it retrieves the best solution of the similar case (line 7). If there is a solution available, then it employs function *exploitSolution* (line 9) to exploit it, thus choosing an agent as responsible for conflict $c^{s'}$. Otherwise, if the descriptions of the current case and the similar case are equal, then it chooses an alternative agent as the source of the conflict (line 11). By contrast, if no similar case was retrieved, it randomly chooses an agent as the source of conflict $c^{s'}$ (line 13). Finally, it retains the case in the case base (lines 14–15).

ALGORITHM 1: Function *getSource*

Input : $d_s, c^{s'}$
Output: ag

```
    /* Retrieve phase:                                                    */
[1] ag ← null;
[2] desc ← ⟨d_s, c^{s'}⟩;
[3] simCase ← getMostSimilarCase(desc);
    /* Reuse phase:                                                       */
[4] if simCase ≠ null then
[5]     simDesc ← getDescription(simCase);
[6]     if desc ≠ simDesc then
[7]         bestSol ← getBestSolution(sim);
[8]         if bestSol ≠ null then
[9]             ag ← exploitSolution(bestSol, simDesc, desc);
[10]    else
[11]        ag ← getAlternativeAgent(simCase);
[12] if ag = null then
[13]    ag ← getRandomAgent(c^{s'});
    /* Retain phase:                                                      */
[14] case ← ⟨desc, {ag, 0}⟩;
[15] CB ← addToCaseBase(case, CB);
[16] return ag;
```

4.4 Evaluating norms' performances

At each time step, BASE evaluates each norm individually to assess how successful it is in regulating conflicts. BASE considers a norm is successful if it (1) avoids conflicts after agents fulfil it (i.e., it is *effective*); and (2) conflicts arise after agents infringe it (i.e., it is not *unnecessary*). Therefore, to evaluate norms, BASE considers a *norm evaluation criterion* $\langle \mu_{success}, \alpha_{success} \rangle$ (see Section 3.2.2), being $\mu_{success}$ a function that computes how successful a norm is in regulating conflicts after the agents fulfil and infringe the norm, and $\alpha_{success} \in [0, 1]$ a *satisfaction degree* that sets the minimum utility $\mu_{success}$ for a norm to be active.

In particular, function $\mu_{success}$ computes the utility of a norm as its ratio of *successful outcomes* for a certain period of time, namely its ratio of successful fulfilments (fulfilments that did not lead to conflicts) and harmful infringements (infringements that led to conflicts). With this aim, it proceeds as follows. Whenever a norm n is fulfilled or infringed at a given time t, BASE first computes the number of successful outcomes of n at time t as follows:

$$so(n, t) = sf_t^n + hi_t^n \tag{4.2}$$

where $n \in \mathcal{N}$ and $t \in \mathbb{N}$, being $\mathcal{N}$ the set of norms in the normative network; and $sf_t^n \in \mathcal{SF}^n$, $hi_t^n \in \mathcal{HI}^n$ stand for the number of successful fulfilments and

harmful infringements of norm n at time t, respectively (see Section 3.3.3).

Analogously, it computes the number of unsuccessful outcomes of n at time t as its number of harmful fulfilments at time t (fulfilments that led to conflicts), and its number of successful infringements at time t (infringements that did not lead to conflicts).

$$uo(n,t) = hf_t^n + si_t^n \tag{4.3}$$

where $hf_t^n \in \mathcal{HF}^n$, $si_t^n \in \mathcal{SI}^n$ stand for the number of harmful fulfilments and successful infringements of norm n at time t, respectively (see Section 3.3.3).

Thereafter, BASE computes a new *evaluation* for norm n at time t that determines how successful it has been in regulating conflicts at time t. With this aim, it employs function *evaluation* described below.

$$evaluation(n,t) = \begin{cases} \dfrac{so(n,t)}{so(n,t) + uo(n,t)} & \text{if } applicable(n,t) = true \\ \\ \bot & \text{otherwise} \end{cases} \tag{4.4}$$

which computes a norm evaluation as the ratio of successful outcomes of n at time t iff the norm was applicable at time t, namely if *applicable(n,t) = true*, and returns an *undefined* evaluations $\bot$ if n has not been applicable at time t. In particular, function *applicable(n,t)* returns *true* if there are compliance outcomes of n at time t, that is, if n has either successful outcomes or unsuccessful outcomes at time t. Formally:

$$applicable(n,t) = \begin{cases} true & \text{if } so(n,t) + uo(n,t) > 0 \\ false & \text{otherwise} \end{cases} \tag{4.5}$$

Finally, BASE computes the utility of a norm ($\mu_{success}$) as its *averaged* evaluations along time. In particular, it only considers the latest q evaluations that are *not undefined*. In other words, it considers the values contained in a *window* containing the latest q defined evaluations. This q-window allows to compute a norm's utility based on the most recent q evaluations of n, allowing to *"forget"* old evaluations that may no longer be relevant at time t.

With this aim, let us first define a set R_l^n of defined evaluations of n from a given time l:

$$\mathcal{DE}_l^n = \{evaluation(n,j) \,|\, l \leq j \leq m, evaluation(n,j) \neq \bot\} \tag{4.6}$$

Then, we consider an index $i(q)$ such that $|de_{i(q)}^n| = q$ and $1 \leq i(q) \leq m$. Thus, set $de_{i(q)}^n$ contains the latest q defined evaluations of n. Finally, BASE computes the utility of a norm n at time t as the average of the values in $de_{i(q)}^n$.

$$\mu_{success}(n,t,q) = \frac{\sum\limits_{de \in \mathcal{DE}_{i(q)}^n} de}{q} \tag{4.7}$$

where $n \in \mathcal{N}$, $t \in \mathbb{N}$, and $1 \leq q \leq t \leq m$.

Note that this approach to norm evaluation supports the indirect evaluation of multiple norms that apply simultaneously in a given state s. Although norms are evaluated individually, their evaluation depends on the state s_c reached after they are fulfilled and infringed, and this state depends on *all* the norms applicable in previous state s. Thus, if the application of a group of norms leads to a non-conflictive situation in s_c, the score of each norm would increase, while if their application leads to a conflict in s_c, the score of each norm would decrease. It is natural to see that this may lead to sporadic erroneous norm evaluations, since multiple norms may be evaluated as unsuccessful at a given time based on a conflict that was caused by only one of these norms. Nevertheless, norm evaluation is performed in an iterative manner, thus cumulating punctual norm evaluations along time. This allows to minimise the impact of sporadic erroneous norm evaluations, making the method quite robust.

4.5 Refining the normative system

During norm refinement, BASE exploits norms' performances to yield normative systems whose norms are successful enough in terms of some norm evaluation criteria. In this way, it tries to synthesise a normative system whose norms are successful to regulate conflicts, hence solving the norm synthesis problem (cf. Definition 18 in Section 3.2.2). In particular, BASE considers that a norm is *unsuccessful* if its utility ($\mu_{success}$) is *below* threshold $\alpha_{success}$, introduced in Section 4.4. This amounts to satisfying the following condition:

$$\mu_{success}(n, t, q) < \alpha_{success} \tag{4.8}$$

where $\mu_{success}(n, t, q)$ stands for the utility of norm n at time t, given a q-window of n's evaluations.

Whenever BASE detects that condition 4.8 above holds for a norm n, then it deactivates the norm in the normative network by invoking operator $deactivate(n, NN)$ described in Table 3.1 in Section 3.4.2. In this way, the norm is removed from the normative system.

4.5.1 Evaluating normative systems

Additionally, we now introduce a means to compute the overall utility of a normative system at a given time. The utility ($\mu_{success}$) of a normative system Ω can be computed as a whole up to a given time t based on the average utility of its norms for the last q utility values up to time t.

$$\mu_{success}(\Omega, t, q) = \frac{\sum_{n \in \Omega} \bar{\mu}_{success}(n, t, q)}{|\Omega|} \tag{4.9}$$

4.6 BASE's synthesis strategy

At this point, we have provided enough background to introduce BASE's synthesis strategy. BASE is intended to be executed y a NSM to synthesise normative systems that effectively avoid conflicts. With this aim, BASE monitors the evolution of a MAS at regular time intervals (i.e., ticks), and carries out a norm synthesis process by performing the three synthesis stages of *norm generation*, *norm evaluation*, and *norm refinement*. During norm refinement, BASE creates norms as described in Section 4.3. During norm evaluation, it evaluates norms as detailed in Section 4.4. Finally, during norm refinement, BASE refines the normative system as described in Section 4.5. During this synthesis process, it changes from one normative system to another by modifying the normative network described in Section 3.4.1 through the collection of operators in Table 3.1 (Section 3.4.2). As a result of norm synthesis, BASE publishes a normative system Ω so that the agents within a MAS become aware of it.

Algorithm 2 illustrates BASE's strategy, which takes as input a tuple with a description of the previous state of a MAS (d_s) and a description of the current MAS state (d_{s_c}), and outputs a normative system to regulate the agents' behaviour. To perform norm synthesis, BASE considers the following globally accessible elements:

- A normative network NN, formally defined in Section 3.4.1.
- A set of operators $\mathcal{O} = \langle add, activate, deactivate \rangle$ described in Section 3.4.2.
- A set Φ of domain-dependent elements (box c in Figure 3.3 in Section 3.4) to perform norm synthesis in a given application domain. Specifically, $\Phi = \langle perceive, getConflicts, getContext, getAction, \mathcal{G} \rangle$. Functions *perceive*, *getConflicts*, *getContext* and *getAction* correspond to the domain-dependent functions introduced in Chapter 3 to (1) describe a MAS state from a global perspective (see Section 3.3.1); (2) retrieve the conflicts in a given MAS state (see Section 3.3.1); (3) yield an agent's context in a given MAS state (see Section 3.3.2); and (4) yield an agent's action in a transition state (see Section 3.3.2). Additionally, $\mathcal{G}$ is a grammar to construct norms, formally described in Section 3.3.2.
- A set Ψ of domain-independent norm synthesis settings (box d in Figure 3.3 in Section 3.4). Particularly, $\Psi = \langle \mathcal{EC}, T \rangle$, being $\mathcal{EC}$ a set of norm evaluation criteria (cf. Section 3.3.3), and T a time interval to determine when a NSM has converged to a normative system (cf. Definition 18 in Section 3.2.2). In particular, BASE will take in $\mathcal{EC} = \langle \mu_{success}, \alpha_{success} \rangle$ (introduced in Section 4.4).
- A set Γ of additional synthesis inputs (box e in Figure 3.3 in Section 3.4), which contains (1) a case base ($\mathcal{CB}$) that BASE employs during norm generation (see Section 4.3); (2) function *getSimilarity* considered by BASE

during norm generation to compute the similarity between two cases (see Section 4.3.2); (3) function *exploitSolution* employed by BASE during norm generation to adapt a exploit a case solution (see Section 4.3.2); and (4) a constant q that BASE employs to compute norms' performances during norm evaluation (see Section 4.4).

ALGORITHM 2: BASE's synthesis strategy

Input : $\langle d_s, d_{s_c} \rangle$
Output : Ω
Initialisations: $\mathcal{NCO} \leftarrow \emptyset$, $\mathcal{P} \leftarrow \emptyset$

[1] $NN \leftarrow$ `normGeneration(`$\langle d_s, d_{s_c} \rangle, \mathcal{P}$`)`;
[2] $\mathcal{P} \leftarrow$ `normEvaluation(`$\langle d_s, d_{s_c} \rangle, \mathcal{NCO}, \mathcal{P}$`)`;
[3] $NN \leftarrow$ `normRefinement(`$\mathcal{NCO}, \mathcal{P}$`)`;
[4] $\Omega \leftarrow \{n \in NN \mid \delta(n) = active\}$;
[5] **return** Ω

Additionally, BASE considers the following data structures that it initialises the first time it is invoked:

1. A structure ($\mathcal{NCO}$) to keep track of the compliance outcomes of each norm (see Section 3.3.3)

2. A structure ($\mathcal{P}$) to keep track of the performance ($\mu_{success}$) of each norm it creates during norm synthesis.

Each time BASE is invoked by a NSM, it starts by carrying out norm generation. With this aim, function *normGeneration* in line 1 detects unregulated conflicts in the current MAS state (s_c) and creates (and adds to the normative network) norms to avoid detected conflicts in the future as discussed in Section 4.3. Next, *normEvaluation* in line 2 evaluates norms in terms of their capability to regulate conflicts as discussed in Section 4.4. As a result, it outputs a set $\mathcal{P}$ of norms' performances, which keeps track of the performances of each norm in regulating conflicts. Then, the norm refinement function (line 3) refines the normative system based on computed norms' performances ($\mathcal{P}$) as detailed in Section 4.5. Finally, BASE publishes a normative system for the agents (line 4). This includes the *active* norms in the normative network. Subsequent sections detail each synthesis stage.

4.6.1 Norm generation

As detailed above, during norm generation, BASE detects conflicts and creates norms. Algorithm 3 depicts BASE's norm generation. It starts by perceiving conflicts in the current MAS state (line 1) by invoking function *getConflicts*. Then, for each detected conflict, it creates a norm as follows. If the conflict is not yet regulated (cf. Definition 9 in Section 3.2), BASE:

1. Guesses the conflict source by choosing an agent as responsible for conflict c^{s_c} (line 4). With this aim, it invokes function *getSource* described in Algorithm 1 (Section 4.3.2).

2. Retrieves the local context of ag in the previous MAS state s (line 5).

3. Retrieves the action that ag performed during the transition from state s to s_c (line 6).

4. Employs grammar $\mathcal{G}$ to create a new norm that prohibits to perform the retrieved action for any agent encountering the retrieved context (line 7).

5. Adds the recently created norm to the normative network (if it does not exist yet), and activates it. With this aim, if employs operator *add* (line 9), and operator *activate* (line 10).

ALGORITHM 3: BASE *normGeneration*

Input : $\langle d_s, d_{s_c} \rangle, \mathcal{P}$
Output: Ω

```
[1]  conflicts_{s_c} ← getConflicts (d_{s_c});
[2]  for c_{s_c} ∈ conflicts_{s_c} do
[3]      if not regulated(c_{s_c}, d_s) then
[4]          ag ← getSource(d_s, c_{s_c}, P);
[5]          context ← getContext(ag, s);
[6]          action ← getAction(ag, s, s_c);
[7]          n ← buildNorm(context, action, G);
[8]          if n ∉ getNorms(NN) then
[9]              NN' ← add(n, NN);
[10]         NN' ← activate(n, NN);
[11] return NN;
```

4.6.2 Norm evaluation

During norm evaluation (line 2 in Algorithm 2), BASE evaluates norms as described in Section 4.4. Algorithm 4 illustrates BASE's *normEvaluation* function, which starts by invoking function *getComplianceOutcomes* (line 1). It updates norms' compliance outcomes ($\mathcal{NCO}$) in current state s_c as described in Section 3.3.3. Next, function *evaluateNorms* (line 2) evaluates norms based on their compliance outcomes by employing Equation 4.7. It outputs a set $\mathcal{P}$ containing updated values of each norm's performance. Finally, in line 3 it invokes function *updateCases*, which employs the set of norms' performances ($\mathcal{P}$) to update the utilities of their corresponding case solutions in the case base (see Section 4.3.2).

ALGORITHM 4: BASE's *normEvaluation*

Input : $\langle d_s, d_{s_c} \rangle, \mathcal{NCO}, \mathcal{P}$
Output: $\mathcal{P}$

```
[1] NCO ← getComplianceOutcomes(⟨d_s, d_sc⟩, NCO);
[2] P   ← evaluateNorms(NCO, P);
[3] CB  ← updateCases(P);
[4] return P;
```

4.6.3 Norm refinement

The final step of the BASE strategy is the refinement of norms (line 3 of Algorithm 2). During this stage, BASE refines the normative system as detailed in 4.5. In short, it discards those norms that are unsuccessful to regulate conflicts. Algorithm 5 illustrates BASE's norm refinement stage. For each norm that has been fulfilled and infringed during the transition to the current MAS state (line 1), it retrieves its utility from set $\mathcal{P}$ (line 2). Then, it deactivates the norm in the normative network in case its utility is under threshold $\alpha_{success}$ (lines 3–4). As a result, the norm will no longer belong to the normative system.

ALGORITHM 5: BASE *normRefinement*

Input : $NN, \mathcal{O}, \Psi, \mathcal{NCO}, \mathcal{P}$
Output: NN

```
[1] for n ∈ getNormsFulfilledInfringedThisState(NCO) do
[2]     u_n ← getUtility(n, P);
[3]     if u_n < α_success then
[4]         NN' ← deactivate(n, NN);
[5] return NN;
```

4.7 Complexity analysis

At this point, the complexity of BASE can be computed. Before that, let us consider the number of norms that can be generated by a particular grammar $\mathcal{G}$. If p is the maximum number of predicates of any norm generated by the grammar, r is the maximum arity of any predicate, and d is the maximum number of terms at any position of any given predicate, the number of norms that can be generated by grammar $\mathcal{G}$ is $d^{r \cdot p}$. Given a grammar $\mathcal{G}$, let $\eta_{\mathcal{G}}$ note the number of norms that $\mathcal{G}$ can generate. Moreover, given a CBR base CB, let η_{CB} note the number of norms in the case base. With all this in mind, BASE's complexity is as follows.

Lemma 1. *The norm synthesis performed by the* BASE *algorithm when employing grammar $\mathcal{G}$ and case base CB when detecting κ conflicts takes time* $O(\kappa \cdot \eta_{CB} + 3 \cdot |Ag| \cdot |NN| + |NN|)$.

Proof. The norm generation stage involves generating norms for each detected conflict. This operation has cost $O(\kappa \cdot \eta_{CB})$. The cost of the norm evaluation process is $O(3 \cdot |Ag||NN|)$, since it involves assessing the applicability of norms ($O(|Ag||NN|)$), assessing the compliance with norms ($O(|Ag||NN|)$), and updating norms' utilities ($O(|Ag||NN|)$). Finally, the cost of norm refinement is $O(|NN|)$, which amounts to the worst case cost of considering all the norms in the normative network to decide either to discard or to preserve them. Putting all together, the resulting worst-case time is $O(\kappa \cdot \eta_{CB} + 3 \cdot |Ag| \cdot |NN| + |NN|)$. □

As a final remark, notice that given a grammar $\mathcal{G}$, the number of normative systems is $2^{\eta_{\mathcal{G}}}$. This is precisely the size of the search space that BASE must explore in search for compact normative systems. However, recall that BASE is an approximate algorithm for norm synthesis, and it does not require exploring the whole search space, as it will be demonstrated in Section 4.8.

4.8 Empirical evaluation

We now empirically evaluate BASE's norm synthesis. The aim is to demonstrate not only BASE's performance, but also the validity of the norm synthesis model introduced in Chapter 3. In these experiments BASE is iteratively executed by a NSM that runs on top of a simulated MAS. For this experiment, we have chosen a scenario we are already familiar with: the road traffic scenario. Hereafter, Section 4.8.1 describes the empirical settings of this evaluation, whereas Section 4.8.2 illustrates its empirical results.

4.8.1 Empirical settings

BASE has been empirically evaluated in a discrete agent-based simulation of the traffic junction scenario introduced in Section 1.3.1, whose graphical representation is depicted here again for convenience. Figure 4.3a shows a 11×11 sub grid that corresponds to the centre of a traffic junction modelled as a 21×21 grid, with cars travelling towards randomly chosen destinations. Each car has been modelled as an agent that can perceive the three cells in front of it (Figure 4.3b). The actions available to the agents are $Ac = \{go, stop\}$. Particularly, agents are permitted to *go* by default unless a norm prohibits them to go forward.

In this scenario, BASE is aimed at synthesising norms that avoid collisions between cars. Therefore, we consider a scenario that potentially leads to a large number of collisions. With this aim, we use a high traffic density (from 41% to 48% of occupied cells) by having three cars entering the scenario per time step (i.e., tick). At each tick, each car decides whether to comply or not with the norms published by BASE according to some probability, namely the *norm infringement rate* ($\mathcal{NIR}$). This norm infringement rate is fixed at the beginning of each simulation and is the same for all cars.

In these experiments, BASE starts each simulation with an empty normative system. As the simulation progresses, collisions among cars occur, and BASE

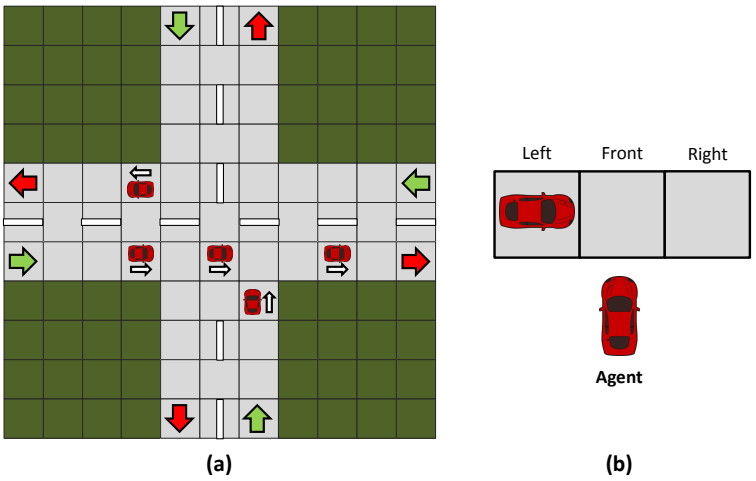

Figure 4.3: a) Central area of our traffic junction scenario. b) An agent's context.

synthesises normative systems to avoid future collisions. A simulation finishes whenever it reaches 10,000 ticks or BASE converges to a stable normative system that avoids unregulated conflicts, hence solving the norm synthesis problem. Subsequent sections describe (1) the implementation of the different domain-dependent elements employed by BASE in this scenario (inputs Φ, Γ in Figure 3.3 in Section 3.4); and (2) the norm synthesis settings that BASE considers during the synthesis process (input Ψ in Figure 3.3 of Section 3.4).

Norm synthesis in the traffic scenario

We now describe how each domain-dependent element required by BASE has been implemented for the traffic scenario. First, we describe each domain-dependent element required by a NSM to perform norm synthesis (Φ). These elements are:

– **Function *perceive***. BASE describes a traffic state (i.e., a grid) by generating an expression of $\mathcal{L}_{MAS}$ that describes each particular cell in the state. With this aim, it employs language $\mathcal{L}_{MAS}$ described in Section 3.2.1 to illustrate the traffic example.

– **Function *getContext***. Given a state s of the traffic scenario, and a car that is part of s, function *context* infers the car's context in s by describing the *three* cells in front of the car from the car's perspective. With this aim, it employs language $\mathcal{L}_{Ag}$ described in Section 3.2.1 to illustrate the traffic example.

– **Function *getAction***. Given a transition between two states s, s' of a traffic junction, and a car that is part of states s, s', function *action* returns action *"go"* if the car has changed its position during the state transition. Otherwise,

it returns action *"stop"*.

– **Function *getConflicts***. Given a state s, function *getConflicts* returns a (possibly empty) set of expressions of $\mathcal{L}_{MAS}$, each describing a conflict in s. An example is illustrated in Section 4.3.1.

– **Grammar $\mathcal{G}$**. The grammar BASE employs to generate norms is the same grammar described in Section 3.3.2 to illustrate norm generation, which we regard here again for convenience:

$$
\begin{array}{lll}
\text{Norm} & ::= & \langle \{LHS\}, RHS \rangle \\
\text{LHS} & ::= & LHS, LHS \mid \rho \\
\text{RHS} & ::= & prh(Ac) \\
Ac & ::= & go \mid stop \\
\rho & ::= & left(\tau) \mid front(\tau) \mid right(\tau) \\
\tau & ::= & car\text{-}heading\text{-}left \mid car\text{-}heading\text{-}right \mid car\text{-}opposite\text{-}heading \mid \\
 & & car\text{-}same\text{-}heading \mid wall \mid nil
\end{array}
$$

In particular, the precondition of each norm has three predicates (*left, front, right*), and each predicate may have one out of 6 different terms. Therefore, the employed grammar can synthesise $6^3 = 216$ different norms, and the number of normative systems to consider amounts to 2^{216} ($> 10^{65}$).

We now describe the implementation for this scenario of each domain-dependent function that BASE's *getSource* function (see Section 4.6.1) requires to guess the source of a given conflict. That is, function *getSimilarity* to compute the similarity between two cases, and function *exploitSolution* to exploit a case solution.

– **Function *getSimilarity***. In this scenario, the similarity between two case descriptions $\langle d_s^a, c_a^{s'} \rangle, \langle d_s^b, c_b^{s'} \rangle$ is computed as the ratio of predicates that are equal in descriptions d_s^a, d_s^b. Figure 4.4 illustrates an example. It illustrates a graphical representation of two states s and s' of a traffic junction, highlighting each cell of state s that is different from its analogous cell in state s'. In the example, 7 of 9 cells are equal in the two states. Therefore, BASE considers that they are 78% similar, that is, $getSimilarity(s, s') = 0.78$. Note though that two states may be *symmetric*, namely they may be equal but one of them may be rotated with respect to the other. Our similarity function considers this possibility while computing the similarity between two states , and compares each pair of states by performing rotations of $0°, 90°, 180°$, and $270°$ to one of them.

– **Function *exploitSolution***. In this scenario, we consider that similar collisions are caused by cars in similar positions. Therefore, given a case to solve (*case*), and a similar, solved case ($case_{sim}$), this function takes the car of the solution in $case_{sim}$, retrieves its position in $case_{sim}$ before colliding, and proposes the car in that position in the case to solve (*case*). Figure 4.4 (Section 4.3.2) depicts an example. There, BASE exploits the solution in $case_{sim}$ (i.e., blaming car a for the collision in $c_{sim}^{s'}$) to choose a source for the car collision in $c^{s'}$. With

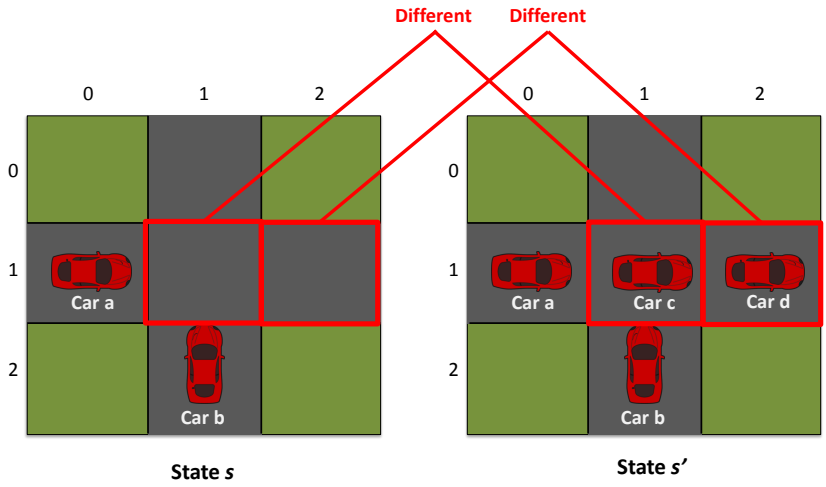

Figure 4.4: Equal cells and different cells in two different traffic states s and s'.

Parameter	Description	Value
q	Number of evaluations considered to compute a norm's utility ($\mu_{success}$)	100
$\alpha_{success}$	Threshold below which a norm is considered as unsuccessful	0.2
α_{sim}	Threshold above which two case problems are considered as similar	0.9
T	Time period considered when assessing convergence (cf. Definition 18 in Section 3.2.2)	5,000

Table 4.1: BASE's norm synthesis settings

this aim, it proposes as the conflict's source the car in the same position of car a in state s_{sim} in $case_{sim}$. Particularly, this process considers that two cases may be symmetric.

Norm synthesis settings

Finally, BASE's norm synthesis parameters have been set as shown in Table 4.1. In short, we have taken a conservative approach to configure BASE. Firstly, BASE considers a great amount of evidences when computing a norm's utility. In particular, the utility of a norm n at time t is computed by considering the last 100 undefined evaluations of n up to time t ($q = 100$). Secondly, BASE refines the normative system by deactivating norms only when they perform very poorly ($\alpha_{success} = 0.2$). Thirdly, BASE considers that two cases are similar when their similarity is greater than 90% ($\alpha_{sim} = 0.9$). Finally, BASE considers

a convergence interval of 5,000 ticks ($T = 5,000$). Thus, BASE is said to have converged to a normative system, hence solving the norm synthesis problem, if during a 5,000-tick period: (i) the normative system remains unchanged; and (ii) no new (unregulated) conflicts are detected.

4.8.2 Empirical results

We now analyse BASE's results. With this aim, this section provides:

- A micro analysis of BASE's convergence process. It analyses *how* BASE manages to synthesise a stable normative system that solves the norm synthesis problem; and *what* type of normative systems BASE synthesises.
- A macro analysis of BASE's performance for a given number of simulations. It analyses the dispersion of the distribution of the normative systems synthesised by BASE out of a number of different simulations.
- An analysis of BASE's robustness (i.e., its convergence rate) for different norm infringement rates ($\mathcal{NIR}$) of the agent society.

Micro analysis: BASE's convergence process

This section illustrates a prototypical execution of BASE in the road traffic scenario to show how it synthesises a normative system that successfully avoids collisions. Figure 4.5 depicts a run of BASE with 0.1 norm infringement rate ($\mathcal{NIR}$). That is, on average, 1 of each 10 agents' decisions lead to norm infringements. On the x-axis, it shows the normative changes (i.e., the ticks at which the the normative network and/or the normative system changed) for a single simulation. On the y-axis, it depicts (1) the ratio of unregulated car collisions at a given tick[1]; (2) BASE's normative network cardinality (the total number of synthesised norms); and (3) the cardinality of the normative system provided to the agents.

At tick 12 (which corresponds to the second normative change), the first collision arises and BASE synthesises the first norm. From that tick onwards, BASE keeps generating norms when needed, hence increasing the cardinality of both the normative network and the normative system. At tick 309 (twelfth normative change), BASE generates the last norm, yielding a normative system with 16 norms. From that point onwards, unregulated collisions are reduced to 0, and the normative system remains stable. By using the resulting normative system, cars that comply with norms do not cause collisions. After 5,000 further ticks (at tick 5309), BASE fulfils the convergence criteria, after synthesising a normative system that, during a 5,000-tick period, remained unchanged and avoided unregulated collisions. Overall, BASE created 16 different norms (out of 216 possible ones), converging to a 16-norm normative system that solved the norm synthesis problem.

[1] Computed as the moving average of unregulated collisions of the last 10 ticks.

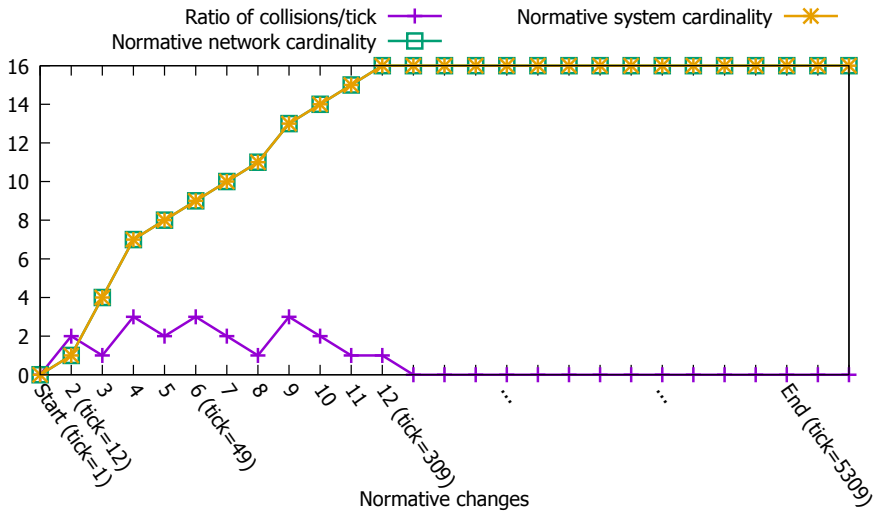

Figure 4.5: Prototype execution of BASE.

Note that both the normative network and the normative system have he same cardinality along all the simulation. This happens because *all* the norms in the normative network remain active during the whole simulation, and thus are included in the normative system. Particularly, synthesised norms are never deactivated since they are successful in regulating collisions.

It is worth noticing that, though in this experiment BASE starts a simulation with an empty normative system, it may also start operating with an initial normative system containing norms. In that case, BASE would provide the initial normative system to the agents, who would be able to fulfil and infringe its norms. BASE would evaluate these norms in terms of their compliance outcomes, and would refine the normative system by discarding those norms that are unsuccessful to avoid conflicts, while preserving successful norms. Therefore, BASE may be employed not only to synthesise a normative system for a given MAS, but also to evaluate how a particular normative system would perform in achieving coordination in a given MAS.

Table 4.2 shows the normative system BASE converged to in this run. Each norm establishes a prohibition for a reference car to go forward whenever it encounters different situations. Norms n_1 to n_6 represent variations of a general left-hand side priority norm. They prohibit a car to go whenever it perceives, to its left, a car heading to the right from its perspective, and different variations to its front and right positions. Norms n_7 to n_{11} represent variations of a general *"keep your distance"* norm. They prohibit a car to go if it perceives a car in front heading to different directions. In these situations, a car should stop "just in case" the car in front stops. Finally, norms n_{12} to n_{16} represent different variations of a general right-hand side priority norms. Note though that left-hand side priority norms and right-hand side priority norms can replace one another

Norm	Pre-condition (θ)			Norm target
n_1	*left(car-to-right)*,	*front(car-to-right)*,	*right(nil)*	*prh(go)*
n_2	*left(car-to-right)*,	*front(car-to-right)*,	*right(car-to-right)*	*prh(go)*
n_3	*left(car-to-right)*,	*front(car-same-heading)*,	*right(nil)*	*prh(go)*
n_4	*left(car-to-right)*,	*front(nil)*,	*right(nil)*	*prh(go)*
n_5	*left(car-to-right)*,	*front(nil)*,	*right(car-to-right)*	*prh(go)*
n_6	*left(car-to-right)*,	*front(car-same-heading)*,	*right(car-to-right)*	*prh(go)*
n_7	*left(nil)*,	*front(car-same-heading)*,	*right(wall)*	*prh(go)*
n_8	*left(nil)*,	*front(car-same-heading)*,	*right(nil)*	*prh(go)*
n_9	*left(nil)*,	*front(car-same-heading)*,	*right(car-to-right)*	*prh(go)*
n_{10}	*left(nil)*,	*front(car-to-right)*,	*right(car-to-right)*	*prh(go)*
n_{11}	*left(car-to-left)*,	*front(car-to-left)*,	*right(nil)*	*prh(go)*
n_{12}	*left(car-to-left)*,	*front(car-same-heading)*,	*right(car-to-left)*	*prh(go)*
n_{13}	*left(car-to-left)*,	*front(car-to-left)*,	*right(car-to-left)*	*prh(go)*
n_{14}	*left(car-to-left)*,	*front(nil)*,	*right(car-to-left)*	*prh(go)*
n_{15}	*left(nil)*,	*front(car-to-left)*,	*right(car-to-left)*	*prh(go)*
n_{16}	*left(nil)*,	*front(car-same-heading)*,	*right(car-to-left)*	*prh(go)*

Table 4.2: A normative system synthesised by BASE upon convergence.

in use. That is, to avoid collisions, a car should give way either to its left (norms n_1 to n_6) or to its right (norms n_{12} to n_{16}), but not to both sides. Therefore, BASE's normative system contains more norms than necessary, and thus over-constrains the agents' freedom. This comes as a consequence of BASE's approach to detect unnecessary norms. As discussed in Section 4.4, BASE evaluates norms *individually* in terms of their effectiveness and necessity. However, individually evaluating norms is not enough to detect if a norm becomes unnecessary when another norm exists. Instead, it is necessary to detect the *synergies* between these norms to be able to disregard one of them, while preserving the other.

Macro analysis

Next, we perform a macro analysis of BASE's performance. First, we analyse the dispersion of the distribution of the normative systems synthesised by BASE. For this purpose, we now focus on the histogram in Figure 4.6. On the x-axis, it shows different normative systems synthesised by BASE in 100 different simulations, each with a 0.1 infringement rate. On the y-axis, it shows the number of times each normative system was synthesised out of these 100 simulations. Overall, BASE converged to 7 different normative systems. Notice that BASE's dispersion is low, since it converged to only 7 different normative systems out of 10^{65} possible normative systems. In fact, BASE synthesised 81 times (81%) the first 2 normative systems (Ω_1, Ω_2), which at convergence time had, on average, respective utilities 0.71 and 0.69 (computed as described in Equation 4.9. To summarise, BASE consistently focused on an area of the space of normative systems where successful normative systems are (namely, normative systems that can avoid unregulated collisions).

Next, we explore BASE's robustness by testing its synthesis capabilities under

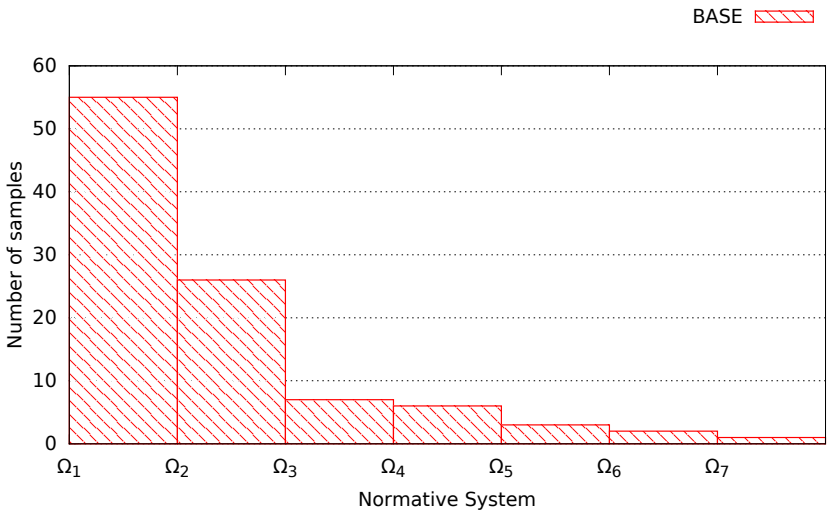

Figure 4.6: Histogram depicting the dispersion of normative systems synthesised by BASE out of 100 simulations.

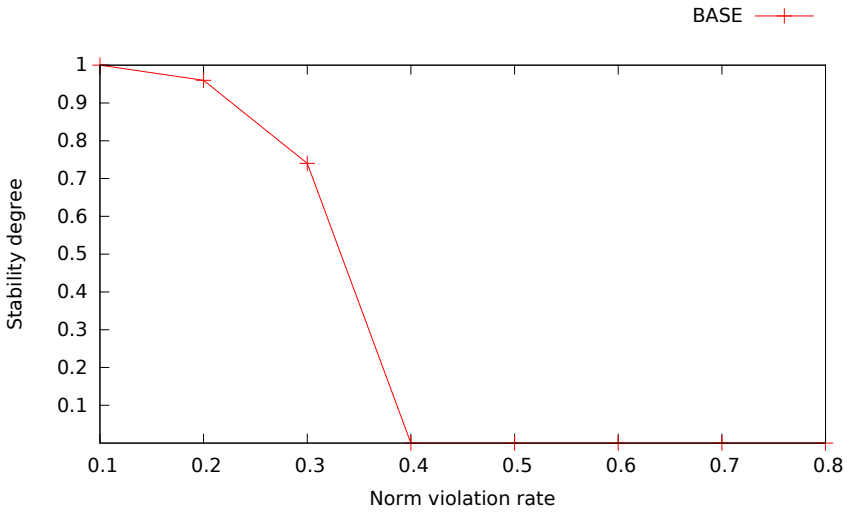

Figure 4.7: Stability degree of BASE for different norm infringement rates.

different norm infringement rates. With this aim, we analyse BASE's stability degree for different norm infringement rates ($\mathcal{NIR}$), ranging from 0.1 to 0.9. Particularly, BASE's stability degree is computed as the ratio of simulations that converged to a stable normative system. We performed 100 simulations per norm infringement rate. The analysis is performed for very low (0.1), low (0.2), medium (0.3 to 0.6) and high (beyond 0.6) norm infringement rates. Figure 4.7 shows, for each norm infringement rate, the ratio of simulations that converged to a stable normative system. For very low norm infringement rates (0.1), BASE successfully converged to a normative system that effectively solved the norm synthesis problem for all the simulations. Nevertheless, beyond low norm infringement rates (0.2), BASE decreased its stability degree (i.e., it became more inefficient in synthesising a stable normative system). In fact, it could not manage to converge to a normative system beyond the 0.3 norm infringement rate, and collisions were never completely eradicated, hence leading to a 0 stability degree.

This lack of stability of BASE stems from its approach to refine the normative system (cf. Section 4.6.3). Briefly, the utility of norms ($\mu_{success}$) fluctuate above and below the deactivation threshold ($\alpha_{success}$) along time, provoking that BASE continuously deactivates/re-activates norms, and making impossible to converge to a stable normative system. Let us illustrate an example with the chart in Figure 4.8. On the x-axis, it shows consecutive evaluations of a norm n as evaluated by BASE. On the y-axis, it shows (1) the utility ($\mu_{success}$) of norm n, (2) the *trend* of n's utility (computed as the power trend of $\mu_{success}$), and (3) threshold $\alpha_{success}$, below which n is considered as unsuccessful. Note that n's long-term utility (i.e., its trend) remains above the deactivation threshold along time. Therefore, in general, n may be considered as successful. However, n's utility ($\mu_{success}$) has short-term fluctuations, punctually going below threshold $\alpha_{success}$ (norm evaluations 9 and 15). When that happens, BASE deactivates the norm and discards it from the normative system. Thereafter, the conflict the norm avoided is no longer regulated, and hence the next time the conflict arises, BASE may create (i.e., re-activate) the norm again, and add it to the normative system. This may happen several times along a simulation, provoking that BASE continuously deactivates and re-activates norms, and making impossible to converge to a stable normative system.

4.9 Conclusions

This chapter has introduced BASE, a synthesis strategy intended to be executed by a Norm Synthesis machine (NSM) to perform norm synthesis. BASE implements the norm synthesis model introduced in Section 3.3 and pursues the synthesis of effective normative systems that successfully avoid conflicts within a MAS. BASE creates new norms by learning from experience. Specifically, once BASE creates a norm to regulate a conflict, it keeps detailed information about the conflict, how BASE determined its source, and the norm created to regulate it. Thereafter, if the norm succeeds in regulating the conflict, BASE employs this

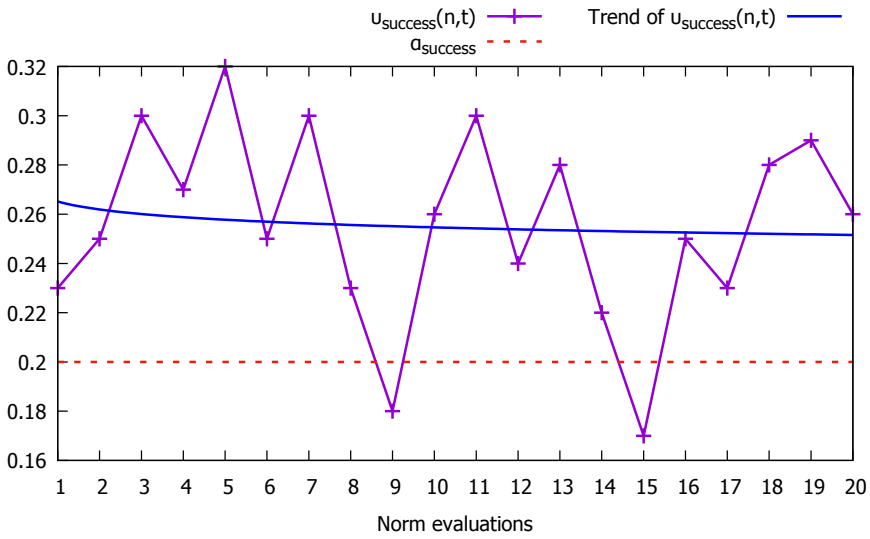

Figure 4.8: Utility ($\mu_{success}$) of a norm n along time.

information to regulate similar conflicts, thus synthesising similar norms.

BASE has been evaluated in an agent-based simulation of a the road traffic scenario introduced in Section 1.3.1. It has been proven to be capable of synthesising a normative system that successfully avoids car collisions as long as cars comply with norms. BASE's performance has been analysed along several dimensions. We have studied the type and distribution of normative systems BASE synthesises, and we have analysed its robustness.

However, from the observations realised in BASE's empirical results, here we argue that it suffers from some limitations:

1. Lack of compactness. As argued in Section 1.1.3, there are further synthesis objectives that can be considered in on-line norm synthesis in addition to simply avoiding conflicts effectively. More specifically, normative systems should be ideally as *compact* as possible to reduce the agents' computational efforts when reasoning about norms. Against this background, here we argue that the normative systems that BASE synthesises could be more compact. As an example, in the road traffic scenario (see Section 4.8), BASE synthesised multiple variations of a general left-hand side priority norm. It is natural to think that these norms could be compactly represented by a more general norm like

$$\langle \{left(car\text{-}to\text{-}right)\},\ prh(go) \rangle$$

which represents a general left-hand side priority norm. It prohibits a car to go if it perceives a car on the left, and no matter what it perceives to its front and right positions. In this way, the number of norms and terms that agents should consider to detect norm applicability could be reduced, thus easing norm reasoning.

2. Lack of liberality. As discussed in Section 1.1.3, a crucial synthesis objective is that of synthesising *liberal* normative systems. Liberality is concerned with not imposing on agents more constraints than necessary to avoid conflicts. BASE aims at pursuing liberality by individually evaluating norms' necessities, and discarding unnecessary norms. However, there may be redundant norms that cannot be detected by means of their individual necessity. Briefly, two norms may replace one another in use, thus becoming one of them unnecessary when the other norm exists. In BASE's empirical results (Section 4.8.2) we saw that it synthesised normative systems containing left-hand side priority norms, and right-hand side priority norms, which together constrain the cars' behaviour to a great extent. This came as a result of BASE's incapability to detect the synergies between these norms to realise that cars should give way either to the left or to the right, but not to both sides.

3. Ill-defined evaluation. As detailed in 4.6.2, BASE evaluates the utility of a norm both in terms of the conflicts it avoids after it has been fulfilled, and the conflicts that arise after it has been infringed. In this sense, we may say that BASE aggregates the effectiveness and the necessity of a norm into a single value representing its overall utility ($\mu_{success}$). This causes a coupling effect between effectiveness and necessity, and makes it impossible to evaluate whether a norm is necessary *independently* of its effectiveness. For instance, a norm may be highly unnecessary, but also highly effective. An example may be a traffic norm like *"remain stopped forever"*, which would be highly effective to avoid collisions, but also highly unnecessary. BASE would never be able to discard such norm, since its high effectiveness would balance its low necessity, leading BASE to evaluate the norm as successful enough.

4. Lack of stability. Norm stability is essential to provide agents with a common framework for their interactions. From the macro analysis of BASE, we observe that it is rather unstable, only being able to synthesise a stable normative system when the number of norm infringements in the system is very low. This means that BASE very frequently changes the *rules of the game* (the norms) to the agents in an agent society. This limitation stems from BASE's approach to norm refinement, which is highly reactive to short-term fluctuations in the utility of norms, leading to continuous deactivations and re-activations of norms that are punctually evaluated as ineffective or unnecessary.

Against these drawbacks, we conclude that there is room for developing alternative strategies that can synthesise more compact and liberal normative systems, while improving BASE's norm evaluation mechanism and overcoming its stability drawbacks. Therefore, the next chapters provide alternative synthesis strategies that extend BASE and outperform it in terms of compactness (Chapters 5 and 6) and liberality (Chapter 7).

Chapter 5

Synthesising compact normative systems: a conservative approach

5.1 Introduction

The previous chapter introduced BASE, a synthesis strategy intended to be executed by a Norm Synthesis Machine (NSM) to perform norm synthesis. BASE was proven to be capable of synthesising effective normative systems that successfully prevent conflicts in a MAS as long as agents comply with their norms. As discussed in Section 4.9, BASE does not consider the *compactness* synthesis objective, which in Section 1.1.3 was identified as essential to reduce agents' norm reasoning efforts. Moreover, BASE suffers from some drawbacks. On the one hand, BASE's norm evaluation approach makes it unable to evaluate whether a norm is necessary independently of if its effectiveness. This may lead to the synthesis of normative systems whose norms are highly effective, but slightly necessary, or the other way around. On the other hand, BASE lacks stability. This comes as a consequence of its norm refinement mechanism, which is highly sensitive to fluctuations in norms' utilities.

Against this background, this chapter introduces IRON (*I*ntelligent *R*obust *O*n-line *N*orm synthesis), a synthesises strategy that extends BASE by considering compactness as a synthesis objective, and by incorporating alternative norm evaluation and norm refinement mechanisms. In this way, this chapter aims at answering research question R2 in Section 1.2. IRON synthesises compact normative systems by performing *norm generalisations*. Briefly, norm generalisations allow to synthesise *general* norms that concisely represent groups of (more specific) norms. For instance, in a road traffic scenario, a norm like *"give way to emergency vehicles"* is a generalisation of (and hence concisely represents) norms to give way to different emergency vehicles (e.g., ambulances, police cars, and

fire-brigades). Thus, norm generalisations allow to reduce the number of norms in the normative system, simplifying agents' norm reasoning process.

IRON approaches norm generalisation from a *conservative* point of view: it requires *full evidence* to generalise. Briefly, IRON will never synthesise a norm like *"give way to emergency vehicles"* until it: (1) has previously synthesised norms to give way to each particular type of emergency vehicle (namely, ambulances, police cars, and fire brigades); (2) has empirically evaluated these norms; and (3) considers that all of them perform well in regulating conflicts. In this way, IRON's norm generalisations do not increase the amount of constraints that a normative system imposes on agents, but they simply represents existing constraints in a compact manner. To generalise norms, IRON is endowed with the computational capabilities to detect *potential norm generalisations*, and to establish generalisation relationships between norms. With this aim, we extend the collection of normative network operators introduced in Chapter 3 by including operators to generalise norms, and to backtrack norm generalisations. Additionally, to overcome BASE's stability limitations, we endow IRON with alternative norm evaluation and norm refinement mechanisms. In order to prove both the performance of IRON in synthesising compact normative systems, and the domain-independence of the computational model introduced in previous chapter, we empirically evaluate IRON in the two application domains introduced in Section 1.3: the road traffic scenario, and the on-line community scenario.

The remainder of this chapter is thus organised as follows. Section 5.2 introduces some preliminary definitions that are essential to understand the concepts of norm generalisation and compactness. Next, Section 5.3 introduces some basic components to perform norm generalisations so that Section 5.4 describes how to generalise norms. Analogously, Section 5.5 details how to backtrack norm generalisations, specialising general norms into more specific norms. Next, Section 5.6 explains how IRON evaluates norms to overcome BASE's evaluation drawbacks. Section 5.7 introduces the IRON strategy, which details how it manages to synthesise compact normative systems for a MAS. Next, Section 5.8 analyses the computational complexity of IRON. Finally, Sections 5.9 and 5.10 show an empirical evaluation of IRON in the two application domains introduced in Section 1.3.

5.2 Preliminary definitions

This section introduces some preliminary definitions to introduce norm generalisation. With this aim, it considers the basic definitions introduced in Section 3.2, and a MAS (cf. Definition 17 in Section 3.2.2) with a set of agents Ag, and a set of actions Ac that these agents can perform. An agent's context can be described as an expression of an agent language $\mathcal{L}_{Ag}$, employed to describe a MAS from the agents' perspective. Norms are of the form $\langle \varphi, \theta(ac) \rangle$, being $\varphi \in \mathcal{L}_{Ag}$ the precondition of the norm, and $\theta(ac)$ its post-condition.

Let us denote the set of terms in language $\mathcal{L}_{Ag}$ by $\mathcal{T}$. We now define a generalisation relationship between the terms in $\mathcal{T}$.

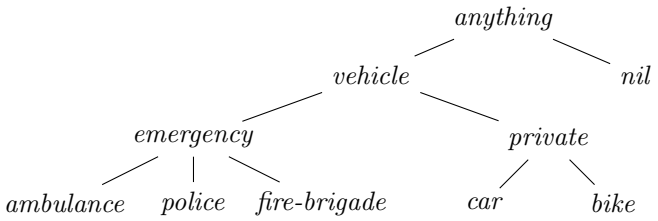

Figure 5.1: Relationships between terms

Definition 20 (Terms generalisation relationship)**.** *Let $\mathcal{T}$ be the set of terms of a language $\mathcal{L}_{Ag}$, $\tau, \tau' \in \mathcal{T}$ two terms, and $\leq$ a relationship that defines a partial order over the elements of $\mathcal{T}$. We denote as $\tau' \leq \tau$ a generalisation relationship between terms τ, τ', where τ is more general than τ'.*

Let us refer to the pair $\mathsf{T} = (\mathcal{T}, \leq)$ as the *taxonomy* over the terms in $\mathcal{T}$. A taxonomy has a (possibly singleton) set of *root terms* $\mathcal{T}^R = \{\tau_1^R, \ldots, \tau_k^R\}$, called *most general terms*, that are not generalised by any other terms. Furthermore, for any given term $\tau \in \mathcal{T}$, there is exactly one (possibly empty) sequence of terms $\tau'_0, \ldots, \tau'_m, \tau'_j \neq \tau, 0 \leq j \leq m$, such that $\tau \leq \tau'_0 \leq \cdots \leq \tau'_m \leq \tau_i^R$ and $\tau_i^R \in \mathcal{T}^R$. Hence, a taxonomy is a *forest* of directed trees rooted at some term $\tau_i^R \in \mathcal{T}^R$, and whose edges capture generalisation relationships.

An example can be illustrated by using again the road traffic example introduced in Section 3.2.1, with travelling cars that can perform actions $Ac = \{go, stop\}$. Each car describes its local context by means of three unary predicates *left*, *front*, *right* that represent the three positions in front of it. However, in this case let us consider a different set of terms that will help illustrate norm generalisation. In particular, each predicate has a single term from $\{$*ambulance*, *police*, *fire-brigade*, *emergency*, *car*, *bike*, *private*, *vehicle*, *nil*, *anything*$\}$ of $\mathcal{L}_{Ag}$, representing different *vehicle* types, term *"nil"* standing for no vehicle, and term *"anything"* representing either any type of vehicle, or nothing. Figure 5.1 illustrates a taxonomy that captures taxonomical knowledge between these terms. On the one hand, ambulances, police cars and fire-brigades can be regarded as different types of *emergency* vehicles. On the other hand, cars and bikes are *private* vehicles, and both *emergency* and *private* vehicles can be generalised as *vehicle*. Finally, a *vehicle* and *nil* can be generalised as *anything*. The ontology is thus rooted at the *"anything"* term (and thus, $\mathcal{T}^R = \{anything\}$). Particularly, term *"emergency"* is more general than *"ambulance"*, term *"vehicle"* is more general than *"emergency"*, and term *"anything"* is more general than *"emergency"*, that is, $ambulance \leq emergency \leq vehicle \leq anything$.

Using this traffic scenario, a norm could be represented to establish a prohibition to go (hence giving way) for any car that observes an ambulance to its left, and a car to its front and right positions, as:

$$n_1 \quad : \quad \langle \{left(ambulance), front(car), right(car)\}, prh(go) \rangle$$

where the norm's precondition has a predicate *left*(*ambulance*) that is true if there is an ambulance to the left of a reference car (i.e., a car evaluating the norm), and *front*(*car*) and *right*(*car*) are predicates that are true when there is a car to the front and right positions of the reference car, respectively.

We say that a norm is *grounded* if all its terms are *grounded*. Particularly, a term is grounded if does not generalise any term, namely it if is a *leaf* in a taxonomy. Formally:

Definition 21 (Grounded term)**.** *Let $\mathcal{T}$ be the set of terms of a language $\mathcal{L}_{Ag}$, and $\tau \in \mathcal{T}$ a term. Term τ is grounded iff there is no term $\tau' \in \mathcal{T}$ such that $\tau' \leq \tau$.*

As an example, n_1 is a grounded norm, since all the terms in its precondition (i.e., *ambulance,car*) are grounded in the taxonomy of Figure 5.1. Next, let us define the subsumption relationship between the terms of a taxonomy.

Definition 22 (Terms subsumption)**.** *For $\tau, \tau' \in \mathcal{T}$, we say that τ subsumes τ', denoted as $\tau' \sqsubseteq \tau$, iff there is a possibly empty sequence of terms $\tau'_0, \ldots, \tau'_m$ such that $\tau' \leq \tau'_0 \leq \cdots \leq \tau'_m \leq \tau$.*

In particular, τ *strictly* subsumes τ', and it is denoted by $\tau' \sqsubset \tau$, iff $\tau' \sqsubseteq \tau$ and $\tau' \neq \tau$. Thus, term *"emergency"* subsumes terms *"emergency"* and *"ambulance"* (that is, *emergency* $\sqsubseteq$ *emergency*, and *ambulance* $\sqsubseteq$ *emergency*), and *strictly* subsumes term *"ambulance"*, that is, *ambulance* $\sqsubset$ *emergency*.

Let us denote by $\bar{\boldsymbol{\tau}}$ a vector of terms of $\mathcal{T}$. We will refer to the i-th component of $\bar{\boldsymbol{\tau}}$ as τ_i. A predicate $p(\bar{\boldsymbol{\tau}}) \in \mathcal{L}_{Ag}$ subsumes another predicate $p(\bar{\boldsymbol{\tau}}') \in \mathcal{L}_{Ag}$ iff, for each pair of terms in $\bar{\boldsymbol{\tau}}, \bar{\boldsymbol{\tau}}'$, term τ_i subsumes term τ'_i. Formally:

Definition 23 (Predicates subsumption)**.** *For $p(\bar{\boldsymbol{\tau}}), p(\bar{\boldsymbol{\tau}}') \in \mathcal{L}_{Ag}$, we say that $p(\bar{\boldsymbol{\tau}})$ subsumes $p(\bar{\boldsymbol{\tau}}')$, denoted as $p(\bar{\boldsymbol{\tau}}') \sqsubseteq p(\bar{\boldsymbol{\tau}})$, iff $\tau'_i \sqsubseteq \tau_i$ for all $1 \leq i \leq m$.*

Given a term $\tau \in \mathcal{T}$, its *potential generalisation* is the most specific term that *strictly* subsumes it. For instance in Figure 5.1, the potential generalisation of term *"ambulance*" is term *"emergency"*, since there is no other term which is more specific and strictly subsumes it. However, the potential generalisation for term *"anything"* does not exist because there is no term strictly subsuming *"anything"*. Formally:

Definition 24 (Potential term generalisation)**.** *For $\tau \in \mathcal{T}$, its potential generalisation is a term $\tau_g \in \mathcal{T}$ such that $\tau \sqsubset \tau_g$ and there is no other term $\tau'' \in \mathcal{T}$ such that $\tau \sqsubset \tau''$ and $\tau'' \sqsubset \tau_g$.*

As an example, consider norms $n_2, \ldots, n_3$ below, which work in conjunction with norm n_1 to regulate priority of emergency vehicles:

$$\begin{array}{lcl} n_2 & : & \langle \{\mathit{left}(\mathit{police}), \quad \mathit{front}(\mathit{car}), \ \mathit{right}(\mathit{car})\}, \quad \mathit{prh}(\mathit{go})\rangle \\ n_3 & : & \langle \{\mathit{left}(\mathit{fire\text{-}brigade}), \ \mathit{front}(\mathit{car}), \ \mathit{right}(\mathit{car})\}, \quad \mathit{prh}(\mathit{go})\rangle \end{array}$$

Let us suppose that n_1, n_2, n_3 comprise a normative system $\Omega = \{n_1, n_2, n_3\}$. Notice that the three norms only differ in their *"left"* predicate, which describes

different *emergency* vehicles. That is, the potential generalisation of terms *ambulance*, *police* and *fire-brigade* is term *emergency*. Therefore, these three norms may be *generalised* into a single, more general norm to give way to *emergency* vehicles coming from the left:

$$n_4 \quad : \quad \langle \{ \mathit{left}(\mathit{emergency}),\ \ \mathit{front}(\mathit{car}),\ \mathit{right}(\mathit{car}) \},\quad \mathit{prh}(\mathit{go}) \rangle$$

yielding a new normative system $\Omega' = \{n_4\}$, containing one single norm. Notice that n_4 generalises the *left* predicate of norms n_1, n_2, n_3 as *left(emergency)*. This generalisation is possible because n_4 caters for the same situations (preconditions) of $\{n_1, n_2, n_3\}$, and establishes the same constraints. Thus, even though norms n_1, n_2, n_3 are no longer explicitly included in the normative system, they are implicitly represented by n_4. Therefore, whenever norms n_1, n_2 or n_3 are applicable, their parent (n_4) will be applicable as well.

Next, a definition of the generalisation relationship between norms is provided:

Definition 25 (Norm generalisation relationship)**.** *A norm $n = \langle \varphi, \theta(ac) \rangle$ is more general than another norm $n' = \langle \varphi', \theta(ac) \rangle$, denoted as $n' \subseteq n$, iff $n \neq n'$, $|\varphi| = |\varphi'|$, and for each predicate $p(\bar{\boldsymbol{\tau}}') \in \varphi'$, there is a predicate $p(\bar{\boldsymbol{\tau}}) \in \varphi$ such that $p(\bar{\boldsymbol{\tau}}') \sqsubseteq p(\bar{\boldsymbol{\tau}})$.*

Note that the applicability condition of n_4 is more general than that of n_1 because a car is prohibited to go if it finds *any* type of *emergency* vehicle (including ambulances) to its left position, and a car to its front and right positions. Specifically, predicate $\mathit{left}(\mathit{ambulance}) \in \varphi_1$ (φ_1 being the precondition of n_1) has a corresponding predicate $\mathit{left}(\mathit{emergency}) \in \varphi_4, \mathit{ambulance} \sqsubseteq \mathit{emergency}$ (φ_4 being the precondition of n_4). Similarly, $\mathit{front}(\mathit{car}) \in \varphi_1$ has $\mathit{front}(\mathit{car}) \in \varphi_4, \mathit{front}(\mathit{car}) \sqsubseteq \mathit{front}(\mathit{car})$, and $\mathit{right}(\mathit{car}) \in \varphi_1$ has $\mathit{right}(\mathit{car}) \in \varphi_4, \mathit{right}(\mathit{car}) \sqsubseteq \mathit{right}(\mathit{car})$. Norm n_4 is thus more general than n_1, namely $n_1 \subseteq n_4$. In general, if a norm n' is *generalised* by a norm n, then we also say that n is *specialised* by n'. If there exists at least some $n'' \in \mathcal{N}$ such that $n' \subset n'' \subset n$, then we say that n is an *ancestor* of n'. Otherwise, n is a *parent* of n'. If n' is not *generalised* by n, we denote it by $n' \not\subseteq n$.

Notice that norm generalisations allow to yield *smaller* normative systems, namely normative systems that include less norms (and hence, less norm predicates and terms). In other words, norm generalisations increase the *compactness* of the normative system. In the example above, the generalisation of norms n_1, n_2, n_3 as norm n_4 leads from a normative system with 3 norms and 9 terms in total (Ω) to another one with 1 norm and 3 terms in total (Ω').

To conclude this section, let us introduce a measure of compactness of a normative system:

Definition 26 (Compactness)**.** *The compactness of a normative system Ω is the total number of terms in the preconditions of its norms, namely $\sum_{n \in \Omega} |n|$, where $|n|$ stands for the number of terms in a norm's precondition.*

5.3 Components for norm generalisation

To perform norm generalisations, IRON requires a means to represent generalisation relationships between norms. With this aim, it is necessary to extend the normative network and the operators for normative networks considered by a NSM to perform norm synthesis (see Section 3.4). Subsequent sections describe IRON's normative network and operators.

5.3.1 A network to represent generalisation relationships

To keep track of synthesised norms, IRON employs a normative network like the one described in Definition 19 (Section 3.4.1). Thus, formally an IRON's normative network is a tuple $\langle \mathcal{N}, \mathcal{R}, \Delta, \delta \rangle$, being $(\mathcal{N}, \mathcal{R})$ a graph such that: $\mathcal{N}$ is a set of norms , and $\mathcal{R} \subseteq \mathcal{N} \times \mathcal{N}$ a set of directed edges representing relationships between norms; $\Delta = \{active, inactive\}$ a set of possible norm states; and δ a function that retrieves a norm's state in the normative network. Particularly, to represent norm generalisations IRON considers that $\mathcal{R} = \{\mathcal{E}_G\}$, being $\mathcal{E}_G \subseteq \mathcal{N} \times \mathcal{N}$ a set of directed edges representing generalisation relationships between norms. Additionally, IRON represents the normative system at a given time as the norms whose state is *"active"* in the normative network at that time.

Figure 5.2 illustrates how IRON transforms a normative network to perform a norm generalisation. Initially, the normative network NN contains three active norms n_1, n_2, n_3 described in Section 5.2 (represented as white circles), hence representing normative system $\Omega = \{n_1, n_2, n_3\}$. As described in Section 5.2, these three norms can be concisely represented by norm n_4, which is more general than $\{n_1, n_2, n_3\}$. Then, n_1, n_2, n_3 are generalised as n_4, and deactivated (represented as gray circles), yielding $NN' = \{n_1, n_2, n_3, n_4\}$ and $\Omega' = \{n_4\}$. Note then that the number of norms of the normative system is reduced from 3 norms to 1, increasing its compactness. Nevertheless, it is worth noticing that norm generalisations do not decrease the number of grounded norms a normative system represents. Therefore, they do not reduce the amount of constraints imposed on agents. In Figure 5.2, normative system Ω' contains a single norm, but it represents 3 grounded norms (give way to ambulances, give way to police cars, and give way to fire brigades).

5.3.2 Operators for norm generalisation

To manage the normative network, IRON considers the basic collection of operators described in Table 3.1 in Section 3.4.2, which can be employed to add, activate, and deactivate norms. Additionally, it considers two new operators to perform the following operations:

1. The *generalisation* of a set of norms in the normative system into a single, more general norm. An example is depicted in Figure 5.2, where norms n_1, n_2, n_3 are generalised into norm n_4. As Table 5.1 shows, the `generalise` operator generalises a set of norms (*children*) into a more

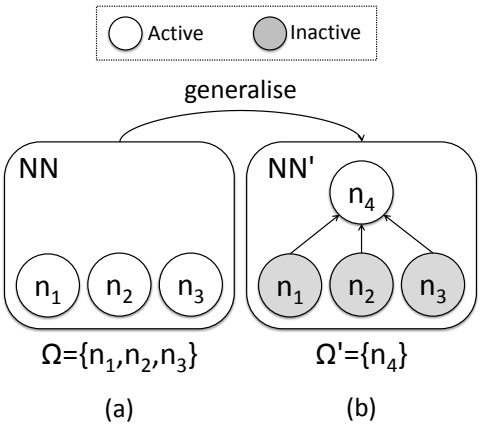

Figure 5.2: Generalisation of norms n_1, n_2, n_3 to n_4.

general norm (*parent*) by performing the following operations. First, if the parent norm does not exist in the normative network, its adds the parent norm to the network and establishes new generalisation relationships between each child norm and its parent. Thereafter, it sets the state of each child to *"inactive"*, and the parent's state to *"active"*. As a result, the child norms will no longer belong to the normative system, while the parent norm will. Note though that the child norms (norms n_1, n_2, n_3) are represented by the parent norm (n_4), and thus they are implicitly represented in the normative system, even though they are not explicitly included.

2. The *specialisation* of a norm into more specific norms. As Table 5.1 shows, this `specialise` operator undoes the result of a generalisation by setting the state of the parent norm to *"inactive"*, and setting its children's to *"active"*. In this way, all the child norms become candidates to belong to the normative system, while the parent norm does not any longer. Figure 5.3 illustrates the specialisation of n_4 into its children (n_1, n_2, n_3) by first deactivating n_4, and finally activating n_1, n_2, n_3.

Note that specialising a parent norm does not imply removing the generalisation relationships it has with its children. The reason of this is that generalisation is a *syntactical* relationship between norms that never changes, independently of the states of the norms. In the example, although norm n_4 is specialised, it will always be more general than its children. Therefore, there is no need to remove this relationship.

5.4 Conservative norm generalisation

As detailed in Section 5.1, IRON takes a conservative approach to norm generalisation that requires *full evidence* to generalise norms. Briefly, IRON will not

Operator	Specification
$generalise(children, parent, NN)$	**if** $parent \notin \mathcal{N}$ $\mathcal{N}' \leftarrow \mathcal{N} \cup \{parent\}$ **for all** $child \in children$ $\mathcal{E}'_G \leftarrow \mathcal{E}_G \cup (child, parent)$ **for all** $child \in children$ $\delta'(child) \leftarrow inactive$ $\delta'(parent) \leftarrow active$ $NN' \leftarrow \langle \mathcal{N}', \mathcal{R}', \Delta, \delta' \rangle$
$specialise(parent, children, NN)$	$\delta'(parent) \leftarrow inactive$ **for all** $child \in children$ **if** $(child, parent) \in \mathcal{E}_G$ $\delta'(child) \leftarrow active$ $NN' \leftarrow \langle \mathcal{N}, \mathcal{R}, \Delta, \delta' \rangle$

Table 5.1: IRON's operators to generalise and specialise norms.

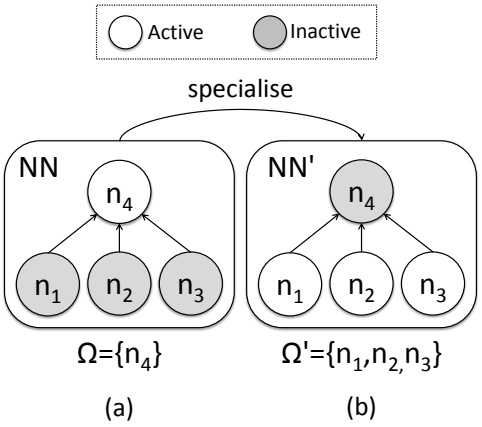

Figure 5.3: Specialisation of norm n_4 to n_1, n_2.

synthesise a general norm until it has first synthesised *each one of* the norms it represents, it has empirically evaluated them, and it considers that they perform well in regulating conflicts. As an example, consider norm n_4 described in Section 5.2, which generalises three norms n_1, n_2, n_3 to give way to different types of emergency vehicles coming from the left (i.e., ambulances, police cars, and fire brigades). IRON would be capable of synthesising a norm like n_4 *only if* it has previously synthesised norms n_1, n_2, n_3, and considers that they perform well in regulating conflicts. If, for any reason, norm IRON never synthesised one of the three norms, (e.g., norm n_3), or if it synthesised the norm but it performed poorly in avoiding conflicts, then IRON would never generalise n_1, n_2, n_3 into n_4.

IRON's generalisation is based on the creation of *potential generalisations*. Briefly, once IRON creates a norm, it creates all the potential generalisations of the norm (namely, those norms that it may be potentially generalised to), to subsequently analyse whether each one of them can be *enacted* or not. During this process it checks, for each potential generalisation, if the norms that are necessary to generalise to a potential parent have been synthesised, and perform well. When this happens, it performs the generalisation by invoking the `generalise` operator described in Section 5.3.2. Subsequent sections detail how to build potential generalisations and how to enact potential generalisations.

5.4.1 Building potential generalisations

Given a norm, the first step to build its potential generalisations is to find those general (parent) norms that it may be potentially generalised to. Specifically, IRON creates a potential generalisation for each potential term generalisation in the norm's precondition. With this aim, it employs a terms taxonomy (T) (cf. Section 5.2) to find the potential generalisation of each term. As an example, consider norm n_1 described in Section 5.2. Its precondition contains predicates *left(ambulance)*, *front(car)*, and *right(car)*. According to the taxonomy in Figure 5.1, the potential generalisations of these predicates are, respectively, *left(emergency)*, *front(private)*, and *right(private)*. Therefore, n_1 has three potential generalisations:

$$
\begin{array}{lcl}
n_4 & : & \langle\{\mathit{left}(\mathit{emergency}), \ \mathit{front}(\mathit{car}), \ \ \mathit{right}(\mathit{car})\}, \ \ \mathit{prh}(\mathit{go})\rangle \\
n_5 & : & \langle\{\mathit{left}(\mathit{ambulance}), \ \mathit{front}(\mathit{private}), \ \mathit{right}(\mathit{car})\}, \ \ \mathit{prh}(\mathit{go})\rangle \\
n_6 & : & \langle\{\mathit{left}(\mathit{ambulance}), \ \mathit{front}(\mathit{car}), \ \ \mathit{right}(\mathit{private})\}, \ \mathit{prh}(\mathit{go})\rangle
\end{array}
$$

where each potential parent norm generalises one of the terms of n_1's precondition. Specifically, norm n_4 generalises term *"ambulance"* of its *left* predicate, while n_5 and n_6 generalise terms *"car"* of its *front* and *right* predicates, respectively.

The second step is to find all the child norms that each potential parent represents. With this aim, IRON employs again the terms taxonomy to retrieve the terms that each potential generalisation represents. As an example, term *"emergency"* of norm n_4's *left* predicate represents terms *"ambulance", "police", "fire-brigade"*. Therefore, the norms that are necessary to generalise to n_4 are

n_1, n_2, n_3. Finally, for each potential generalisation, IRON generates and records a triple $\langle n, n', S\rangle$, where n is a generalisable norm, n' is a potential generalisation of n, and S is a (possibly empty) set of norms that n' generalises (disregarding n). In our example, the resulting potential generalisation is $\langle n_1, n_4, \{n_2, n_3\}\rangle$, containing a generalisable norm (n_1), its potential generalisation (n_4), and the siblings that are necessary to generalise (n_2, n_3).

Algorithm 7 illustrates IRON's *createPotentialGeneralisations* function, which it employs to create the potential generalisations of a norm n. It inputs a norm n, and outputs a set of n's potential generalisations. First, it takes the precondition φ of a norm n (line 1) and employs function *getPotentialGeneralisations* (line 3) to obtain its parent preconditions (P_φ) by using a taxonomy T. Thereafter, in lines 4–5, for each parent precondition $\varphi' \in \Phi$, it builds n' (a potential parent) based on the general precondition φ' and the same consequent $\theta(ac)$ from n (that is, $n' = \langle\varphi', \theta(ac)\rangle$). Next, function *generateChildren* (line 6) computes (using again taxonomy T) the child norms that are necessary to generalise into norm n', disregarding n. In other words, it generates the siblings of n. Finally, it builds a new potential generalisation $\langle n, n', S\rangle$, which is finally added to the potential generalisations of norm n, namely $potential_n$ (line 7).

ALGORITHM 6: IRON's *createPotentialGeneralisations* function

Input : n
Output: $potential_n$
1 **let** $n = \langle\varphi, \theta(ac)\rangle$;
2 $potential_n \leftarrow \emptyset$;
3 $\mathrm{P}_\varphi \leftarrow$ `getPotentialGeneralisations`(φ, T);
4 **for** $\varphi' \in \mathrm{P}_\varphi$ **do**
5 $\quad n' \leftarrow \langle\varphi', \theta(ac)\rangle$;
6 $\quad S \leftarrow$ `generateNecessaryChildren`(n', T, n);
7 $\quad potential_n \leftarrow potential_n \cup \{\langle n, n', S\rangle\}$;
8 **return** $potential_n$

5.4.2 Enacting potential generalisations

A potential generalisation $\langle n, n', S\rangle$ can be enacted if, and only if, both n and its siblings (the norms in S) satisfy the following conditions:

1. They belong to the normative network. That is, IRON has previously created them, and has added them to the normative network.
2. They are active in the normative network (and thus, they belong to the normative system).
3. They *perform well* in regulating conflicts. Particularly, the way in which IRON considers that a norm performs well is detailed in Section 5.6.1.

If the conditions above conditions hold, a potential generalisation is enacted to transform both the normative network and the normative system. With this

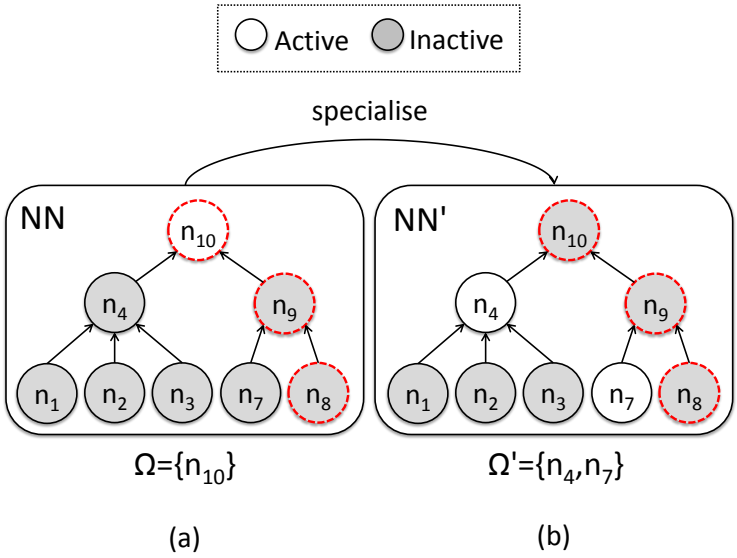

Figure 5.4: Specialisation of norm n_{10} into norms n_4 and n_7.

aim, IRON invokes operator `generalise` (cf. Table 5.1) to generalise norm n and its siblings (the norms in S) into norm n'. As an example, consider the potential generalisation $\langle n_1, n_4, \{n_2, n_3\}\rangle$, and the normative network $NN = \{n_1, n_2, n_3\}$ in Figure 5.2, which contains n_1, n_2, n_3 as active norms, and represents normative system $\Omega = \{n_1, n_2, n_3\}$. Considering that norms n_1, n_2, n_3 perform well in regulating conflicts, the aforementioned potential generalisation can be enacted by invoking operator $generalise(\{n_1, n_2, n_3\}, n_4)$, hence generalising norms n_1, n_2, n_3 into n_4, and yielding normative system $\Omega' = \{n_4\}$.

By generalising norms, IRON can provide the agents with normative systems containing a reduced number of norms and norm predicates. In this way, it can reduce the agents' computational efforts when reasoning about norms. However, note that even though agents are provided with general norms, IRON keeps track of both general norms and the (more specific) norms they represent in the normative network. As an example, although agents are provided with norm n_4, IRON keeps track in the normative network of both n_4 and the norms it represents (n_1, n_2, n_3).

5.5 Backtracking norm generalisations

IRON performs *norm specialisations* as a dual operation to generalisation, allowing to *backtrack* generalisations of norms that *under-perform* in regulating conflicts (see Section 5.6). The idea is that a norm that under-performs must not be included in the normative system. Initially, one may think that it should

be enough with deactivating the norm in the normative network, thus ensuring that it does not belong to the normative system. However, recall that an inactive norm can be implicitly represented in the normative system by a more general norm. Therefore, whenever IRON detects that a norm under-performs, it deactivates the norm along with all those norms that generalise it. In other words, it specialises its *ancestors* in the normative network. In this way, the norm will no longer be neither explicitly included nor implicitly represented in the normative system. Let us illustrate an example with the normative network (NN) depicted in Figure 5.4a. It contains a single active norm n_{10}, and thus represents normative system $\Omega = \{n_{10}\}$. The network contains norms n_1, n_2, n_3 and n_4 (defined in Section 5.2), together with:

$$\begin{aligned} n_7 &: \langle\{\mathit{left}(\mathit{car}), \quad \mathit{front}(\mathit{car}), \mathit{right}(\mathit{car})\},\ \mathit{prh}(\mathit{go})\rangle \\ n_8 &: \langle\{\mathit{left}(\mathit{bike}), \quad \mathit{front}(\mathit{car}), \mathit{right}(\mathit{car})\},\ \mathit{prh}(\mathit{go})\rangle \\ n_9 &: \langle\{\mathit{left}(\mathit{private}), \mathit{front}(\mathit{car}), \mathit{right}(\mathit{car})\},\ \mathit{prh}(\mathit{go})\rangle \\ n_{10} &: \langle\{\mathit{left}(\mathit{vehicle}), \mathit{front}(\mathit{car}), \mathit{right}(\mathit{car})\},\ \mathit{prh}(\mathit{go})\rangle \end{aligned}$$

Norms n_7 and n_8 prohibit a car from proceeding whenever there is a car or a bike to its left, and car in front and to its right. Norm n_9 is a generalisation of norms n_7 and n_8 to give way to private vehicles coming from the left, whereas norm n_{10} is a generalisation of all norms from n_1 to n_9 to give way to any vehicle coming from the left. Let us now consider that giving way to bicycles is not convenient to avoid conflicts (i.e., collisions), and thus n_8 under-performs (depicted as a dashed red circle). Even though norm n_8 is inactive, it is implicitly represented by norm n_{10}. Therefore, IRON proceeds by recursively invoking operator `specialise` (cf. Table 5.1 in Section 5.3.2) to perform the the following steps:

1. Specialisation of norm n_{10}. This operation involves deactivating norm n_{10} and activating its children (n_4, n_9). As a result, norms n_4, n_9, which were implicitly represented by n_{10}, will now be explicitly included in the normative system.

2. Specialisation of norm n_9. It deactivates norm n_9, and activates norms n_7, n_8 to explicitly include them in the normative system.

3. Deactivation of norm n_8.

The resulting normative network (NN') is depicted in 5.4b. It contains two active norms n_4, n_7, hence representing normative system $\Omega' = \{n_4, n_7\}$. Note that norm n_8 is no longer represented in the normative system neither by itself nor by a general norm. Moreover, IRON has preserved the generalisation of norms n_1, n_2, n_3 as norm n_4, keeping the normative system as compact as possible.

5.6 Evaluating norms' performances

IRON incorporates a novel norm evaluation mechanism to overcome BASE's stability drawbacks (described in Section 3.5). In short, IRON's norm evaluation

mechanism is an improvement of the one described in 4.4. Briefly, IRON performs two main improvements:

1. It *decouples effectiveness and necessity*, allowing to evaluate the necessity of a norm independently of its effectiveness. In other words, IRON evaluates norms in terms of the norm evaluation criteria of effectiveness and necessity (see Section 3.2.2). Formally, it considers a set of evaluation criteria $\mathcal{EC} = \{\langle \mu_{\mathit{eff}}, \alpha_{\mathit{eff}} \rangle, \langle \mu_{nec}, \alpha_{nec} \rangle\}$, being $\mu_{\mathit{eff}}, \mu_{nec}$ two functions to compute a norm's effectiveness and necessity, respectively, and $\alpha_{\mathit{eff}}, \alpha_{nec}$ two thresholds that set the minimum effectiveness and necessity of a norm to be included in the normative system, respectively.

2. It *computes additional metrics* to make more informed decisions when refining the normative system. Specifically, after evaluating a norm, IRON computes its effectiveness and necessity *performance ranges*, which represent the ranges in which the effectiveness and the necessity of the norm move along time. In this way, it smooths out short-term fluctuations of effectiveness and necessity, and highlights longer-term trends when detecting under-performance.

Briefly, IRON computes a norm's effectiveness by iteratively aggregating its effectiveness rewards along time. In particular, it computes a effectiveness reward of a norm n that has been fulfilled at a given time t as its ratio of *successful fulfilments* at time t (see Section 3.3.3). With this aim, it employs the equation below.

$$r_{\mathit{eff}}(n,t) = \begin{cases} \dfrac{sf_t^n}{sf_t^n + hf_t^n} & \text{if } \mathit{fulfilled}(n,t) = \mathit{true} \\ \\ \bot & \text{otherwise} \end{cases} \tag{5.1}$$

where $n \in \mathcal{N}$, $t \in \mathbb{N}$ (being $\mathcal{N}$ the set of norms in the normative network), and $sf_t^n \in \mathcal{SF}^n$ adn $hf_t^n \in \mathcal{HF}^n$ are the number of successful fulfilments and harmful fulfilments of n at time t, respectively (see Section 3.3.3). Specifically, function r_{eff} computes a effectiveness reward as the ratio of successful fulfilments of n at time t iff n has been fulfilled at time t, namely if *fulfilled(n,t) = true*, and returns an *undefined* reward value $\bot$ if n has not been fulfilled at time t. Particularly, function *fulfilled* returns *true* if there are either successful fulfilments or unsuccessful fulfilments of n at time t. Formally:

$$\mathit{fulfilled}(n,t) = \begin{cases} \mathit{true} & \text{if } sf(n,t) + hf(n,t) > 0 \\ \mathit{false} & \text{otherwise} \end{cases} \tag{5.2}$$

Thereafter, IRON computes the effectiveness of norm n at time t as:

$$\mu_{\mathit{eff}}(n,t) = \begin{cases} (1 - \gamma_{eff}) \times \mu_{\mathit{eff}}(n,t') + \gamma_{\mathit{eff}} \times r_{\mathit{eff}}(n,t) & \text{if } r_{\mathit{eff}}(n,t) \neq \bot \\ \bot & \text{otherwise} \end{cases} \tag{5.3}$$

where $\mu_{\mathit{eff}}(n,t') \neq \perp$; $0 \leq \gamma_{\mathit{eff}} \leq 1$ is a *learning rate*; $t' < t$; and there is no t'' such that $\mu_{\mathit{eff}}(n,t'') \neq \perp$ and $t' < t'' < t$. In particular, at an initial time t_0 the effectiveness of a norm is set to an initial constant value $k_{\mathit{eff}} \in [0,1]$. Formally, $\mu_{\mathit{eff}}(n,t_0) = k_{\mathit{eff}}$

Notice therefore that IRON's norm evaluation approach is akin to reinforcement learning [Sutton and Barto, 1998], since the effectiveness of a norm (μ_{eff}) is somehow *learned* by iteratively aggregating rewards computed from the outcomes of agents' norm compliance. In this way, a norm's effectiveness in IRON evolves more smoothly than in the case of BASE, avoiding high fluctuations in its values.

Analogously, IRON computes a norm's necessity by iteratively aggregating its *necessity rewards* along time. It computes the necessity reward of a norm n that has been infringed at a given time t as its ratio of *harmful infringements* at time t (see Section 3.3.3). For this purpose, it employs the following equation.

$$r_{nec}(n,t) = \begin{cases} \dfrac{hi_t^n}{hi_t^n + si_t^n} & \text{if } \mathit{infringed}(n,t) = \mathit{true} \\ \\ \perp & \text{otherwise} \end{cases} \tag{5.4}$$

where $n \in \mathcal{N}$, $t \in \mathbb{N}$, and $hi_t^n \in \mathcal{HI}^n$ and $si_t^n \in \mathcal{SI}^n$ are the number of harmful infringements and successful infringements of n at time t, respectively (see Section 3.3.3). More specifically, function r_{nec} computes a necessity reward as the ratio of harmful infringements of n at time t iff n has been infringed at time t, and returns an *undefined* reward value $\perp$ if n has not been infringed at time t. Particularly, function *infringed* returns *true* if there are either harmful infringements or successful infringements of n at time t. Formally:

$$\mathit{infringed}(n,t) = \begin{cases} \mathit{true} & \text{if } hi(n,t) + si(n,t) > 0 \\ \mathit{false} & \text{otherwise} \end{cases} \tag{5.5}$$

Thereafter, IRON computes the necessity of norm n at time t as:

$$\mu_{nec}(n,t) = \begin{cases} (1-\gamma_{nec}) \times \mu_{nec}(n,t') + \gamma_{nec} \times r_{nec}(n,t) & \text{if } r_{nec}(n,t) \neq \perp \\ \perp & \text{otherwise} \end{cases} \tag{5.6}$$

where $\mu_{nec}(n,t') \neq \perp$; $0 \leq \gamma_{nec} \leq 1$ is a learning rate; $t' < t$; and there is no t'' such that $\mu_{\mathit{eff}}(n,t'') \neq \perp$ and $t' < t'' < t$. Likewise in the case of effectiveness, at an initial time t_0 the necessity of a norm is set to an initial constant value $k_{nec} \in [0,1]$. Formally, $\mu_{nec}(n,t_0) = k_{nec}$.

As previously introduced, IRON computes additional metrics to make more informed decisions when refining the normative system. With this aim, once IRON has computed new effectiveness/necessity values for a norm n ($\mu_{\mathit{eff}}/\mu_{nec}$), it computes the effectiveness/necessity *performance ranges* of n to smooth out short-term fluctuations and to highlight long-term trends of its effectiveness/necessity. These ranges are essential to determine whether a norm performs well, or it under-performs, in order to perform generalisations and specialisations.

Basically, a performance range consists in the *Bollinger Bands* [Bollinger, 2001] of a q number of effectiveness/necessity values ($\mu_{\mathit{eff}}/\mu_{nec}$), and is composed of: (i) a q-period moving average; (ii) an *upper band* at a q-period standard deviation *above* the moving average; and (iii) a *lower band* at a q-period standard deviation *below* the moving average. IRON computes the effectiveness range of a norm n at a given time t by considering the latest q effectiveness values (μ_{eff}) of n that are not undefined at time t. First, it computes the q-period moving average of the effectiveness of n at time t. With this aim, let us define a set Eff_l^n of defined effectiveness values of n from a given time l:

$$\mathit{Eff}_l^n = \{\mu_{\mathit{eff}}(n,j) \,|\, l \leq j \leq m, \mu_{\mathit{eff}}(n,j) \neq \perp\} \tag{5.7}$$

Then, we consider an index $i(q)$ such that $|\mathit{Eff}_{i(q)}^n| = q$ and $1 \leq i(q) \leq m$. Thus, set $\mathit{Eff}_{i(q)}^n$ contains the latest q defined effectiveness values of n. IRON computes the average effectiveness of norm n as the average of the values in $\mathit{Eff}_{i(q)}^n$.

$$\hat{\mu}_{\mathit{eff}}^n = \frac{\sum\limits_{\mathit{eff} \in \mathit{Eff}_{i(q)}^n} \mathit{eff}}{q} \tag{5.8}$$

where $n \in \mathcal{N}$, $q \in \mathbb{N}$, and $1 \leq q \leq m$.

Next, IRON computes the standard deviation of the last q effectiveness values in set $\mathit{Eff}_{i(q)}^n$ to be able to compute the higher bound and the lower bound of n's effectiveness range. It computes a standard deviation as follows.

$$\sigma_{\mathit{eff}}^n = \sqrt{\frac{1}{q} \sum_{\mathit{eff} \in \mathit{Eff}_{i(q)}^n} (\mathit{eff} - \hat{\mu}_{\mathit{eff}}^n)^2} \tag{5.9}$$

Finally, the effectiveness range of norm n is a tuple $\mathcal{ER}^n = \langle lb_{\mathit{eff}}^n, ub_{\mathit{eff}}^n \rangle$, being lb_{eff}^n the effectiveness lower bound of the range and lb_{eff}^n is the effectiveness upper bound of the range, which are computed as follows:

$$lb_{\mathit{eff}}^n = \hat{\mu}_{\mathit{eff}}^n - \sigma_{\mathit{eff}}^n \tag{5.10}$$

$$ub_{\mathit{eff}}^n = \hat{\mu}_{\mathit{eff}}^n + \sigma_{\mathit{eff}}^n \tag{5.11}$$

Analogously, IRON computes the necessity range of a norm n at a given time t by considering the latest q necessity values (μ_{nec}) of n that are not undefined at time t. It first computes the q-period moving average of the necessity of n at time t. With this aim, let us define a set Nec_l^n of defined necessity values of n from a given time l:

$$\mathit{Nec}_l^n = \{\mu_{nec}(n,j) \,|\, l \leq j \leq m, \mu_{nec}(n,j) \neq \perp\} \tag{5.12}$$

Then, we consider an index $i(q)$ such that $|\mathit{Nec}_{i(q)}^n| = q$ and $1 \leq i(q) \leq m$. Thus, set $\mathit{Nec}_{i(q)}^n$ contains the latest q defined necessity values of n. IRON computes the average necessity of norm n as the average of the values in $\mathit{Nec}_{i(q)}^n$.

$$\hat{\mu}_{nec}^n = \frac{\sum\limits_{nec \in \mathit{Nec}_{i(q)}^n} nec}{q} \tag{5.13}$$

where $n \in \mathcal{N}$, $q \in \mathbb{N}$, and $1 \leq q \leq m$.

Next, IRON computes the standard deviation of the last q necessity values in set $Nec^n_{i(q)}$ in order to compute the higher bound and the lower bound of n's necessity range. It computes this standard deviation as follows.

$$\sigma^n_{nec} = \sqrt{\frac{1}{q} \sum_{nec \in Nec^n_{i(q)}} (nec - \hat{\mu}^n_{nec})^2} \tag{5.14}$$

Finally, the necessity range of norm n is a tuple $\mathcal{NR}^n = \langle lb^n_{nec}, ub^n_{nec} \rangle$, being lb^n_{eff} the necessity lower bound of the range and lb^n_{eff} is the necessity upper bound of the range, which are computed as follows:

$$lb^n_{nec} = \hat{\mu}^n_{nec} - \sigma^n_{nec} \tag{5.15}$$

$$ub^n_{nec} = \hat{\mu}^n_{nec} + \sigma^n_{nec} \tag{5.16}$$

5.6.1 Detecting good performance

As described in Section 5.4, IRON generalises norms only whenever it considers they *perform well* in regulating conflicts. This section details how IRON manages to check that condition. Briefly, IRON considers that a norm n performs well at a given time t iff the lower bounds of its effectiveness *and* necessity ranges for the last q values up to time t are *above* some satisfaction (generalisation) thresholds. This amounts to satisfying the following generalisation conditions:

$$lb^n_{eff} > \alpha^{gen}_{eff} \tag{5.17}$$

$$lb^n_{nec} > \alpha^{gen}_{nec} \tag{5.18}$$

where lb^n_{eff}, lb^n_{nec} are the lower bounds of the n's effectiveness range ($\mathcal{ER}^n$) and n's necessity range ($\mathcal{N}^n$), respectively; and $\alpha^{gen}_{eff} \in [0, 1]$ and $\alpha^{gen}_{nec} \in [0, 1]$ are thresholds that set the minimum effectiveness/necessity of a norm to be considered to perform well enough to be generalised. Note that these conditions are highly conservative, since generalising a norm requires that both its effectiveness and its necessity range are above a some thresholds.

Let us illustrate this with an example. Consider now the normative network NN in Figure 5.2, the effectiveness and necessity ranges in Table 5.2, and $\alpha^{gen}_{eff} = 0.5, \alpha^{gen}_{nec} = 0.4$ as generalisation thresholds. Initially, the normative network contains three active norms n_1, n_2, n_3. From Table 5.2, we can observe that these three norms fulfil the generalisation conditions in equations 5.17 and 5.18, namely they perform well in terms of effectiveness and necessity. Thus, the three norms can be considered to be generalised as detailed in Section 5.4.2, enacting their corresponding potential generalisation.

5.6.2 Detecting under performance

IRON takes a conservative approach when assessing if a norm under-performs. In short, it requires to cumulate greats amounts of negative evidences about a

	$\mathcal{ER}$	$\mathcal{NR}$
n_1	[0.6, 0.7]	[0.5, 0.6]
n_2	[0.7, 0.7]	[0.6, 0.7]
n_3	[0.7, 0.8]	[0.7, 0.8]
n_4	[0.8, 0.9]	[0.4, 0.5]
n_7	[0.6, 0.7]	[0.6, 0.7]
n_8	[0.0, 0.1]	[0.7, 0.8]
n_9	[0.4, 0.4]	[0.6, 0.7]
n_{10}	[0.5, 0.6]	[0.5, 0.6]

Table 5.2: Effectiveness and necessity ranges for the norms in Figures 5.2 and 5.4. Each range contains two values representing the lower bound and the upper bound of the range, respectively.

norm's performance to ensure that it is either ineffective or unnecessary. This implies that, once IRON has activated a norm, it is resistant to deactivating it. Briefly, IRON considers that a norm n under-performs at time t if the *higher bounds* of its effectiveness *or* necessity performance ranges of the last q values up to time t are *below* some satisfaction thresholds. This amounts to satisfying either (or both) of the two following conditions.

$$ub^n_{\mathit{eff}} < \alpha_{\mathit{eff}} \tag{5.19}$$

$$ub^n_{nec} < \alpha_{nec} \tag{5.20}$$

where $ub^n_{\mathit{eff}}, ub^n_{nec}$ are the higher bounds of n's effectiveness range ($\mathcal{ER}^n$) and n's necessity range ($\mathcal{NR}^n$), respectively; and $\alpha_{\mathit{eff}} \in [0,1]$ and $\alpha_{nec} \in [0,1]$ are the two thresholds of IRON's evaluation criteria ($\mathcal{EC}$, described at the beginning of Section 5.6), which set the minimum effectiveness/necessity of a norm to be included in the normative system. Let us illustrate an example with normative network NN' in Figure 5.4b, the effectiveness and necessity ranges in Table 5.2, and $\alpha_{\mathit{eff}} = 0.3, \alpha_{nec} = 0.1$ as effectiveness and necessity thresholds. As the table shows, the higher bound of n_8's effectiveness range (0.2) is under threshold α_{eff} (0.3). Therefore, n_8 is considered to under-perform in terms of effectiveness and must be deactivated, triggering the specialisation of its parent norms, namely n_9, n_{10}.

To conclude, note that, when agents fulfil and infringe general norms, IRON is capable of evaluating both these general norms, and the norms they represent in the normative network. For instance, whenever agents fulfil norm n_4 in Figure 5.2 (Section 5.4) in the specific context described by n_2, IRON will detect that both n_4 and n_2 were applicable to the agent, and thus will evaluate both norms in terms of their effectiveness. This allows IRON to detect whenever grounded norms under-perform, even though they are not explicitly included in the normative system. This is possible because IRON does not remove generalised norms, but simply deactivates them in the normative network.

5.6.3 Evaluating normative systems

We now introduce a means to compute the overall effectiveness and necessity of a normative system at a given time. On the one hand, the effectiveness (μ_{eff}) of a normative system Ω can be computed as a whole up to a given time t based on the average effectiveness of its norms for the last q utility values up to time t. On the other hand, the necessity (μ_{nec}) of a normative system Ω can be computed as a whole up to a given time t based on the average necessity of its norms for the last q utility values up to time t.

$$\mu_{eff}(\Omega, t, q) = \frac{\sum_{n \in \Omega} \mu_{eff}(n, t, q)}{|\Omega|} \qquad \mu_{nec}(\Omega, t, q) = \frac{\sum_{n \in \Omega} \mu_{nec}(n, t, q)}{|\Omega|} \tag{5.21}$$

5.7 IRON's synthesis strategy

Next, IRON's synthesis strategy is described in detail. IRON is intended to be iteratively executed by a NSM in order to synthesise effective and compact normative systems. Based on BASE, IRON iteratively monitors a MAS at runtime, and performs norm synthesis by carrying out three subsequent synthesis stages: norm generation, norm evaluation, and norm refinement. During norm generation, IRON detects conflicts, and creates new norms as detailed in Section 4.3. During norm evaluation, it evaluates norms in terms of their effectiveness and necessity as described in Section 5.6. Thereafter, during norm refinement, it tries to generalise norms that perform-well as described in Section 5.4, and specialises under-performing norms as described in Section 5.5. While performing norm synthesis, IRON changes from one normative system to another by applying changes to the normative network described in Section 5.3.1 through the collection of operators introduced in Section 5.3.2. As a result of norm synthesis, IRON publishes a normative system Ω so that the agents within a MAS become aware of it.

Algorithm 7 illustrates IRON's strategy, which takes as input a tuple with a description of the previous state of a MAS (d_s) and a description of the current MAS state (d_{s_c}), and outputs a normative system to regulate the agents' behaviour. To perform norm synthesis, IRON considers the following globally accessible elements:

- A normative network NN, formally defined in Section 5.3.1.
- A set of operators $\mathcal{O} = \langle add, activate, deactivate, generalise, specialise \rangle$ described in Section 5.3.2.
- A set Φ of domain-dependent elements (box c in Figure 3.3 in Section 3.4) to perform norm synthesis in a given application domain. Specifically, $\Phi = \langle perceive, getConflicts, getContext, getAction, \mathcal{G} \rangle$, being $perceive$, $getConflicts$, $getContext$ and $getAction$ the domain-dependent

functions introduced in Section 3.2, and $\mathcal{G}$ a grammar to construct norms, formally described in Section 3.3.2.

- A set Ψ of domain-independent norm synthesis settings (box d in Figure 3.3 in Section 3.4). Particularly, $\Psi = \langle \mathcal{EC}, T \rangle$, being $\mathcal{EC}$ the evaluation criteria introduced in Section 5.6, and T a time interval to determine when a NSM has converged to a normative system.

- A set Γ of additional synthesis inputs (box e in Figure 3.3 in Section 3.4), which contains BASE's additional synthesis parameters (described at the beginning of Section 4.6), along with the following inputs: (1) a terms taxonomy (T) capturing taxonomical knowledge between the terms in $\mathcal{L}_{Ag}$; (2) two constants $k_{\mathit{eff}}, k_{nec}$ to set the initial effectiveness and necessity values of a norm n; and (3) the two generalisation thresholds introduced in Section 5.6.1 ($\alpha^{gen}_{\mathit{eff}}, \alpha^{gen}_{nec}$).

ALGORITHM 7: IRON's synthesis strategy

Input : $\langle d_s, d_{s_c} \rangle$
Output : Ω
Initialisations: $\mathcal{NCO} \leftarrow \emptyset$, $\mathcal{P} \leftarrow \emptyset, \mathcal{ER} \leftarrow \emptyset, \mathcal{NR} \leftarrow \emptyset, \mathcal{PTG} \leftarrow \emptyset$

[1] NN $\leftarrow$ `normGeneration`$(\langle d_s, d_{s_c} \rangle, \mathcal{P}, \mathcal{PTG})$;
[2] $(\mathcal{P}, \mathcal{ER}, \mathcal{NR})$ $\leftarrow$ `normEvaluation`$(\langle d_s, d_{s_c} \rangle, \mathcal{NCO}, \mathcal{P}, \mathcal{ER}, \mathcal{NR})$;
[3] NN $\leftarrow$ `normRefinement`$(\mathcal{NCO}, \mathcal{P}, \mathcal{ER}, \mathcal{NR}, \mathcal{PTG})$;
[4] Ω $\leftarrow \{n \in NN \mid \delta(n) = \mathit{active}\}$;
[5] **return** Ω

Additionally, IRON considers the following data structures that it initialises the first time it is invoked:

1. A structure ($\mathcal{NCO}$) to keep track of the compliance outcomes of each norm (see Section 3.3.3)

2. A structure ($\mathcal{P}$) to keep track of the effectiveness (μ_{eff_i}) and the necessity (μ_{nec_i}) of each synthesised norm (n_i).

3. A structure ($\mathcal{ER}$) containing the effectiveness lower bound and upper bound of each norm (see section 5.6).

4. A structure ($\mathcal{NR}$) containing the necessity lower bound and upper bound of each norm (see section 5.6).

5. A structure ($\mathcal{PTG}$) to keep track of the potential generalisations of each synthesised norm (see section 5.4.1).

Each time IRON is invoked by a NSM, it starts by carrying out norm generation. With this aim, it invokes function *normGeneration* (line 1), which detects conflicts in the current MAS state, and creates norms to avoid conflicts. Next,

normEvaluation in line 2 evaluates norms in terms of their effectiveness and necessity to avoid conflicts. Such function updates norms' utilities ($\mathcal{P}$), along with norms' effectiveness and necessity performance ranges ($\mathcal{ER}, \mathcal{NR}$). These ranges are finally employed by function *normRefinement* to refine the normative system by (1) discarding norms that under-perform; and (2) trying to generalise norms that perform well. Finally, IRON outputs a normative system as the norms that are active in the normative network (line 4). Subsequent sections describe in more detail each synthesis stage.

ALGORITHM 8: IRON's *normGeneration* function

Input : $\langle d_s, d_{s_c} \rangle, \mathcal{P}, \mathcal{PTG}$
Output: Ω

```
[1]  conflicts_{s_c} ← getConflicts (d_{s_c});
[2]  for c_{s_c} ∈ conflicts_{s_c} do
[3]      if not regulated(c_{s_c}, d_s) then
[4]          n ← create(c_{s_c}, d_s);
[5]          if n ∉ getNorms(NN) then
[6]              NN' ← add(n, NN);
[7]          NN' ← activate(n, NN);
[8]          potential_n ← createPotentialGeneralisations(n);
[9]          PTG ← PTG ∪ potential_n
[10] return NN;
```

5.7.1 Norm generation

IRON's *normGeneration* function is depicted in Algorithm 8. It starts by detecting conflicts within the current MAS state (line 1). Thereafter, for each conflict it detects, it creates a norm aimed at avoiding the conflict in the future. With this aim, it invokes function *create* (line 4), which creates a norm from a conflict as BASE does (Algorithm 3 in Section 4.6.1). Thereafter, it adds the norm to the normative network (line 6), and activates it (line 7). Nevertheless, IRON adds a final step to this process. For each created norm, it creates and records its potential generalisations by invoking function *createPotentialGeneralisations* (lines 8–9), described in Algorithm 6 in Section 5.4.1.

5.7.2 Norm evaluation

Algorithm 9 illustrates IRON's *normEvaluation* function, which carries out norm evaluation. First, it invokes function *getComplianceOutcomes* (line 1), which gathers the compliance outcomes of each norm that has been fulfilled and infringed in the transition from s to s' ($\mathcal{NCO}$). Next, function *evaluateNorms* (line 2) evaluates each fulfilled and infringed norm as detailed in Section 5.6, and outputs set $\mathcal{P}$, containing the current performances of each norm in terms of its effectiveness and its necessity. Thereafter, it updates the effectiveness and

necessity ranges ($\mathcal{ER}, \mathcal{NR}$) of each norm that has been evaluated during the current time step (lines 3–4). It computes these ranges as detailed in Section 5.6. Finally, in line 5, it updates the utilities of each case solution in the case base as BASE does (see Algorithm 4 in Section 4.3.2).

ALGORITHM 9: IRON's *normEvaluation* function

Input : $\langle d_s, d_{s_c} \rangle, \mathcal{NCO}, \mathcal{P}, \mathcal{ER}, \mathcal{NR}$
Output: $\mathcal{P}$

```
[1] NCO ← getComplianceOutcomes(⟨d_s, d_{s_c}⟩, NCO);
[2] P   ← evaluateNorms(NCO, P);
[3] ER  ← updateEffectivenessRange(P, ER);
[4] NR  ← updateNecessityRange(P, NR);
[5] CB  ← updateCases(P);
[6] return (P, ER, NR);
```

5.7.3 Norm refinement

IRON refines the normative system via norm generalisations (cf. Section 5.4) and norm specialisations (cf. Section 5.5). Algorithm 10 illustrates IRON's *normRefinement* function, which is in charge of executing norm refinement. It proceeds as follows. For each norm n fulfilled or infringed during the transition to the current MAS state (s_c), it checks the following conditions:

- If norm n under-performs in terms of its effectiveness *or* its necessity (cf. Section 5.6.2), then it deactivates norm n and specialises its *ancestors* in the normative network. In other words, it *deactivates up* norm n by invoking function *deactivateUp*.

- If norm n performs well in terms of both its effectiveness *and* its necessity (cf. Section 5.4.2), then it tries to generalise norm n by invoking function *generaliseUp*, which checks each potential generalisation of n, enacting them when possible.

ALGORITHM 10: IRON's *normRefinement* function.

Input : $\mathcal{NCO}, \mathcal{P}, \mathcal{ER}, \mathcal{NR}, \mathcal{PTG}$
Output : NN

```
1 for n ∈ getNormsFulfilledInfringedThisState(NCO) do
2     if ub^n_eff < α^gen_eff or ub^n_nec < α^gen_nec then
3         NN ← deactivateUp(n, NN);
4     else if lb^n_eff > α^gen_eff and lb^n_nec > α^gen_nec then
5         NN ← generaliseUp(n, NN, ER, NR, PTG);
6 return NN
```

Next, we provide the algorithms that IRON employs to generalise norms and to specialise norms.

Norm generalisation algorithm

Algorithm 11 illustrates IRON's *generaliseUp* function, which is in charge of enacting the potential generalisations of a norm n. It works as follows. First, it retrieves the potential generalisations of norm n (line 1). Then, for each potential generalisation, it checks if each sibling performs well in regulating conflicts as described in Section 5.6.1. If the generalisation condition is satisfied, then it invokes operator *generalise* (Table 5.1 in Section 5.3.2) to generalise norm n and its siblings into norm n' (line 8).

ALGORITHM 11: IRON's *generaliseUp* function

Input : $n, NN, \mathcal{ER}, \mathcal{NR}, \mathcal{PTG}$
Output: NN

```
1 potential_n ← potentialGeneralisations(n, PTG);
2 for ⟨n, n', S⟩ ∈ potential_n do
3     enact ← true;
4     for sibling ∈ S do
5         if lb_eff^sibling <= α_eff^gen  or  lb_nec^sibling <= α_nec^gen then
6             enact ← false;
7     if enact = true then
8         NN ← generalise(S ∪ {n}, n', NN);
9 return NN
```

Norm specialisation algorithm

To deactivate a norm along with its ancestors, IRON employs function *deactivateUp* depicted in Algorithm 12. This function proceeds by subsequently specialising each ancestor of n from top to bottom in the normative network, and finally deactivating n. First, it retrieves the parents of n (line 1), namely those norms that *directly* generalise it. Then, it recursively deactivates up each parent (lines 2–3). This recursive invocation makes that IRON starts by specialising the highest ancestor of n in the hierarchy, and subsequently specialises its descendants until it gets to n. For each specialisation, it checks if the n has no children in the normative network, namely it is a *leaf* (line 4). In that case, it deactivates the norm (line 5) by invoking operator *deactivate* (Table 3.1 in Section 3.4.2). Otherwise, it specialises norm n (line 7) by applying operator *specialise*(n, NN) depicted in Table 5.1 in Section 5.3.2. This operator deactivates norm n and activates its children.

ALGORITHM 12: IRON's *deactivateUp* function

Input : n, NN
Output: NN

```
parents ← getParents(n, NN)
foreach parent ∈ parents do
    deactivateUp(parent, NN)
if isLeaf(n, NN) then
    deactivate(n, NN)
else
    specialise(n, NN)
return NN
```

5.8 Complexity analysis

We now provide an analysis of the computational complexity of IRON. Before that, let us consider the number of norms that can be generated by a particular grammar $\mathcal{G}$. If p is the maximum number of predicates of any norm generated by the grammar, r is the maximum arity of any predicate, and d is the maximum number of terms at any position of any given predicate, the number of norms that can be generated by grammar $\mathcal{G}$ is $d^{r \cdot p}$. Given a grammar $\mathcal{G}$ we shall note the number of norms it can generate as $\eta_{\mathcal{G}}$. Moreover, given a CBR base CB, we shall note as η_{CB} the number of norms in the case base. Now we are ready to compute IRON's complexity.

Lemma 2. *The norm synthesis performed by the* IRON *algorithm when employing grammar $\mathcal{G}$ and case base CB when detecting κ conflicts takes time $O(\kappa \cdot \eta_{CB} + 3 \cdot |Ag| \cdot |NN| + \eta_{\mathcal{G}}(|Ag| \cdot |NN| + 1))$.*

Proof. The norm generation stage involves (i) generating norms for all detected conflicts and (ii) generating all potential generalisations for each new norm. The cost of the first step is $O(\kappa \cdot \eta_{CB})$, whereas the cost of the second step is $O(\eta_{\mathcal{G}})$. The cost of the norm evaluation process is $O(3 \cdot |Ag||NN|)$, since it involves assessing the applicability of norms ($O(|Ag||NN|)$), assessing the compliance with norms ($O(|Ag||NN|)$), and updating norms' utilities and performance ranges ($O(|Ag||NN|)$). Finally, the cost of norm refinement is $O(\eta_{\mathcal{G}} \cdot |Ag||NN|)$, which amounts to the worst case cost of generalising norms, since the cost of specialising norms is $O(|Ag||NN|^2)$. Putting all together, the resulting worst-case time is $O(\kappa \cdot \eta_{CB} + 3 \cdot |Ag| \cdot |NN| + \eta_{\mathcal{G}}(|Ag| \cdot |NN| + 1))$. □

Observe that the computation time of IRON is larger than BASE. Nonetheless, as it will be demonstrated in the experiments carried out in Sections 5.9 and 5.10, this is the price paid by IRON in order to significantly outperform BASE in terms of compactness.

As a final remark, notice that given a grammar $\mathcal{G}$, the number of normative systems is $2^{\eta_{\mathcal{G}}}$. This is precisely the size of the search space that IRON must

explore in search for compact normative systems. However, recall that IRON is an approximate algorithm for norm synthesis, and it does not require exploring the whole search space, as it will be demonstrated in Sections 5.9 and 5.10.

5.9 Empirical evaluation considering a traffic scenario

We now analyse IRON's norm synthesis in the road-traffic scenario. In these experiments, a NSM runs on top of a simulated traffic junction MAS, and iteratively executes IRON to perform norm synthesis. The remainder of this section is organised as follows. Section 5.9.1 describes the experimental settings, whereas Section 5.9.2 provides the results of this empirical evaluation.

5.9.1 Empirical settings

The road traffic scenario consists in a traffic junction modelled as a 21×21 grid. To compare IRON with BASE, the scenario has been configured as described in Section 4.8.1. Briefly, cars enter the scenario by randomly chosen entry points, and travel towards randomly chosen destinations. While travelling, cars can perform actions $Ac = \{go, stop\}$. In particular, cars perform action *"go"* by default, unless a norm prohibits it. At each time step, cars decide whether to fulfil or infringe the norms that apply to them according to a norm infringement rate ($\mathcal{NIR}$).

Each experiment consists of a set of 100 different simulations for both IRON and BASE. Both strategies start each simulation with an empty normative system, and it finishes whenever it reaches 50,000 ticks, or IRON/BASE converge to a stable normative system. IRON/BASE are assumed to have converged to a normative system, hence solving the NSP (see Definition 18 in Section 3.2.2), if during a 5,000-tick period, the normative system remains unchanged and no unregulated conflicts arise. Hereafter, we detail how both approaches have been configured to synthesise traffic norms.

Norm synthesis in the road traffic scenario

To perform norm synthesis in the traffic scenario, IRON considers BASE's implementation of the domain-dependent elements, which are described in Section 4.8.1. Nevertheless, IRON employs a different grammar $\mathcal{G}$ to be able to construct general norms, which is as follows.

Norm	::=	$\langle \{LHS\}, RHS \rangle$
LHS	::=	$LHS, LHS \mid \rho$
RHS	::=	$prh(Ac)$
Ac	::=	$go \mid stop$
ρ	::=	$left(\tau) \mid front(\tau) \mid right(\tau)$
τ	::=	*car-heading-left, car-heading-right, car-opposite-heading, car-same-heading, wall, nil, anything*

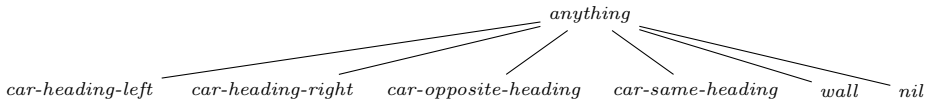

Figure 5.5: Taxonomy of terms of the road traffic scenario

The precondition of each norm contains three unary predicates *left*, *front*, *right*, which represent the 3 cells in the context of a reference car. Each predicate contains one term out of a set of seven terms {*car-heading-left, car-heading-right, car-opposite-heading, car-same-heading, wall, nil, anything*}. The first four terms represent a car along with the direction it is heading to, while term *"wall"* represents a wall, *"nil"* represents no car, and term *"anything"* represents whether car, a wall, or nothing. Figure 5.5 depicts a taxonomy capturing generalisation relationships between these terms. The taxonomy is rooted at term *anything*, which represents (generalises) the remaining terms to describe a car heading to different directions, a wall, or nothing. That is, *car-heading-left* $\sqsubseteq$ *anything*, *car-heading-right* $\sqsubseteq$ *anything*, *car-opposite-heading* $\sqsubseteq$ *anything*, *car-same-heading* $\sqsubseteq$ *anything*, *wall* $\sqsubseteq$ *anything*, *nil* $\sqsubseteq$ *anything*.

With this grammar, IRON can create norms of the form:

$$n : \langle\{left(car\text{-}heading\text{-}right),\ front(nil),\ right(nil)\},\ prh(go)\rangle$$
$$n' : \langle\{left(car\text{-}heading\text{-}right),\ front(anything),\ right(anything)\},\ prh(go)\rangle$$

Norm n prohibits a reference car from moving on (hence giving way) if the cell to its left contains a car heading towards its right, and the cells in front and right contain nothing. Norm n' is a general norm (an ancestor of n) prohibiting a car to go forward if there is a on its left which is heading right from its perspective, and no matter what it perceives to its front and right cells. With this grammar, IRON can synthesise $7^3 = 343$ different norms, and the number of normative systems to consider amounts to 2^{343} ($> 10^{103}$).

To reduce agents' norm reasoning efforts, before providing norms to the agents IRON *removes* from a norm's precondition those predicates whose term is the root of a tree in the taxonomy, namely whose terms are most general. As an example, consider that IRON publishes a normative system containing norm n' above. In that norm, predicates *front* and *right* contain term *"anything"*, hence stating that no matter what a car represents to its front and right positions. Therefore, IRON will represent norm n' as

$$\langle\{left(car\text{-}heading\text{-}right)\},\ prh(go)\rangle$$

which prohibits a car to go if it perceives a car coming from its left which is heading to the right. Since it does not contain *front* and *right* predicates, then *no matters* what a car perceives to its front and right positions. Therefore, it represents the same constraint than n', even though it has only one predicate.

Parameter	Description	Value
q	Number of effectiveness values (μ_{eff}) and necessity values (μ_{nec}) considered to compute a norm's effectiveness and necessity ranges (see Section 5.6).	100
$k_{\mathit{eff}}, k_{nec}$	Default norm effectiveness and necessity values ($\mu_{\mathit{eff}}(n, t_0), \mu_{nec}(n, t_0)$).	0.5
$\alpha_{\mathit{eff}}, \alpha_{nec}$	Thresholds below which a norm is considered to under-perform in terms of its effectiveness/necessity.	0.2
$\alpha_{\mathit{eff}}^{gen}$	Threshold above which a norm is considered to perform well in terms of its effectiveness.	0.6
α_{nec}^{gen}	Threshold above which a norm is considered to perform well in terms of its necessity.	0.4
T	Time period considered when assessing convergence (cf. Definition 18 in Section 3.2.2).	1,000

Table 5.3: IRON's norm synthesis settings in the road traffic scenario.

Norm synthesis settings

Finally, IRON's and BASE's norm synthesis parameters have been configured as follows. In the case of BASE, it has been configured as shown in Table 4.1 in Section 4.8.1. As to IRON, it has been configured as depicted in Table 5.3. In short, we have taken a conservative approach to configure IRON. Firstly, IRON considers a great amount of evidences when computing a norm's effectiveness/necessity ranges. Given a norm n, its effectiveness range ($\mathcal{ER}^n$) and its necessity range ($\mathcal{NR}^n$) are computed by considering the last 100 undefined effectiveness/necessity values of n ($q = 100$). Secondly, after creating a norm, IRON sets its initial effectiveness and necessity to 0.5 ($k_{\mathit{eff}} = 0.5, k_{nec} = 0.5$). Thirdly, IRON deactivates norms only when they perform very poorly ($\alpha_{\mathit{eff}} = 0.2, \alpha_{nec} = 0.2$). Fourthly, IRON considers high generalisation thresholds ($\alpha_{\mathit{eff}}^{gen} = 0.6, \alpha_{nec}^{gen} = 0.4$). Finally, IRON considers a convergence interval of 5,000 ticks ($T = 5,000$).

5.9.2 Empirical results

Next, we analyse IRON's empirical results. With this aim, we provide:

1. A micro analysis that illustrates how IRON reaches convergence, namely how it manages to converge to a compact normative system that solves the norm synthesis problem.

2. A macro analysis that demonstrates that IRON is able to converge despite a large proportion of non-compliant behaviours of the agent society.

3. A comparison of IRON and BASE in terms of their stability and the compactness of the normative systems they synthesise.

Micro analysis: IRON's convergence process

Let us now analyse a prototype execution of IRON to illustrate its convergence process. Figure 5.6 shows the results of an IRON's execution for a single road traffic simulation with 0.3 norm infringement rate. That is, on average, 3 of each 10 agents' decisions lead to norm infringements. On the x-axis, it shows the normative changes (i.e., the time steps in which the the normative network and/or the normative system change). On the y-axis, it shows:

1. The normative network's cardinality, namely the total number of synthesised norms.
2. The cardinality of the normative system, namely the number of active norms in the normative network.
3. The number of grounded norms the normative system represents.
4. The compactness of the normative system, namely the number of terms in the preconditions of its norms (see Definition 26 in Section 5.2).
5. The ratio of unregulated car collisions at a given tick[1].

At tick 13 (which corresponds to the first normative change), the first collision arises and IRON synthesises the first norm. From that tick onwards, IRON keeps generating norms when needed, hence increasing the cardinality of both the normative network and the normative system. As a consequence, the number of terms in the normative system increases as well. At tick 20 (sixth normative change), IRON performs the first norm generalisation, reducing the cardinality of the normative system from 7 to 6 norms. As a result of this norm generalisation, the number of norms and terms in the normative system decreases, thus increasing its compactness. Up to tick 2,213 (seventeenth normative change), IRON keeps generating and generalising norms when possible. Norm generalisations reduce the total number of terms of the normative system. At tick 2,213 IRON performs the last norm generalisation, hence synthesising a compact normative system of 6 norms with 10 terms in total, which represents 16 grounded norms. From tick 2,213 onwards, the normative system remains stable. By using the resulting normative system, cars that comply with norms do not cause collisions. However, those cars may collide with other cars that infringe norms. Recall that those collisions that have arisen from norm infringements are not taken into account when assessing convergence. After 5,000 further ticks, IRON reaches the convergence criteria (tick 7,213). Overall, IRON explored 35 different norms (out of 343 possible ones), which were generalised into 6 norms, to find a 6-norm normative system that represents 18 different grounded norms. By employing this normative system, IRON successfully prevents collisions as long as cars comply with its norms.

The 6-norm normative system that IRON converged to is shown in Table 5.4. As previously detailed, IRON provides the agents with norms that do not contain

[1]Computed as the moving average of unregulated collisions of the last 10 ticks.

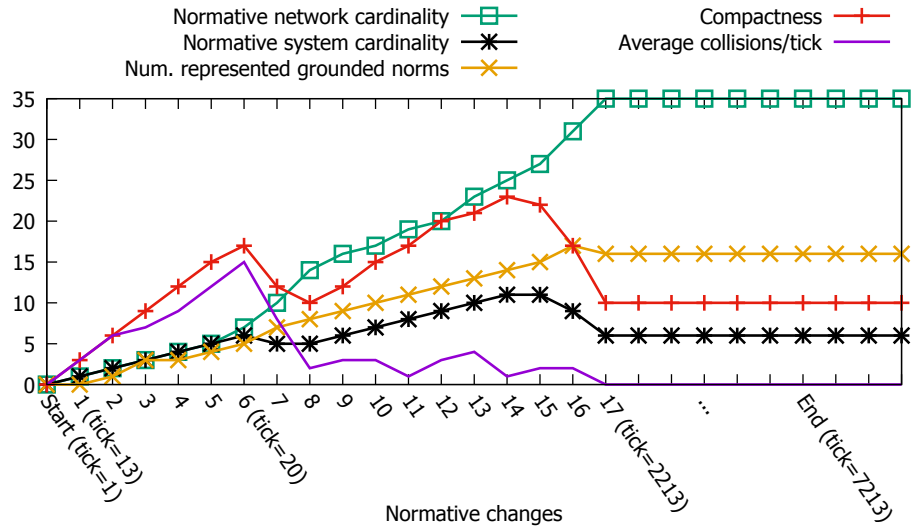

Figure 5.6: Norm synthesis along a single simulation. The x-axis corresponds to normative changes (i.e., changes in the normative network and/or the normative system), and the y-axis corresponds to the different measures in the legend.

Norm	Pre-condition (θ)		Norm target	$\hat{\mu}^n_{eff}$	$\hat{\mu}^n_{nec}$
n_1	*left*(*car-heading-right*)		*prh*(*go*)	0.93	0.87
n_2	*left*(*car-heading-left*),	*front*(*car-heading-left*)	*prh*(*go*)	0.95	0.64
n_3	*front*(*car-same-heading*)		*prh*(*go*)	0.83	0.33
n_4	*front*(*car-heading-left*),	*right*(*car-heading-left*)	*prh*(*go*)	0.81	0.75
n_5	*front*(*nil*),	*right*(*car-heading-left*)	*prh*(*go*)	0.95	0.64
n_6	*left*(*nil*),	*right*(*car-heading-left*)	*prh*(*go*)	0.92	0.26

Table 5.4: A normative system upon convergence

predicates containing root terms (e.g., term *"anything"*). Norm n_1 is a general left-hand side priority norm specifying that a car must stop when it observes a car to its left which is heading to its right, and no matter what it perceives to its front and right positions. Upon convergence, norm n_1 had very high average values of effectiveness ($\hat{\mu}^n_{eff} = 0.93$) and necessity ($\hat{\mu}^n_{nec} = 0.97$), since n_1 successfully avoids collisions in this situation when it is fulfilled and, when it is infringed, collisions arise most of the times. Norms n_2 and n_3 represent variations of a general *"keep your distance"* norm. They prohibit a car to go when it observes a car in front heading different directions. These norms would avoid a collision if the car in front suddenly stopped. This situation rarely leads to collisions, since the car in front rarely stops. As a consequence, its has low average necessity ($\hat{\mu}_{nec} = 0.33$). Finally, norms n_4 to n_6 are variations of a general right-hand side priority norm. They prohibit a car to go forward if it perceives a car to its right, and different variations to its front and left. Note that IRON significantly outperforms BASE in terms of compactness. As shown in Table 4.2 in Section 4.8.2, BASE provided cars with 16 (grounded) norms and 48 terms in total. By contrast, IRON provided agents with 6 norms and 10 terms in total, which is significantly more compact.

Macro analysis

We now explore the limits of IRON by testing its synthesis capabilities under different norm infringement rates. Specifically, IRON's convergence ratio is analysed for different norm infringement rates, ranging from 0.1 to 0.9. With this aim, IRON has been executed in 100 simulations per norm infringement rate. Figure 5.7 shows averaged results of the effectiveness and necessity of the synthesised normative systems. For the sake of clarity, standard deviations are not plotted. However, it is worth mentioning that the standard deviations for effectiveness and necessity for each norm infringement rate range within $[0.006, 0.011]$ and $[0.080, 0.0137]$ respectively. The different series show:

1. The convergence ratio, namely the number of simulations that converged to a stable normative system out of the 100 simulations.
2. The effectiveness degree (i.e., the averaged effectiveness) of the normative system up to convergence.
3. The necessity degree (i.e., the averaged necessity) of the normative system up to convergence.
4. The dispersion (variability) degree, namely the dispersion of the distribution of the normative systems that IRON converged to.

The analysis is performed for very low (0.1), low (0.2 to 0.3), medium (0.4 to 0.6) and high (beyond 0.6) norm infringement rates. In particular, up to medium norm infringement rates (up to 0.4), IRON's convergence rate is 1, namely it successfully converged 100% of the times. Furthermore, it converged to normative systems with high averaged effectiveness (0.88) and necessity (0.71). Between

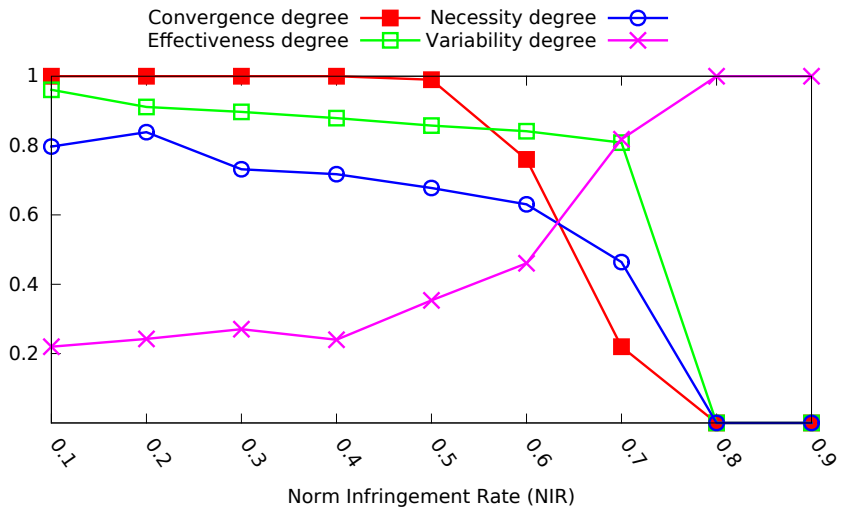

Figure 5.7: Stability analysis depending on norm infringements. The x-axis represents the different norm infringement rates, and the y-axis represents the different degrees that are given in the legend.

medium and high norm infringement rates (0.4 and 0.6), the convergence ratio decreases (due to oscillations in the normative systems), and it is for very high norm infringement rates (from 0.8 onwards) that IRON cannot converge to a stable normative system. Overall, IRON proved to be highly resilient to non-compliant behaviours during the synthesis process, managing to successfully synthesise norms despite up to 40% norm infringement rate of agents.

As to dispersion, below medium norm infringement rates (0.4), it remains near 0.2 (i.e., 100 executions converged to 20 different normative systems). Thereafter, its dispersion increases as the norm infringement rate increases. This happens because, as long as the norm infringement increases, most unnecessary norms (e.g., norm n_6 in Table 5.4) become unstable (deactivated and re-activated back and forth). As a result, IRON takes longer to synthesise a stable normative system, hence exploring new, different normative systems that it did not explore with lower norm infringement rates.

IRON vs. BASE: Comparison of stability and compactness

Next, IRON and BASE are compared in terms of their stability and compactness.

– **Stability analysis.** The stability of IRON and BASE is computed as their convergence ratio. Both IRON and BASE have been executed in 100 simulations per norm infringement rate. Figure 5.8 illustrates how normative systems change along a single, sample simulation with 0.4 norm infringement rate. The switch frequency between different normative systems of BASE is much higher than

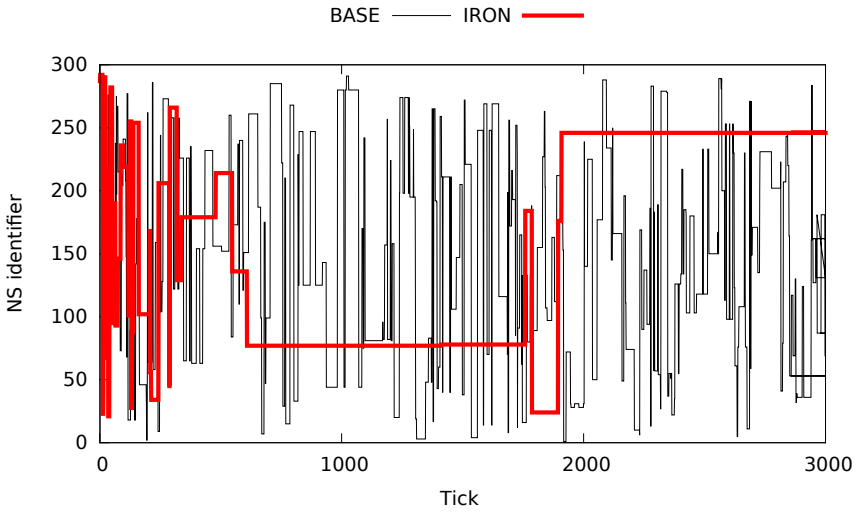

Figure 5.8: Normative system changes along time.

IRON's, which stabilises normative systems for longer periods of time until it converges. At the end of the simulation, BASE explored 251 different normative systems and was not able to converge, while IRON explored 41 different normative systems out of 10^{103} possible normative systems, and managed to converge to a normative system.

Figure 5.9 compares the stability degree (i.e., the convergence ratio) of both norm synthesis methods over 100 simulations. For very low norm infringement rates (0.1), both methods successfully converged to a normative system that effectively avoids unregulated collisions. Nevertheless, beyond low norm infringement rates (0.2), BASE dramatically decreases its stability degree (i.e., it becomes less efficient in converging to a normative system that avoids collisions). In fact, it never manages to converge to a normative system (i.e., it has 0 stability degree) beyond the 0.3 norm infringement rate, since it continuously deactivates/re-activates norms, and thus unregulated collisions are never completely eradicated. As for IRON, it converges for low norm infringement rates (up to 0.3) and totally removes unregulated collisions 100% of the times. Still, for medium norm infringement rates (0.4 to 0.6), its stability degree ranges between 0.78 and 0.96, namely it converges between 78% and 96% of the simulations. The stability degree of IRON tends to decrease beyond high norm infringement rates (0.6), and it is for very high norm infringement rates (0.8) that it fails to converge. Overall, IRON is much more stable than BASE, allowing to converge for much higher norm infringement rates.

– **Compactness savings.** Next, both methods are compared in terms of their compactness. Recall from Definition 26 that the compactness of a normative system is measured as its overall number of norm terms. Figure 5.10 illustrates

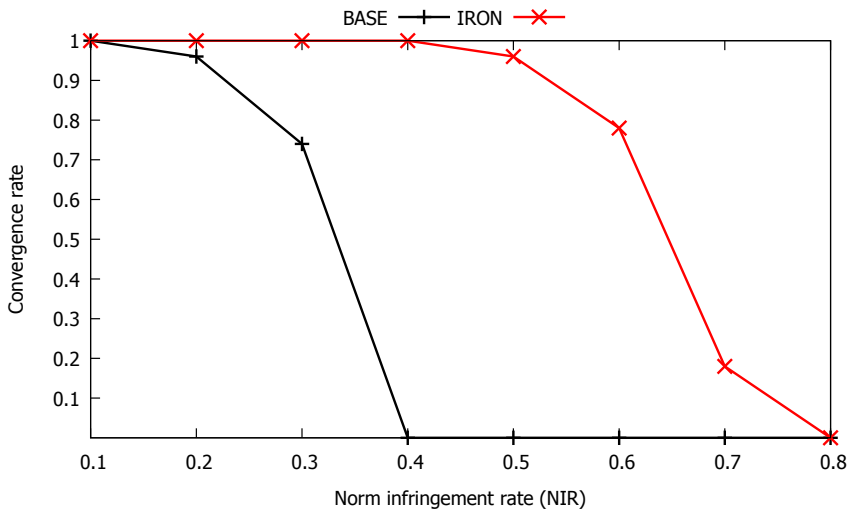

Figure 5.9: Comparison of the stability degree of IRON and BASE

Figure 5.10: Compactness savings of IRON with respect to BASE

the compactness savings achieved by IRON with respect to BASE: it saves compactness for all norm infringement rates, achieving its best results for medium norm infringement rates. For low norm infringement rates, (up to 20% norm infringement rates) IRON manages to converge to normative systems that are between 30% and 40% more compact (have between 30% and 40% fewer terms) than those synthesised by BASE. As for medium norm infringement rates (0.3 to 0.6), IRON obtains its best savings in compactness (up to 70%: from 36.2 down to 10.5 terms on average). Here IRON benefits from its stability, whereas BASE is penalised by its instability. This is due to the fact that, when BASE does not converge, it typically outputs normative systems with a low compactness. Specifically, while IRON manages to converge between 90% and 100% of the time, BASE just converges 72% of the time for a norm infringement rate of 0.3, while it never converges beyond a 0.4 norm infringement rate. As a result, IRON synthesises normative systems that are up to 70% more compact than those synthesised by BASE. As for high norm infringement rates (beyond 0.6), IRON's savings in compactness tend to decrease since its convergence rate decreases as well. Overall, IRON synthesised normative systems that are much more compact than those synthesised by BASE.

5.10 Empirical evaluation considering an on-line community scenario

Next, IRON is empirically evaluated in the on-line community scenario introduced in Section 1.3.2. The aim is to demonstrate the domain-independence of both the norm synthesis model introduced in Section 3.3 and the IRON strategy. First, Section 5.10.1 describes the empirical settings of IRON in this scenario. Thereafter, Section 5.10.2 depicts IRON's empirical results. Specifically, it provides a micro analysis illustrating IRON's convergence process, and a macro analysis illustrating how often IRON converges, and the types of normative systems it synthesises.

5.10.1 Empirical settings

In our particular notion of an on-line community, a population of users continuously interact by exchanging contents into different sections: *forum, the-reporter*, and *multimedia*. While section *forum* allows users to chat and exchange opinions, section *the-reporter* is used for publishing news and section *multimedia* is employed to exchange media contents such as videos and photos. The actions available to the users are $Ac = \{upload, view, complain\}$, and thus users are allowed to:

1. *Upload* contents to the different sections.
2. *View* uploaded contents.
3. *Complain* about those viewed contents that they consider as *inappropriate*.

Uploaded contents may have a *type* out of $\{correct, spam, porn, violent, insult\}$, being *spam, porn, violent* and *insult* inappropriate contents that can trigger users' complaints.

The overall goal in this scenario is to achieve *healthy* on-line communities [Hinds and Lee, 2011], namely communities wherein frictions rarely occur, and contents are exchanged fluidly. In particular, the goal is to avoid situations in which a user feels discomfort about contents uploaded by other users, thus reporting complaints. Note therefore that users' complaints identify situations in which a user has uploaded inappropriate contents. Therefore, complaints are regarded as *conflicts* that drive IRON's norm synthesis. Note that, in this case, the users guide IRON's norm synthesis through their complaints. IRON synthesises norms to prohibit the type of contents that users complain about. In this sense, we may say that IRON is employed to build a *participatory regulatory mechanism*, which synthesises normative systems that are aligned with users' preferences.

In these experiments, IRON starts each simulation with an empty normative system. Each simulations considers a *"warm-up"* period of 500 ticks that allows it to reach normal conditions in on-line communities, where the users enter in the community and there already exist contents to view and complain about. Thus, from tick 0 to 500, users only upload contents, and from tick 500 onwards, users upload, view, and complain about contents. As a simulation progresses, users upload and view appropriate and inappropriate contents, and complain about inappropriate contents. When that happens, IRON synthesises normative systems to prohibit to upload the type of contents identified as inappropriate by the users. A simulation finishes whenever it reaches 5,000 ticks. At the end of each simulation, IRON is considered to have converged, hence solving the NSP (cf. Definition 18 in Section 3.2.2, when during a 1,000-tick period: (1) the normative system remains unchanged; and (2) no unregulated conflicts arise (see Definition).

Subsequent sections describe the settings of the on-line community scenario; the implementation of the different domain-dependent elements employed by IRON in this scenario (inputs Φ, Γ in Figure 3.3 in Section 3.4); and the norm synthesis settings that IRON considers during the synthesis process (input Ψ in Figure 3.3 in Section 3.4).

On-line community settings

These experiments consider a discrete agent-based simulator that we have implemented to simulate the on-line community scenario described above. Particularly, each section of the community is implemented as a *FIFO* queue with capacity for 1000 different contents. Thus, whenever a section is full and a new content arrives, the oldest content is removed in order to place the newly arriving content.

The aim is to regulate a simulated on-line community of 10,000 users. However, as described in [Nielsen, 2006], within an on-line community only the 1% of its users are *heavy contributors*, while the remaining 99% are *intermittent*

contributors and *lurkers* (namely, users that barely contribute to the community). Thus, for the sake of simplicity, a 10,000-user population is represented by means of a population of 100 heavy contributors. At each time step (i.e, tick) each user: (i) views one content with probability 1; and (ii) uploads one content with probability 0.05. Notice that the probability to upload contents is much lower than the probability to view contents. With this setting, we aim at simulating users' behaviour in real on-line communities, in which users view contents more often than they upload. Moreover, at each tick a user decides whether to fulfil or infringe those norms that apply to it according to a *norm infringement rate*.

A user behaves within the community according to its *profile*, which describes *when* and *how* it uploads, views, and complains about contents. Specifically, a user's profile is composed of three sub-profiles:

- *The upload profile*, which defines (i) its *upload frequency*, namely its probability to upload contents at a given time; and (ii) different *upload probabilities* for each content type.
- *The view profile*, which describes its probability to view contents from a given section at a given time step.
- *The complain profile*, which defines its probability to complain about each type of inappropriate content it views.

As an example, Table 5.5 depicts the profile of what we call a *moderate* user. That is, a user that only uploads *correct* contents, and complains about all the inappropriate contents it views. More specifically, Table 5.5 describes a user with probability 1 to upload *correct* contents, and probability 0 to upload inappropriate contents (i.e., *spam, porn, violent, insult*). As described in the complain profile, it complains about each type of inappropriate content with probability 1, and complaints about *correct* contents with probability 0. Finally, the view profile defines the different probabilities of the user to view contents from each section.

Upload Profile		**View Profile**		**Complain Profile**	
Type	**Prob**	**Section**	**Prob**	**Type**	**Prob**
Correct:	1	Forum:	0.34	Correct:	0
Spam:	0	The Reporter:	0.33	Spam:	1
Porn:	0	Multimedia:	0.33	Porn:	1
Violent:	0	**TOTAL**	1	Violent:	1
Insult:	0	***View Mode***		Insult:	1
TOTAL:	1	By Order	•		
Upload Frequency		Most Viewed	○		
Frequency:	1	Random	○		

Table 5.5: A user's profile

The experiments consider three different populations composed of *moderates* and *spammers* in different proportions. On the one hand, moderates behave

according to the profile depicted in Table 5.5. On the other hand, spammers upload *spam* contents with probability 1, and never complain about *spam*. We then consider the following populations:

1. A population with a *majority* of moderate users, composed of 70% moderates and 30% spammers (hereafter 30M-70S).
2. A *balanced* population of 50% moderates and 50% spammers (50M-50S).
3. A population with a *minority* of moderate users, composed of 30% moderates and 70% spammers (70M-30S).

Norm synthesis in the on-line community scenario

In this scenario, IRON considers the following implementation of each domain-dependent element required by a NSM (Φ in Figure 3.3 in Section 3.4).

– **Function *perceive***. IRON describes a state of the on-line community by generating an expression of $\mathcal{L}_{MAS}$ that describes the contents in each section of the community. In particular, each content is represented by describing (1) its *identifier*; (2) its *proprietary*, namely the user who uploaded it; (3) the *section* the content has been uploaded to; and (4) its *type* (e.g., *correct, spam, etc.*).

– **Function *getConflicts***. Given a state, function *getConflicts* returns a (possibly empty) set of conflicts, where each conflict describes a particular content that has received user' complaints in state that state.

– **Function *getContext***. Given a state, and a user that uploaded a content in that state, function *context* infers the user's context in the state as an expression of $\mathcal{L}_{Ag}$, which describes: (1) the user's *identifier*, (2) the *type* of the content it has uploaded; and (3) the *section* of the community whereto the content has been uploaded. With this aim, IRON aims at inferring the *state of mind* the user had when uploading the state.

– **Function *getAction***. Given a transition between two states s, s', and a user that is part of states s, s', function *action* returns action *"upload"* if the user has uploaded a new content during the state transition. With this aim, it compares the contents in each state to assess if the user has uploaded new contents.

– **Grammar** $\mathcal{G}$: The grammar considered in this scenario is as follows.

Norm	::=	$\langle \{LHS\}, RHS \rangle$
LHS	::=	$LHS, LHS \mid \rho$
RHS	::=	$prh(Ac)$
Ac	::=	$view \mid upload \mid complain$
ρ	::=	$user(\tau_u) \mid section(\tau_s) \mid cntType(\tau_c)$
τ_u	::=	$u_1, \ldots, u_{100}$, $anyUser$
τ_s	::=	*forum, the-reporter, multimedia, anySection*
τ_c	::=	*spam, porn, violent, insult, inappropriate, appropriate* *anyContentType*

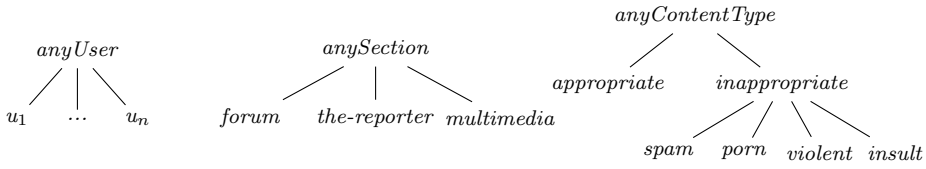

Figure 5.11: Taxonomy of terms in the on-line community scenario.

where: terms $u_1, \ldots, u_{100}$ represent each one of the 100 users of the population, and term *"anyUser"* represents any possible user; terms *"forum", "the-reporter", "multimedia"* represent the three different sections of the community, and term *"anySection"* generalises terms for each section; terms *"spam"*, *"porn"*, *"violent"*, *"insult"* represent different types of *inappropriate* contents, and *"appropriate"* contents are those uploaded by moderate users; and term *"anyContentType"* represents both appropriate and inappropriate contents.

Note therefore that a norm's precondition contains three unary predicates *user, section, cntType* that can contain, respectively, 101 different terms (one for each user, and one general term), 4 terms (one for each section, and one general term), and 7 terms (one for each content type, and three general terms). Therefore, the grammar this grammar allows to synthesise $101 \times 4 \times 7 = 2828$ different norms, and the number of normative systems to consider amounts to $2^{2828} (> 10^{851})$.

Figure 5.11 illustrates a taxonomy between the terms of the grammar, in which all users can be compactly represented by means of a general term *anyUser*, and analogously for sections and content types. With this grammar, IRON can synthesise norms of the following form:

$$\begin{aligned}
n^I &: \langle \{user(u_1),\ section(forum),\ cntType(spam)\},\ prh(upload)\rangle \\
n^{II} &: \langle \{user(u_1),\ section(the\text{-}reporter),\ cntType(spam)\},\ prh(upload)\rangle \\
n^{III} &: \langle \{user(u_1),\ section(multimedia),\ cntType(spam)\},\ prh(upload)\rangle \\
n^{IV} &: \langle \{user(u_1),\ section(anySection),\ cntType(spam)\},\ prh(upload)\rangle
\end{aligned}$$

Norms n^I, n^{II}, n^{III} prohibit user u_1 to upload *spam* contents to sections *forum*, *the-reporter*, and *multimedia*, respectively. Norm n^{IV} compactly represents norms n^I, n^{II}, n^{III}, prohibiting user u_1 to upload spam to *anySection*.

We now describe (1) the way IRON represents cases in the CBR mechanism considered by function *getConflictSource* to retrieve a conflict's source (see Section 4.6.2); and (2) the implementation of each domain-dependent required by such function to carry out a complete cycle of the unsupervised CBR cycle. That is, functions *getSimilarity* and *exploitSolution*.

– ***CBR Cases***. IRON employs function *getConflictSource* to determine a conflict's source. In particular, such function employs an unsupervised CBR mechanism that exploits previous experiences to determine the agent responsible for a given conflict. In this scenario, a conflict describes a content that has received

complaints, which contains the identifier of its *proprietary*, (namely, the user who uploaded it). Therefore, the conflict description itself contains information enough to determine the user responsible for the conflict (i.e., the content's proprietary). In this sense, IRON does not require to exploit cases of the case base. Therefore, a case in this scenario simply describes the conflictive content. Eventually, IRON will employ this information to blame the content's proprietary for the conflict.

– **Function *getSimilarity***. Since IRON does not require to exploit CBR cases to determine a conflicts' source, this function returns always 0. Therefore, two cases will be always different.

– **Function *exploitSolution***. Since case solutions are never exploited, this function carries out no operations.

Norm synthesis settings

IRON's has been configured as depicted in Table 5.6. For each new norm n, IRON sets its initial necessity to 0.5 ($\mu_{nec}(n, t_0) = 0.5$). Note that, in this case, IRON does not initialise a norm's effectiveness. The reason is that in this scenario IRON cannot evaluate norms in terms of their effectiveness, since norm fulfilments are *not observable*. As an explanation, recall from Section 5.6 that IRON computes a norm's effectiveness based on its fulfilments. In this scenario, when a user fulfils a norm, it implies that the she decides not to upload a content she had the intention to upload. This (absence of) action is not observable by IRON, which cannot detect norm fulfilments in order to compute norms' effectiveness.

A norm' necessity range ($\mathcal{NR}$) is computed by considering its last $q = 100$ defined necessity values (μ^n_{nec}). IRON's necessity generalisation threshold is set to 0 ($\alpha^{gen}_{nec} = 0$) so that it generalises any norm that is necessary enough to be active. IRON deactivates a norm whenever its necessity range is below a necessity threshold (α_{nec}). Thus, α_{nec} directly affects to the preservation and discard of norms, and hence to the convergence to a normative system. Therefore, IRON's norm synthesis is analysed for different necessity thresholds. More specifically, $\alpha_{nec} \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$.

5.10.2 Empirical results

Now, IRON's empirical results are analysed. First, a micro analysis of IRON's convergence process is provided. The aim is to shed light on how IRON manages to synthesise norms from scratch based on users' complaints, yielding a normative system that avoids conflicting situations. Thereafter, a macro analysis studies IRON's convergence rate for different populations and necessity thresholds, and analyse the type of normative networks that IRON synthesises.

Parameter	Description	Value
q	Number of effectiveness values (μ_{eff}) and necessity values (μ_{nec}) considered to compute a norm's effectiveness and necessity ranges (see Section 5.6).	100
k_{nec}	Default norm effectiveness and necessity values ($\mu_{\mathit{eff}}(n, t_0), \mu_{nec}(n, t_0)$).	0.5
α_{nec}	Threshold below which a norm is considered to under-perform in terms of its necessity.	(0.1, 0.3, 0.5, 0.7, 0.9)
α_{nec}^{gen}	Threshold above which a norm is considered to perform well in terms of its necessity.	0
T	Time period considered when assessing convergence (cf. Definition 18 in Section 3.2.2).	5,000

Table 5.6: IRON's norm synthesis settings in the on-line community scenario.

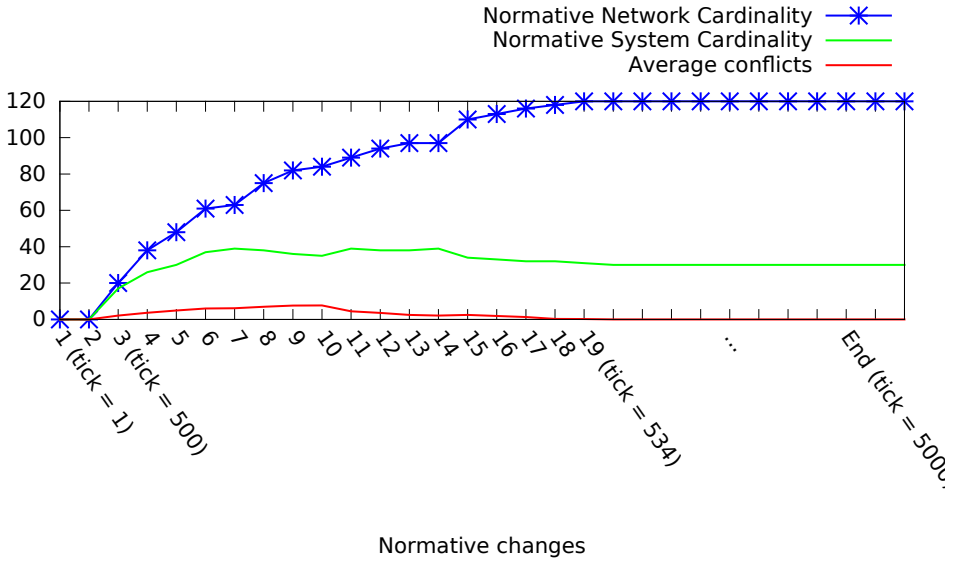

Figure 5.12: IRON's convergence process for a population of 70 moderate users and 30 spammers.

Micro analysis: IRON's convergence process

This first analysis focuses on how IRON manages to synthesise norms for an example population with a majority of moderate users. In particular, we consider a population of 70 moderate users and 30 spammers with a 0.3 norm infringement rate. Figure 5.12 shows a prototypical execution of IRON with this configuration. On the x-axis, it shows the different normative changes that IRON performed along time. On the y-axis, it shows:

1. The cardinality of the normative network.
2. The cardinality of the normative system.
3. The ratio of unregulated user complaints (conflicts) at a given tick.

During the *warm-up* period (up to tick 500) users only upload contents, and therefore there are no user complaints. At tick 500 (third normative change), moderate users start complaining about spam contents that they view. These complaints are considered as conflicts by IRON, which, from tick 500 onwards, creates new norms to prevent spammers from uploading spam contents in the different sections of the community (examples of these norms are depicted in Figure 5.13). As a consequence, the cardinality of the normative network increases, as well as the cardinality of the normative system. As IRON starts creating norms, that is, from tick 500 onwards, it performs norm generalisations when possible. In this way, IRON keeps the cardinality of the normative system oscillating around 30 norms, while the cardinality of the normative network keeps on increasing as it creates new (specific) norms.

At tick 534 (nineteenth normative change), IRON manages to synthesise a compact normative system of 30 general norms that regulate the on-line community. Note that, by contrast, the normative network contains 120 norms. That is: (1) 90 grounded norms to prohibit the thirty spammers to upload spam into the three sections of the community (30 spammers $\times$ 3 sections $=$ 90 norms); and (2) 30 general norms that compactly represent these 90 norms. These 30 general norms compose the normative system that is finally provided to the agents. Figure 5.13 shows these norms ($n_{91}, \ldots, n_{120}$), which prohibit users $u_{71}, \ldots, u_{100}$ (i.e., the spammers) to upload spam contents in *any section* of the on-line community. Note that these norms do not contain the *"section"* predicate since, before publishing a norm for the agents, IRON removes those predicates containing root terms (i.e., term *"anySection"*). After the synthesis of these 30 norms, the simulation converges and no new unregulated conflicts (complaints) are detected from tick 534 onwards.

Macro analysis: IRON's convergence outcomes

Next, a macro analysis of IRON's convergence is provided. First, we analyse for which combinations of population/norm necessity threshold IRON manages to converge, and the type of normative systems it synthesises in each case. Table 5.7 depicts averaged results after 100 simulations. Each cell contains, for each

$$
\begin{aligned}
n_{91} &: \langle (user(u_{71}),\ cntType(spam)),\ prh(upload) \rangle \\
&\dots \\
n_{120} &: \langle (user(u_{100}),\ cntType(spam)),\ prh(upload) \rangle
\end{aligned}
$$

Figure 5.13: Norms that IRON synthesises to regulate spammers' behaviour.

	Deactivation threshold (α_{nec})				
Population	0.1	0.3	0.5	0.7	0.9
30M-70S	70	X	X	X	X
50M-50S	50	50	X	X	X
70M-30S	30	30	30	X	X

Table 5.7: IRON's normative systems size upon convergence.

population and necessity threshold: either (i) the size of the normative system upon convergence; or (ii) symbol "X" if it was not able to converge to a normative system. Observe that, for a population of 30 moderates and 70 spammers (30M-70S) IRON was able to converge to a stable normative system of 70 norms whenever the necessity threshold is low ($\alpha_{nec} < 0.3$). By contrast, it did not converge for medium and high necessity thresholds ($\alpha_{nec} \geq 0.3$). Along the same lines, for population 50M-50S, it converged to 50 norms only for low and medium necessity thresholds ($\alpha_{nec} < 0.5$), and converged to 70 norms for population 70M-30S for low, medium and high thresholds ($\alpha_{nec} < 0.7$). In particular, IRON converged to one norm for *each spammer* to prevent her from uploading spam in *any* section. For instance, for a population of 70 spammers it generated 70 norms similar to the following norm below.

$$\langle \{user(u_{71}), section(anySection), cntType(spam)\}, prh(upload) \rangle$$

which prohibits a spammer user with identifier u_{71} to upload *spam* to *any* section. Note that IRON is capable of generalising the *"section"* predicate of norms, but it is not capable of generalising their *"user"* predicate. That is, it cannot synthesise a general norm that prohibits *all* the spammers to upload spam to any section. This comes as a consequence of its conservative approach to norm generalisation, which will never generalise the *"user"* predicate of norm's preconditions until it has synthesised a norm to prohibit to upload spam to *each user* of the community. However, moderate users (i.e., users with identifiers $u_1, \dots, u_{70}$) never upload spam, and hence no norms are generated for them.

Note that moderates are the only users that complain about spam contents, and hence the only users that believe that norms to prohibit spam are necessary.

Case	Convergence
Complain power > consensus degree ($\mathsf{C}_{Pw} > \alpha_{nec}$)	Yes
Complain power = consensus degree ($\mathsf{C}_{Pw} = \alpha_{nec}$)	No
Complain power < consensus degree ($\mathsf{C}_{Pw} < \alpha_{nec}$)	No

Table 5.8: Summary of IRON's macro analysis.

Therefore, the proportion of moderates in a population is directly related to the necessity of norms that prohibit spam. As an example, in population 30M-70S, containing 30% moderate users, norms to prohibit spam are around 30% necessary, since three of each ten content views will lead to complaints. Hereafter, we will refer to the proportion of moderates in a population as its *complain power* (C_{Pw}), since moderates represent the proportion of users that complain about spam. As a general rule, IRON converges to a stable normative system whenever the complain power of a population *is above* the necessity threshold considered by IRON to deactivate norms (cf. Section 5.6.2). Namely, it converges when $\mathsf{C}_{Pw} > \alpha_{nec}$. As an example, with population 30M-70S norms that prohibit spam have a necessity value around 30%, and they are preserved only whenever the necessity threshold is $\alpha_{nec} < 0.3$.

Additionally, if the complain power of a population *is equal to* the necessity threshold (that is, $\mathsf{C}_{Pw} = \alpha_{nec}$), IRON cannot converge, since the norms' necessities fluctuate around 0.3, continuously going above and below the necessity threshold. As a consequence, IRON continuously deactivates and re-activates norms, being unable to converge to a stable normative system.

Finally, let us analyse what happens when the complain power of a population *is under* the necessity threshold (that is, $\mathsf{C}_{Pw} < \alpha_{nec}$). Initially, one may expect that IRON should converge to an empty normative system, since norms' necessities in this case will be below the necessity threshold (α_{nec}). However, IRON is not capable to converge to a stable normative system. This comes at a consequence of IRON's norm generation mechanism, which is *highly reactive* to conflicts. Briefly, when IRON creates a new norm, it sets its initial necessity to 0.5, and evaluates it along time. Once the norm's necessity goes below the necessity threshold, IRON deactivates it. Thereafter, the conflict the norm regulated is unregulated again. The next time a user complaints, and the conflict arises again, IRON immediately reacts by creating (re-activating) the norm. Eventually, IRON will evaluate the norm again as unnecessary, thus deactivating it. This leads IRON into a cycle of continuous norm deactivations/re-activations, making IRON incapable to converge. In this scenario, IRON should consider larger amounts of evidences to assess if re-activating a norm is really necessary. Namely, it should be more *deliberative*. Table 5.8 shows a summary of IRON's convergence for different combinations of complaint power/consensus degree.

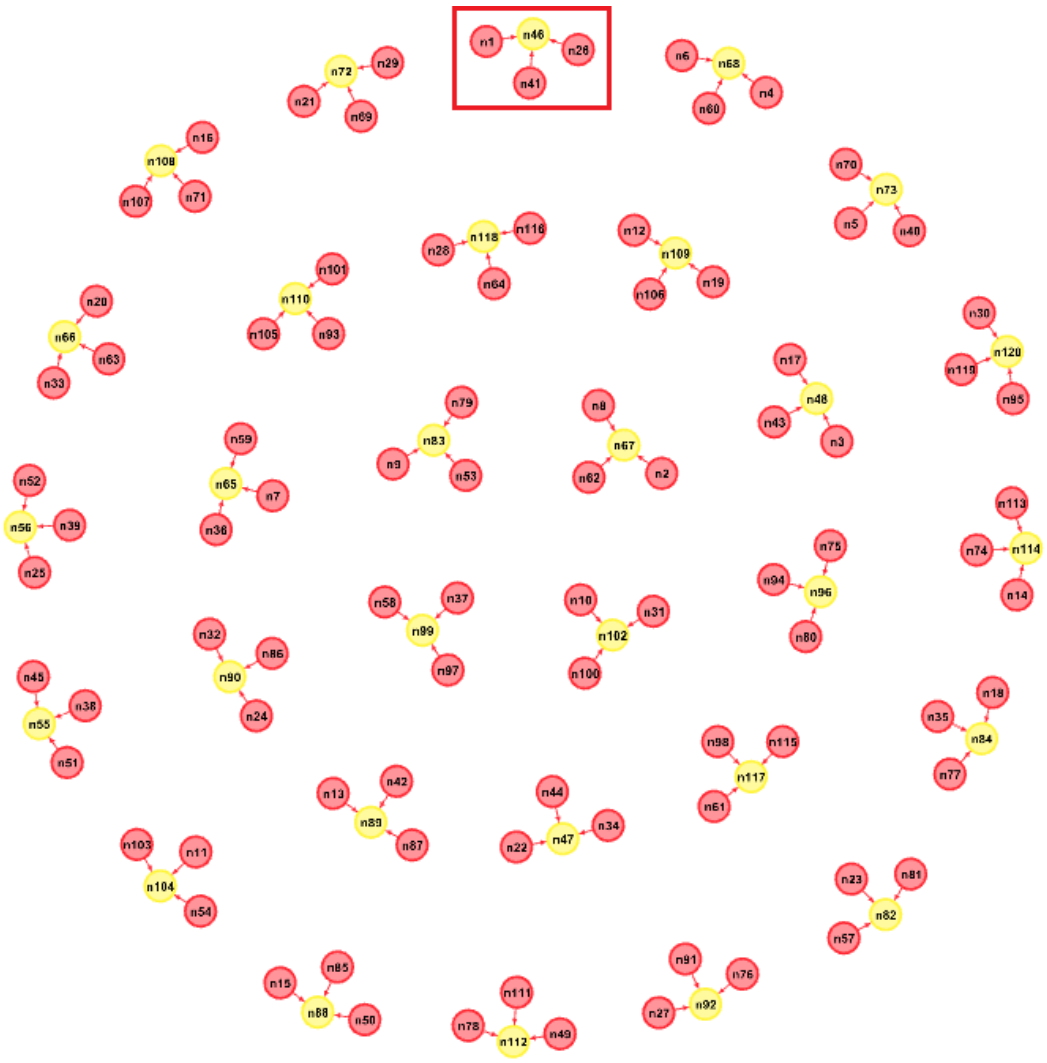

Figure 5.14: Example of a prototypical normative network synthesised by IRON.

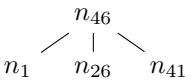

Figure 5.15: An example generalisation of IRON in the on-line community scenario.

Analysis of synthesised normative networks

Finally, the size and the structure of the normative networks synthesised by IRON are analysed. The aim is to understand *how* IRON manages to synthesise compact normative systems. Figure 5.14 depicts a prototypical normative network synthesised by IRON for a population of 70 moderates and 30 spammers. There, each circle represents a norm, and each edge represents a generalisation relationship between two norms. In particular, *yellow* circles represent *general* norms, and *pink* circles represent grounded norms. The depicted network contains thirty general (active) norms that concisely prohibit each one of the thirty spammers to upload spam in any section. In particular, each general norm generalises three norms that prohibit a user to upload spam in the three different sections of the on-line community. Figure 5.15 illustrates an example of the generalisation of norms n_1, n_{26}, n_{41} as norm n_{46}, marked with a red square in Figure 5.14. These norms are described below.

$$
\begin{array}{rcl}
n_1 & : & \langle (user(u_{71}),\ section(forum),\ \ \ \ \ \ cntType(spam)),\ prh(upload)\rangle \\
n_{26} & : & \langle (user(u_{71}),\ section(the\text{-}reporter),\ cntType(spam)),\ prh(upload)\rangle \\
n_{41} & : & \langle (user(u_{71}),\ section(multimedia),\ \ cntType(spam)),\ prh(upload)\rangle \\
n_{46} & : & \langle (user(u_{71}),\ section(anySection),\ \ cntType(spam)),\ prh(upload)\rangle
\end{array}
$$

Note therefore that IRON is capable of generalising only the *section* predicate of norms, hence performing generalisations of three norms to one. As a result, it synthesises normative systems with several norms that prohibit *each spammer* to upload spam to *any* section. However, it is not capable to synthesise a general norm that prohibits *all the users* to upload spam. This happens because IRON requires full evidence to generalise, namely it requires the previous synthesis of a norm to prohibit to upload spam to *each user* in order to generalise the *user* predicate.

5.11 Conclusions

This chapter has focused on answering research question R2 introduced in Section 1.2. For this purpose, it has introduced IRON, a synthesis strategy that extends BASE and considers compactness as a synthesis objective. As BASE, IRON is conflict-driven and is intended to be iteratively executed by a NSM to synthesise normative systems that effectively avoid conflicts within a MAS. IRON incorporates alternative norm evaluation and norm refinement mechanisms to

overcome BASE's stability limitations introduced in Section 3.5. Briefly, IRON computes additional metrics during norm evaluation that allow it to make more informed decisions when refining the normative system.

IRON pursues the synthesis of *compact* normative systems by performing norm generalisations. It structures domain knowledge as a tree term taxonomy, which it employs to synthesise (general) norms representing groups of (more specific) norms. In this way, IRON can reduce the number of norms and norms' predicates and terms in a normative system, hence increasing its compactness. In particular, IRON takes a conservative approach to generalisation that requires full evidence to generalise. That is, it never synthesises a general norm until it has previously synthesised each grounded norm it represents. In this way, IRON's norm generalisations do not increase the number of constraints a normative system imposes on agents.

In order to assess both IRON's performance and the generality of the norm synthesis model introduced in Chapter 3, we have empirically evaluated IRON in two different scenarios. Of these, the first one has been the road traffic scenario described in Section 1.3.1. In this scenario, we have compared IRON's norm synthesis with BASE's in terms of its stability (i.e., their capability to converge to a stable normative system) and the compactness of the normative systems they synthesise. The empirical results demonstrate that IRON significantly outperforms BASE in terms of:

1. *Compactness.* IRON manages to converge to normative systems that are between 30% and 70% more compact (i.e., have between 30% and 70% less norm predicates) than those synthesised by BASE.

2. *Stability.* IRON is *highly stable*, capable of synthesising enduring normative systems that avoid conflicts despite a high number of infringements in the agent society.

Additionally, we have empirically evaluated IRON in the on-line community scenario described in Section 1.3.2. In this case, we have used IRON to build a participatory regulatory mechanism that considers users' complaints to synthesise norms based on their preferences. Initial results demonstrate that IRON can synthesise normative systems aimed at prohibiting the users from uploading those contents that the community considers to be inappropriate. Furthermore, norm generalisations allow IRON to synthesise compact normative systems that reduce the agents' computational efforts when reasoning about norms.

IRON's improvements mainly derive from the fact that it takes more informed and fine-grained decisions than BASE. With this aim, IRON employs: (i) a generalisation function that requires complete evidence prior to perform norm generalisations; and (ii) a specialisation function that allows to perform a fine-grained backtracking of norm generalisations.

However, from IRON's empirical results, we observe that it has one major limitation. In short, its approach to norm generalisation jeopardises the compactness of synthesised normative systems. In the on-line community scenario

(Section 5.10), IRON was capable of generalising norms by their *"section"* predicate. However, it was not capable to generalise their *"user"* predicate. As a result, it was not able to synthesise a general norm prohibiting *all users* to upload spam. This limitation stems from IRON's conservative approach to generalisation, which requires full evidence to generalise.

Against this background, we conclude that there is room for developing alternative strategies that can synthesise normative systems more compact than those synthesised by IRON. With this aim, the next chapter introduces an alternative strategy aimed at outperforming IRON in terms of compactness.

Chapter 6

Synthesising compact normative systems: an optimistic approach

6.1 Introduction

The previous chapter tackled the synthesis of compact normative systems to answer research question R2. With this aim it introduced IRON, a strategy to be executed by a NSM (see Section 3.4) to perform norm synthesis. IRON pursues compactness by performing norm generalisations. In particular, IRON takes a conservative approach to norm generalisation that requires full evidence to generalise. Briefly, it will never synthesise a general norm until it has previously synthesised *all* the norms it represents. In this way, IRON's norm generalisations do not increase the amount of constraints a normative system imposes on agents. However, as discussed in Section 5.11, conservative norm generalisation is not appropriate if it is not possible to gather full evidence of norms. This limitation was illustrated with the empirical evaluation in the on-line community scenario (see Section 5.10). There, IRON synthesised norms to prohibit the users of an on-line community to upload spam contents. Even though IRON managed to perform norm generalisations, it was never capable of synthesising a single, general norm to prohibit *all* the users of a community to upload spam.

Against this background, this chapter introduces SIMON (*SI*mple *M*inimal *O*n-line *N*orm synthesis), a strategy that extends BASE and synthesises compact normative systems via *optimistic* norm generalisations. SIMON incorporates a norm generalisation mechanism that does not require full evidence to generalise norms. Instead, it generalises norms with partial evidence, making the *optimistic* assumption that general norms will perform well. SIMON's generalisation is inspired in the *anti-unification* of terms (in the context of logical reasoning) [Armengol and Plaza, 2000]. Anti-unification consists of generalising feature

terms to their *least common subsumer* or most specific generalisation, generalising pairs of terms to the most specific term that represents both of them. Along these lines, SIMON generalises norms by (1) detecting *pairs* of norms that are *generalisable*, and (2) generalising them to their most specific generalisation. In this way, SIMON is capable of performing further norm generalisations than IRON, hence yielding normative systems more compact than those synthesised by IRON. Moreover, SIMON employs IRON's norm evaluation mechanism, as well as its approach to detect norms' good-performance and under-performance. In this way, SIMON is capable of not only outperforming IRON in terms of compactness, but also preserving its stability.

The remainder of this chapter is thus organised as follows. Section 6.2 describes SIMON's information model and introduces some preliminary definitions that will be essential to understand optimistic norm generalisation. Thereafter, Section 6.3 details how SIMON performs optimistic norm generalisations, and Section 6.4 explains how it backtracks over-generalisations. Next, Section 6.5 introduces the SIMON strategy, which details how it operates to produce compact normative systems for a MAS via optimistic norm generalisations. Finally, Sections 6.6 and 6.7 show an empirical evaluation of SIMON in the two scenarios introduced in Section 1.3: the road traffic scenario, and the on-line community scenario.

6.2 Information model and basic definitions

To represent synthesised norms along with their generalisation relationships, SIMON employs the normative network described in Section 5.3.1. Moreover, SIMON represents a normative system as the set of norms that are active in the normative network. To apply changes to the normative network, SIMON incorporates the normative network operators employed by described in Table 3.1 (Section 5.3.2) and Table 5.1 (Section 5.3.2). These operators allow SIMON to: (i) *add* norms to the normative network; (ii) *activate* norms so that they become part of the normative system; (iii) *deactivate* so that they no longer belong to the normative system; (iv) *generalise* several norms to a parent norm, resulting in a more compact representation of the normative system; and (iv) *specialise* norms that do not perform well, as a method to backtrack norm generalisations.

SIMON considers the formal model described in Section 5.2. Let us recall some basic concepts to help understand the remainder of this section. SIMON considers a MAS with a set of agents Ag that can perform a set of actions Ac. The representation language used by agents is denoted by $\mathcal{L}_{Ag}$. Norms are of the form $\langle \varphi, \theta(ac) \rangle$, where φ is the precondition of the norm, θ is a deontic operator (e.g., a *prohibition*) and ac is an action available to the agents. The precondition φ of a norm is a set of first-order predicates $p(\tau_1, \ldots, \tau_k)$, where p is a predicate symbol and $\tau_1, \ldots, \tau_k$ are terms of the language $\mathcal{L}_{Ag}$. Let us denote by $\bar{\boldsymbol{\tau}}$ a vector of terms in $\mathcal{T}$. We refer to the i-th component of $\bar{\boldsymbol{\tau}}$ as τ_i. Moreover, the set of terms in language $\mathcal{L}_{Ag}$ is denoted by $\mathcal{T}$, and the taxonomy between the terms in $\mathcal{T}$ as T. Let us consider a subsumption relationship $\sqsubseteq$ that establishes

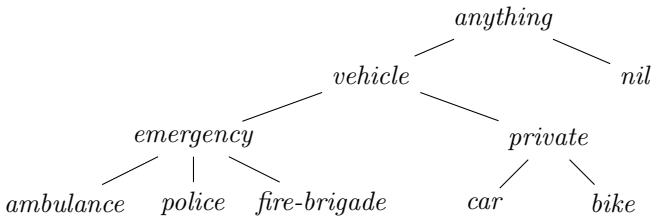

Figure 6.1: A taxonomy capturing generalisation relationships between terms

generalisation (subsumption) relationships between the terms of a taxonomy. In particular, a term τ *strictly* subsumes another term τ', denoted as $\tau' \sqsubset \tau$, iff $\tau' \sqsubseteq \tau$ and $\tau' \neq \tau$.

Hereafter, some new, basic definitions are provided. Considering that the taxonomy of terms has a *tree* structure, the *intersection* between two terms is the most specific term subsumed by these two terms. Let us illustrate an example with the road traffic example, and the taxonomy of terms introduced in Section 5.2. Figure 6.1 depicts this taxonomy. According to it, the intersection between terms *"ambulance"* and *"emergency"* is the *"ambulance"* term. The general formal definition is:

Definition 27 (Terms intersection)**.** *For $\tau, \tau' \in \mathcal{T}$, their intersection $\tau \sqcap_t \tau'$ is:*

$$\tau \sqcap_t \tau' = \begin{cases} \tau & \text{if } \tau \sqsubseteq \tau' \\ \tau' & \text{if } \tau' \sqsubseteq \tau \\ \emptyset & \text{otherwise} \end{cases}$$

Analogously, the intersection between two predicates $p(\bar{\boldsymbol{\tau}}), p(\bar{\boldsymbol{\tau}}') \in \mathcal{L}_{Ag}$ is another predicate with the intersection of each corresponding pair of terms in $\bar{\boldsymbol{\tau}}, \bar{\boldsymbol{\tau}}'$, whenever such intersection exists for all of them.

Definition 28 (Predicates intersection)**.** *For $p(\bar{\boldsymbol{\tau}}), p(\bar{\boldsymbol{\tau}}') \in \mathcal{L}_{Ag}$, if $\tau_i \sqcap_t \tau_i' \neq \emptyset, 1 \leq i \leq n$, then their intersection $p(\bar{\boldsymbol{\tau}}) \sqcap_\pi p(\bar{\boldsymbol{\tau}}')$ is $p(\bar{\boldsymbol{\tau}}'')$ such that $\tau_i'' = \tau_i \sqcap_t \tau_i'$ for all $1 \leq i \leq n$.*

As an example, the intersection of *left*(*ambulance*) and *left*(*emergency*) is *left*(*ambulance*). Formally, *left(ambulance)* $\sqcap_t$ *left(emergency)* = *left(ambulance)*.

Taking inspiration from the anti-unification of terms proposed in [Armengol and Plaza, 2000], let us define the *most specific generalisation* of two terms. Given terms $\tau, \tau' \in \mathcal{T}, \tau \neq \tau'$, their most specific generalisation is the most specific term that *strictly* subsumes both of them. For instance in Figure 6.1, the most specific generalisation of terms *"ambulance"* and *"car"* is term *"vehicle"*, since there is no other term which is more specific and strictly subsumes both of them. However, the most specific generalisation for *"ambulance"* and *"anything"* does not exist because there is no term strictly subsuming *"anything"*. Formally:

Definition 29 (Most specific generalisation of two terms)**.** *For $\tau, \tau' \in \mathcal{T}, \tau \neq \tau'$, their* most specific generalisation*, denoted as $\tau \sqcup_t \tau'$, is a term $\tau_s \in \mathcal{T}$ such that $\tau \sqsubset \tau_s$ and $\tau' \sqsubset \tau_s$, and $\nexists \tau'' \in \mathcal{T}$ such that $\tau \sqsubset \tau'', \tau' \sqsubset \tau''$ and $\tau'' \sqsubset \tau_s$.*

Analogously, the most specific generalisation of two predicates is another predicate containing the most specific generalisation of their terms.

Definition 30 (Most specific generalisation of two predicates)**.** *Predicates $p(\bar{\boldsymbol{\tau}}), p(\bar{\boldsymbol{\tau}}') \in \mathcal{L}_{Ag}$ have a* generalisation *iff $\exists \tau \in \bar{\tau}, \tau' \in \bar{\tau}'$ such that $\tau \sqcup_t \tau' \neq \emptyset$. Their most specific generalisation, denoted as $p(\bar{\boldsymbol{\tau}}) \sqcup_\pi p(\bar{\boldsymbol{\tau}}')$, is another predicate $p(\bar{\boldsymbol{\tau}}'')$ such that $\tau'' = \tau \sqcup_t \tau'$.*

In the running example, the most specific generalisation of predicates *left(ambulance)* and *left(car)* is predicate *left(vehicle)*, with the most specific generalisation of terms *"ambulance"* and *"car"*.

6.3 Optimistic norm generalisation

Next, SIMON's optimistic approach to norm generalisation is introduced. Briefly, SIMON's generalisation consists of three phases:

1. *Monitoring* when the norms of the normative system start *performing well*. SIMON considers that a norm starts performing well at a given time as described in Section 5.6.1. That is, whenever the *lower boundaries* of its effectiveness *and* necessity ranges overpass certain generalisation thresholds.

2. *Checking* if the identified norms are *generalisable* with the rest of the norms of the normative system.

3. *Generalising each pair* of norms that are generalisable. It generalises a pair of norms by retrieving their most specific general (parent) norm that represents them, and generalising these two norms into such parent.

We say that two norms are generalisable if *at least one* of their predicates have a most specific generalisation, and their remaining predicates are equal. Formally:

Definition 31 (Generalisable norms)**.** *Two norms $n = \langle \varphi, \theta(ac) \rangle$ and $n' = \langle \varphi', \theta(ac) \rangle$, $n \neq n'$, are generalisable iff for each predicate $p(\bar{\boldsymbol{\tau}}) \in \varphi$ either: (i) $p(\bar{\boldsymbol{\tau}}) \in \varphi'$; or (ii) there is a predicate $p(\bar{\boldsymbol{\tau}}') \in \varphi', p(\bar{\boldsymbol{\tau}}) \sqcup_\pi p(\bar{\boldsymbol{\tau}}') \neq \emptyset$.*

Note therefore that SIMON performs *pairwise* norm generalisations. Let us illustrate an example with norms $n_1, \ldots, n_4$ introduced in Section 5.2, which we recall here:

$$
\begin{array}{lcl}
n_1 & : & \langle \{left(ambulance), \ front(car), \ right(car)\}, \ prh(go) \rangle \\
n_2 & : & \langle \{left(police), \ front(car), \ right(car)\}, \ prh(go) \rangle \\
n_3 & : & \langle \{left(fire\text{-}brigade), \ front(car), \ right(car)\}, \ prh(go) \rangle \\
n_4 & : & \langle \{left(emergency), \ front(car), \ right(car)\}, \ prh(go) \rangle
\end{array}
$$

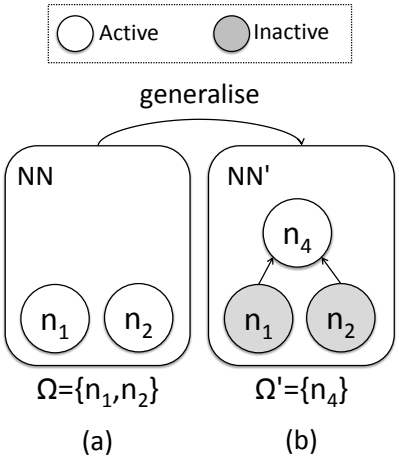

Figure 6.2: Direct generalisation of norms n_1, n_2 to n_4.

Notice that n_1, n_2 satisfy the conditions of Definition 31: following the relationships between terms in Figure 6.1, we see that predicate $\mathit{left}(\mathit{ambulance}) \in \varphi_1$ (being φ_i the precondition of norm n_i) has a corresponding predicate $\mathit{left}(\mathit{police}) \in \varphi_2$ such that their most specific generalisation (cf. Definition 30) is $\mathit{left}(\mathit{ambulance}) \sqcup_\pi \mathit{left}(\mathit{police}) = \mathit{left}(\mathit{emergency}$. Furthermore, predicate $\mathit{front}(\mathit{car}) \in \varphi_1$ is equal to $\mathit{front}(\mathit{car}) \in \varphi_2$, and analogously for $\mathit{right}(\mathit{car}) \in \varphi_1$ and $\mathit{right}(\mathit{car}) \in \varphi_2$. Hence, n_1, n_2 are generalisable to a new norm that can be computed as follows:

Definition 32 (Norm generalisation)**.** *The generalisation of two norms* $n = \langle \varphi, \theta(ac) \rangle, n' = \langle \varphi', \theta(ac) \rangle$ *is a norm* $n'' = \langle \varphi'', \theta(ac) \rangle$ *such that, for each predicate* $p(\bar{\boldsymbol{\tau}}) \in \varphi$ *and* $p(\bar{\boldsymbol{\tau}}') \in \varphi'$*, there is a predicate* $p(\bar{\boldsymbol{\tau}}'') \in \varphi''$ *obtained as:*

$$p(\bar{\boldsymbol{\tau}}'') = \begin{cases} p(\bar{\boldsymbol{\tau}}) & \text{if } \tau_i = \tau_i', \forall i \in [1..m] \\ p(\bar{\boldsymbol{\tau}}) \sqcup_\pi p(\bar{\boldsymbol{\tau}}') & \text{otherwise} \end{cases}$$

As a result, norms n_1, n_2 can be generalised as norm n_4, which contains (i) predicate $\mathit{left}(\mathit{emergency})$, as the most specific generalisation of predicates $\mathit{left}(\mathit{ambulance}) \in \varphi_1$ and $\mathit{left}(\mathit{police}) \in \varphi_2$; and (ii) predicates $\mathit{front}(\mathit{car}), \mathit{right}(\mathit{car})$ of φ_1 and φ_2. Figure 6.2 illustrates this generalisation. It depicts a normative network *NN* (Figure 6.2a) that contains two active norms n_1, n_2 that are generalisable. Thus, SIMON computes norm n_4 as their most specific generalisation, and generalises n_1, n_2 to n_4 (NN' in Figure 6.2b). As a result, the cardinality of the normative system is reduced from two norms ($\Omega = \{n_1, n_2\}$) to one ($\Omega' = \{n_4\}$).

Hereafter we will refer to this generalisation as *direct*, or *shallow*, generalisation. Its procedure is described in function *shallowNormGeneralisation* (Algorithm 13). It receives as inputs (i) a norm n; (ii) a normative network *NN*; and (iii) a taxonomy of terms T. Additionally, it receives a generalisation step

G_S (larger than 1) that states the number of terms that can be simultaneously generalised. Notice that, from definition 32, a generalisation can involve from 1 term up to the total number of term in the precondition of a norm. Thus, a generalisation can take up to at most G_S terms at the same time. In the example above, norm n_4 is a generalisation of 1 term (that is, $\mathsf{G}_S = 1$). However, setting G_S to 2 would mean that a generalisation can involve both 1 and 2 terms.

Function *shallowGeneralisation* proceeds by performing pairwise comparisons of n with each norm n' in the normative system, checking if they are generalisable (lines 1–3). For each pair of generalisable norms n, n', it computes their most specific generalisation (c.f. Definition 32) as norm n_π (line 4). Thereafter, it checks that n_π does not represent any norm that under-performs (line 5). In this way, SIMON ensures that norm generalisations do not introduce in the normative system specific norms that do not perform well in regulating conflicts. If that condition is satisfied, then it generalises norms n, n' as n_π. With this aim, it first adds the new parent norm n_π to the normative network if it does not exist (line 6), and then invokes the *generalise* operator (see Table 5.1) to activate norm n'' and to deactivate norms n, n' (line 7).

ALGORITHM 13: Function *shallowNormGeneralisation*

Input : $n, NN, \mathsf{T}, \mathsf{G}_S$
Output: NN

```
1 Ω ← {n ∈ NN | δ(n) = active};
2 foreach n' ∈ Ω do
3     if areGeneralisable(n, n', T, G_S) then
4         n_π ← mostSpecificGeneralisation(n, n', T, G_S);
5         if not representsUnderperformingNorms(n_π, NN) then
6             NN ← add(n_π, NN);
7             NN ← generalise({n, n'}, n_π, NN);
8 return NN
```

Note that SIMON generalises norms with partial evidence. As an example, SIMON can synthesise n_4 by generalising n_1, n_2, even though n_3 has never been synthesised and no evidence has been gathered about its performance. However, this generalisation increases the number of grounded norms that a normative system represents, namely norms with grounded terms (see Definition 21 in Section 5.2). After this norm generalisation, n_4 *implicitly* represents n_3, and thus includes it in the normative system, despite it has no guarantees that it is effective or necessary to avoid conflicts. By contrast, as detailed in Section 5.4, IRON requires *full evidence* to generalise norms. It would generalise n_1, n_2 to n_4 only if n_3 has been previously created, added to the normative network, and is active. Therefore, we say that SIMON takes an *optimistic* approach to generalisation that requires less evidences than IRON to synthesise compact normative systems.

Moreover, it is also worth noticing that SIMON's norm generalisation only takes into account those norms in the normative system (that is, those that

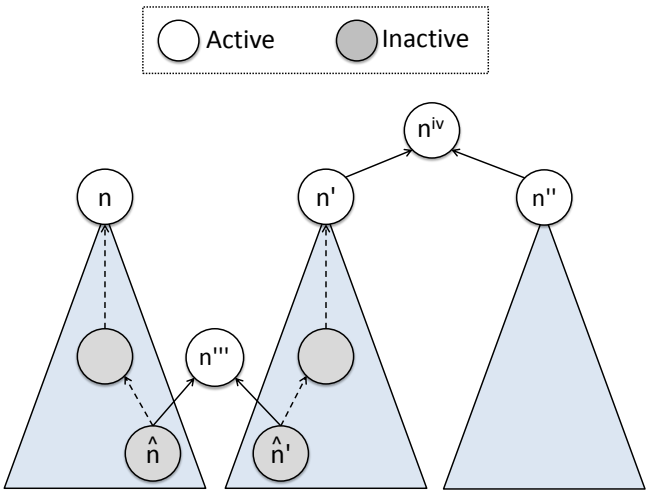

Figure 6.3: Indirect generalisation of norms n and n' and direct generalisation of norms n' and n''.

are active in the normative network), whereas IRON generalises norms taking into account all norms in the normative network. Nevertheless, as Figure 6.3 shows, two norms in the normative system implicitly represent other norms in the normative network which in turn may be generalisable. Specifically, two norms n, n' in the normative system may not be directly generalisable, but they may include two other norms $\hat{n} \subseteq n$, $\hat{n}' \subseteq n'$ that will be generalisable to a norm n''' if the conditions of Theorem 1 hold:

Theorem 1. *Let $n = \langle \varphi, \theta(ac) \rangle, n' = \langle \varphi', \theta(ac) \rangle, n \neq n'$ be two norms satisfying the following conditions:*

1. *for each predicate $p(\bar{\boldsymbol{\tau}}) \in \varphi$ there is a predicate $p(\bar{\boldsymbol{\tau}}') \in \varphi'$ such that either $p(\bar{\boldsymbol{\tau}}) \sqcup_\pi p(\bar{\boldsymbol{\tau}}') \neq \emptyset$ or $p(\bar{\boldsymbol{\tau}}) \sqcap_\pi p(\bar{\boldsymbol{\tau}}') \neq \emptyset$; and*
2. *there is at least one pair of predicates $p(\bar{\boldsymbol{\tau}}) \in \varphi$ and $p(\bar{\boldsymbol{\tau}}') \in \varphi'$ such that $p(\bar{\boldsymbol{\tau}}) \sqcap_\pi p(\bar{\boldsymbol{\tau}}') = \emptyset$ and $p(\bar{\boldsymbol{\tau}}) \sqcup_\pi p(\bar{\boldsymbol{\tau}}') \neq \emptyset$.*

Then there exist two norms $\hat{n} \subseteq n$, $\hat{n}' \subseteq n'$ that are generalisable.

Proof. The proof proceeds by constructing norms $\hat{n}$, $\hat{n}'$ from n and n'. We define $\hat{n}$, $\hat{n}'$ as the intersection of each pair of predicates $p(\bar{\boldsymbol{\tau}}) \in \varphi$ and $p(\bar{\boldsymbol{\tau}}') \in \varphi'$ as follows.

$$\hat{n} = \begin{cases} p(\bar{\boldsymbol{\tau}}) \sqcap_\pi p(\bar{\boldsymbol{\tau}}') & \text{if } p(\bar{\boldsymbol{\tau}}) \sqcap_\pi p(\bar{\boldsymbol{\tau}}') \neq \emptyset \\ p(\bar{\boldsymbol{\tau}}) & \text{otherwise} \end{cases}$$

$$\hat{n}' = \begin{cases} p(\bar{\boldsymbol{\tau}}) \sqcap_\pi p(\bar{\boldsymbol{\tau}}') & \text{if } p(\bar{\boldsymbol{\tau}}) \sqcap_\pi p(\bar{\boldsymbol{\tau}}') \neq \emptyset \\ p(\bar{\boldsymbol{\tau}}') & \text{otherwise} \end{cases}$$

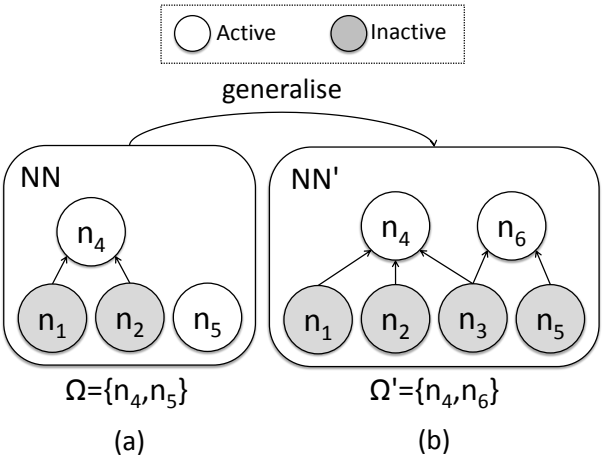

Figure 6.4: Indirect generalisation of norms n_3, n_5 to n_6.

Note that the method used to construct $\hat{n}, \hat{n}'$ guarantees that they share the predicates whose intersection is not empty. Therefore, some of the predicates in both norms will be equal. When the intersection between two predicates is empty, we know from the assumptions in the theorem that there is a most specific generalisation for both predicates. In that case, we just copy the predicate of n into $\hat{n}$, and the predicate of n' into $\hat{n}'$. Therefore, by construction we ensure that $\hat{n}, \hat{n}'$ are generalisable. That is, for each pair or predicates in $\hat{n}, \hat{n}'$ either they are equal or there is a most specific generalisation for them. □

As an example, consider the normative network in Figure 6.4a, norms $n_1, \ldots, n_4$ described above, and the following norms:

$$\begin{aligned} n_5 &: \langle \{\mathit{left}(\mathit{fire\text{-}brigade}),\ \mathit{front}(\mathit{car}),\ \mathit{right}(\mathit{bike})\},\ \ \mathit{prh}(\mathit{go})\rangle \\ n_6 &: \langle \{\mathit{left}(\mathit{fire\text{-}brigade}),\ \mathit{front}(\mathit{car}),\ \mathit{right}(\mathit{private})\},\ \mathit{prh}(\mathit{go})\rangle \end{aligned}$$

Initially, (Figure 6.4a) the normative network contains two active norms n_4, n_5 that have an intersection in predicates their *"left"* and *"front"* predicates, and a generalisation in their *"right"* predicates. Formally, (i) $\mathit{left}(\mathit{emergency}) \in \varphi_4 \sqcap_\pi \mathit{left}(\mathit{fire\text{-}brigade}) \in \varphi_5 = \mathit{left}(\mathit{fire\text{-}brigade})$; (ii) $\mathit{front}(\mathit{car}) \in \varphi_4 \sqcap_\pi \mathit{front}(\mathit{car}) \in \varphi_5 = \mathit{front}(\mathit{car})$; and (iii) $\mathit{right}(\mathit{car}) \in \varphi_4 \sqcup_\pi \mathit{right}(\mathit{bike}) \in \varphi_5 = \mathit{right}(\mathit{private})$. Then, SIMON can compute two norms $\hat{n} \subseteq n_4$, $\hat{n}' \subseteq n_5$ that are generalisable along the lines of the proof of Theorem 1. In this particular example, $\hat{n} = n_3$ and $\hat{n}' = n_5$. Norm n_3 contains (i) predicate $\mathit{left}(\mathit{fire\text{-}brigade})$, namely the intersection of $\mathit{left}(\mathit{emergency}) \in \varphi_4$ and $\mathit{left}(\mathit{emergency}) \in \varphi_5$; (ii) predicate $\mathit{front}(\mathit{car}) \in \varphi_4$; and (ii) predicate $\mathit{right}(\mathit{car}) \in \varphi_4$. Then, in Figure 6.4b (normative network NN'), norms n_3, n_5 are generalised to a norm n_6, which contains (i) the intersection ($\mathit{left}(\mathit{fire\text{-}brigade})$ of predicates $\mathit{left}(\mathit{emergency}) \in \varphi_4$ and $\mathit{left}(\mathit{emergency}) \in \varphi_5$; the intersection ($\mathit{front}(\mathit{car})$ of predicates $\mathit{front}(\mathit{car}) \in$

φ_4 and $front(car) \in \varphi_5$; and (ii) predicate $right(private)$ as the most specific generalisation of predicates $right(car) \in \varphi_4$ and $right(bike) \in \varphi_5$.

Hereafter, we will refer to this generalisation as *indirect* or *deep* generalisation, whose procedure is described in function *deepNormGeneralisation* (Algorithm 14). It receives as inputs: (1) a norm n to generalise; (2) a normative network NN; (3) a terms taxonomy T; and (4) a generalisation step G_S, which sets the number of terms that SIMON can simultaneously generalise. It performs pairwise comparisons of norm n with each norm n' in the normative system. For each pair of norms n, n', it checks if they represent two norms $\hat{n} \subset n, \hat{n}' \subset n'$ that are generalisable. With this aim, it employs function *getRepresentedGeneralisableNorms* (line 3), which retrieves these norms as described in Theorem 1. If norms $\hat{n} \subset n, \hat{n}' \subset n'$ exist, then it then computes the most specific generalisation of $\hat{n}, \hat{n}'$ (line 5) and generalises them if the general norm does not represent any norm of the normative network that under-performs (lines 6–8).

ALGORITHM 14: Function *deepNormGeneralisation*

Input : $n, NN, \mathsf{T}, \mathsf{G}_S$
Output: NN

```
1 Ω ← {n ∈ NN | δ(n) = active};
2 foreach n' ∈ Ω do
3     (n̂, n̂') ← getRepresentedGeneralisableNorms(n, n', T, G_S);
4     if n̂ ≠ null and n̂' ≠ null then
5         n_π ← mostSpecificGeneralisation(n, n', T, G_S);
6         if not representsUnderperformingNorms(n_π, NN) then
7             NN ← add({n̂, n̂', n_π}, NN);
8             NN ← generalise({n̂, n̂'}, n_π, NN);
9 return NN
```

6.4 Revising over-generalisations

At this point, we know that optimistic norm generalisations allow SIMON to generalise norms with partial evidence. Nevertheless, optimistic norm generalisations may lead to *over-generalisations*, since general norms may implicitly represent (and hence implicitly include in the normative system) norms that may under-perform. For instance, let us suppose that n_4 in Figure 6.2 does not perform well whenever agents fulfil it in the situation described by norm n_3. In that case, SIMON has to specialise n_4, activating n_1, n_2 and deactivating n_3. However, norm n_3 is not explicitly represented in the normative network, and hence it cannot be deactivated. Therefore, SIMON must previously create norm n_3 in order to keep track of it and finally deactivate it. With this aim, in addition to norm specialisation, SIMON incorporates a method for refining norm generalisations. It specifies a general norm n that under-performs in the

situation described by a specific norm $n_{sp} \subset n$ that has not been created yet. This *norm specification* process proceeds as follows: (1) it detects when a general norm n under-performs in the situation described by a specific norm n_{sp} it implicitly represents; (2) it *specifies* norm n_{sp} by creating it, and adding it to the normative network; (3) it establishes a generalisation relationship from n_{sp} to n; and (4) it searches for alternative generalisation relationships that n_{sp} may have with other norms in the normative network. We regard this process as a revision of generalisations, with the intention to *backtrack* the search for normative systems to a less general set of norms.

ALGORITHM 15: Function *buildGeneralisations*

Input : n, n', NN, T
Output: NN

```
1  linked ← False
2  nChildren ← getChildren(n, NN)
3  n'Children ← getChildren(n', NN)
4  if n = n' then
5      return NN
6  if subsumes(n, n', T) then                          /* If n subsumes n' and n */
7      if not isAncestor(n, n', NN) then               /* is not an ancestor of n', */
8          generalise(n', n, NN)                       /* set n as a parent of n' */
9          removeGeneralisation(n'Children, n, NN)
10         linked ← True
11 else if subsumes(n', n, T) then                     /* If n is subsumed by n' and */
12     if not subsumes(n'Children, n, T) then          /* no child of n' subsumes n, */
13         if not isAncestor(n, n', NN) then           /* then set n as a child of n' */
14             generalise(n, n', NN)
15             linked ← True
16         n'ChildrenSubsumedByN ← subsumed(n'Children, n)        /* If n */
17         foreach n'_ch ∈ n'ChildrenSubsumedByN do    /* subsumes any child */
18             removeGeneralisation(n'_ch, n', NN)     /* of n, insert n between */
19             generalise(n'_ch, n, NN)                /* n' and that particular child */
20 if not linked then                                  /* If n was not linked to n', then */
21     foreach n'_ch ∈ n'Children do                   /* keep on trying with n' children */
22         link(n, n'_ch, NN, T)
23 else                                                 /* Even if n was linked */
24     n'ChildrenNotSubsumedByN ← notSubsumed(n'Children, n)   /* keep on */
25     foreach n'_ch ∈ n'ChildrenNotSubsumedByN do     /* searching in those */
26         link(n, n'_ch, NN, T)                       /* children of n' that */
                                                       /* n does not subsume */
27 return NN
```

This last step is an important stage in the refinement of norms, since it preserves the consistency of the normative network. It concerns search-

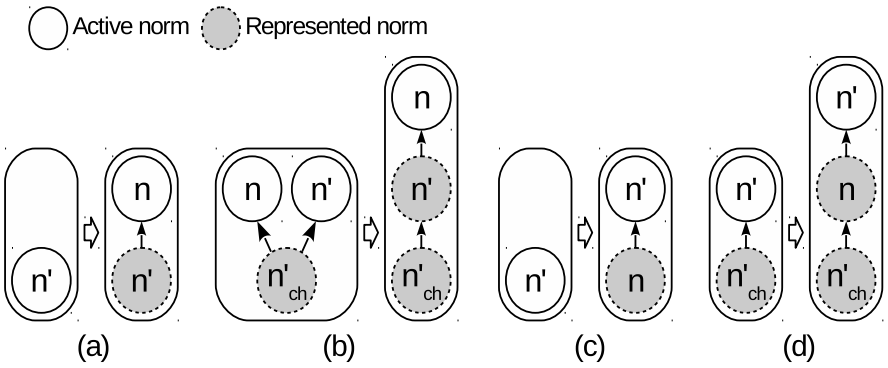

Figure 6.5: Different situations in which SIMON may link norm n to a norm n'

ing for alternative generalisation relationships a specified norm n_{sp} may have with other norms in the normative network. Algorithm 15 describes function *buildGeneralisations*, which performs this step. It receives as input a normative network NN, a taxonomy of terms T, and two norms n, n' to check if they may have generalisation relationships. Then, it checks the following conditions, and performs specific operations when they are satisfied:

- **Norm n directly generalises n'** (Figure 6.5a). SIMON checks if n subsumes (represents) n' with respect to taxonomy T, and n is not already an ancestor of n' (lines 6–7). If these conditions are fulfilled, then n is established as a parent of n' (line 8). depicts an example of this situation.

- **Norm n' is *between* n and one of its children in the hierarchy** (Figure 6.5b). Next, SIMON checks if any of the children n'_{ch} of n' is also a child of n. In that case, it removes the generalisation from n'_{ch} to n', so that the network remains consistent (line 9).

- **Norm n is generalised by norm n'** (Figure 6.5c). In case n cannot be a parent of n', it checks if it can be a child of n'. If n' subsumes n, no child of n' subsumes n, and n is not an ancestor of n', it establishes a generalisation relationship from n' to n (lines 11–14).

- **Norm n is *between* n' and one of its children in the hierarchy** (Figure 6.5d). SIMON checks if n subsumes any child n'_{ch} of n'. If that is the case, it *inserts* n between n' and its child n'_{ch} (lines 16–19).

Finally, SIMON keeps on searching generalisation relationships between n and each child n'_{ch} of n', if whether (i) n has not been linked yet; or (ii) n has been linked, but n does not subsume the child n'_{ch}.

6.5 SIMON's synthesis strategy

We have so far described how SIMON manages to perform optimistic norm generalisations, and to revise/backtrack over-generalisations. Now, SIMON's synthesis strategy is introduced. SIMON is intended to be iteratively executed by a NSM to perform norm synthesis (see Section 3.4). With this aim, SIMON operates by perceiving a MAS, and subsequently carrying out norm generation, norm evaluation, and norm refinement. In particular, SIMON generates new norms as described in Section 4.6.1, and it evaluates norms as described in Section 5.7.2. Crucially, the norm refinement phase is novel. During norm refinement, it specifies under-performing grounded norms that are implicitly represented by general norms. It performs this operation as described in Section 6.4. Thereafter, it generalises norms by performing optimistic norm generalisations as described in Section 6.3, and backtracks norm generalisations by specialising under-performing norms as described in Section 6.4.

Algorithm 16 illustrates SIMON's strategy, which takes as input a tuple with a description of the previous state of a MAS (d_s) and a description of the current MAS state (d_{s_c}), and outputs a normative system to regulate the agents' behaviour. To perform norm synthesis, SIMON considers the globally accessible elements described at the beginning of Section 5.7. That is, a normative network, a collection of operators, a set of domain-dependent elements, a set of domain-independent settings, and a set of additional synthesis inputs. Moreover, SIMON considers the following additional inputs:

1. SIMON's generalisation mode ($\mathsf{G}_M \in \{shallow, deep\}$).
2. SIMON's generalisation step (G_S), introduced in Section 6.3.

ALGORITHM 16: SIMON's synthesis strategy

Input : $\langle d_s, d_{s_c} \rangle$
Output : Ω
Initialisations: $\mathcal{NCO} \leftarrow \emptyset$, $\mathcal{P} \leftarrow \emptyset, \mathcal{ER} \leftarrow \emptyset, \mathcal{NR} \leftarrow \emptyset$

[1] $NN \leftarrow$ `normGeneration(`$\langle d_s, d_{s_c} \rangle, \mathcal{P}$`);`
[2] $(\mathcal{P}, \mathcal{ER}, \mathcal{NR}) \leftarrow$ `normEvaluation(`$\langle d_s, d_{s_c} \rangle, \mathcal{NCO}, \mathcal{P}, \mathcal{ER}, \mathcal{NR}$`);`
[3] $NN \leftarrow$ `normRefinement(`$\mathcal{NCO}, \mathcal{P}, \mathcal{ER}, \mathcal{NR}$`);`
[4] $\Omega \leftarrow \{n \in NN \mid \delta(n) = active\}$;
[5] **return** Ω

Additionally, SIMON considers the data structures described in Section 5.7 to keep track of norms' compliance outcomes ($\mathcal{NCO}$), norms' effectiveness and necessities ($\mathcal{P}$), and norms' effectiveness and necessity ranges ($\mathcal{ER}, \mathcal{NR}$).

Each time SIMON is invoked, it starts by performing norm generation (line 1). Briefly, it detects unregulated conflicts within the current MAS state (s_c), and creates norms to avoid detected conflicts. Thereafter, it carries out norm evaluation (line 2), in which it evaluates norms in terms of their effectiveness and necessity, and computes their effectiveness and necessity ranges ($\mathcal{ER}, \mathcal{NR}$). Finally,

SIMON carries out norm refinement (line 3), in which: (1) it *specifies* grounded norms that are implicitly represented by general norms and under-perform; (2) it *specialises* norms that under-perform in terms of their effectiveness or their necessity; and (3) *generalises* norms when possible.

ALGORITHM 17: SIMON's *normRefinement* function.

Input : $\mathcal{NCO}, \mathcal{P}, \mathcal{ER}, \mathcal{NR}$
Output: NN

```
1  Ω ← {n ∈ NN | δ(n) = active};
2  NRG ← getNegativelyRewardedGeneralNorms(P);
3  foreach n_g ∈ NRG do
4      n_sp ← retrieveSpecificUnderperformingNorm(n_g, NN);
5      NN ← add(n_sp, NN);
6      NN ← generalise({n_sp}, n_g, NN);
7      foreach n' ∈ Ω do
8          buildGeneralisations(NN, n_sp, n');
9  for n ∈ getNormsFulfilledInfringedThisState(NCO) do
10     if underPerforms(n, ER, NR, α_eff, α_nec) then
11         NN ← deactivateUp(n, NN);
12     else if performsWell(n, ER, NR, α^gen_eff, α^gen_nec) then
13         NN ← generaliseUp(n, NN, T, G_M, G_S);
14 return NN
```

Algorithm 17 illustrates SIMON's *normRefinement* function, which operates as follows. First, it retrieves a set *NRG* of *negatively rewarded general norms* (line 2), namely those general norms that have been evaluated as ineffective or unnecessary during the current state transition. Next, for each general (under-performing) norm (n_g), it retrieves the specific norm (n_{sp}) in whose context n_g has been negatively rewarded (line 4). Thereafter, it adds the specific norm (n_{sp}) to the normative network, and establishes a generalisation relationship between n_{sp} and n_g (lines 5–6). Then, it searches for alternative generalisation relationships that may hold between n_{sp} and other norms in the normative network. With this aim, it invokes function *buildGeneralisations* to try to link n_{sp} to other norms in the normative system (lines 7–8).

The next step is the specialisation and generalisation of norms. First, SIMON checks if any of the norms fulfilled or infringed during the current time step under-performs as described in Section 5.6.2. In that case, it deactivates the under-performing norm, propagating up its deactivation to its ancestors in the normative network (lines 11–12). With this aim, it invokes function *deactivateUp*, described in Algorithm 12 from Section 5.7.3. Otherwise, it checks if each norm performs well enough to be generalised as detailed in Section 5.6.1. In that case, it tries to generalise up the norm (lines 12–13) by invoking function *generaliseUp*, whose procedure is depicted in Algorithm 18. In particular, such function generalises a norm n as follows. If SIMON is configured to perform shal-

low norm generalisations, then it invokes function *shallowNormGeneralisation* (line 2), whose procedure is depicted in Algorithm 13. Otherwise, it invokes function *deepNormGeneralisation*, whose procedure is described in Algorithm 14.

ALGORITHM 18: Function *generaliseUp*

Input : $n, NN, \mathsf{T}, \mathsf{G}_M, \mathsf{G}_S$
Output: NN

```
1 if isShallow(G_M) then
2     NN ← shallowNormGeneralisation(n, NN, T, G_S);
3 else if isDeep(G_M) then
4     NN ← deepNormGeneralisation(n, NN, T, G_S);
5 return NN
```

6.6 Empirical evaluation considering a road traffic scenario

Next, the performance of SIMON is compared with IRON's along several dimensions. With this aim, both strategies are empirically evaluated in the road traffic scenario described in Section 1.3.1. First, they are compared in terms of the quality (of compactness, effectiveness and necessity) of the normative systems they synthesise, as well as the convergence time required by their synthesis. Thereafter, a micro analysis studies the distributions of normative systems synthesised by IRON and SIMON. The aim is to shed light on how the different generalisation mechanisms employed by IRON and SIMON affect their norm synthesis processes. This will help us understand the differences in quality of the normative systems that they synthesise. Finally, an analysis of the computational costs of both approaches is provided.

6.6.1 Empirical settings

As detailed in Section 6.3, SIMON can operate with two alternative generalisation modes (G_M), either a *shallow* mode (namely, by performing shallow norm generalisations), or a *deep* mode (namely, by performing deep generalisations). Hereafter, S-SIMON will stand for SIMON operating in *shallow* generalisation mode, and D-SIMON will stand for SIMON operating in *deep* generalisation mode.

The experiments use the scenario configuration and experimental settings described in Section 5.9.1. Thus, they employ a discrete-event simulator of a traffic junction, wherein agents are autonomous cars, and conflicts their collisions. Each simulation uses either IRON, S-SIMON, or D-SIMON as a norm synthesis strategy. At each time step, cars decide whether to fulfil or infringe the norms that apply to them according to a norm infringement rate ($\mathcal{NIR}$), which

is fixed to 0.3 and is the same for all cars. That is, on average, 3 of each 10 agents' decisions lead to norm infringements. Norms have at most three unary predicates with predicate symbols *"left", "front", "right"* that represent the three positions in front of a reference car. Each predicate contains one term out of a set of terms {*car-heading-right, car-heading-left, car-opposite-heading, car-same-heading, nil, anything*}, whose generalisation relationships are established by the taxonomy in Figure 6.1 (Section 6.2).

Both SIMON and IRON have been configured as shown in Table 6.1. Briefly, we have taken a conservative approach to configure both approaches. Firstly, the effectiveness/necessity ranges of a norm ($\mathcal{ER}, \mathcal{NR}$) are computed by considering the last $q = 100$ undefined effectiveness/necessity values of that norm. Secondly, the initial effectiveness and necessity of a norm is set to 0.5 ($k_{\mathit{eff}} = 0.5, k_{nec} = 0.5$). Thirdly, both approaches deactivate norms only when they perform very poorly ($\alpha_{\mathit{eff}} = 0.2, \alpha_{nec} = 0.2$). Fourthly, they consider high generalisation thresholds ($\alpha^{gen}_{\mathit{eff}} = 0.6, \alpha^{gen}_{nec} = 0.4$). Finally, they consider a convergence interval of 5,000 ticks ($T = 5,000$). Additionally, SIMON's specific settings, namely its generalisation step (G_S), considers values within [1..3]. Observe that, unlike S-SIMON and D-SIMON, IRON can only perform generalisations of a single term (which is equivalent to fixing the generalisation step to $\mathsf{G}_S = 1$).

Each experiment consists of a set of 200 different simulations for each synthesis strategy, namely IRON, S-SIMON, and D-SIMON. Each simulation starts with an empty normative system, and it finishes whenever it reaches 50,000 ticks, or it has converged to a stable normative system. A simulation is assumed to have converged to a normative system, hence solving the NSP (see Definition 18 in Section 3.2.2), if during a 5,000-tick period, the normative system remains unchanged and no unregulated conflicts arise.

6.6.2 Empirical results

This section analyses the results of this empirical evaluation. First, it illustrates a convergence analysis of SIMON to show *how* it manages to converge to a compact normative system that solves the norm synthesis problem. Next, it provides a macro analysis that compares IRON and SIMON in terms of the quality of the normative systems they synthesise. It shows that SIMON outperforms IRON in terms of compactness, while obtaining similar results in terms of effectiveness and necessity. Then, it presents a micro analysis that shows *why* SIMON manages to outperform IRON. Finally, it provides a comparison of IRON and SIMON in terms of their computational costs.

SIMON's convergence analysis

Let us now analyse how SIMON reaches convergence in a prototypical simulation. With this aim, let us focus on the chart depicted in Figure 6.6. On the x-axis, it illustrates the normative changes (i.e., the time steps in which the the normative network and/or the normative system change) for a single simulation with 0.3 norm infringement rate. On the y-axis, it shows:

Parameter	Description	Value
q	Number of effectiveness values (μ_{eff}) and necessity values (μ_{nec}) considered to compute a norm's effectiveness and necessity ranges (see Section 5.6).	100
$k_{\mathit{eff}}, k_{nec}$	Default norm effectiveness and necessity values ($\mu_{\mathit{eff}}(n, t_0), \mu_{nec}(n, t_0)$).	0.5
$\alpha_{\mathit{eff}}, \alpha_{nec}$	Thresholds below which a norm is considered to under-perform in terms of its effectiveness/necessity.	0.2
$\alpha_{\mathit{eff}}^{gen}$	Threshold above which a norm is considered to perform well in terms of its effectiveness.	0.6
α_{nec}^{gen}	Threshold above which a norm is considered to perform well in terms of its necessity.	0.4
T	Time period considered when assessing convergence (cf. Definition 18 in Section 3.2.2).	5,000

Table 6.1: SIMON's and IRON's norm synthesis settings in the road traffic scenario.

1. The cardinality of the normative system, namely the number of active norms in the normative network.

2. The number of grounded norms the normative system represents.

3. The number of terms that the normative system contains, namely its compactness.

4. The ratio of new unregulated car collisions per tick along time.

For the sake of visibility, the cardinality of the normative network is not included, since it reaches a size of 54 norms at tick 8301 (27th normative system).

At tick 13 (which corresponds to the 1st normative change), the first collision arises and SIMON synthesises the first norm. From that tick onwards, SIMON keeps generating norms when needed, hence increasing the cardinality of both the normative network and the normative system. As a consequence, the number of terms in the normative system increases as well. At tick 15 (fourth normative change), SIMON performs the first norm generalisation, reducing the cardinality of the normative system from 5 to 3 norms, and reducing the number of norm predicates of the normative system from 9 to 8. Up to tick 63 (fifteenth normative change), SIMON keeps generating and generalising norms when possible, thus reducing the number of terms in the normative system. At tick 63 (fifteenth normative change), SIMON performs an over-generalisation: it generalises all the norms of the normative system to one single norm like:

$$n^* \quad : \quad \langle \{\mathit{left}(\mathit{anything}), \mathit{front}(\mathit{anything}), \mathit{right}(\mathit{anything})\}, \mathit{prh}(\mathit{go}) \rangle$$

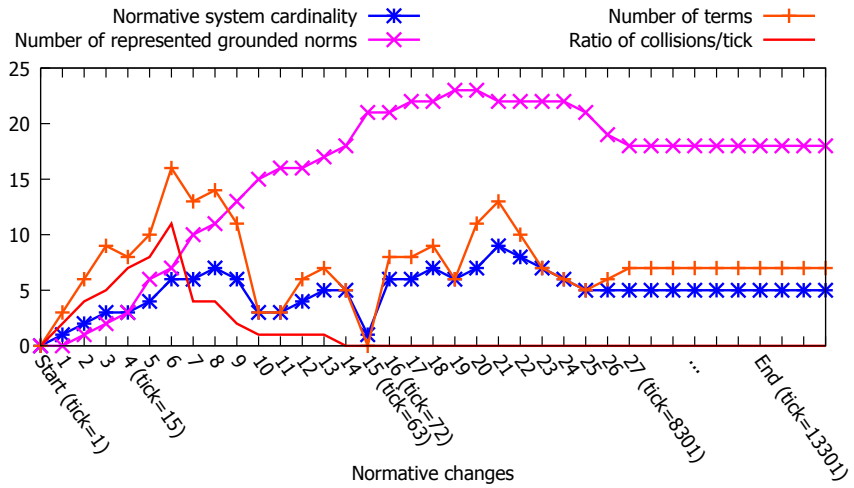

Figure 6.6: A prototypical execution of SIMON.

which concisely prohibits a car to stop in *any* situation it encounters (even if there is no approaching car). Note that this norm over-constrains the car's freedom, since it prohibits them to move in situations that imply no collision risk. Therefore, at tick 72 (sixteenth normative change), SIMON backtracks this generalisation by specialising from 1 norm to 6 child norms. From then onwards, SIMON keeps on specialising norms when required to remove from the normative system those specific norms that are unnecessary. At tick 8,301, SIMON performs the last norm specialisation, hence synthesising a compact normative system with 5 norms, which represents 18 grounded norms, and contains 7 terms in total. From tick 8,301 onwards, the normative system remains stable. By using the resulting normative system, cars that comply with norms do not cause collisions. After 5,000 further ticks (i.e., at tick 13,301), SIMON reaches convergence. Overall, IRON explored 54 different norms out of 343 possible norms (see grammar $\mathcal{G}$ in Section 5.9.1). These 54 norms were generalised into 5 norms, to find a 5-norm normative system that successfully prevents collisions as long as cars comply with norms.

Norm	Pre-condition (θ)	Norm target	$\bar{\mu}_{eff}$	$\bar{\mu}_{nec}$
n_1	$left(car\text{-}heading\text{-}right)$	$prh(go)$	0.81	0.26
n_2	$front(car\text{-}heading\text{-}right), right(car\text{-}heading\text{-}right)$	$prh(go)$	0.82	0.55
n_3	$left(car\text{-}heading\text{-}left), front(car\text{-}heading\text{-}left)$	$prh(go)$	0.92	0.55
n_4	$front(car\text{-}same\text{-}heading)$	$prh(go)$	0.91	0.25
n_5	$right(car\text{-}heading\text{-}left)$	$prh(go)$	0.89	0.33

Table 6.2: A normative system of SIMON upon convergence

The 5-norm normative system that SIMON converged to is shown in Table 6.2. This normative system contains a norm n_1 that represents a left-hand side priority norm. It specifies that a car must stop when it observes a car to its left which is heading to its right, and no matter what it perceives to its front and right positions. Additionally, norms n_2, n_3, n_4 prohibit a car to go in different situations. Note though that this normative system contains a fifth norm (n_5), which represents a general right-hand side priority norm. It is not difficult to see that either norm n_1 or norm n_5 are unnecessary, since they can replace one another in use. We already discussed this phenomenon in Section 4.8.2, since BASE had the same problem. Briefly, this comes as a consequence of SIMON's approach to norm evaluation. Likewise BASE, SIMON evaluates norms individually, and it is not able to detect when two norms are unnecessary only whenever another norm exists in the normative system. In other words, SIMON is not capable of detecting the *synergies* between these norms, and thus keeps both of the in the normative system.

Macro analysis

The next analysis compares the quality of the normative systems obtained by S-SIMON, D-SIMON, and IRON with respect to their effectiveness, necessity, and compactness. In this analysis, the effectiveness and the necessity of a normative system is computed by means of Equation 5.21 in Section 5.6.3, and the compactness of a normative system is computed as described in Definition 26 (Section 5.2). In particular, a normative system's compactness is computed as the overall number of terms in its norms' predicates. Figure 6.7 illustrates S-SIMON's and D-SIMON's savings with respect to IRON as the generalisation step G_S increases, for $\mathsf{G}_S \in \{1, 2, 3\}$. Additionally, Table 6.3 shows numerical data corresponding to Fig.6.7. The results below show the *interquartile mean* in the third quartile of the different measures employed in our comparison.

We initially analyse the quality obtained by the three algorithms being compared. Notice that S-SIMON with $\mathsf{G}_S = 1$ already synthesises normative systems that are up to 30.14% more compact (have 30.14% less terms) than those synthesised by IRON, while keeping effectiveness and necessity at very similar values. Moreover, its normative systems contain up to 30.67% less norms. Here S-SIMON clearly benefits from its more optimistic norm generalisation, which allows it to generalise further than IRON, synthesising more compact normative systems. When increasing the generalisation step, $\mathsf{G}_S = 2$, S-SIMON obtains further benefits in terms of compactness (37.2%). Analogously, its normative systems contain 34.47% less norms. Nonetheless, increasing the generalisation step, $\mathsf{G}_S = 3$, makes S-SIMON obtain worse results than for $\mathsf{G}_S = 2$. Therefore, S-SIMON cannot take advantage of the largest generalisation step. Recall that a large generalisation step enables S-SIMON to eventually carry out large generalisations. However, this may lead to over-generalisations. Then, we have observed that after undoing an over-generalisation, S-SIMON ends up with a normative network where it finds difficult to perform generalisations. A norm representing an over-generalisation causes the synthesis in the normative network of child

norms that are negatively rewarded. When backtracking an over-generalisation S-SIMON ends up with a larger number of norms (than before generalising) that it finds difficult to generalise.

Figure 6.7 also shows that D-SIMON clearly outperforms both IRON and S-SIMON, thus being the best in class. When using a low generalisation step, $\mathsf{G}_S = 1$, D-SIMON already synthesises normative systems that are up to 59.19% more compact than those synthesised by IRON and contain 41.01% less norms, while keeping effectiveness and necessity at very similar values. D-SIMON also outperforms S-SIMON, obtaining benefits in terms of compactness (and number of norms), namely the normative systems synthesised by D-SIMON have fewer terms (and norms) than those synthesised by S-SIMON. Notice also that D-SIMON slightly benefits in terms of compactness as the generalisation step increases. That means that in this particular domain, $\mathsf{G}_S = 1$ is enough for D-SIMON to obtain very compact normative systems.

When we consider the convergence time required by S-SIMON, D-SIMON, and IRON, we observe that, in general, S-SIMON and D-SIMON require more time to converge (in terms of the number of ticks). With the exception of S-SIMON with a generalisation step $\mathsf{G}_S = 1$ (which converges 14.79% faster), all S-SIMON and D-SIMON configurations require between 16.01% and 30.51% more time than IRON to converge. This may sound contradictory, since SIMON generates and evaluates norms along the lines of IRON, and can generalise norms faster than IRON. Therefore, it is sensible to think that it should converge faster. However, as discussed above, SIMON is more prone than IRON to over-generalise norms. Recall that detecting a norm representing an over-generalisation requires to explicitly represent its under-performing child norms, and to accumulate evidences about their performances to be able to deactivate them (and thus, to backtrack the over-generalisation). As a consequence, SIMON requires extra time to detect and deactivate under-performing norms, and hence to converge. Additionally, as the generalisation step increases, S-SIMON requires more time to converge, whereas D-SIMON requires less time to converge. This indicates that increasing the generalisation step helps D-SIMON reach convergence. In other words, the generalisation steps favours D-SIMON's synthesis process. This is not the case for S-SIMON. As discussed above, the largest value of the generalisation step is detrimental to S-SIMON, leading to extra work to undo over-generalisations.

To summarise D-SIMON with $\mathsf{G}_S = 3$ is the best-in-class algorithm since it achieves the best (lowest) values of compactness and number of norms at a low cost of extra convergence time with respect to IRON.

Micro analysis

Let us now investigate why D-SIMON and S-SIMON both outperform IRON in terms of minimality and simplicity. Figures 6.8, 6.9 and 6.10 show, respectively, three histograms of the normative systems synthesised by D-SIMON, S-SIMON and IRON. These histograms consider D-SIMON with $\mathsf{G}_S = 3$ and S-SIMON with $\mathsf{G}_S = 2$, namely the best configurations for the D-SIMON and S-SIMON algorithms. The x-axis in each histogram shows the different normative systems synthesised by

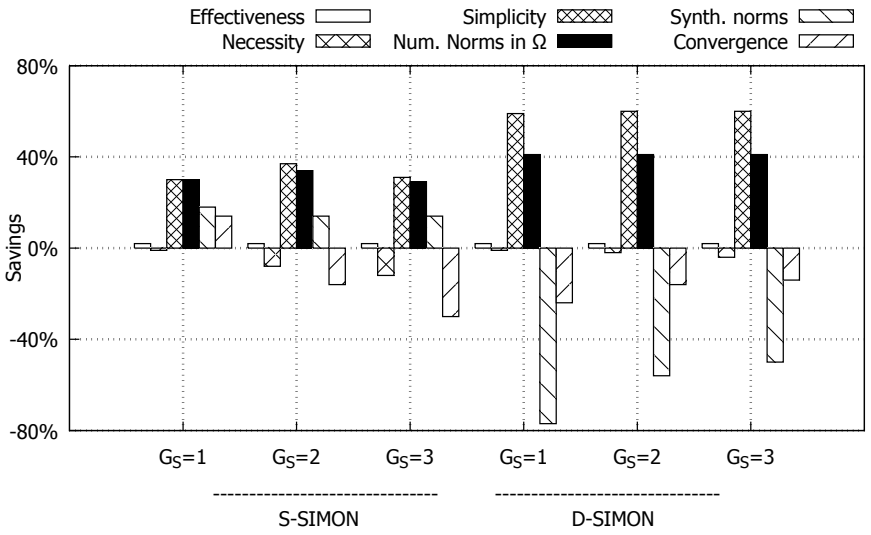

Figure 6.7: Savings of S-SIMON and D-SIMON with respect to IRON.

	IRON	S-SIMON			D-SIMON		
		$G_S = 1$	$G_S = 2$	$G_S = 3$	$G_S = 1$	$G_S = 2$	$G_S = 3$
Effectiveness	0.853	2.6%	2.32%	2.68%	2.66%	2.29%	2.6%
Necessity	0.487	-1%	-8.1%	-12.9%	-1.7%	-2.1%	-4.1%
Compactness	18.313	30.14%	37.2%	31.19%	59.19%	60.1%	60.68%
Num. norms in Ω	8.76	30.67%	34.47%	29.52%	41.01%	41.85%	41.93%
Synth. norms	23.5	18.55%	14.38%	14.38%	-77%	-56.82%	-50.41%
Convergence	6,323.6	14.79%	-16.56%	-30.51%	-24.68%	-16.01%	-14.18%

Table 6.3: Numerical data of S-SIMON's and D-SIMON's savings with respect to IRON.

the three algorithms, while the y-axis shows the number of times each normative system was synthesised. Overall, the three algorithms together managed to synthesise 314 *different* normative systems.

Observe in Figure 6.10 that IRON synthesises 173 different normative systems, ranging from Ω_{142} to Ω_{314}. Now, focus on Figure 6.9, depicting S-SIMON's histogram. Observe that S-SIMON synthesises 111 normative systems, mostly ranging from Ω_{31} to Ω_{141}. Therefore, S-SIMON explores an area of the space of normative systems that is not at all addressed by IRON. Finally, consider Figure 6.8, depicting D-SIMON's histogram. Note that D-SIMON synthesises 30 different normative systems, mostly ranging from Ω_1 to Ω_{30}. Although there is some intersection between the normative systems synthesised by D-SIMON and S-SIMON, we notice that, again, D-SIMON mostly explores an area of the space of normative systems that is not reached by S-SIMON or IRON. To summarise, the three algorithms explore different areas of the search space of normative systems.

Next, let us analyse the dispersion of the three distributions of synthesised

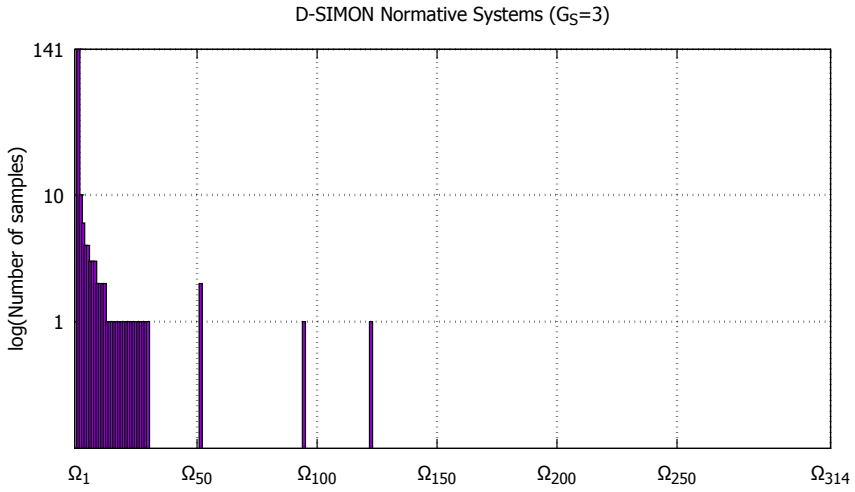

Figure 6.8: Histogram of normative systems synthesised by D-SIMON

normative systems. Notice that D-SIMON's dispersion is low (it only synthesises 30 different normative systems out of 200 different simulations), whereas S-SIMON's is medium (it synthesises 111 normative systems), and IRON's is high (it synthesises 173 different normative system). In fact, D-SIMON synthesises 141 times (70.5%) normative system Ω_1, and the first 10 normative systems are synthesised 89% of the times. Against this, S-SIMON synthesises normative system Ω_{31} only 15 times (7.5% of the total), and IRON synthesises Ω_{142} only 4 times (2% of the total).

To summarise, D-SIMON consistently focuses on an area of the search space where more compact normative systems are. This explains why it outperforms S-SIMON and IRON in terms of compactness.

Computational cost analysis

Finally, D-SIMON, S-SIMON, and IRON are compared with respect to the number of norms they synthesised (one should not confuse the number of synthesised norms with the number of normative systems). The results are shown in the Table 6.3. While S-SIMON synthesises fewer norms than IRON (between 14.38% and 18.55%, depending on the generalisation step), D-SIMON does synthesise more norms than IRON (between 50.41% and 77%, depending on the generalisation step). This must not be regarded negatively because this is precisely what allows D-SIMON to synthesise normative systems that are reached by neither S-SIMON nor IRON. In fact, D-SIMON synthesises more norms that its competitors because it manages to perform more norm generalisations than them, which amounts to synthesising new (general) norms.

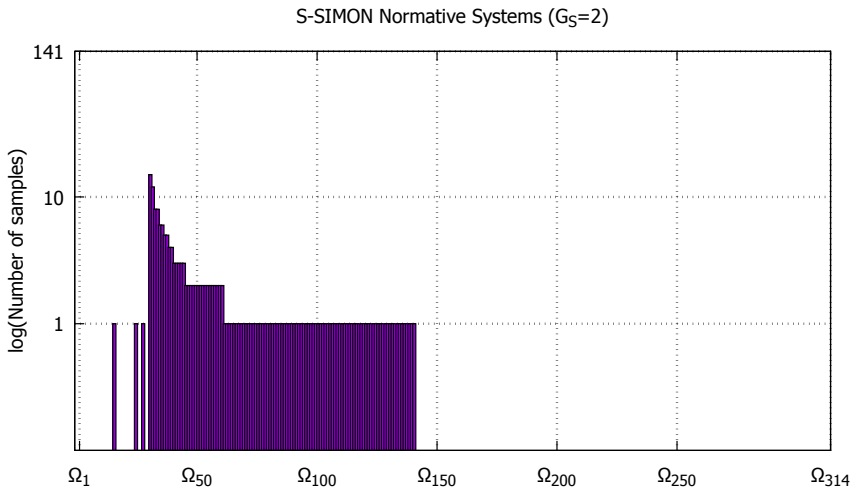

Figure 6.9: Histogram of normative systems synthesised by D-SIMON

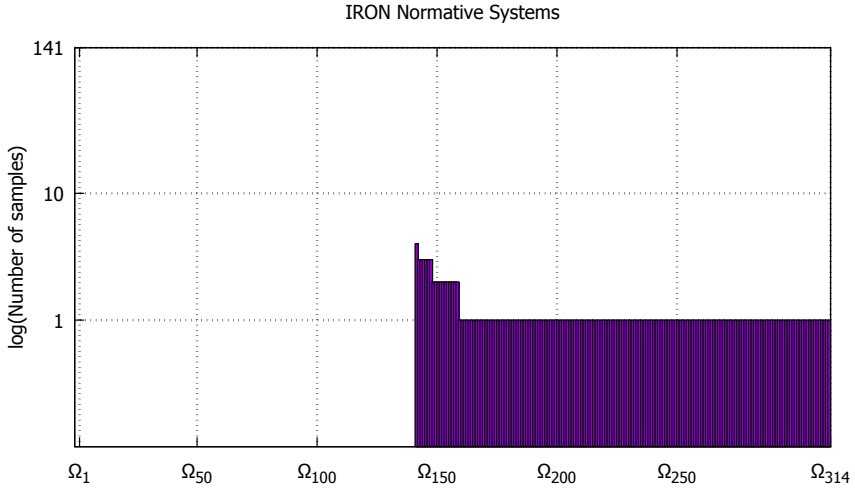

Figure 6.10: Histogram of normative systems synthesised by D-SIMON

6.7 Empirical evaluation considering an on-line community scenario

The empirical evaluation of SIMON ends with a comparison of IRON's and SIMON's norm synthesis in the on-line community scenario described in Section 1.3.2. Next, Section 6.7.1 details the empirical settings of these experiments. Thereafter, Section 6.7.2 empirically shows the differences between both approaches and their results for this scenario. More specifically, it first illustrates a micro analysis of SIMON's norm synthesis in a single simulation, showing how it manages to synthesise a compact normative system that solves the NSP. Then, it provides a macro analysis of the convergence of both approaches, illustrating *when* they converge, and *what* types of normative systems they synthesise. Finally, it analyses the type of normative networks that IRON and SIMON synthesise. Initial results prove that SIMON is capable of performing further generalisations than IRON, which allows it to synthesise more compact normative systems.

6.7.1 Empirical settings

These experiments employ the discrete agent-based on-line community simulator described in Section 5.10.1. Such simulator simulates an on-line community in which a population of users can upload contents, view contents, and complain about those contents they feel uncomfortable with. The on-line community is divided into three different *sections* on which users can upload different types of contents. In particular, users can upload either appropriate (correct) contents, or inappropriate contents (i.e., spam, porn, violent, or insults).

Each simulation finishes when it reaches 5,000 ticks, and it considers an initial *"warm-up"* period of 500 ticks to reach normal conditions in on-line communities, where the users enter in the community and there already exist contents to view and complain about. Thus, from tick 0 to 500, users only upload contents, and from tick 500 onwards, users upload, view, and complain about contents. We consider that a simulation has converged whenever the normative system remains unchanged during a 1000-tick period.

The experiments consider an on-line community of 10,000 users that are modelled as a population of 100 *heavy contributors*. Particularly, three different populations are considered, which are composed of *moderates* and *spammers* in different proportions:

1. A population with a *majority* of moderate users, composed of 70% moderates and 30% spammers (hereafter 30M-70S).
2. A *balanced* population of 50% moderates and 50% spammers (50M-50S).
3. A population with a *minority* of moderate users, composed of 30% moderates and 70% spammers (70M-30S).

At each time step (i.e, tick) each user: (i) views one content with probability 1; (ii) uploads one content with probability 0.05; and (iii) complaints about those

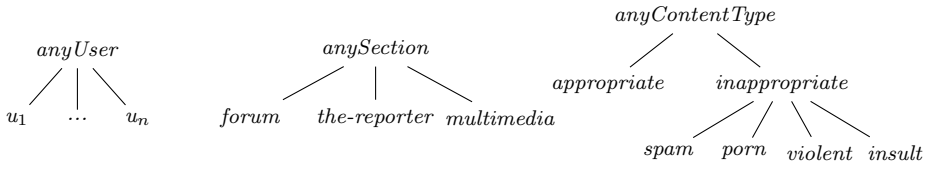

Figure 6.11: Taxonomy of terms in the on-line community scenario.

inappropriate contents it views according to its preferences (i.e., its profile).

Norms have at most three unary predicates with predicate symbols {*user,section,cntType*} that represent a user of the community, a section, and the type of a content a user aims at uploading. In particular, the *"user"* predicate contains one term out of a set $\{u_1, \ldots, u_{100}, anyUser\}$, the *"section"* predicate contains one term out of {*forum, the-reporter, multimedia, anySection*}, and the *"cntType"* predicate has one term out of {*spam, porn, violent, insult, anyContentType*}. Figure 6.11 captures the generalisation relationships between these terms.

SIMON's and IRON's synthesis parameters have been configured as shown in Table 6.4. Briefly, for each new norm n, both approaches set its initial necessity to 0.5 ($\mu_{nec}(n, t_0) = 0.5$), and compute a norm's necessity at a given time by considering its last 100 defined necessity values ($q = 100$). Recall from Section 5.10.1 that, in this scenario, a norm's effectiveness cannot be evaluated, since norm fulfilments are not observable. SIMON's and IRON's necessity generalisation thresholds are set to 0 ($\alpha_{nec}^{gen} = 0$) so that they generalise any norm that is necessary enough to be active. SIMON's and IRON's norm synthesis is analysed for low, medium, and high deactivation thresholds. More specifically, $\alpha_{nec} \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. As to SIMON's specific settings, it has been configured to perform *deep* generalisations (G_M = *deep*) and to generalise three terms at a time ($\mathsf{G}_S = 3$), since with this configuration SIMON obtained its best results in the empirical evaluation in the road traffic scenario (Section 5.9.2).

6.7.2 Empirical results

Next, SIMON's empirical results are analysed. First, a micro analysis of SIMON shows *how* it manages to converge to a compact normative system by synthesising norms from scratch. Thereafter a macro analysis analyses *when* SIMON converges and *what* type of normative systems it synthesises for different populations and norm necessity thresholds. SIMON and IRON are compared to demonstrate that SIMON significantly outperforms IRON in terms of compactness. Finally, SIMON's synthesised normative networks are analysed and compared with those synthesised by IRON. The aim is to shed light on how SIMON structures norms in a normative network to synthesise compact normative systems.

Parameter	Description	Value
q	Number of effectiveness values (μ_{eff}) and necessity values (μ_{nec}) considered to compute a norm's effectiveness and necessity ranges (see Section 5.6).	100
k_{nec}	Default norm effectiveness and necessity values ($\mu_{eff}(n, t_0), \mu_{nec}(n, t_0)$).	0.5
α_{nec}	Thresholds below which a norm is considered to under-perform in terms of its effectiveness/necessity.	(0.1, 0.3, 0.5, 0.7, 0.9)
α_{nec}^{gen}	Threshold above which a norm is considered to perform well in terms of its necessity.	0
T	Time period considered when assessing convergence (cf. Definition 18 in Section 3.2.2).	1,000
G_M	SIMON's generalisation mode.	*deep*
G_S	SIMON's generalisation step.	3

Table 6.4: SIMON's norm synthesis settings in the on-line community scenario.

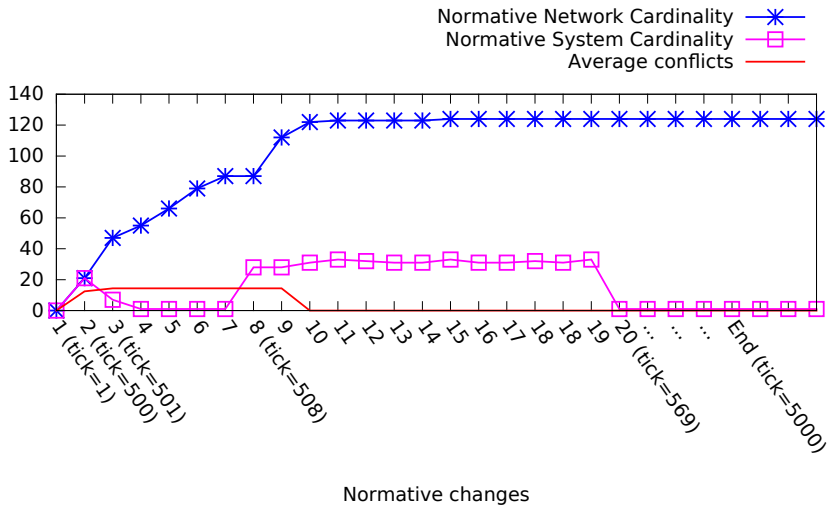

Figure 6.12: SIMON's convergence process for a population of 70 moderate users and 30 spammers.

Micro analysis: SIMON's convergence process

Let us first illustrate how SIMON manages to synthesise from scratch a normative system that avoids unregulated conflicts in our on-line community scenario. Figure 6.12 shows a prototypical execution of SIMON for population of 70 moderate users and 30 spammers, and a 0.3 norm infringement rate. The x-axis shows the different normative changes that SIMON performed along time, and the y-axis shows: (1) the cardinality of the normative network, (2) the cardinality of the normative system, and (3) the ratio of unregulated user complaints (conflicts) per tick. From tick 0 to 500 (i.e., during the *warm-up* period), users only upload and view contents, but never complain. Therefore, no conflicts arise and SIMON does not generate norms. At tick 500 (second normative change), moderate users start complaining about spam contents they view, which triggers SIMON's norm generation. Thus, from tick 500 on, SIMON subsequently generates and adds norms to the normative network to prevent spammers from uploading spam. At tick 501 (third normative change) SIMON performs the first norm generalisation, reducing the number of norms in the normative system from 47 to 7. At tick 503 (fourth normative change), SIMON managed to synthesise the following compact norm:

$$n^* \quad : \quad \langle (user(anyUser),\ section(anySection), cntType(spam)),\ prh(upload) \rangle$$

which concisely prohibits *all* the users of the community to upload spam contents in *any* section. This comes as a result of SIMON's optimistic approach to norm generalisation, which does not require to synthesise norms to prohibit each user to upload spam before generalising into n^*. Particularly, SIMON provides the agents with a simplified version of this norm that does not contain neither the *user* nor the *section* predicates, since these predicates have *root terms*.

Notice then that SIMON generalises norms very rapidly, synthesising a compact normative system of one norm in a few time steps. Thereafter, SIMON specifies those specific norms that are implicitly represented by n^*, and that under-perform at some point. As a result, the cardinality of the normative network keeps on increasing, while the cardinality of the normative system remains stable. At tick 508 (eighth normative change), SIMON considers that some specific norms that n^* represents under-perform in terms of necessity, and therefore specialises n^* into its children. This results in the increment of the normative system's cardinality from 1 norm to 31 norms. At tick 569 (twentieth normative change), SIMON considers that all the norms that n^* represents perform well again (i.e., they are necessary enough), and re-activates them, generalising again to norm n^*. Thereafter, the normative system remains stable. At the end of the simulation, SIMON has synthesised a normative system with one general norm that prohibits spam contents within the community. By contrast, the normative network contains 121 norms. It contains: (i) 90 norms to prohibit the three spammers to upload spam in the three sections of the community (30 spammers $\times$ 30 sections = 90 norms); (ii) 30 general norms to prohibit each spammer to upload spam in any section; and (iii) one general norm to prohibit to any user to upload spam in any section.

	Consensus degree (α_{nec})				
Population	0.1	0.3	0.5	0.7	0.9
30M-70S (0.3 complain power)	1	X	X	X	X
50M-50S (0.5 complain power)	1	1	X	X	X
70M-30S (0.7 complain power)	1	1	1	X	X

Table 6.5: Number of norms that SIMON converged to.

Macro analysis. SIMON versus IRON: convergence outcomes

Next, SIMON is compared with IRON in terms of their convergence and the type of normative systems they synthesise. First, recall from Section 5.10.2 that the proportion of moderate users in a population represents the *complain power* of that population (C_{Pw}), since moderates are the only users who complain about spam. Thus, the complain power of a population is directly related with the necessity of the norms to prohibit spam. The more complaints, the more necessary the norms. Moreover, the necessity threshold (α_{nec}) can be seen as a *consensus degree* that establishes the minimum proportion of users that must consider norms as necessary so that they are included in the normative system. Therefore, the relationship between the complain power of a population and the established consensus degree determines the type of normative systems that SIMON converges to.

Table 6.7 depicts averaged results after 100 simulations of SIMON. Each cell contains, for each population and specialisation threshold: either (i) the size of the normative system upon convergence; or (ii) symbol "X" if it was not able to converge to a normative system. Observe that SIMON converged to a normative system whenever the complain power of a population (that is, its proportion of moderates) was *above* the consensus degree (i.e., the norm necessity threshold), namely when $\mathsf{C}_{Pw} > \alpha_{nec}$. This happens because the proportion of moderates in a population determines the necessity of the norms to prohibit spam. That is, the more users complain about spam, the more necessary the norms to prohibit spam. For instance, for a population of 30 moderates and 70 spammers (30M-70S), norms to prohibit spam had around 0.3 necessity. Therefore it converged to a stable normative system when the consensus degree was below 0.3 ($\alpha_{nec} < 0.3$). Along the same lines, for population 50M-50S, it converged to 50 norms when the consensus degree was below 0.5 ($\alpha_{nec} < 0.5$), and converged to 70 norms for population 70M-30S and consensus degrees below 0.7 ($\alpha_{nec} < 0.7$). When SIMON converged, it managed to synthesise a normative system with one single norm like norm n^* above, which prohibits any user to upload spam in any section. Here SIMON benefited from its optimistic approach to norm generalisation, which allows it to generalise both the *"section"* and the *"user"* predicates of norms .

Case	Convergence
Complain power > consensus degree ($\mathsf{C}_{Pw} > \alpha_{nec}$)	Yes
Complain power = consensus degree ($\mathsf{C}_{Pw} = \alpha_{nec}$)	No
Complain power < consensus degree ($\mathsf{C}_{Pw} < \alpha_{nec}$)	No

Table 6.6: Summary of SIMON's macro analysis.

By contrast, SIMON was not able to converge whenever the complain power was *equal* the consensus degree (namely, $\mathsf{C}_{Pw} = \alpha_{nec}$). In this case, norms' necessities fluctuated above and below the consensus degree, and thus norms were continuously deactivated and re-activated. Therefore, SIMON could not find a stable normative system. Finally, when the complain power of a population was *below* the consensus degree (namely, $\mathsf{C}_{Pw} \leq \alpha_{nec}$), SIMON could not converge. However, one may expect that SIMON should have converged to an empty normative system, since norms' necessities, in this case, should be always below the necessity threshold. The reason of this was already introduced in Section 5.10.2 (since IRON had the same problem). Briefly, when SIMON creates a new norm, it sets its initial necessity to 0.5, and iteratively evaluates it. Once the norm's necessity goes below the necessity threshold, SIMON deactivates it. Thereafter, the conflict the norm regulated becomes unregulated again. The next time a user complaints, and the conflict arises again, SIMON immediately reacts by creating (re-activating) the norm. Eventually, SIMON will evaluate the norm again as unnecessary, thus deactivating it. This leads SIMON into a cycle of continuous norm deactivations/re-activations, making IRON incapable to converge. In this scenario, SIMON should consider larger amounts of evidences to assess if re-activating a norm is really necessary. Namely, it should be more *deliberative*. Table 6.6 summarises SIMON's results for different combinations of complaint power/consensus degree.

Let us now compare SIMON's results with IRON's, which are shown in Table 6.7. Briefly, IRON shows similar convergence conditions to SIMON. It converged to a stable normative system only whenever the complain power of a given population is over the consensus degree (namely, $\mathsf{C}_{Pw} > \alpha_{nec}$). However, IRON was not capable of synthesising the 1-norm normative system that SIMON managed to synthesise. Instead, it synthesised normative systems containing norms to prohibit *each spammer* to upload spam in any section. These empirical results demonstrate that SIMON significantly outperforms IRON in terms of compactness.

Analysis of synthesised normative networks

Let us now analyse the size and structure of SIMON's synthesised normative networks and compare them to those synthesised by IRON. Figure 6.13 depicts a normative network that SIMON synthesised to regulate a population with 70 moderates and 30 spammers. There, each circle represents a norm, and each edge represents a generalisation relationship between two norms. In particular, *yellow* circles represent norms that generalise the term *"section"* predicate, and

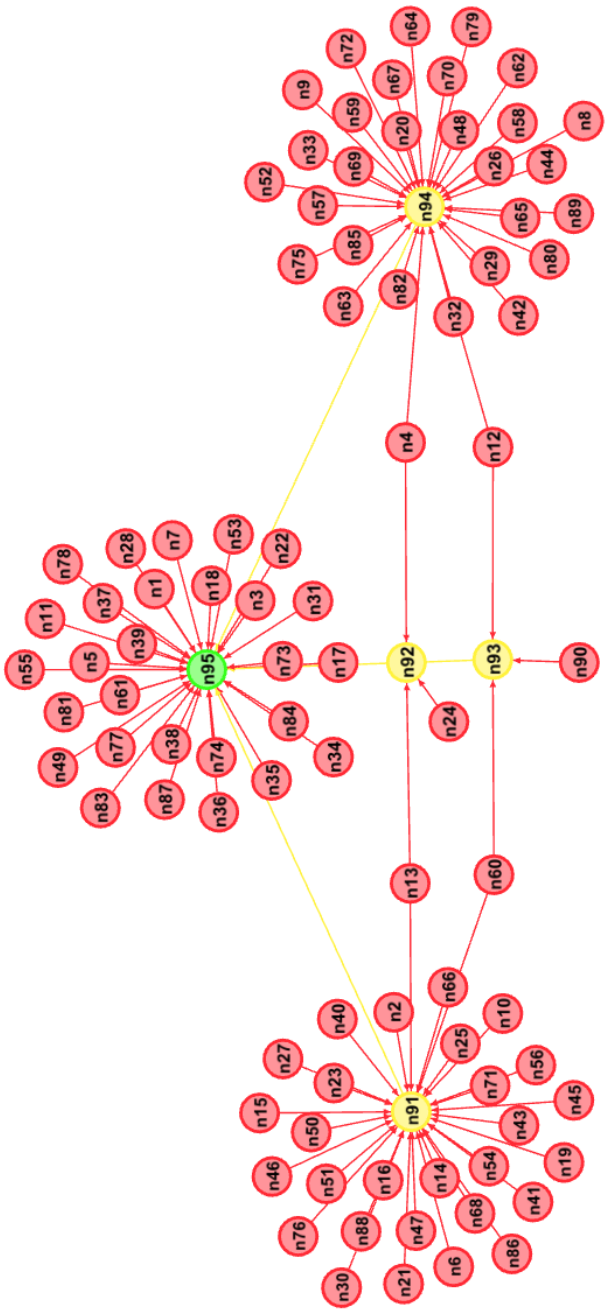

Figure 6.13: Example of a prototypical normative network synthesised by IRON.

	Consensus degree (α_{nec})				
Population	0.1	0.3	0.5	0.7	0.9
30M-70S (0.3 complain power)	70	X	X	X	X
50M-50S (0.5 complain power)	50	50	X	X	X
70M-30S (0.7 complain power)	30	30	30	X	X

Table 6.7: Number of norms that IRON converged to.

a *green* circle represents a norm that generalises both the *"section"* and the *"user"* predicates. Particularly, such network contains one single active norm n_{95}, which corresponds to norm n^* described above. Therefore, it represents a normative system with one single norm that prohibits *any* user to upload spam in *any* section. In fact, norm n_{95} generalises (and hence represents) each one of the norms in the normative network.

Figure 6.14 illustrates an example generalisation of norms $n_2, n_6, n_{91}, n_{70}, n_{72}, n_{94}$ described below, which belong to the normative network depicted in Figure 6.13, and are generalised as norm n_{95} described above. Figure 6.14 illustrates the optimistic generalisation of: (1) norms n_2 and n_6 as n_{91}, which prohibits user u_{71} to upload *spam* in *anySection*; (ii) norms n_{70} and n_{72} as n_{94}, which prohibits user u_{72} to upload *spam* to *anySection*; and finally (iii) norms n_{91} and n_{94} as n_{95}, which prohibits any user to upload spam in any section.

$$
\begin{array}{rcl}
n_2 & : & \langle(user(u_{71}), \quad section(forum), \quad cntType(spam)), \ prh(upload)\rangle \\
n_6 & : & \langle(user(u_{71}), \quad section(the\text{-}reporter), \ cntType(spam)), \ prh(upload)\rangle \\
n_{91} & : & \langle(user(u_{71}), \quad section(anySection), \ cntType(spam)), \ prh(upload)\rangle \\
n_{70} & : & \langle(user(u_{72}), \quad section(forum), \quad cntType(spam)), \ prh(upload)\rangle \\
n_{72} & : & \langle(user(u_{72}), \quad section(the\text{-}reporter), \ cntType(spam)), \ prh(upload)\rangle \\
n_{94} & : & \langle(user(u_{72}), \quad section(anySection), \ cntType(spam)), \ prh(upload)\rangle
\end{array}
$$

We then observe a structure which is different from the normative network synthesised by IRON (depicted in Figure 5.14 from Section 5.10.2). SIMON's normative network shows a hierarchical structure of two levels, where the most general norm generalises two predicates. By contrast, IRON's normative network shows a hierarchical structure of one level, since it is only capable of generalising the *"section"* predicate of norms. Therefore, we conclude that SIMON manages to outperform IRON in terms of compactness of the normative system since it generates normative networks with a structure that IRON is not capable of generating.

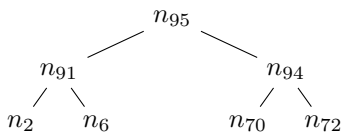

Figure 6.14: Example of SIMON's generalisation.

6.8 Conclusions

This chapter has addressed the synthesis of compact normative systems from an optimistic point of view. It has introduced SIMON, a synthesis strategy that performs optimistic norm generalisations. SIMON generalises norms with partial evidence, which allows it to yield very compact normative systems. Its generalisation mechanism is inspired in the *anti-unification* of terms (in the context of logical reasoning) [Armengol and Plaza, 2000]. Anti-unification consists of generalising feature terms to their *least common subsumer* or most specific generalisation, generalising pairs of terms to the most specific term that is common to all of them. Along the same lines, SIMON generalises norms by detecting pairs of norms that are generalisable, and generalising them to their most specific generalisation. SIMON incorporates a mechanism to handle over-generalisation whereby it backtracks generalisations of norms that under-perform, leading to more specific norms and increasing the precision of the normative system.

SIMON's performance has been analysed and compared with IRON's by considering two different scenarios. Of these, the first has been the road traffic scenario. There, SIMON has been empirically evaluated with different operation modes (namely, deep and shallow), as well as for different generalisation steps (namely, the number of terms that SIMON can simultaneously generalise). The empirical results prove that SIMON significantly outperforms IRON in terms of compactness. It synthesises normative systems that have fewer norms and terms. Moreover, SIMON offers a fine-grained control of how to generalise the pre-condition of norms. Thereafter, SIMON's norm synthesis has been compared with IRON's in the on-line community scenario. There, SIMON has been again proven to significantly outperform IRON in terms of compactness, synthesising a compact normative system with 1 norm that concisely prohibits all the users in a community to upload spam in any section.

From SIMON's empirical analysis, we conclude that SIMON is capable of synthesising compact normative systems in domains wherein gathering full evidence of norms to generalise is not possible. One should choose SIMON as a synthesis strategy if achieving compactness is more important than achieving liberality (i.e., the agents' freedom). The reason of this is that SIMON's norm generalisations increase the number of grounded norms a normative system represents, and thus it is prone to over-generalise. Therefore, even though SIMON can synthesise very compact normative systems, over-generalisations may penalise the agents' freedom.

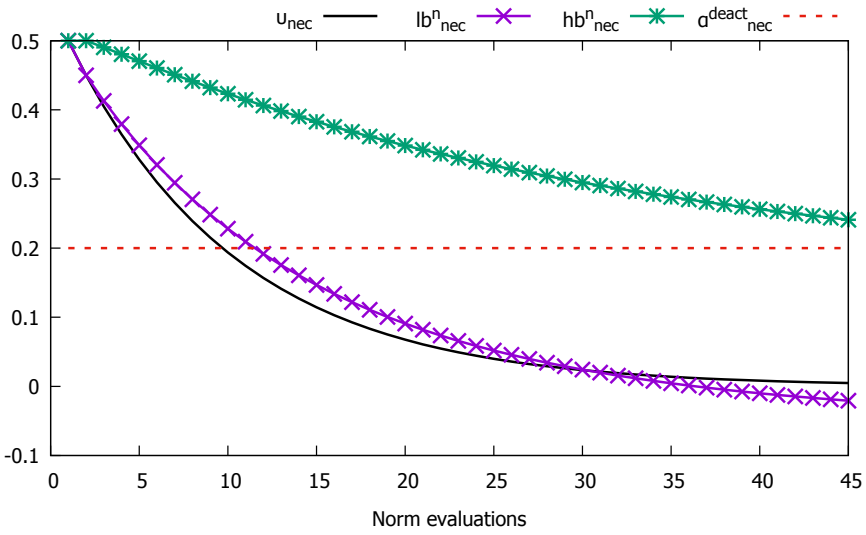

Figure 6.15: Evolution of a norm's necessity ranges along time.

However, from the observed in SIMON's empirical results, here we argue that SIMON has some limitations:

1. Lack of liberality. SIMON pursues liberality by individually evaluating norms in terms of their necessity, and discarding unnecessary norms. However, evaluating norms individually is not enough to detect when a norm becomes unnecessary if another norm exists in the normative system. As discussed in Section 6.6, in the road traffic scenario, SIMON synthesised normative systems containing left-hand side priority norms, and right-hand side priority norms, which together restrict agents' behaviours to a great extent. This happens because SIMON cannot detect the synergies between these two norms to disregard one of them, thus increasing the liberality of the normative system.

2. Inefficiency in detecting under-performing norms. Put crudely, SIMON's mechanism to detect under-performance is highly conservative. SIMON decides that a norm under-performs based on its effectiveness and necessity ranges (see Equations 5.10, 5.15, 5.11, 5.16 in Section 5.6). These ranges aggregate, respectively, the effectiveness and the necessity values of a norm along time, which in turn are also aggregated values, computed by a reinforcement learning formula (see Equations 5.3 and 5.6 in Section 5.6). Therefore, SIMON makes its decisions based on information that has been aggregated *two times*, and thus is over-smoothened. As a result, SIMON requires greats amounts of evidences to detect that a norm under-performs, even though it could detect with fewer evidences.

Let us illustrate an example with Figure 6.15. It shows the evolution of a norm's necessity along time as evaluated by SIMON. On the x-axis, it depicts

different norm evaluations. On the y-axis, it shows:

1. A norm's necessity (μ_{nec}).
2. The lower boundary of the norm's necessity range (lb_{nec}).
3. The higher boundary of the norm's necessity range (hb_{nec}).
4. A norms' necessity threshold (α_{nec}).

Observe that the evolution of the norm's necessity (μ_{nec}) hints that it is completely unnecessary. From the very beginning, it monotonically decreases, until it rapidly goes below the necessity threshold (α_{nec}) in the tenth evaluation. However, the norm's necessity range decreases much more slowly, since it is over-smoothed. In particular, its higher boundary (hb^n_{nec}), which SIMON employs to detect under-performance, has not gone below the necessity threshold after forty-five norm evaluations. This delays the detection of the norm's under-performance, which may have been detected in the tenth norm evaluation.

3. High reactivity to conflicts. In short, SIMON decides to activate a norm, hence including it in the normative system, based on one single conflict evidence. Moreover, SIMON is not capable of considering larger amounts of evidences (conflicts) when making this decision. This may be counter-productive in domains wherein being more *deliberative* may help yield better normative systems. An example was illustrated with the empirical evaluation of SIMON in the on-line community scenario (see the macro analysis in Section 6.7.2). There, SIMON could not converge when the complaint power of a population (i.e., its proportion of moderates) was *below* the consensus degree (i.e., the necessity threshold). This happened because SIMON reacted to single complaints by re-activating norms that were unnecessary. Eventually, this led to a cycle of continuous norm de-activations and re-activations, making SIMON unable to converge. With that setting, being more *deliberative* would have allowed SIMON to realise that, despite punctual users' complaints, norms were indeed unnecessary, and they thus should have never been activated.

Against this background, we conclude that there is room for designing alternative synthesis strategies that synthesise more liberal normative systems, and that can perform norm synthesis by considering different degrees of *reactivity* to conflicts. Therefore, the next two chapters introduce a strategy to synthesise liberal normative systems (Chapter 7) and to synthesise normative systems with different degrees of reactivity to conflicts (Chapter 8).

Chapter 7

Synthesising liberal normative systems

7.1 Introduction

Chapters 5 and 6 tackled the synthesis of compact normative systems to answer research question R2 introduced in Section 1.2. They introduced two synthesis strategies (IRON and SIMON) to synthesise compact normative systems that avoid conflicts while reducing the amount of norms that agents are provided with. Both approaches pursue the compactness of a normative system by performing norm generalisations. On the one hand, IRON takes a conservative approach to norm generalisation that requires full evidence. On the other hand, SIMON takes an optimistic approach that generalises norms with partial evidence.

However, as we discussed in Section 6.8, SIMON synthesises normative systems that are not enough *liberal*. Liberality can be seen as an attempt to preserve agents' freedom as much as possible when regulating their behaviour. Intuitively, the smaller the number of constraints in a normative system, the greater freedom for agents (and thus, the more liberal the normative system). In the road traffic scenario, SIMON synthesised normative systems containing left-hand side priority norms and right-hand side priority norms, which can replace one another in use, and together constrain agents' behaviours to a great extent. This came as a result of their incapability to detect the synergies between norms.

Against this background, this chapter introduces LION (*LI*beral *O*n-line *N*orm synthesis), a strategy to synthesise liberal normative systems. In this way, this chapter answers research question R3 introduced Section 1.2. LION builds on SIMON, since it obtained the best results in terms of compactness in its empirical evaluation (see Sections 6.6 and 6.7). Thus, LION aims at synthesising normative systems that (1) avoid conflicts within a MAS; (2) are as compact as possible; and (3) preserve agents' freedom to the greatest possible extent.

The key to the success of LION in this multi-objective synthesis process is that it is able to detect *semantic* relationships between norms, thus detecting norms

that can replace one another in use. With this aim, it detects and exploits norm synergies. More precisely, LION can detect *substitutability* and *complementarity* relationships between norms. Substitutability and complementarity [Samuelson, 1974, McKenzie, 1977] are well-known concepts in economics that apply to a wide range of domains [Ahlfeldt and Maennig, 2010, Ward et al., 2009, van Smoorenburg and van der Velden, 2000]. While two substitute goods may replace one another in use, complementary goods are better used together. Along these lines, we say that two norms like *"give way to the left"* and *"give way to the right"* are substitutable, since they can replace one another in avoiding collisions. Analogously, we say that two norms like *"give way to the left"* and *"keep your distance"* are complementary, since both of them are necessary to avoid collisions. Overall, LION exploits generalisation relationships to pursue compactness, exploits complementarity relationships to safeguard performance, and exploits substitutability relationships to pursue liberality.

Additionally, LION incorporates alternative approaches to evaluate and to refine norms. In this way, it overcomes SIMON's drawbacks with respect to the detection of under-performing norms. LION is endowed with alternative mechanisms to evaluate norms and to detect norms' under-performance that exploit norm compliance evidences more efficiently than SIMON. All this allows LION to detect under-performing norms faster than SIMON, and thus to converge in a reasonable amount of time, even though it requires extra time to detect and exploit norm synergies.

The remainder of this chapter is organised as follows. Section 7.2 formalises the notions of substitutability and complementarity. Thereafter, Section 7.3 details how to detect and exploit substitutability and complementarity relationships to synthesise liberal normative systems, and Section 7.4 details how to evaluate norms and to detect under-performing norms. Next, Section 7.5 introduces LION, and Section 7.6 provides means to analyse and compare normative systems in terms of their liberality. Finally, Section 7.7 shows an empirical evaluation of LION and a comparison with SIMON in the road traffic scenario, and Section 7.8 draws some conclusions.

7.2 Characterising norm synergies

Next, we introduce some definitions to describe when two norms are substitutable, and when two norms are complementary. With this aim, let us consider the formal model employed in previous chapters, with a set of agents Ag, a set of actions Ac available to agents; a function *context(ag,s)* to retrieve an agent's context in a given state; and an *action(ag,s,s')* function to retrieve the action an agent performed in a state transition. Additionally, let us consider a function to retrieve the *scope* of an agent. Given an agent, such function describes which other agents she can perceive. Formally, this function is $scope : Ag \times S \rightarrow \mathcal{P}(Ag)$, which returns the set of agents that an agent perceives at a given state. In particular, when an agent ag' *is in* the scope of agent ag in a state s, we say that ag' is detected by ag in s, and we denote it by $ag' \in scope(ag, s)$.

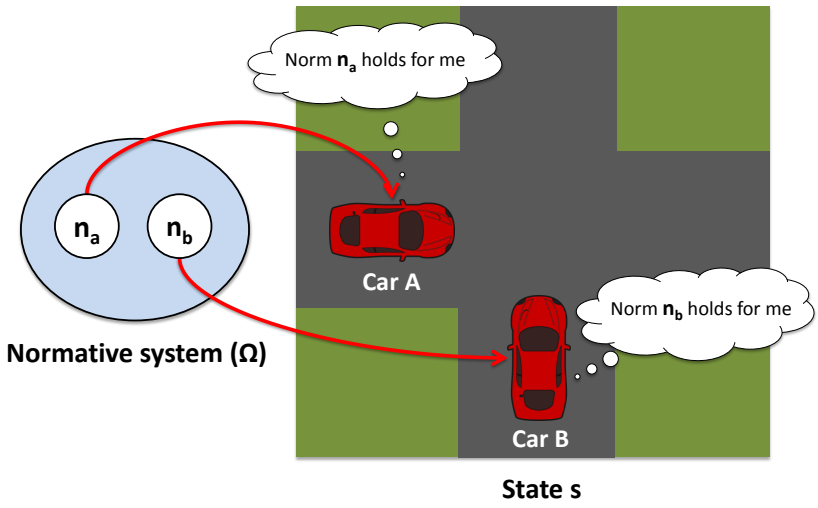

Figure 7.1: Example of two substitutable norms.

Let us now describe when two norms are substitutable or complementary. In short, two norms are substitutable if they satisfy the same regulatory needs and, therefore, can substitute one another. By contrast, two norms are complementary if they perform better when used together. Therefore, in order to assess whether any of these two relationships holds between a pair of norms, it is necessary to assess the difference in outcome between their simultaneous and individual fulfilments. Thus, substitutability between two norms will hold when the concurrent fulfilment of the two norms avoids conflicts, but also does the individual fulfilment of only one of the two norms. Let us illustrate it with an example using again the road traffic example employed in previous chapters, wherein a car describes her local context by means of three predicates with predicate symbols *"left", "front", "right"*, representing the three positions in front of it, and each predicate contains one term out of a set of terms {*car-heading-right, car-heading-left, car-opposite-heading, car-same-heading, nil, anything*}, whose generalisation relationships are established by the taxonomy in Figure 6.1. Consider now the traffic situation in Figure 7.1, and a normative system containing norms n_a, n_b described below:

$$n_a : \langle \{left(nil), front(nil), right(car\text{-}heading\text{-}left)\}, prh(go) \rangle$$
$$n_b : \langle \{left(car\text{-}heading\text{-}right), front(nil), right(nil)\}, prh(go) \rangle$$

Norm n_a is a right-hand side priority norm, aimed at giving way to cars on the right. Analogously, norm n_b is a left-hand side priority norm aimed at giving way to cars on the left. Figure 7.1 depicts a situation in which two different cars (A and B) perceive one another in a state s. In particular, car A perceives

car B to its right position, and car B perceives car A to its left position. Thus, norm n_a applies to car A, and norm n_b applies to car B. Notice that, in this situation, either n_a or n_b are necessary to avoid conflicts (a collision), but not both. Therefore, employing both norms would over-constrain this situation. We say then that both norms substitute one another, namely they are substitutable, since only one of them could satisfactorily regulate the situation. Therefore, only one of these norms is enough to avoid collisions in this situation. Table 7.1 summarises the different outcomes resulting from two substitutable norms n, n' that are jointly fulfilled and/or infringed.

		n'	
		Fulfilment	Infringement
n	Fulfilment	No conflicts	No conflicts
	Infringement	No conflicts	Conflicts

Table 7.1: Outcomes of two substitutable norms that are jointly fulfilled/infringed.

By contrast, complementarity between two norms will hold when only the concurrent fulfilment of both norms avoids conflicts, whereas the individual fulfilment of only one of them does not. Consider now the road traffic situation in Figure 7.2, and a normative system with norms n_c, n_d described below:

$$n_c : \langle \{left(nil), front(car\text{-}same\text{-}heading), right(car\text{-}heading\text{-}left)\}, prh(go) \rangle$$
$$n_d : \langle \{left(car\text{-}heading\text{-}right), front(car\text{-}heading\text{-}right), right(nil)\}, prh(go) \rangle$$

Norm n_c prohibits a car to go if it perceives a car heading left to its right position, a car heading to its same direction in front, and nothing to its left, and norm n_d prohibits a car to go if it perceives a car heading to its left on its left and front positions, and nothing to its right. The figure depicts a situation similar to that of Figure 7.1, with the exception that, in this case, there is an additional car C on the centre of the junction. Notice that norm n_c applies to car A, and norm n_d applies to car B. In this situation, cars A and B should comply with their applicable norms to avoid collisions, since car C may suddenly stop, and either car A or car B would collide with it. Therefore, we say that norms n_c, n_d are complementary, since they complement one another in avoiding conflicts in this situation. In other words, both of them are necessary to avoid conflicts. Table 7.2 summarises the different outcomes resulting from two complementary norms n, n' that are jointly fulfilled and/or infringed.

Let us formalise substitutability and complementarity. With this aim, it is necessary to formalise when two norms concurrently apply. First, let us consider the notion of joint context. We say that two agents share a joint context in a given state if they can detect (perceive) one another. Formally,

Definition 33 (Joint context)**.** *Let $s \in S$ be a state, and $ag, ag' \in Ag$ two agents with contexts $ctxt(ag, s)$, $ctxt(ag', s)$ and scopes $sctxt(ag, s)$, $sctxt(ag', s)$.*

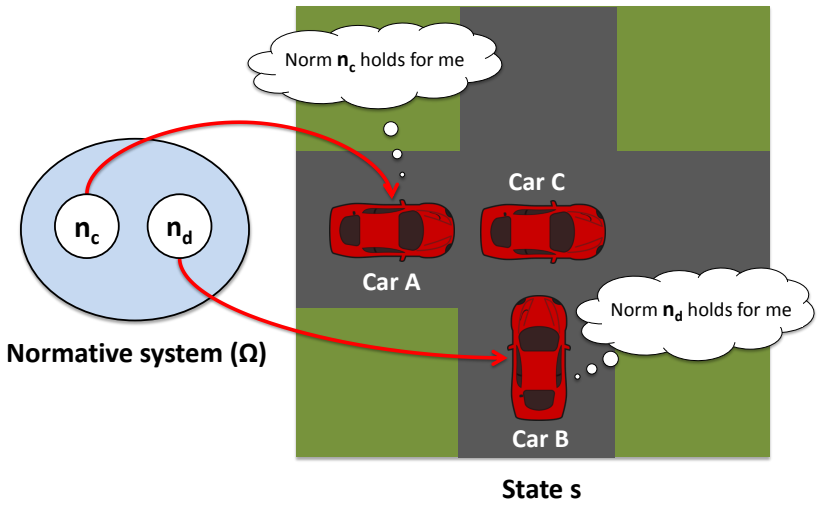

Figure 7.2: Example of two complementary norms n_c, n_d.

		n'	
		Fulfilment	Infringement
n	Fulfilment	No conflicts	Conflicts
	Infringement	Conflicts	Conflicts

Table 7.2: Outcomes of two complementary norms that are jointly fulfilled/infringed.

We say that $\langle ctxt(ag, s), ctxt(ag', s)\rangle$ *is a joint context shared by the agents iff* $ag' \in scope(ag, s)$ *and* $ag \in scope(ag', s)$.

Then, when two agents share a joint context and there are two norms such that there is one norm that applies to each agent, we say that the norms concurrently apply. Formally,

Definition 34 (Concurrent applicability)**.** *Let* $s \in S$ *be a state,* $ag, ag' \in Ag$ *two agents that share a joint context* $\langle ctxt(ag, s), ctxt(ag', s)\rangle$*, and* $n = \langle \varphi, \theta(ac)\rangle$*,* $n' = \langle \varphi', \theta'(ac')\rangle$ *two different norms. We say that* n*,* n' *concurrently apply in the joint context iff* $ctxt(ag, s) \models \varphi$ *and* $ctxt(ag', s) \models \varphi'$.

Let us now formalise the notion of substitutability. It is worth noticing, though, that here it is considered that substitutability only holds between grounded norms, namely between norms with grounded terms (see Definition 21 in Section 5.2). Thus, for example, norms $n_a, \ldots, n_d$ described above are grounded, since all the terms in their precondition are grounded in the taxonomy depicted in Figure 6.1. Then, we say that two grounded norms that concurrently

apply are substitutable in a particular state when only one of them is required to avoid a transition to an undesirable state (namely, a state containing conflicts). Formally,

Definition 35 (Substitutability)**.** *Let $s \in S$ be a state, $C \subseteq S$ a set of undesirable states, and $ag, ag' \in Ag$ two agents that share a joint context $\langle ctxt(ag, s), ctxt(ag', s)\rangle$. Let n, n' be two different, specific norms such that n applies to ag and n' applies to ag', and hence concurrently apply in the agents' joint context. Let $\langle s, A, s'\rangle$ be the transition that results after ag and ag' perform actions $ac \in A$ and $ac' \in A$ respectively. Norms n, n' are substitutable in s iff the following conditions hold: (i) a conflict occurs ($s' \in C$) when both agents infringe their norms; and (ii) no conflict occurs ($s' \notin C$) when at least one of the agents fulfils its applicable norm.*

The substitutability relationship was illustrated above with two norms n_a, n_b (Figure 7.1). Notice that both norms concurrently apply together in a situation in which (i) two cars perceive each other; and (ii) each norm applies to only one of the cars. In that situation, collisions can be avoided by employing only one of the norms.

From the definition above, we say that two norms are substitutable in a set of states if they are substitutable at *each state*. Consider now all the states of a system where two norms concurrently apply. We say that the two norms are *fully substitutable* if they are substitutable in all those states. Particularly, this chapter focuses on detecting substitutability, but not on detecting fully substitutable norms. We notice that the substitutability relationship is *irreflexive*, and *symmetric*, but *non-transitive*.

Now, let us formalise the notion of complementarity. Two norms that concurrently apply are complementary in a particular state when the two of them are required to avoid a transition to an undesired state. Formally,

Definition 36 (Complementarity)**.** *Let $s \in S$ be a state, $C \subseteq S$ a set of undesired states, and $ag, ag' \in Ag$ two agents that share a joint context $\langle ctxt(ag, s), ctxt(ag', s)\rangle$. Let n, n' be two different, specific norms such that n applies to ag and n' applies to ag', and hence concurrently apply in the agents' joint context. Say that $\langle s, A, s'\rangle$ is the transition that results after ag and ag' perform actions $ac \in A$ and $ac' \in A$ respectively. Then, norms n, n' are complementary in s iff the following conditions hold: (i) a conflict occurs ($s' \in C$) when at least one of the agents infringes its applicable norm; and (ii) no conflict occurs ($s' \notin C$) when both agents fulfil their norms.*

From the definition above, we say that two norms are complementary in a set of states if they are complementary at each state. We say that the two norms are *fully complementary* if they are complementary in all the states of the system where they concurrently apply. Like substitutability, the complementarity relationship is irreflexive, and symmetric, but non-transitive.

At this point, we know that norms may have generalisation relationships, as well as substitutability and complementarity relationships. Against this background, it is natural to ask if there exists any relationship between generalisation,

substitutability and complementarity. The answer is provided in the observation below.

Observation 1. *Generalisation, substitutability, and complementarity are mutually exclusive relationships.*

To see this, first observe that substitutability and complementarity are mutually exclusive relationships: two norms that are substitutable cannot be complementary at the same time, and the other way around. This directly follows from the conditions in definitions 35 and 36. Furthermore, generalisation and substitutability are mutually exclusive. This also comes from the definition of generalisation and substitutability. According to definition 35, substitutability only holds between specific norms, whereas generalisation always requires at least a non-specific norm. From this, it follows that generalisation and substitutability cannot hold at the very same time between two norms. Following the same line of reasoning, since complementarity only holds between specific norms, generalisation and complementarity are also mutually exclusive. This observation tells us that IRON (Chapter 5) and SIMON (Chapter 6), which only learned generalisation relationships, never learned any substitutability relationship nor any complementarity relationship.

7.3 Synthesising liberal normative systems

This section illustrates how substitutability and complementarity relationships between norms can be detected, and how these relationships can be exploited to synthesise liberal normative systems. The remainder of this section is organised as follows. Section 7.3.1 introduces a normative network to represent substitutability and complementarity relationships between norms. Thereafter, Section 7.3.2 details how to detect substitutability and complementarity so that Section 7.3.3 illustrates how to exploit such relationships to synthesise normative systems that are as liberal as possible, while preserving compactness to the greatest possible extent.

7.3.1 Representing norm relationships

LION represents synthesised norms and their relationships as nodes and edges in a normative network. With this aim, LION considers the normative network formally defined in Definition 19 (Section 3.4.1), with a set of norms $\mathcal{N}$, and a set $\mathcal{R} \subseteq \mathcal{N} \times \mathcal{N}$ of relationships between norms. Particularly, LION considers three different relationships between norms: generalisation relationships, substitutability relationships, and complementarity relationships. Thus, $\mathcal{R} = \{\mathcal{E}_G, \mathcal{E}_S, \mathcal{E}_C\}$, where $\mathcal{E}_G \subset \mathcal{N} \times \mathcal{N}$ is a set of directed edges standing for generalisation relationships; $\mathcal{E}_S \subset \mathcal{N} \times \mathcal{N}$ is a set of undirected edges standing for substitutability relationships; $\mathcal{E}_C \subset \mathcal{N} \times \mathcal{N}$ is a set of undirected edges standing for complementarity relationships; and $\mathcal{E}_G, \mathcal{E}_S, \mathcal{E}_C$ are pair-wise disjoint.

Figure 7.4 shows a normative network built by LION after detecting substitutability and complementarity relationships between norms. Note that grounded norms n_2 and n_3 hold a substitutability relationship; specific norms n_2 and n_{11} hold a complementarity relationship; and the rest of relationships (directed edges) stand for generalisations – n_{10} generalises $n_3, \ldots, n_6$, and n_8, n_9 at different generalisation levels, and n_7 generalises norms n_1, n_2. Since norms n_7, n_{10}, n_{11} are the only active norms (denoted by white circles), the normative network represents the normative system $\Omega = \{n_7, n_{10}, n_{11}\}$.

7.3.2 Detecting norm relationships

This section describes how LION discovers, at run-time, substitutability and complementarity relationships between grounded norms. Recall from Section 7.2 that two concurrently applicable norms are *substitutable* if no conflicts arise whenever at least one of them is fulfilled. Similarly, they are considered as *complementary* if conflicts arise whenever at least one of them is infringed. LION is an on-line process that iteratively gathers evidences about norms' compliance outcomes to assess whether a pair of norms are either substitutable or complementary. Thus, LION considers that two norms are substitutable if the substitutability conditions of Definition 35 hold for these two norms for a sufficient number of times.The same applies to complementarity. When LION detects that either a substitutability relationship or a complementarity relationship holds between two norms, it represents such relationship in the normative network.

With this aim, LION proceeds as follows. For each pair of concurrently applicable norms n, n' that LION detects (see Definition 34 in Section 7.2), it creates a tuple of finite series

$$\langle \mathcal{FF}^{n,n'}, \mathcal{FI}^{n,n'}, \mathcal{IF}^{n,n'} \rangle$$

that accumulate their frequency in avoiding conflicts whenever they are fulfilled ($\mathcal{F}$) or infringed ($\mathcal{I}$) along time. Formally, these series are:

$$\mathcal{FF}^{n,n'} = \langle ff_1^{n,n'}, \ldots, ff_m^{n,n'} \rangle \in [0..1]^m \quad (7.1)$$

$$\mathcal{FI}^{n,n'} = \langle fi_1^{n,n'}, \ldots, fi_m^{n,n'} \rangle \in [0..1]^m \quad (7.2)$$

$$\mathcal{IF}^{n,n'} = \langle if_1^{n,n'}, \ldots, if_m^{n,n'} \rangle \in [0..1]^m \quad (7.3)$$

where each element $ff_i^{n,n'}$ is a binary value that gathers the evidence related to the i-th time that n and n' were both fulfilled ($\mathcal{FF}$). Specifically, $ff_i^{n,n'}$ is set to 1 when both fulfilments did not lead to a conflict, and set to 0 otherwise. Analogously, series $\mathcal{FI}^{n,n'}$ keeps track of the evidence related to the fulfilment of n and the infringement n', and series and $\mathcal{IF}^{n,n'}$ keeps track of this evidence when n is infringed and n' is fulfilled.

Gathering pair-wise evidences does not cover dependencies with other norms in the current normative system, although accounted conflicts may in fact be caused by third norms. These "noisy" evidences may cause fluctuations in the binary series. Therefore, as it is usually the case for data streams, LION employs

the *cumulative moving average* [Chou, 1969] to smooth out short-term fluctuations and to highlight longer-term trends when detecting substitutability and complementarity. With this aim, for each pair of concurrently applicable norms n, n' that LION detects, it creates a tuple of finite series

$$\langle \mathcal{U}^{\mathcal{FF}}_{n,n'}, \mathcal{U}^{\mathcal{FI}}_{n,n'}, \mathcal{U}^{\mathcal{IF}}_{n,n'} \rangle$$

that accumulate their utility in avoiding conflicts whenever they are fulfilled ($\mathcal{F}$) or infringed ($\mathcal{I}$) along time. Formally, these series are:

$$\mathcal{U}^{\mathcal{FF}}_{n,n'} = \langle u^{\mathcal{FF}}_1, \ldots, u^{\mathcal{FF}}_m \rangle \in \mathbb{N}^m \tag{7.4}$$

$$\mathcal{U}^{\mathcal{FI}}_{n,n'} = \langle u^{\mathcal{FI}}_1, \ldots, u^{\mathcal{FI}}_m \rangle \in \mathbb{N}^m \tag{7.5}$$

$$\mathcal{U}^{\mathcal{IF}}_{n,n'} = \langle u^{\mathcal{IF}}_1, \ldots, u^{\mathcal{IF}}_m \rangle \in \mathbb{N}^m \tag{7.6}$$

where each element $u^{\mathcal{FF}}_i$ represents the evolution of the utility in avoiding conflicts of a pair of norms n, n' that are both fulfilled. Specifically, each value $u^{\mathcal{FF}}_i$ is computed as follows:

$$u^{\mathcal{FF}}_i = \frac{\sum\limits_{u_j \in \mathcal{U}^{\mathcal{FF}}_{n,n'}, 1 \leq j \leq i} u^{\mathcal{FF}}_j}{i} \tag{7.7}$$

Analogously, the elements in $\mathcal{U}^{FI}_{n,n'}$ and $\mathcal{U}^{IF}_{n,n'}$ are computed as follows.

$$u^{\mathcal{FI}}_i = \frac{\sum\limits_{u_j \in \mathcal{U}^{\mathcal{FI}}_{n,n'}, 1 \leq j \leq i} u^{\mathcal{FI}}_j}{i} \tag{7.8}$$

$$u^{\mathcal{IF}}_i = \frac{\sum\limits_{u_j \in \mathcal{U}^{\mathcal{IF}}_{n,n'}, 1 \leq j \leq i} u^{\mathcal{IF}}_j}{i} \tag{7.9}$$

LION considers that two concurrently applicable norms are substitutable if they perform similarly in avoiding conflicts whenever both of them are fulfilled, and when one of them is fulfilled, and the other is infringed. Specifically, it considers that two norms n, n' are substitutable iff their three utility series $(\mathcal{U}^{\mathcal{FF}}_{n,n'}, \mathcal{U}^{\mathcal{FI}}_{n,n'}, \mathcal{U}^{\mathcal{IF}}_{n,n'})$ *are similar*. Since the Euclidean distance is one of the most used and efficient time series (dis)similarity measures [Wang et al., 2013], LION assesses the similarity between these series by means of the *averaged Euclidean distance* as follows:

$$Distance(\mathcal{U}^{\mathcal{FF}}_{n,n'}, \mathcal{U}^{\mathcal{FI}}_{n,n'}) = \frac{\sqrt{\sum\limits_{i=1}^{l_{min}} (u^{\mathcal{FF}}_i - u^{\mathcal{FI}}_i)^2}}{l_{min}} \tag{7.10}$$

being l_{min} the minimum length of series $\mathcal{U}^{\mathcal{FF}}_{n,n'}$ and $\mathcal{U}^{\mathcal{FI}}_{n,n'}$, namely $l_{min} = min(|\mathcal{U}^{\mathcal{FF}}_{n,n'}|, |\mathcal{U}^{\mathcal{FI}}_{n,n'}|)$.

Next, LION determines that two concurrently applicable norms n and n' are substitutable iff: (i) it accumulates a minimum of evidence regarding the outcomes of the concurrent application of both norms; and (ii) the distance between series $\mathcal{U}^{FF}_{n,n'}$ and $\mathcal{U}^{FI}_{n,n'}$ as well as the distance between series $\mathcal{U}^{FF}_{n,n'}$ and $\mathcal{U}^{IF}_{n,n'}$ are both *below* a given threshold $\alpha^{\mathcal{U}}_{sim}$, such that $\alpha^{\mathcal{U}}_{sim} > 0$. This amounts to verifying whether the following conditions hold:

$$|\mathcal{U}^{\mathcal{FF}}_{n,n'}| \geq e_{min}, |\mathcal{U}^{\mathcal{FI}}_{n,n'}| \geq e_{min}, |\mathcal{U}^{\mathcal{FI}}_{n,n'}| \geq e_{min} \tag{7.11}$$

$$Distance(\mathcal{U}^{\mathcal{FF}}_{n,n'}, \mathcal{U}^{\mathcal{FI}}_{n,n'}) \leq \alpha^{\mathcal{U}}_{sim} \tag{7.12}$$

$$Distance(\mathcal{U}^{\mathcal{FF}}_{n,n'}, \mathcal{U}^{\mathcal{IF}}_{n,n'}) \leq \alpha^{\mathcal{U}}_{sim} \tag{7.13}$$

where e_{min} is the minimum number of evidences that are required to assess if two concurrently applicable norms are substitutable of complementary.

Analogously, LION considers that two concurrently applicable norms are complementary if they perform better in avoiding conflicts when they are both fulfilled than when one of them is fulfilled and the other infringed. Specifically, it considers that two norms n, n' are complementary if series $\mathcal{U}^{\mathcal{FF}}_{n,n'}$ is *above* series $\mathcal{U}^{\mathcal{FI}}_{n,n'}$ and $\mathcal{U}^{\mathcal{IF}}_{n,n'}$. This amounts to verifying if condition 7.11 above holds, along with the following conditions below:

$$Average(\mathcal{U}^{\mathcal{FF}}_{n,n'}) - Average(\mathcal{U}^{\mathcal{FI}}_{n,n'}) > \alpha^{\mathcal{U}}_{sim} \tag{7.14}$$

$$Average(\mathcal{U}^{\mathcal{FF}}_{n,n'}) - Average(\mathcal{U}^{\mathcal{IF}}_{n,n'}) > \alpha^{\mathcal{U}}_{sim} \tag{7.15}$$

where $Average(\mathcal{U}^{\mathcal{FF}}_{n,n'})$ is the average of the utility values in series $\mathcal{U}^{\mathcal{FF}}_{n,n'}$, and is computed as follows:

$$Average(\mathcal{U}^{\mathcal{FF}}_{n,n'}) = \frac{\sum_{i=1}^{m} u_i^{\mathcal{FF}}}{m} \tag{7.16}$$

being m the number of values in series $\mathcal{U}^{\mathcal{FF}}_{n,n'}$.

Whenever LION detects that two norms are substitutable or complementary, it establishes the corresponding relationship in the normative network.

7.3.3 Exploiting norm relationships

As previously detailed, LION exploits norm relationships to discard (deactivate) norms involved in substitutability relationships. During this process, it aims at preserving the compactness of a normative system to the greatest possible extent, while safeguarding a normative system's performance in avoiding conflicts. The heuristic employed to achieve these outcomes is simple: given two substitutable norms, choose to deactivate the norm that causes less *compactness loss*, provided that it is not part of any complementarity relationships. The compactness loss of a norm is related to the decrement in compactness that the normative system would suffer in case a norm was deactivated – recall from

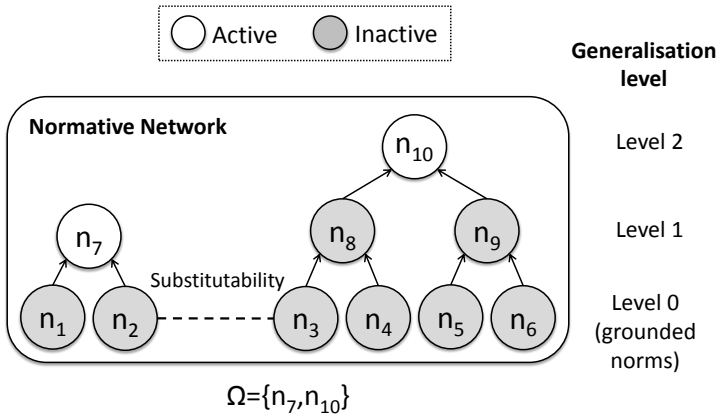

Figure 7.3: A normative network containing generalisation and substitutability relationships: solid arrows stand for (non-symmetric) generalisation relationships; and dashed lines for substitutability (symmetric) relationships.

Section 6.4 that deactivating a norm leads to specialising its ancestors in the hierarchy, and hence to losing compactness. Therefore, LION chooses to deactivate the norm that minimises such loss. Moreover, it does not deactivate any norm that is complementary with another norm, since complementary norms are all necessary to avoid conflicts.

When LION detects two substitutable norms n, n', it chooses one of them to be deactivated as follows. First, it considers the ancestors of n and n' to determine their corresponding compactness loss. Thereafter, it chooses the norm with the lowest compactness loss value, provided that it has no complementarity relationships. As an example, consider normative network in Figure 7.3 representing the normative system $\Omega = \{n_7, n_{10}\}$. It contains two substitutable norms n_2, n_3 that are generalised by other norms. In particular, n_2 is represented by an active norm n_7 whose generalisation level is 1, and n_3 is represented by an active norm n_{10} whose generalisation level is 2. In short, the generalisation level of a norm represents its *height* in the generalisation hierarchy, namely how general it is. Therefore, n_{10} is more general than n_7, and thus compactly represents a greater number of norms. At this point, LION must decide whether to discard either n_2 or n_3 because they are substitutable. Since deactivating n_3 would imply specialising n_{10}, LION considers that n_3 has a higher specialisation cost than n_2, and thus, LION will deactivate norm n_2 instead.

Specifically, LION uses equation 7.17 to compute the compactness loss of a norm n as the sum of the generalisation degrees of its ancestors in the normative network.

$$\mathcal{C}_{loss}(n, NN) = \sum_{n' \in ancestors(n)} g_{deg}(n', NN) \tag{7.17}$$

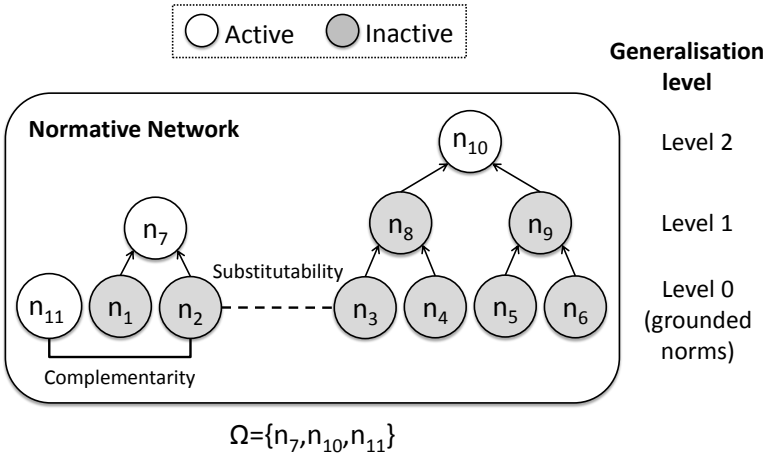

Figure 7.4: A normative network containing generalisation, substitutability and complementarity relationships: solid arrows stand for (non-symmetric) generalisation relationships; dashed lines for substitutability (symmetric) relationships; and solid lines for complementarity (symmetric) relationships.

where $g_{deg}(n', NN)$ is the generalisation degree of a norm n' in the normative network, which is computed by means of equation 7.18:

$$g_{deg}(n, NN) = |children(n)| \cdot \kappa^{level(n)} \tag{7.18}$$

where: *children(n)* is a function that assesses the norms that n directly generalises (those norms n has an incoming generalisation relationship with); *level(n)* is the generalisation level of n; and κ is a constant factor, where $\kappa > 1$. Note that $\mathcal{C}_{loss}$ is finite, since the generalisation relationship is transitive and asymmetric, and thus does not allow cycles.

Figure 7.3 helps to illustrate these computations. The compactness loss of norm n_3 is the sum of the generalisation degrees of its ancestors (namely, n_8 and n_{10}). In particular, since n_8 has a generalisation level of 1, and it has two children (n_3, n_4), its generalisation degree is $g_{deg}(n_8, NN) = 2\kappa$. Analogously, norm n_{10} has a generalisation degree $g_{deg}(n_{10}, NN) = 2\kappa^2$ because its generalisation level is 2 and it has 2 children (n_8 and n_9). As a result, $\mathcal{C}_{loss}(n_3, NN) = 2\kappa + 2\kappa^2$, which is greater than $\mathcal{C}_{loss}(n_2, NN) = 2\kappa$. Therefore, LION chooses to deactivate n_2 (and hence, to specialise n_7).

Regarding complementarity relationships, Figure 7.4 depicts an alternative situation during the synthesis process where the normative network contains, not only substitutable norms n_2, n_3, but also a complementarity relationship between norm n_2 and norm n_{11}. In this case, since LION prioritises the preservation of complementary norms, it will keep n_2 active and will choose to deactivate n_3.

Algorithm 19 details how LION exploits norm relationships to choose which

ALGORITHM 19: Function *chooseToDeactivate*.

Input : n, n', NN, κ
Output: *normToDeactivate*

```
1  normToDeactivate ← null;
2  n_Complementary ← hasComplementarityRelationships(n, NN);
3  n'_Complementary ← hasComplementarityRelationships(n', NN);
4  if n_Complementary and not n'_Complementary then
5      normToDeactivate ← n';
6  else if n'_Complementary and not n_Complementary then
7      normToDeactivate ← n;
8  else if not n_Complementary and not n'_Complementary then
9      if C_loss(n, NN, κ) > C_loss(n', NN, κ) then
10         normToDeactivate ← n';
11     else if C_loss(n, NN, κ) < C_loss(n', NN, κ) then
12         normToDeactivate ← n;
13     else
14         normToDeactivate ← chooseRandomly(n, n');
15 return normToDeactivate
```

norm to deactivate from a pair of substitutable norms. It receives as inputs two substitutable norms n, n', a normative network NN, and a constant k to compute the *generalisation degree* of a norm in the normative network (cf. Equation 7.18). First, it checks whether n and n' are complementary to other norms in the normative network (lines 2–3). Thereafter, it checks the following conditions:

1. Norm n is complementary, and n' is not. In this case, it chooses norm n' to be deactivated (lines 4–5).

2. Norm n' is complementary, and n is not. Therefore, it chooses norm n to be deactivated (lines 6–7).

3. Neither norm n nor norm n' are complementary. Then, it chooses to deactivate the norm with the lowest compactness loss (lines 8–12). Particularly, in case their compactness losses are equal, then LION randomly chooses to deactivate one of them (line 14).

4. Both norm n and norm n' are complementary. In this case, LION does not choose any norm to deactivate (i.e., *normToDeactivate=null*).

Then, LIONreturns the norm that has been chosen to be deactivated (line 15). Note that LION checks complementarity relationships before analysing compactness loss. In this way, LION prioritizes the preservation of a normative system's performance over the preservation of compactness. In fact, LION may choose to deactivate the norm with the highest compactness loss if it was necessary to preserve complementary norms.

7.4 Enhancing norm evaluation

As introduced in Section 7.1, LION incorporates a novel norm evaluation approach to overcome SIMON's norm evaluation drawbacks. As SIMON, LION evaluates a norm's performance in terms of its effectiveness and its necessity, computed as its ratio of successful fulfilments and harmful infringements along time (see Definitions 13 and 16 in Section 3.2). However, unlike SIMON, LION (1) does not employ reinforcement learning to compute a norm's punctual effectiveness and necessity; and (2) does not require the computation of a norm's effectiveness and necessity performance ranges. Instead, it computes a norm's effectiveness and necessity as its averaged *punctual* effectiveness and necessity evaluations along time.

Whenever a norm n is fulfilled by some agent at time t, LION computes its punctual effectiveness evaluation at time t as its ratio of successful fulfilments at time t. With this aim, it employs formula ev_{eff} below.

$$ev_{\mathit{eff}}(n,t) = \begin{cases} \dfrac{sf_t^n}{sf_t^n + hf_t^n} & \text{if } \mathit{fulfilled}(n,t) = \mathit{true} \\ \\ \bot & \text{otherwise} \end{cases} \tag{7.19}$$

where $n \in \mathcal{N}$, $t \in \mathbb{N}$ (being $\mathcal{N}$ the set of norms in the normative network), and $sf_t^n \in \mathcal{SF}^n$ and $hf_t^n \in \mathcal{HF}^n$ are the number of successful fulfilments and harmful fulfilments of n at time t, respectively (see Section 3.3.3). Specifically, function ev_{eff} computes a punctual effectiveness as the ratio of successful fulfilments of n at time t iff n has been fulfilled at time t, namely if *fulfilled(n,t) = true*, and returns an *undefined* reward value $\bot$ if n has not been fulfilled at time t. Function *fulfilled* was introduced in Definition 5.2 from Section 5.6, but we recall it here for convenience. It returns *true* if there are either successful fulfilments or unsuccessful fulfilments of n at time t. Formally:

$$\mathit{fulfilled}(n,t) = \begin{cases} \mathit{true} & \text{if } sf(n,t) + hf(n,t) > 0 \\ \mathit{false} & \text{otherwise} \end{cases} \tag{7.20}$$

Then, LION computes the q-period cumulative moving average of the punctual effectiveness evaluations of n at time t. With this aim, let us define a set Eff_l^n of defined effectiveness values of n from a given time l:

$$\mathit{Eff}_l^n = \{ev_{\mathit{eff}}(n,j) \,|\, l \leq j \leq m, ev_{\mathit{eff}}(n,j) \neq \bot\} \tag{7.21}$$

Then, we consider an index $i(q)$ such that $|\mathit{Eff}_{i(q)}^n| = q$ and $1 \leq i(q) \leq m$. Thus, set $\mathit{Eff}_{i(q)}^n$ contains the latest q defined effectiveness values of n. LION computes the effectiveness of norm n at time t as the average of the values in $\mathit{Eff}_{i(q)}^n$ at time t.

$$\mu_{\mathit{eff}}(n,t) = \frac{\sum\limits_{ev_{\mathit{eff}} \in \mathit{Eff}_{i(q)}^n} ev_{\mathit{eff}}}{q} \tag{7.22}$$

where $n \in \mathcal{N}$, $q \in \mathbb{N}$, and $1 \leq q \leq m$.

Analogously, whenever a norm n is infringed by some agent at time t, LION computes its punctual necessity evaluation at time t as its ratio of harmful infringements at time t. With this aim, it employs formula ev_{nec} below.

$$ev_{nec}(n,t) = \begin{cases} \dfrac{hi_t^n}{hi_t^n + si_t^n} & \text{if } \mathit{infringed}(n,t) = \mathit{true} \\ \\ \perp & \text{otherwise} \end{cases} \tag{7.23}$$

where $n \in \mathcal{N}$, $t \in \mathbb{N}$, and $hi_t^n \in \mathcal{HI}^n$ and $si_t^n \in \mathcal{SI}^n$ are the number of harmful infringements and successful infringements of n at time t, respectively (see Section 3.3.3). More specifically, function ev_{nec} computes a necessity reward as the ratio of harmful infringements of n at time t iff n has been infringed at time t, and returns an *undefined* reward value $\perp$ if n has not been infringed at time t. Function *infringed* (introduced in Equation 5.5 from Section 5.6) returns *true* if there are either harmful infringements or successful infringements of n at time t. Formally:

$$\mathit{infringed}(n,t) = \begin{cases} \mathit{true} & \text{if } hi(n,t) + si(n,t) > 0 \\ \mathit{false} & \text{otherwise} \end{cases} \tag{7.24}$$

Then, LION computes the q-period cumulative moving average of the punctual necessity evaluations of n at time t. With this aim, let us define a set Nec_l^n of defined necessity values of n from a given time l:

$$Nec_l^n = \{ev_{nec}(n,j) \,|\, l \leq j \leq m, ev_{nec}(n,j) \neq \perp\} \tag{7.25}$$

Then, we consider an index $i(q)$ such that $|Nec_{i(q)}^n| = q$ and $1 \leq i(q) \leq m$. Thus, set $Nec_{i(q)}^n$ contains the latest q defined effectiveness values of n. LION computes the effectiveness of norm n at time t as the average of the values in $Nec_{i(q)}^n$ at time t.

$$\mu_{nec}(n,t) = \frac{\sum\limits_{ev_{nec} \in Nec_{i(q)}^n} ev_{nec}}{q} \tag{7.26}$$

where $n \in \mathcal{N}$, $q \in \mathbb{N}$, and $1 \leq q \leq m$.

7.4.1 Detecting norms' performances

Let us now see how LION detects that a norm *performs well* in regulating conflicts, and how it detects that a norm *under-performs* in regulating conflicts. On the one hand, LION considers that a norm performs well whenever it is sufficiently effective *and* necessary to be included in the normative system. On the other hand, it considers that a norm under-performs if it is not effective *or* necessary enough to be included in the normative system.

Detecting good performance

LION considers that a norm n performs well at a given time t if *both* conditions below are satisfied:

1. Its effectiveness at time t ($\mu_{\mathit{eff}}(n,t)$) is *above* a certain *effectiveness uncertainty area*, which represents a range of values above which a norm is considered to perform well in terms of effectiveness. LION implements this area as a value (ϵ) above and below the effectiveness threshold α_{eff} included in LION's evaluation criteria ($\mathcal{EC}$, see Section 5.6). Formally, an effectiveness uncertainty area is a tuple $\langle \alpha^{+}_{\mathit{eff}}, \alpha^{-}_{\mathit{eff}} \rangle$, being $\alpha^{+}_{\mathit{eff}}$ the higher boundary of the uncertainty area, and $\alpha^{-}_{\mathit{eff}}$ the lower boundary of the uncertainty area, which are computed as follows:

$$\alpha^{+}_{\mathit{eff}} = \alpha_{\mathit{eff}} + \epsilon \tag{7.27}$$

$$\alpha^{-}_{\mathit{eff}} = \alpha_{\mathit{eff}} - \epsilon \tag{7.28}$$

2. Its necessity at time t ($\mu_{\mathit{eff}}(n,t)$) is *above* a certain *necessity uncertainty area*, which represents a range of values above which a norm is considered to perform well in terms of necessity. LION implements this area as a value (ϵ) above and below the necessity threshold α_{nec} included in LION's evaluation criteria ($\mathcal{EC}$, see Section 5.6). Formally, a necessity uncertainty area is a tuple $\langle \alpha^{+}_{nec}, \alpha^{-}_{nec} \rangle$, being α^{+}_{nec} the higher boundary of the uncertainty area, and $\alpha^{-}_{\mathit{eff}}$ the lower boundary of the uncertainty area, which are computed as follows:

$$\alpha^{+}_{nec} = \alpha_{nec} + \epsilon \tag{7.29}$$

$$\alpha^{-}_{nec} = \alpha_{nec} - \epsilon \tag{7.30}$$

Thus, LION considers that a norm n performs well at time t if its effectiveness and its necessity at time t are above the higher boundary of the effectiveness and necessity uncertainty areas, respectively. Formally, it corresponds to checking if *both* conditions below are satisfied:

$$\mu_{\mathit{eff}}(n,t) > \alpha^{+}_{\mathit{eff}} \tag{7.31}$$

$$\mu_{nec}(n,t) > \alpha^{+}_{\mathit{eff}} \tag{7.32}$$

Note that, unlike SIMON, LION no longer decides that a norm performs well based on some generalisation thresholds (thresholds $\alpha^{gen}_{\mathit{eff}}, \alpha^{gen}_{nec}$ described in Section 5.6.1). Instead, LION considers that a norm performs well if it is sufficiently effective and necessary to be active in the normative network (and thus, to be included in the normative system).

Detecting under performance

LION considers that a norm n under-performs at a given time t it its effectiveness *or* its necessity at time t is *below* the lower boundary of respective uncertainty areas. Formally, it corresponds to checking if *at least one* of the two conditions below are satisfied:

$$\mu_{\mathit{eff}}(n,t) < \alpha^{+}_{\mathit{eff}} \tag{7.33}$$

$$\mu_{nec}(n,t) < \alpha^{+}_{\mathit{eff}} \tag{7.34}$$

Note that this approach to detecting norms' performances allows LION to be more stable, since is more resilient to oscillations in the norms' effectiveness and necessities. Specifically, these uncertainty areas allow LION not to continuously make decisions regarding a norm's performance whenever its effectiveness or its necessity oscillate above and below thresholds α_{eff} and α_{nec}, respectively.

7.5 LION's synthesis strategy

Next, LION's synthesis strategy is described. Based on SIMON, LION incorporates the computational capabilities to detect and to exploit norm synergies during the norm synthesis process. More precisely, LION identifies when norms are either substitutable or complementary, book-keeping these relationships between norms in the normative network. Then, it exploits this knowledge (together with the generalisation relationships) to synthesise *liberal* normative systems that are *effective* to avoid conflicts, while being as *compact* as possible. This requires carefully handling the interplay between all norm relationships at synthesis time.

As SIMON, LION iteratively observes agents' interactions within a MAS, and subsequently carries out the three synthesis stages of norm generation, norm evaluation, and norm refinement. Particularly, LION performs norm generation as described in Section 4.6.1. Crucially, the norm evaluation and norm refinement stages are novel. During norm evaluation, LION evaluates norms as described in Section 7.4, and gathers evidences about concurrently applicable norms and their utility in avoiding conflicts. Thereafter, during norm refinement, it generalises norms as described in 6.3, and specialises under-performing norms as described in 6.4. Additionally, it detects substitutability and complementarity relationships between norms as described in Section 7.3.2, and exploits these relationships as described in Section 7.3.3.

Algorithm 20 describes LION's norm synthesis strategy, which receives as input a tuple of a description of the previous state of a MAS (d_s) and a description of the current MAS state (d_{s_c}). Eventually, it outputs a normative system (Ω) aimed at regulating agents' behaviours. To perform norm synthesis, LION considers the same globally-accessible elements considered in Section 6.5. That is, a normative network, a collection of operators, a set of domain-dependent elements, a set of domain-independent settings, and a set of additional synthesis inputs. Additionally, it considers the following extra synthesis parameters:

- A constant e_{min} (cf. Section 7.3.2) that sets the minimum number of evidences in series $\mathcal{FF}^{n,n'}, \mathcal{FI}^{n,n'}, \mathcal{IF}^{n,n'}$ to detect substitutability and complementarity between norms (see Section 7.3.2)

- A threshold ($\alpha^{\mathcal{U}}_{sim}$) to determine if the series in Equations 7.4, 7.5 and 7.6 are similar (see Section 7.3.2).

- A constant κ to compute the *generalisation degree* of a norm in the normative network (see Section 7.3.3)

- A constant ϵ to compute the effectiveness and necessity uncertainty areas described in Section 7.4.1.

ALGORITHM 20: LION's synthesis strategy

Input : $\langle d_s, d_{s_c} \rangle$
Output : Ω
Initialisations: $\mathcal{NCO} \leftarrow \emptyset$, $\mathcal{P} \leftarrow \emptyset$, $\mathcal{PCO} \leftarrow \emptyset$, $\mathcal{U} \leftarrow \emptyset$

[1] $NN \leftarrow$ `normGeneration(`$\langle d_s, d_{s_c} \rangle, \mathcal{P}$`)`;
[2] $(\mathcal{P}, \mathcal{U}) \leftarrow$ `normEvaluation(`$\langle d_s, d_{s_c} \rangle, \mathcal{NCO}, \mathcal{P}, \mathcal{PCO}, \mathcal{U}$`)`;
[3] $NN \leftarrow$ `normRefinement(`$\mathcal{NCO}, \mathcal{P}, \mathcal{PCO}, \mathcal{U}$`)`;
[4] $\Omega \leftarrow \{n \in NN \mid \delta(n) = active\}$;
[5] **return** Ω

LION considers the data structures described in Section 5.7 to keep track of norms' compliance outcomes ($\mathcal{NCO}$), norms' effectiveness and necessities ($\mathcal{P}$), and the following additional structures:

1. A structure ($\mathcal{PCO}$) to keep track of the compliance outcomes of each *pair* of concurrently applicable norms (namely, series $\mathcal{FF}^{n,n'}, \mathcal{FI}^{n,n'}, \mathcal{IF}^{n,n'}$, see Section 7.3.2).

2. A structure ($\mathcal{U}$), to keep track of the *utility* in avoiding conflicts of each *pair* of concurrently applicable norms whenever they are jointly fulfilled, or whenever one is fulfilled and the other infringed (see Equations 7.4, 7.5, 7.6 in Section 7.3.2).

LION performs norm synthesis by first carrying out norm generation (line 1), in which it detects conflicts and creates norms as described in Algorithm 3 from Section 4.6.1. Next, it carries out norm evaluation (line 2), in which it evaluates norms' individual performances, and the utility in avoiding conflicts of each pair of concurrently applicable norms. Finally, during norm refinement (line 3), it specialises and generalises norms as SIMON does. Additionally, LION detects substitutability and complementarity relationships, and exploits these relationships to remove substitutable norms. The algorithm ends by returning the (possibly updated) normative system, which contains the norms that will be communicated to the agents (lines 4–5). Subsequent sections detail LION's *normEvaluation* and *normRefinement* functions.

7.5.1 Norm evaluation

Algorithm 21 depicts LION's *normEvaluation* function. It receives as input the compliance outcomes of individual norms ($\mathcal{NCO}$), the individual performances of each norm ($\mathcal{P}$), the compliance outcomes of each pair of concurrently applicable norm ($\mathcal{PCO}$), and the utilities in avoiding conflicts of each pair of norms that are concurrently fulfilled and infringed ($\mathcal{U}$). It starts by retrieving the compliance

outcomes of each norm that has been fulfilled and infringed in the transition to the current MAS state (line 1). Thereafter, it retrieves the compliance outcomes of *each pair of norms* that have been concurrently fulfilled and infringed during the state transition (line 2). Then, it evaluates each fulfilled and infringed norm in terms of its effectiveness and necessity as described in Section 7.4 (line 3). Next, in line 4, it computes the utility in avoiding conflicts of each pair of norms that have been concurrently fulfilled and infringed as described in Equations 7.7, 7.8, and 7.9 from Section 7.3.2. Finally, in line 5, it updates the utilities of each case solution in the case base as BASE does (see Algorithm 4 in Section 4.3.2).

ALGORITHM 21: LION's *normEvaluation* function

Input : $\langle d_s, d_{s_c} \rangle, \mathcal{NCO}, \mathcal{P}, \mathcal{PCO}, \mathcal{PU}$
Output: $\mathcal{P}, \mathcal{PU}$

[1] $\mathcal{NCO} \leftarrow$ `getComplianceOutcomes(`$\langle d_s, d_{s_c} \rangle, \mathcal{NCO}$`)`;
[2] $\mathcal{PCO} \leftarrow$ `getPairsComplianceOutcomes(`$\langle d_s, d_{s_c} \rangle, \mathcal{PCO}$`)`;
[3] $\mathcal{P}$ $\leftarrow$ `evaluateNorms(`$\mathcal{NCO}, \mathcal{P}$`)`;
[4] $\mathcal{PU}$ $\leftarrow$ `evaluatePairs(`$\mathcal{PCO}, \mathcal{PU}$`)`;
[5] $\mathcal{CB}$ $\leftarrow$ `updateCases(`$\mathcal{P}$`)`;
[6] **return** $(\mathcal{P}, \mathcal{PU})$;

7.5.2 Norm refinement

LION's *normRefinement* function is depicted in Algorithm 22. First, it performs SIMON's norm refinement operations to:

1. *Specify under-performing grounded norms.* During this step, LION revises over-generalisations by specifying general norms that under-perform (lines 1-3). More details about this particular process can be found in Section 6.4.

2. *Specialise under-performing norms.* Next, LION deactivates norms that under-perform, along with their ancestors (lines 5–6). Particularly, LION detect norms' under-performance as described in Section 5.6.2, and to deactivate a norm it employs function *deactivateUp*, depicted in Algorithm 12 from Section 5.7.3.

3. *Generalise norms.* Then, LION generalises norms that do not under-perform (lines 7–8). With this aim, it invokes function *generaliseUp*, described in Algorithm 18 from Section 6.5.

Thereafter, LION proceeds to detect substitutability and complementarity relationships. With this aim, it invokes functions *detectSubstitutability* (line 9) and *detectComplementarity* (line 10). Such functions detect substitutability and complementarity relationships between pairs of norms as described in Section 7.3.2, and keep track of such relationships in the normative network.

Once LION has detected substitutability and complementarity, it proceeds to exploit these relationships. With this aim, function *getSubstitutableNorms* (line 11) retrieves pairs of norms that are substitutable and are both included in the normative system. That is, substitutable norms that are both either active in the normative network, or implicitly represented by an active ancestor. For each pair of norms satisfying these conditions, it first chooses one of them to be deactivated as detailed in Section 7.3.3 (line 13). Briefly, it chooses the norm whose deactivation implies the lowest compactness loss, provided that it is not complementary with other norms. Finally, it deactivates the norm by invoking function *deactivateUp* (line 14), which deactivates the norm and specialises its ancestors in the normative network (see Algorithm 12 in Section 5.7.3).

ALGORITHM 22: LION's *normRefinement* function

Input : $\mathcal{NCO}, \mathcal{P}, \mathcal{PCO}$
Output: NN

```
   /* Norm specification:                                                    */
1  Ω ← {n ∈ NN | δ(n) = active};
2  NRG ← getNegativelyEvaluatedNorms(P);
3  specifyUnderperformingNorms(NRG, NN, Ω);
   /* Norm specialisation and generalisation:                               */
4  for n ∈ getNormsFulfilledInfringedThisState(NCO) do
5      if underPerforms(n, P, α_eff, α_nec, ε) then
6          NN ← deactivateUp(n, NN);
7      else
8          NN ← generaliseUp(n, NN, ⊤, G_M, G_S);
   /* Substitutability and complementarity detection:                       */
9  NN ← detectSubstitutability(PCO);
10 NN ← detectComplementarity(PCO);
11 SN ← getSubstitutableNorms(NN);
   /* Substitutability removal:                                             */
12 foreach ⟨n, n'⟩ ∈ SN do
13     n_subs ← chooseToDeactivate(n, n', NN, κ);
14     NN ← deactivateUp(n_subs, NN);
15 return NN
```

7.6 Analysing normative systems

Next, let us establish how to measure and compare normative systems in terms of the synthesis objectives pursued by LION, namely in terms of *regulative performance*, *compactness*, and *liberality*. On the one hand, the regulative performance

of a normative system can be measured in terms of its effectiveness and its necessity to avoid conflicts. We resort to the measures of effectiveness and necessity defined in Section 5.6.3. These allow us to quantify how good a normative system is at regulating a MAS and how necessary its norms are for regulating. On the other hand, the compactness of a normative system can be measured in terms of the number of terms in the preconditions of its norms. A formal definition of a normative system's compactness was provided in Definition 26 from Section 5.2.

As to liberality, we have seen along this chapter that it can be pursued by reducing the amount of constraints a normative system imposes on agents. In other words, liberality can be understood as attempting to reduce the number of grounded norms a normative system represents, namely those norms whose all terms are grounded in a taxonomy (see Definition 21 in Section 5.2). Let us now formally define the *representation* of a normative system:

Definition 37 (Representation of a normative system)**.** *The representation of a normative system Ω, denoted as $\mathcal{R}(\Omega)$, is the set of grounded norms it represents, namely $\mathcal{R}(\Omega) = \cup_{n \in \Omega} grounded(n)$, where $grounded(n)$ stands for the grounded norms represented by norm n.*

From this definition follows that the fewer the representation of a normative system, namely the fewer the number of grounded norms it represents, the greater a normative system's liberality.

In order to compare whether a normative system is more liberal than another, we rely on the *"more liberal than"* relationship between normative systems introduced by Ågotnes et al. in [Ågotnes et al., 2007]. There, Ågotnes et al. stated that a normative system is *more liberal* (less restrictive) than another normative system if it places fewer constraints on agents. Thus, following a traffic example, we can state that a normative system with a single *"Give way to left"* norm is *more liberal* than one with a single norm *"Give way to any approaching car"*, since it places fewer constraints on the agents. Hence, we say that a normative system Ω is more liberal than another Ω' if Ω represents fewer grounded norms than Ω', and the grounded norms represented by Ω are included in those represented by Ω'.

Definition 38 (Liberality relationship)**.** *Given two normative systems Ω, Ω' such that $\Omega \neq \Omega'$, we say that Ω is more liberal than Ω' iff $\mathcal{R}(\Omega) \subset \mathcal{R}(\Omega')$.*

We will also assess the substitutability of a normative system as the number of substitutability relationships between the norms it contains. Formally:

Definition 39 (Substitutability of a normative system)**.** *Left $\mathcal{E}_S$ be the set of substitutability relationships between the norms in a normative network. The substitutability of a normative system is $S(\Omega) = |(n, n) \in \mathcal{E}_S | n, n \in \Omega|$.*

This provides a measure of the lack of liberality of a normative system: the larger the substitutability of a normative system, the more unnecessary norms it will represent, and the bigger the opportunity to synthesise a more liberal one.

7.7 Empirical evaluation

Next, LION's norm synthesis is empirically evaluated and compared SIMON's along several dimensions. For these experiments, the chosen scenario has been the road traffic scenario. The reason of this is because, in this scenario, SIMON synthesised normative systems that contained substitutable norms. An example was depicted in Table 6.2 from the Section 6.6.2, which contained left-hand side priority norms, and right-hand side priority norms. By contrast, the normative systems that SIMON synthesised in the on-line community (see Section 6.7.2) did not contain any substitutability and complementarity relationships. This is because, in our particular on-line community scenario, users do not have joint contexts, and thus synthesised norms do not have synergies between them (i.e., substitutability or complementarity relationships). Therefore, in the on-line community scenario it does not pay off to invest LION's extra computational cost to detect such relationships.

Thus, the aim now is to analyse whether LION can remove substitutability relationships in the traffic scenario, hence yielding normative systems that contain either norms left-hand side priority norms, or right-hand side priority norms, but not both. Hereafter, Section 7.7.1 details the empirical settings of these experiments, and Section 7.7.2 provides an analysis of LION's empirical results.

7.7.1 Empirical settings

These experiments use the same scenario and experimental settings described in Section 6.6.1. Each simulation employs either LION or SIMONas a norm synthesis strategy, which monitors the simulation and synthesises norms for the cars. At each tick, a car decides whether to fulfil or infringe norms according to some norm infringement probability, which is fixed to 0.3 and is the same for all cars. This means that, on average, 3 of each 10 agents' decisions lead to norm infringements.

LION and SIMON have been configured as described in from Section 5.9.1. These settings are shown in Table 7.3. Thus, LION computes a norm's effectiveness and necessity by considering the 100 last effectiveness and necessity evaluations ($q = 100$), and analogously for SIMON and the computation of its effectiveness and necessity performance ranges. A norm's effectiveness and necessity is initially set to 0.5 ($\mu_{\mathit{eff}}(n.t_0) = 0.5, \mu_{nec}(n, t_0) = 0.5$). We have set low effectiveness and necessity thresholds ($\alpha_{\mathit{eff}} = 0.2, \alpha_{nec} = 0.2$) to deactivate norms only when they perform very poorly. In particular, both LION and SIMON have been configured to perform deep generalisations ($\mathsf{G}_M = \mathit{deep}$), and it generalises three terms at a time ($\mathsf{G}_S = 3$). The reason of this is because, with these settings, SIMON obtained its best results in the road traffic scenario (Section 6.6.2).

As for LION's specific parameters, they are shown in Table 7.4. Thus, to detect substitutability and complementarity, LION requires a minimum of 25 evidences ($e_{min} = 25$) in series $\mathcal{FF}^{n,n'}, \mathcal{FI}^{n,n'}, \mathcal{IF}^{n,n'}$ (see Section 7.3.2). The

Parameter	Description	Value
q	Number of effectiveness values (μ_{eff}) and necessity values (μ_{nec}) considered to compute a norm's effectiveness and necessity ranges (see Section 5.6).	100
k_{eff}, k_{nec}	Default norm effectiveness and necessity values ($\mu_{eff}(n, t_0), \mu_{nec}(n, t_0)$).	0.5
$\alpha_{eff}, \alpha_{nec}$	Thresholds below which a norm is considered to under-perform in terms of its effectiveness/necessity.	0.2
T	Time period considered when assessing convergence (cf. Definition 18 in Section 3.2.2).	5,000
G_M	Generalisation mode.	*deep*
G_S	Generalisation step.	3

Table 7.3: LION's and SIMON's norm synthesis settings in the road traffic scenario.

Parameter	Description	Value
e_{min}	Number of evidences required to detect substitutability and complementarity (see Section 7.3.2).	25
$\alpha^{\mathcal{U}}_{sim}$	Threshold to determine if the series in Equations 7.4, 7.5, and 7.6 are similar.	0.05
κ	Constant to compute the generalisation degree of a norm in the normative network (see Section 7.3.3).	10
ϵ	Constant to compute the effectiveness and necessity uncertainty areas described in Section 7.4.1.	0.05

Table 7.4: LION's specific settings in the road traffic scenario.

effectiveness and necessity uncertainty areas considered by LION are computed as a constant value 0.05 *above* and *below* thresholds α_{eff} and α_{nec} ($\epsilon = 0.05$). Thus, LION considers effectiveness and necessity uncertainty areas to be like $\langle 0.25, 0.15 \rangle$. Moreover, two series are considered as similar if their Euclidean distance is very low. Therefore, $\alpha^{\mathcal{U}}_{sim}$ has been set to 0.05. Finally, the constant factor κ employed to compute the generalisation degree of a norm is set to 10.

Each experiment consists of a set of 200 different simulations for each synthesis strategy, namely LION, and SIMON. Each simulation starts with an empty normative system, and it finishes whenever it reaches 50,000 ticks, or it has converged to a stable normative system. A simulation is assumed to have converged to a normative system, hence solving the norm synthesis problem, if during a 5,000-tick period, the normative system remains unchanged and no unregulated conflicts arise.

7.7.2 Empirical results

Let us now analyse LION's empirical results. First, a micro analysis is provided to show LION's convergence process, which manages to synthesise liberal normative systems from scratch. Thereafter, a liberality analysis illustrates the liberality of the normative systems synthesised by both approaches. Finally, a further analysis quantifies the performance of SIMON's multi-objective synthesis.

Micro analysis: LION's convergence process

We performed a simulation of the road traffic scenario with LION as a norm synthesis strategy. Figure 7.5 depicts, on the x-axis, the normative changes along time (i.e., the time steps in which the the normative network and/or the normative system change) for this single simulation. On the y-axis, the figure shows:

1. The cardinality of the normative system, namely the number of active norms in the normative network.
2. The number of grounded norms the normative system represents.
3. The number of terms that the normative system contains, namely its compactness.
4. The compactness of the normative system, namely the number of terms in the preconditions of its norms (see Definition 26 in Section 5.2).
5. The ratio of unregulated car collisions at a given tick[1].

For the sake of visibility, the cardinality of the normative network is not included, since it reaches a size of 55 norms at tick 3804 (twenty fifth normative change).

At tick 12 (which corresponds to the second normative change), the first collision arises and LION synthesises the first norm. From that tick onwards, LION keeps generating norms when needed, hence increasing the cardinality of both the normative network and the normative system. As a consequence, the number of grounded norms the normative system represents, as well as the number of terms in their norm preconditions, increases. Also, LION generalises norms when possible, trying to reduce the number of terms in the normative system (increasing its compactness). Norm generalisations lead LION to over-generalise at tick 97 (tenth normative change). Specifically, LION synthesises of one single, general norm like the one depicted below

$$n^* \quad : \quad \langle \{ left(anything), front(anything), right(anything) \}, prh(go) \rangle$$

which prohibits a car to go forward in any situation. In particular, LION provides the agents with a compacted version of this norm that does not contain root terms (i.e., terms *"anything"*). This is why, at tick 97 (tenth normative change),

[1] Computed as the moving average of unregulated collisions of the last 10 ticks.

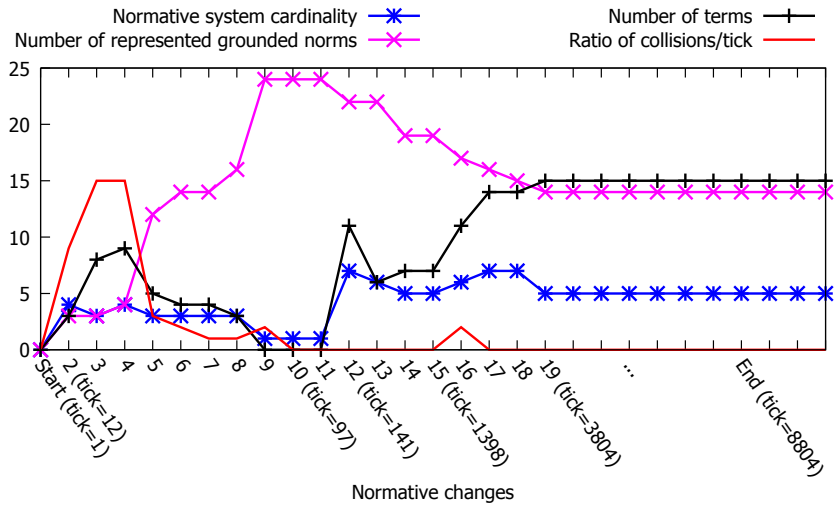

Figure 7.5: A prototypical execution of LION.

the number of terms in the normative system is 0. Norm n^* is clearly a case of over-generalisation, and thus LION specialises it at tick 141 (twelfth normative change), leading from a normative system of one norm to another with seven norms. From that tick onwards, LION subsequently deactivates grounded norms that are implicitly represented and unnecessary, hence reducing the number of grounded norms the normative system represents.

At tick 1398 (fifteenth normative change), LION detects and deactivates the first substitutable norm, reducing in one the number of constraints represented by the normative system. This deactivation triggers the specialisation of its ancestors, which leads to a loss of compactness: the number of terms in the normative system leads from seven to eleven. Thereafter, LION keeps on detecting and deactivating substitutable norms, which results in an increment of terms (i.e., the loss of compactness). At tick 3804 (nineteenth normative change), LION removes the last substitutable norm, and performs the last norm generalisation. In this way, it yields a compact normative system that effectively avoids unregulated car collisions, while preserving cars' freedom as much as possible. Finally, at tick 8804, LION converges, and the simulation stops. LION has yield a 5-norm normative system that does not contain substitutable norms. Table 7.5 depicts this normative system. It contains:

1. Two norms (n_1, n_2) to give way to the *right* in different situations.

2. Three security-distance norms to prevent collisions *just in case* the car in front of a reference car suddenly stops.

Let us now compare this normative system with the one synthesised by SIMON in Table 6.2 from Section 6.6.2. There, SIMON's normative system contained a

Norm	Pre-condition (θ)	Norm target	μ_{eff}	μ_{nec}
n_1	*front*(*nil*), *right*(*car-heading-left*)	*prh*(*go*)	0.87	0.73
n_2	*left*(*car-heading-right*), *right*(*car-heading-right*)	*prh*(*go*)	0.91	0.78
n_3	*front*(*car-same-heading*)	*prh*(*go*)	0.96	0.37
n_4	*front*(*car-heading-right*), *right*(*car-heading-right*)	*prh*(*go*)	0.92	0.37
n_5	*left*(*car-heading-left*), *front*(*car-heading-left*)	*prh*(*go*)	0.91	0.35

Table 7.5: LION's most-frequently synthesised normative system upon convergence (Ω_7 in Figure 7.6).

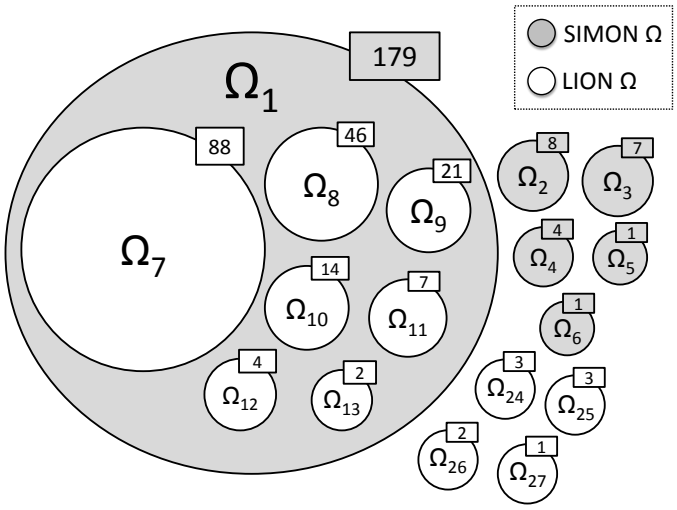

Figure 7.6: Normative systems synthesised by LION and SIMON.

general norm to give way to the left (norm n_1), and a general norm to give way to the right (norm n_5), which are substitutable. By contrast, LION (Table 7.5) contains norms n_1, n_2 to give way to the right, but it does not contain norms to give way to the left. This comes as a result of LION's capability to detect the substitutability between norms, which allowed it to disregard the left-hand side priority norm.

Liberality analysis

Next, LION and SIMON are compared in terms of the liberality of the normative systems they synthesised upon convergence. Particularly, both LION and SIMON converged the 100% of simulations. Figure 7.6 graphically represents the relationship *"more liberal than"* between LION's and SIMON's normative systems. Each circle represents a different normative system. The squared figure on top of each circle stands for the number of times (out of 200 simulations) it was synthesised. White circles represent LION's normative systems, while gray circles represent SIMON's normative systems. For instance, Ω_1 is a normative system

Norm	Pre-condition (θ)	Norm target	$\bar{\mu}_{eff}$	$\bar{\mu}_{nec}$
n_1	*left*(*car-heading-right*)	*prh*(*go*)	0.81	0.26
n_2	*front*(*car-heading-right*), *right*(*car-heading-right*)	*prh*(*go*)	0.82	0.55
n_3	*left*(*car-heading-left*), *front*(*car-heading-left*)	*prh*(*go*)	0.92	0.55
n_4	*front*(*car-same-heading*)	*prh*(*go*)	0.91	0.25
n_5	*right*(*car-heading-left*)	*prh*(*go*)	0.89	0.33

Table 7.6: SIMON's most-frequently synthesised normative system upon convergence (Ω_1 in Figure 7.6).

that was synthesised by SIMON 179 times. The "more liberal than" relationship is represented by the subset relationship between circles. For instance, since Ω_7 is contained in Ω_1 then we say that Ω_7 is more liberal than Ω_1.

Observe that SIMON converged to 6 different normative systems ($\Omega_1, \ldots, \Omega_6$). Specifically, 90% of the times (179 out of 200 simulations), it converged to normative system Ω_1, which corresponds to the normative system depicted in Table 7.6. It contains substitutable norms to give way to the left (norm n_1) and to give way to the right (norm n_5). As to LION, it synthesised 21 different normative systems ($\Omega_7, \ldots, \Omega_{27}$). Nevertheless, those normative systems where not evenly distributed. Thus, 90% of the simulations (180 out of 200) it synthesised just 6 normative systems ($\Omega_7, \ldots, \Omega_{12}$), whereas the remaining normative systems ($\Omega_{13}, \ldots, \Omega_{27}$) where only synthesised by 10% of the simulations. As shown in Figure 7.6, 81% of LION's normative systems (from Ω_7 to Ω_{23}) are more liberal than (are contained in) Ω_1, namely SIMON's most frequent normative system[2]. 81% of LION's normative systems were synthesised in 96% of simulations, and thus we can state that LION converged to normative systems more liberal than Ω_1 for 96% of simulations. The remaining normative systems have similar (slightly better) metrics than SIMON's Ω_1.

Now, let us analyse *why* 81% of LION's normative systems are more liberal than Ω_1. We have computed compute the number of substitutability relationships in LION's and SIMON's normative systems by means of simulation. For each normative system, we have run a simulation that performs pairwise comparison between its norms to check if they are substitutable. For each pair of norms in the normative system, the simulation proceeds by having two cars fulfil/infringe the norms, checking whether conflicts arise or not after fulfilments/infringements, and hence detecting substitutability according to Definition 35. As depicted in Figure 7.7, on average, LION's normative systems managed to get rid of 90% of the substitutability relationships that SIMON's Ω_1 contains. Furthermore, it managed to detect and preserve 100% of complementary norms in SIMON's Ω_1.

[2]For clarity, Figure 7.6 only shows 7 out of the 17 LION's normative systems that are more liberal than Ω_1. Table 7.5 depicts normative system Ω_7, which is LION's most-frequently synthesised normative system. Normative systems $\Omega_{14}, \ldots, \Omega_{23}$ are all more liberal than Ω_1 but each one is only synthesised once by LION.

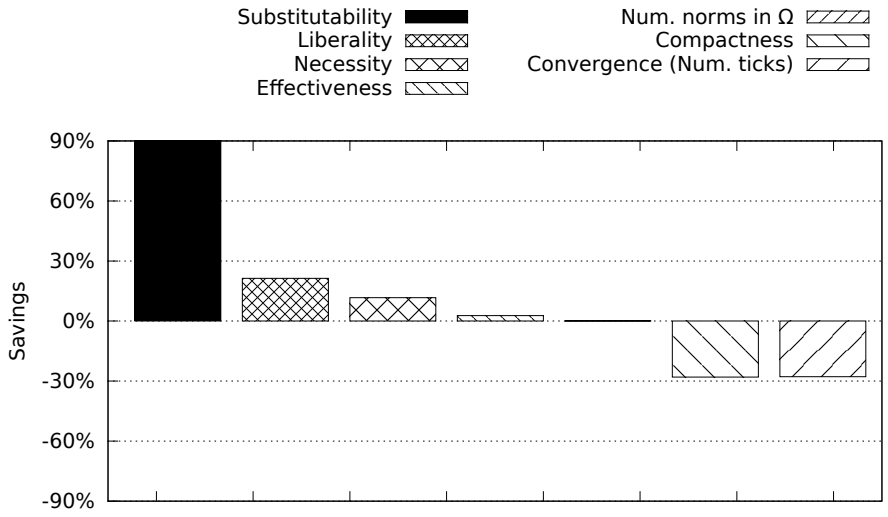

Figure 7.7: LION savings with respect to SIMON's most frequently synthesised normative system (Ω_1 in Figure 7.6).

Multi-objective synthesis analysis

Figure 7.7 summarises the average savings obtained by LION's multi-objective synthesis process with respect to SIMON. First, following the observation above, the success in detecting and discarding substitutable norms translates into a reduction in representation minimality, which is our measure to quantify the gain in liberality. More precisely, the normative systems synthesised by LION contain 21% less specific norms than those synthesised by SIMON. Therefore, LION's normative systems are less restrictive than SIMON's normative systems. Second, regarding effectiveness and necessity, LION's normative systems are similarly effective in avoiding conflicts (in fact, slightly, 2.73%, more effective), while they are more necessary (11% higher). This comes at no surprise because SIMON's normative systems contain substitutable norms. We recall from Section 7.2 that two substitutable norms make unnecessary one another. Since they satisfy the same regulatory needs, only one of them (but at least one of them) is actually necessary. Therefore, a normative system containing substitutability relationships contains unnecessary norms, which decreases its overall necessity. Removing substitutability allows to increase the necessity of the normative system, since *all* the norms it contains are necessary to avoid conflicts. Third, LION's normative systems contain 0.1 more norms than SIMON's normative systems. However, LION's norms are less compact. In fact, LION's normative systems are 28% less compact (have 28% more terms in their norm preconditions) than SIMON's normative systems. In other words, LION's normative systems have as many norms as SIMON's, but the norms in LION's normative systems are more specific. This is

reasonable if we consider that discarding substitutable norms involves specialising general norms, and hence leads to normative systems whose norms are more specific (and thus have a larger number of terms). Finally, LION requires 27.81% extra time to converge, since it performs extra tasks to capture norm synergies, as well as to deactivate substitutable norms.

7.8 Conclusions

This chapter has addressed the synthesis of liberal normative systems, thus answering research question R3 introduced in Section 1.2. With this aim, the chapter has introduced LION, a strategy to synthesise liberal normative systems. LION incorporates the on-line detection and exploitation of norm synergies (substitutability and complementarity relationships) between norms. On the one hand, two substitutable norms are characterised by satisfying the same regulatory needs, and thus only one of them is necessary to avoid conflicts. On the other hand, two complementary norms are both necessary to avoid conflicts. LION is capable of detecting such relationships between the norms in a normative system, and removing substitutable norms while preserving complementary norms. As a result, LION synthesises normative systems that are more liberal than those synthesised by SIMON. This comes at the cost of employing: (1) a mechanism to detect substitutability and complementarity between pairs of norms; and (2) a mechanism to exploit substitutability and complementarity relationships to remove unnecessary constraints, hence increasing the liberality of the normative system.

We have provided an empirical evaluation of LION in the road traffic scenario to assess its quality and relevance. Also, we have compared LION's performance with SIMON's. We reported on experiments in which 96% of the time LION synthesised normative systems that are more liberal than those produced by SIMON. Specifically, LION managed to remove 90% of its substitutability relationships. This was accomplished without compromising effectiveness, or necessity, but at the cost of a 28% reduction in compactness.

To conclude, it pays off using LION to perform norm synthesis only in scenarios wherein synergies between norms (substitutability and/or complementarity) exist, and can be detected. If these conditions are not accomplished (e.g., in the on-line community scenario), one should choose SIMON as a strategy to synthesise norms, since it performs the same synthesis of LION, with the difference that it does not detect and exploit norm synergies.

Chapter 8

Deliberative synthesis of normative systems

8.1 Introduction

Chapters 5 and 6 introduced IRON and SIMON as two alternative strategies to synthesise compact normative systems. Both approaches pursue compactness by performing norm generalisations. Furthermore, Chapter 7 introduced LION, which synthesises compact and liberal normative systems that preserve agents' freedom to the greatest possible extent. As introduced in Section 1.1.3, it is worth to be able to consider different degrees of *reactivity* during norm synthesis. As stated in [Knight, 1993], being reactive allows to take decisions with little information, but cannot guarantee efficient solutions (in our case, norms). By contrast, being more *deliberative* (i.e., considering larger amounts of information) is expected to generate more efficient solutions (norms). In Section 6.8, we observed that IRON and SIMON are highly reactive to conflicts, and thus they are not effective in converging to a stable normative system when norms should be deliberatively synthesised. An example was illustrated with the on-line community scenario (see Sections 5.10.2 and 6.7.1), where neither IRON nor SIMON could converge to a normative system with no norms when the majority of the users considered norms as unnecessary.

Against this background, this chapter aims at answering research question R4 introduced in Section 1.2, and introduces DESMON (*DE*liberative *S*imple *M*inimal *O*n-line *N*orm synthesis), a strategy for the *deliberative* synthesis of compact normative systems. Based on SIMON, DESMON can synthesise norms by considering different degrees of reactivity to conflicts. Thus, DESMON can synthesise norms by being either reactive or deliberative. With this aim, DESMON is endowed with an alternative norm generation mechanism that adds a norm to a normative system only when it gathers enough evidence to consider it is *necessary*. To demonstrate DESMON's performance, an empirical evaluation is provided in which DESMON's norm synthesis is compared with IRON's and SI-

MON's in the on-line community scenario. In short, being more deliberative allows DESMON to converge to a stable normative system in situations in which neither IRON nor SIMON can converge.

The remainder of this chapter is organised as follows. Section 8.2 analyses in more detail why being reactive can be detrimental when synthesising normative systems. Then, Section 8.3 details how to synthesise norms deliberatively. Thereafter, Section 8.4 details DESMON's norm synthesis, which is empirically evaluated in Section 8.5. Finally, Section 8.6 draws some conclusions.

8.2 The reactivity problem: causes and effects

This section analyses why the high reactivity of IRON and SIMON can be detrimental to their convergence to a stable normative system. Let us illustrate this problem with the on-line community scenario employed in the empirical evaluations of IRON (Section 5.10) and SIMON (Section 6.7). There, each user's complaint was regarded as a conflict that revealed the need for regulation, and thus triggered the creation of a norm. In particular, both IRON and SIMON reacted to each complaint by creating a new norm, and *immediately* adding it to the normative system. (i.e., activating it in the normative network).

Consider we aim at synthesising normative systems that are aligned with the preferences of the *majority* of the users of a population. Consider now a population composed of a minority of *moderate* users and a majority of *spammers*. On the one hand, moderates consider spam as inappropriate, and thus always complain about it. On the other hand, spammers always upload spam, and never complain about spam. In this case, it is natural to see that, since the majority of the users are spammers, no regulation is needed to prohibit spam. Therefore, IRON and SIMON should converge to a normative system with no norms. Nevertheless, they will never be able to converge. This lack of convergence can be illustrated with the process below:

1. Consider a moderate user complains about a spam content. IRON/SIMON rapidly react by creating a norm to prohibit spam, then adding it to the normative system (i.e., activating the norm in the normative network).

2. Eventually, IRON/SIMON gather enough evidence to consider that the norm is unnecessary, since only a small proportion of the population (i.e., the moderates) consider the norm as necessary. In other words, there is no enough *consensus* to keep the norm in the normative system.

3. IRON/SIMON deactivate the norm, thus removing it from the normative system. Thereafter, the conflict the norm regulated becomes again unregulated.

4. Whenever a user issues a complain about spam, IRON/SIMON rapidly react by re-activating the norm, thus including it in the normative system.

Thereafter, steps 2–4 are continuously repeated along time, leading to a cycle of continuous norm deactivations/re-activations. This makes IRON/SIMON unable to converge to a stable normative system. Thus, the problem of IRON and SIMON is that they decide the activation of a norm based on a single conflict. In our on-line community scenario, being reactive is counter-productive, since it makes impossible to gather enough evidences about the need for a norm before activating it. In this case, being more deliberative may have helped IRON and SIMON converge to a stable normative system.

8.3 Deliberative norm synthesis

This section introduces DESMON, a synthesis strategy aimed at avoiding the reactivity problem described in Section 8.2. DESMON builds on SIMON, because it obtained the best compactness results in the empirical evaluation introduced in Section 6.7. DESMON's operation follows SIMON's on-line approach, and thus iteratively executes its three synthesis stages: *norm generation, norm evaluation* and *norm refinement*. However, DESMON introduces some changes in each synthesis stage that allows it to deliberatively synthesise norms.

Subsequent sections detail DESMON's components and norm synthesis. First, Sections 8.3.1 and 8.3.2 introduce DESMON's normative network and additional operators. Thereafter, Sections 8.3.3, 8.3.4, and 8.3.5 detail how each DESMON's synthesis stages proceed to perform deliberative synthesis.

8.3.1 A normative network for deliberative norm synthesis

DESMON synthesises normative systems by applying changes to a normative network, which keeps norms as nodes and their relationships as edges. With this aim it employs the normative network described in Section 5.3.1. However, DESMON incorporates an alternative set of possible norm states. This allows DESMON to develop a new norm life cycle to support deliberative norm synthesis. Figure 8.1 depicts a diagram of the possible states of a norm in a DESMON's normative network, along with their transitions. Briefly, a norm in a DESMON's normative network has a state out of a set of states $\Delta = \{hibernated, active, inactive, represented\}$. DESMON computes a normative system (i.e., the norms provided to the agents) as the norms that are currently *active* in the normative network.

Note therefore that DESMON extends the initial set of states described in Section 3.4.1 (that is, $\Delta = \{active, inactive\}$) by including two new states: *hibernated*, and *represented*. Let us now focus on the semantics of each particular state, and then detail the state transitions.

– **Hibernated norm**. A norm whose state is *"hibernated"* is a norm that has been created from a conflict, but has not been activated yet. Therefore, hibernated norms do not belong to the normative system, and thus agents are not aware of them.

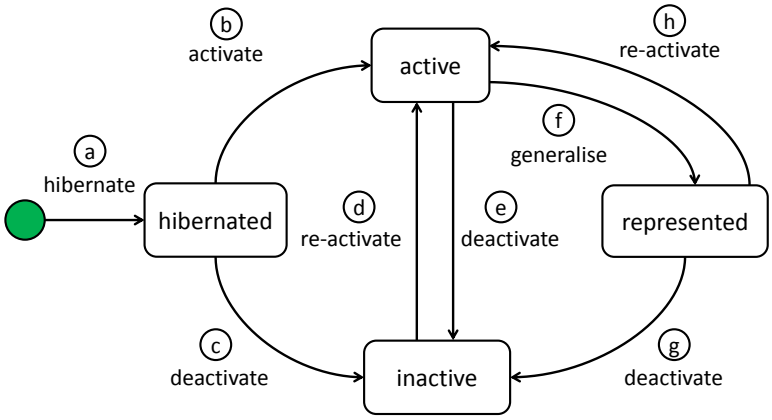

Figure 8.1: Stage diagram of a norm in a DESMON's normative network.

– **Active norm**. An *active* norm is a norm that is considered as effective and necessary enough to avoid conflicts, and thus is included in the normative system.

– **Inactive norm**. A norm is *inactive* if, at some point, it has been considered as either ineffective or unnecessary, and thus it has been removed from the normative system.

– **Represented norm**. A norm is *represented* if it is effective and necessary enough to be included in the normative system, but it is represented by an *active*, more general norm (i.e., an ancestor), which implicitly represents it in the normative system. As an example, an active norm like *"Users are prohibited to upload spam"* implicitly represents a *represented* norm like *"User 1 is prohibited to upload spam"*. In this case, we may say that this latter norm is an instance of the former norm.

Thus, at a given time, a normative system explicitly includes those norms whose state is *"active"*, and implicitly represents those whose state is *"represented"*. In other words, the agents are aware only of *active* and *represented* norms, while they are not aware of *hibernated* and *inactive* norms.

After creating a norm, DESMON hibernates its by setting its state to *"hibernated"*. In this way, it is not included in the normative system. The transition labelled with *"a"* in Figure 8.1 (referred as Figure 8.1a as a shorthand), illustrates this operation. Eventually, a hibernated norm may be added to the normative system by setting its state to *"active"* (transition b), or deactivated by setting its state to *"inactive"* (transition c). Also, at some point, an active

norm may under-perform, thus being removed from the normative system by setting its state to *"inactive"* (transition d). Additionally, an inactive norm could be re-added to the normative system by re-activating it (transition e).

After an active norm is generalised, its state is set to *"represented"* (transition f). In this way, it is no longer explicitly included in the normative system, since it is implicitly represented by another active, more general norm. At some point, a represented norm may under-perform. In this case, the represented norm is deactivated (transition g) and its ancestors are specialised (see Section 5.5). These norm specialisations lead to re-activations of other norms that the specialised ancestors represent and do not under-perform (transition h). This specific process is explained in more detail in Section 8.3.5.

8.3.2 Operators for DESMON's normative network

To operate over the normative network, DESMON employs the operators described in Table 3.1 from Section 3.4.2, which allow to add, activate, and deactivate norms. Moreover, it considers the *specialise* operator described in Table 5.1 (Section5.3.2), which allows to specialise norms in the normative network. Additionally, DESMON includes novel operators to hibernate a norm (Figure 8.1a), and to generalise a norm in the normative network (Figure 8.1f). These operators are:

- A *hibernate* operator. The implementation of this `hibernate` operator is depicted in Table 8.1. It sets the state of a given norm n to *"hibernated"* ($\delta(n) = \mathit{hibernated}$), and thus it is not included in the normative system.
- A *generalise* operator. Essentially, DESMON's `generalise` operator works similarly to IRON's (described in Table 5.1 from Section 5.3.2). It generalises a set of norms (*children*) into a more general norm (*parent*) by performing the following steps. First, if the parent norm does not exist in the normative network, its adds the parent norm to the network and establishes new generalisation relationships between each child norm and its parent. In the next step, DESMON introduces a slight change with respect to IRON's `generalise` operator: it sets the state of each child to *"represented"* (instead of setting it to *"inactive"*. Then it sets the parent's state to *"active"*. As a result, the child norms will no longer belong to the normative system, while the parent norm will.

8.3.3 Creating new norms

During norm generation, DESMON creates new norms as detailed in Section 4.6.1. Briefly, whenever it detects a new, *unregulated* conflict, it creates a new norm aimed at avoiding the conflict in the future. Then, it adds the newly created norm to the normative network. However, DESMON does not immediately activate it. In this way, the norm is not added to the norm to the normative system, and thus it is not not provided to the agents yet. Instead, DESMON *hibernates* the

Operator	Specification
$hibernate(n, NN)$	$\delta(n) \leftarrow hibernated$ $NN' \leftarrow \langle \mathcal{N}, R, \Delta, \delta' \rangle$
$generalise(children, parent, NN)$	**if** $parent \notin \mathcal{N}$ $\mathcal{N}' \leftarrow \mathcal{N} \cup \{parent\}$ **for all** $child \in children$ $\mathcal{E}'_G \leftarrow \mathcal{E}_G \cup (child, parent)$ **for all** $child \in children$ $\delta'(child) \leftarrow represented$ $\delta'(parent) \leftarrow active$ $NN' \leftarrow \langle \mathcal{N}', \mathcal{R}', \Delta, \delta' \rangle$

Table 8.1: DESMON's additional operators.

norm (transition a in Figure 8.1), and considers the conflict as *regulated*. Thereafter, the norm remains hibernated while DESMON gathers enough evidence to decide if it is necessary enough to be activated. In other words, it *deliberates* about the norm's necessity before deciding either to add it to the normative system, or to discard it. With this aim, it iteratively evaluates the norm during the norm evaluation stage (Section 8.3.4). Eventually, when DESMON has gathered enough evidence about the norm's performance, it decides either to activate it, or to deactivate it, during the norm refinement stage (Section 8.3.5).

8.3.4 Evaluating norms

During norm evaluation, DESMON evaluates norms as described in Section 7.4. Briefly, it evaluates a norm's performance in terms of its effectiveness and its necessity. While it computes a norm's effectiveness based on the outcomes of its *fulfilments*, it computes a norm's necessity based on the outcomes of its *infringements*. However, note that hibernated norms are not available to the agents (since they do not belong to the normative system), and thus agents cannot assess their applicability. Consequently, agents cannot fulfil hibernated norms, which makes DESMON incapable of evaluating them in terms of their effectiveness. Therefore, DESMON can evaluate hibernated norms only in terms of their necessity. More specifically, DESMON evaluates norms as follows:

- *Active and represented norms*: As introduced in Section 8.3.1, agents only are aware of active and represented norms, and thus can decide whether to fulfil or infringe them. Therefore, active and represented norms can be evaluated in terms of both their effectiveness (from their fulfilments) and their necessity (from their infringements).

- *Hibernated norms*: As described above, DESMON can evaluate hibernated norms only in terms of their necessity.

8.3.5 Refining the normative system

During the norm refinement stage, DESMON decides the activation, the deactivation and the generalisation of those norms it has created during norm generation. To make these decisions, it considers the norms' performance evidences it has accumulated during norm evaluation (Section 8.3.4). In particular, DESMON generalises norms as SIMON does (see Section 6.3). Additionally, DESMON incorporates novel mechanisms to activate and to deactivate norms. In what follows, we detail how DESMON: (1) activates norms that perform well; and (2) deactivates norms that under-perform.

Activating well-performing norms

DESMON activates a norm provided that it performs well in regulating conflicts. As depicted in Figure 8.1, DESMON may activate a hibernated norm (transition *a*), or it may re-activate a norm that has been previously deactivated (transition *d*). As detailed in Section 8.3.4, DESMON can evaluate a hibernated norm only in terms of its necessity. Therefore, it activates a hibernated norm provided that it is *necessary*. By contrast, a norm may be inactive because it was previously activated (transition *b* in Figure 8.1), and then deactivated because it was either ineffective or unnecessary (transition *e* in Figure 8.1). Therefore, to activate an inactive norm, DESMON checks both its effectiveness and its necessity to make sure it does not include an ineffective norm in the normative system. At a given time t DESMON activates a norm n as follows:

– **Hibernated norm**. If a norm is *hibernated*, activate it iff:

1. There is evidence enough to decide if it is necessary.
2. Its necessity at time t is *above* a necessity *uncertainty area* (see Section 7.4).

This amounts to satisfying both conditions below:

$$|Nec_0^n| \geq nv_{min} \tag{8.1}$$

$$\mu_{nec}(n,t) > \alpha_{nec}^{+} \tag{8.2}$$

where Nec_0^n (defined in Equation 7.25 from Section 7.4) is the set of defined necessity values of norm n from an initial time ($t = 0$); nv_{min} stands for the minimum number of necessity values that are required to exist in set Nec_0^n to decide if a norm is necessary; and condition 8.2 corresponds to condition 7.32 in Section 7.4.1. Specifically, $\mu_{nec}(n,t)$ (described in Section 7.4) stands for the necessity of a norm n at time t; and α_{nec}^{+} is the *upper boundary* of a necessity uncertainty area $\langle \alpha_{nec}^{+}, \alpha_{nec}^{-} \rangle$ (also introduced in Section 7.4.1).

– **Inactive norm**. If a norm is *inactive*, activate it iff both conditions below are satisfied:

1. There is evidence enough to decide if it is necessary.

2. Its necessity at time t is *above* a necessity uncertainty area.

and either both or neither or the two conditions below are satisfied:

1. There is evidence enough to decide if it is effective.

2. Its effectiveness at time t is *above* an uncertainty area.

This amounts to satisfying conditions 8.1 and 8.2 above, and either both or neither of the conditions below:

$$|\mathit{Eff}_0^n| \geq ev_{min} \tag{8.3}$$

$$\mu_{\mathit{eff}}(n,t) > \alpha^+_{\mathit{eff}} \tag{8.4}$$

where Eff_0^n (defined in Equation 7.25 from Section 7.4) is the set of defined effectiveness values of norm n from an initial time ($t = 0$); ev_{min} stands for the minimum number of effectiveness values that are required to exist in set Eff_0^n to decide if a norm is effectiveness; and condition 8.4 corresponds to condition 8.4 in Section 7.4.1. Specifically, $\mu_{\mathit{eff}}(n,t)$ (described in Section 7.4) stands for the effectiveness of a norm n at time t; and α^+_{eff} is the *upper boundary* of the effectiveness threshold band $\langle \alpha^+_{\mathit{eff}}, \alpha^-_{\mathit{eff}} \rangle$ introduced in Section 7.4.1.

Note that, by considering condition 8.1, DESMON implements the activation of a norm as a deliberative process in which a norm can be activated only when DESMON has cumulated enough evidence to decide that it is necessary. Here, the difference between being reactive or more deliberative relies on the minimum amount of necessity evidences (nv_{min}) that DESMON considers to decide if a norm is necessary. Thus, the higher nv_{min}, the more evidences DESMON will require to activate a norm, and the more deliberative it will be. Also, setting nv_{min} to one, would make DESMON to be purely reactive, since it would activate norms with a single evidence, namely immediately after their creation (likewise IRON and SIMON).

As depicted in Figure 8.1, DESMON can activate hibernated norms for the first time (transition b), or it can re-activate norms that have been previously deactivated (transition e). As previously detailed, deactivating a norm requires the specialisation of its ancestors in the normative network (see Section 5.5). Thus, norm deactivations lead to the loss of compactness of the normative system. Therefore, when DESMON activates a norm, it tries to regain the compactness of the normative system by checking if its ancestors can be re-activated (namely, the norm can be re-generalised). Figure 8.2 illustrates an example. Initially, a normative network (*NN* in Figure 8.2a) contains norms n_1 to n_8, being $\{n_1, n_2, n_3, n_4, n_5\}$ grounded norms that are generalised by n_6, n_7, and n_8 the most general norm. Note that n_5's state is *"inactive"*, and thus, its ancestors (n_7, n_8) are inactive so that n_5 is not represented in the normative system. Thus, the normative network represents normative system $\Omega = \{n_2, n_4, n_6\}$. At a given time, DESMON activates n_5 to include it in the normative system (transition d in Figure 8.1). Thereafter, it tries to regain the normative system's compactness by re-generalising n_5 into its ancestors. With this aim, it re-activates n_8, and

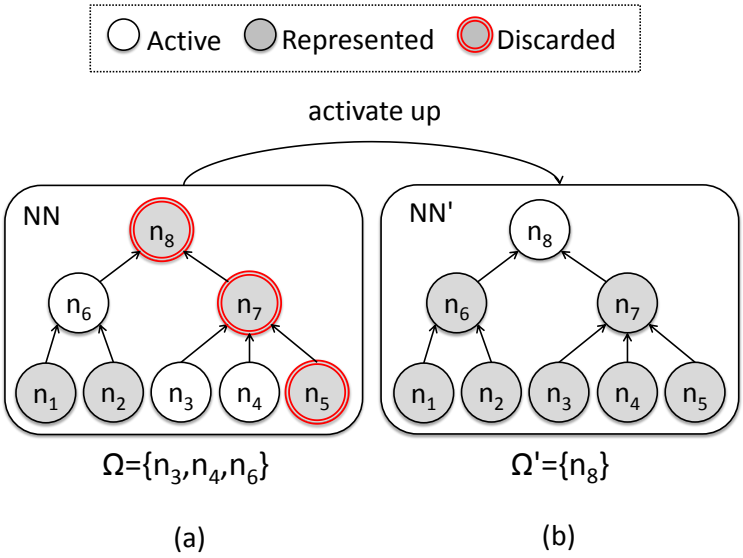

Figure 8.2: Upwards propagation of norm n_5's re-activation.

sets the state of n_5, n_7 to *"represented"*. As a result, it yields normative network NN' (Figure 8.2b), which represents $\Omega' = \{n_8\}$. By propagating up this n_5's re-activation, DESMON managed to synthesise again a compact normative system of a single norm.

Algorithm 23 depicts DESMON's *activateUp* function, which recursively activates a norm and tries to re-activate its ancestors. It receives a norm to activate (n), and a normative network (NN). If n is not already represented by an active parent, it activates it (lines 1–2). Next, it sets the state of all its children to *"represented"* so that they are no longer explicitly included in the normative system (lines 5–7). Finally, for each parent of n, it checks if it represents any norm whose state is *"inactive"*. If a parent does no represent inactive norms, then it is also activated up (lines 8–10). Eventually, the re-activation of its parent norm will imply that n will be set to state *"represented"*.

Deactivating under-performing norms

DESMON deactivates a norm provided that it under-performs in regulating conflicts. As previously introduced, DESMON can evaluate hibernated norms only in terms of their necessity, while it can evaluate active and represented norms in terms of their effectiveness and their necessity. Therefore, DESMON discards a norm n at time t as follows:

– **Hibernated norm**. If a norm is hibernated, discard it if:

1. There is evidence enough to decide if it is unnecessary.

ALGORITHM 23: Function *activateUp*

Input : n, NN
Output: NN

```
1  if not isRepresented(n) then
2      NN ← activate(n, NN)
3      children ← getChildren(n, NN)
4      parents ← getParents(n, NN)
5      foreach child ∈ children do
6          if isRepresented(child) then
7              NN ← setRepresented (n, NN)
8      foreach parent ∈ parents do
9          if not representsDiscardedNorms(parent) then
10             activateUp(n, NN)
11 return NN
```

2. Its necessity at time t is *below* a necessity *uncertainty area* (see Section 7.4).

This amounts to satisfying condition 8.1 above, and the following condition (which corresponds to condition 7.34 in Section 7.4.1):

$$\mu_{nec}(n,t) < \alpha^{-}_{nec} \tag{8.5}$$

where $\mu_{nec}(n,t)$ stands for the necessity of a norm n at time t; and α^{-}_{nec} is the *lower boundary* of the necessity uncertainty are $\langle \alpha^{+}_{nec}, \alpha^{-}_{nec} \rangle$ described in Section 7.4.1.

– **Active/represented norm**. If a norm is either active or represented, discard it if one of the conditions below are satisfied:

1. There is evidence enough to decide if it is effective, and its effectiveness is *below* an effectiveness *uncertainty area* (see Section 7.4).

2. There is evidence enough to decide if it is necessary, and its necessity is *below* a necessity *uncertainty area* (see Section 7.4).

This amounts to satisfying either both conditions 8.3, 8.6, *or* both conditions 8.1, 8.5.

$$\mu_{\mathit{eff}}(n,t) < \alpha^{-}_{\mathit{eff}} \tag{8.6}$$

where $\mu_{\mathit{eff}}(n,t)$ stands for the effectiveness of a norm n at time t; and $\alpha^{-}_{\mathit{eff}}$ is the *lower boundary* of the effectiveness threshold band $\langle \alpha^{+}_{\mathit{eff}}, \alpha^{-}_{\mathit{eff}} \rangle$ introduced in Section 7.4.1.

Once DESMON considers that a norm under-performs, it proceeds to deactivate it by setting its state and its ancestors' to *"inactive"*. In this way, the norm is completely removed from the normative system, since it is no longer

represented by itself or an active ancestor. With this aim, it employs function *deactivateUp* depicted in Algorithm 12 from Section 5.7.3. Briefly, it recursively deactivates a norm, along with its ancestors in the normative network.

8.4 DESMON's synthesis strategy

So far, we have seen how DESMON generates, evaluates, and refines norms to perform deliberative norm synthesis. Next, let us introduce DESMON's overall strategy, which runs the three synthesis stages of norm generation, norm evaluation, and norm refinement. Algorithm 24 illustrates DESMON's strategy, which takes as input a tuple with a description of the previous state of a MAS (d_s) and a description of the current MAS state (d_{s_c}), and outputs a normative system to regulate the agents' behaviour. To perform norm synthesis, DESMON considers the globally accessible elements described in Section 7.5. That is, a normative network, a collection of operators, a set of domain-dependent elements, a set of domain-independent settings, and a set of additional synthesis inputs. Additionally, DESMON considers two globally accessible constants ev_{min}, nv_{min} (see Section 8.3.5) that represent the minimum number of effectiveness and necessity values that DESMON requires to decide if a norm is either effective or necessary.

ALGORITHM 24: DESMON's synthesis strategy

Input : $\langle d_s, d_{s_c} \rangle$
Output : Ω
Initialisations: $\mathcal{NCO} \leftarrow \emptyset$, $\mathcal{P} \leftarrow \emptyset$

[1] $NN \leftarrow$ `normGeneration`$(\langle d_s, d_{s_c} \rangle, \mathcal{P})$;
[2] $\mathcal{P} \leftarrow$ `normEvaluation`$(\langle d_s, d_{s_c} \rangle, \mathcal{NCO}, \mathcal{P})$;
[3] $NN \leftarrow$ `normRefinement`$(\mathcal{NCO}, \mathcal{P})$;
[4] $\Omega \leftarrow \{n \in NN \mid \delta(n) = active\}$;
[5] **return** Ω

During norm generation (line 1), DESMON perceives the current state of the MAS, detects conflicts, and creates norms to avoid each detected conflict. Then, it carries out norm evaluation (line 2), in which it evaluates norms as detailed in Section 7.4. Thereafter, it carries out norm refinement (line 3), in which it activates norms that perform well, deactivates norms that under-perform, and generalises norms that do not under-perform. Subsequent sections detail DESMON's norm generation and norm refinement stages.

8.4.1 Norm generation

Algorithm 25 depicts DESMON's *normGeneration* function. It starts by detecting conflicts in the current MAS state, and creating a norm for each unregulated conflict it detects (lines 1–4). Then, it adds the recently created norm to the normative network if it does not exist yet (lines 5–6). Thereafter, it hibernates

the norm (line 7) by invoking the *hibernate* operator described in Table 8.1 from Section 8.3.2. In this way, the norm is not included in the normative system, since DESMON will deliberate about (i.e., will iteratively evaluate) its necessity. Eventually, the hibernated norm will be either activated or deactivated during the norm refinement stage.

ALGORITHM 25: DESMON's *normGeneration* function

Input : $\langle d_s, d_{s_c} \rangle, \mathcal{P}$
Output: Ω

```
[1] conflicts_{s_c} ← getConflicts (d_{s_c});
[2] for c_{s_c} ∈ conflicts_{s_c} do
[3]     if not regulated(c_{s_c}, d_s) then
[4]         n ← create(c_{s_c}, d_s);
[5]         if n ∉ getNorms(NN) then
[6]             NN' ← add(n, NN);
[7]         NN' ← hibernate(n, NN);
[8] return NN;
```

ALGORITHM 26: DESMON's *normRefinement* function

Input : $\mathcal{NCO}, \mathcal{P}, \mathcal{PCO}$
Output: NN

```
/* Norm specification:                                              */
1  Ω ← {n ∈ NN | δ(n) = active};
2  NRG ← getNegativelyEvaluatedNorms(P);
3  specifyUnderperformingNorms(NRG, NN, Ω);
/* Norm activation, norm deactivation and generalisation:           */
4  for n ∈ getNormsFulfilledInfringedThisState(NCO) do
5      if performsWell(n, NN, α_eff, α_nec, ev_min, nv_min, ϵ) then
6          activateUp(n, NN);
7      else if underPerforms(n, P, α_eff, α_nec, ϵ) then
8          NN ← deactivateUp(n, NN);
9      else
10         NN ← generaliseUp(n, NN, T, G_M, G_S);
11 return NN
```

8.4.2 Norm refinement

Algorithm 26 depicts DESMON's *normRefinement* function. It starts by specifying general norms that under-perform (lines 1-3). More details about this particular process can be found in Section 6.4. Then, it activates those norms

that perform well in regulating conflicts (lines 5–6), and discards those norms that under-perform (lines 7–8). It performs these steps as described in Section 8.3.5. Thereafter, in lines 9–10, it generalises norms that do not under-perform as SIMON does (see Algorithm 18 in Section 6.5). In particular, to generalise norms, function `generaliseUp` invokes the *generalise* operator described in Table 8.1 from Section 8.3.2.

8.5 Empirical evaluation

Next, DESMON's norm synthesis is empirically evaluated in the on-line community scenario, and compared with IRON's and SIMON's. The aim is to show that DESMON is capable of converging to a stable normative system in situations in which neither IRON nor SIMON are able to converge. These benefits stem from DESMON's capability to deliberatively synthesise norms, which allows it to make more informed decisions during norm synthesis.

8.5.1 Empirical settings

The experiments have been performed with the on-line community simulator employed in the empirical evaluations of IRON (Section 5.10) and SIMON (Section 6.7). In particular, we have configured the on-line community scenario as described in Section 6.7.1.

DESMON has been configured with the same settings as SIMON in Section 6.7.1. These settings are shown in Table 8.2. Briefly, DESMON computes a norm's necessity at a given time by considering its last 100 defined necessity values ($q = 100$), and, for each new norm n that DESMON creates, it sets its initial necessity to 0.5 ($\mu_{nec}(n, t_0) = 0.5$). Recall from Section 5.10.1 that, in this scenario, a norm's effectiveness cannot be evaluated, since norm fulfilments are not observable. DESMON's norm synthesis is analysed for low, medium, and high necessity thresholds. More specifically, $\alpha_{nec} \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. DESMON has been configured to perform *deep* generalisations ($\mathsf{G}_M = \mathit{deep}$) and to generalise three terms at a time ($\mathsf{G}_S = 3$). The effectiveness and necessity uncertainty areas considered by DESMON are computed as a constant value 0.05 *above* and *below* threshold α_{nec} ($\epsilon = 0.05$). Thus, for instance, with threshold α_{nec} set to 0.2, DESMON considers a necessity uncertainty area like $\langle 0.25, 0.15 \rangle$. Finally, DESMON has been configured to be highly deliberative. Particularly, DESMON requires a minimum of 50 effectiveness and necessity values ($ev_{min} = 50, nv_{min} = 50$) to decide if a norm is necessary.

Each simulation finishes when it reaches 5,000 ticks, and a simulation is considered to have converged whenever the normative system remains unchanged during a 1000-tick period, and hence DESMON is considered to have solved the norm synthesis problem.

Parameter	Description	Value
q	Number of effectiveness values and necessity values (μ_{nec}) considered to compute a norm's effectiveness and necessity ranges (see Section 5.6).	100
k_{nec}	Default norm effectiveness and necessity values ($\mu_{\mathit{eff}}(n, t_0), \mu_{nec}(n, t_0)$).	0.5
α_{nec}	Threshold below which a norm is considered to under-perform in terms of its necessity.	(0.1, 0.3, 0.5, 0.7, 0.9)
T	Time period considered when assessing convergence (cf. Definition 18 in Section 3.2.2).	1,000
G_M	Generalisation mode.	*deep*
G_S	Generalisation step.	1
ϵ	Constant to compute the effectiveness and necessity uncertainty areas described in Section 7.4.1.	0.05
ev_{min}	Number of effectiveness values required to determine a norm's effectiveness (see Section 8.3.5)	50
nv_{min}	Number of necessity values required to determine a norm's necessity(see Section 8.3.5)	50

Table 8.2: IRON's norm synthesis settings in the on-line community scenario.

8.5.2 Empirical results

Let us now analyse DESMON's results. First, recall from Section 5.10.2 that the proportion of moderate users in a population represents the *complain power* of that population (C_{Pw}), since moderates are the only users who complain about spam. Thus, the complain power of a population is directly related with the necessity of the norms to prohibit spam. The more complaints, the more necessary the norms. Moreover, the necessity threshold (α_{nec}) can be seen as a *consensus degree* that establishes the minimum proportion of users that must consider norms as necessary so that they are included in the normative system. Therefore, the relationship between the complain power of a population and the established consensus degree determines the type of normative systems that DESMON converges to.

Table 8.3 depicts averaged results of 100 different simulations for each population and necessity threshold. Each cell contains, either: (i) the size of the normative system that DESMON converged to; or (ii) symbol "X" if DESMON was not able to converge to a normative system. The results in the table can be grouped as follows:

– **Convergence to a single norm.** Whenever the complain power of the population is *above* the consensus degree (to the left of the diagonal of the table), DESMON converges 100% of the simulations to a normative system with one general norm

$$n^* \quad : \quad \langle (user(anyUser), section(anySection), cntType(spam)), prh(upload) \rangle$$

that concisely prohibits any user to upload spam in any section. Let us illustrate this situation with population 30M-70S (0.3 complain power), and consensus degree 0.1 ($\alpha_{nec} = 0.1$). With that population, norms to prohibit spam are around 0.3 necessary, since 30% of the users believe that spam should be prohibited. Moreover, the consensus degree establishes that norms must be at least 0.1 necessary to be active, namely at least 10% of the users must agree on the need for norms. Therefore, norms to prohibit spam are included in the normative system, since they are considered as necessary. Eventually, DESMON benefits from norm generalisations to synthesise a compact normative system with single norm n^* described above.

– **Lack of convergence.** When the complain power of the population is equal to the consensus degree (i.e., the diagonal of the table), DESMON cannot converge to a stable normative system. This happens because norms' necessities fluctuate above and below the necessity threshold. As a result, DESMON continuously deactivates and re-activates norms, being unable to converge to a stable normative system.

– **Convergence to no norms**. When the complain power of the population is *below* the consensus degree, DESMON converges to an empty normative system (to the right of the diagonal). This comes as a result of DESMON's capability to deliberate about the need for norms before including them in the normative sys-

	Consensus degree (α_{nec})				
Population	0.1	0.3	0.5	0.7	0.9
30M-70S (0.3 complain power)	1	X	0	0	0
50M-50S (0.5 complain power)	1	1	X	0	0
70M-30S (0.7 complain power)	1	1	1	X	0

Table 8.3: Number of norms that DESMON converged to.

Case	Convergence
Complain power > consensus degree ($\mathsf{C}_{Pw} > \alpha_{nec}$)	Yes
Complain power = consensus degree ($\mathsf{C}_{Pw} = \alpha_{nec}$)	No
Complain power < consensus degree ($\mathsf{C}_{Pw} < \alpha_{nec}$)	Yes

Table 8.4: Summary of DESMON's convergence analysis.

tem. In this case, the complaints of moderate users trigger the creation of norms to prohibit spam upload. However, instead of activating them, DESMON iteratively evaluates (i.e., deliberates about) their necessity. Eventually, when it has gathered enough evidence, DESMON considers that created norms are not necessary enough to be activated, and deactivates them (i.e., it performs transition c in Figure 8.1). As a result, the normative system remains stable with 0 norms during all the simulation allowing DESMON to converge. Table 8.4 summarises DESMON's results for different combinations of complain power/consensus degree.

Figure 8.3 illustrates how DESMON managed to converge to an empty normative system with population 70M-30S (0.7 complain power) and 0.9 consensus degree ($\alpha_{nec} = 0.9$). The x-axis shows different normative changes that DESMON performed along time, and the y-axis shows:

1. The cardinality of the normative network.
2. The cardinality of the normative system.
3. The ratio of unregulated user complaints (conflicts) per tick.

From tick 0 to 500 (i.e., during the *warm-up* period), DESMON does not generate norms since users do not complain. From tick 500 onwards, moderates start complaining about those spam contents they view, and DESMON creates (and hibernates) norms to prohibit spam. At tick 511 (sixth normative change), DESMON has created ninety norms to prohibit each of the thirty spammers to upload spam in the three sections. However, all of them remain hibernated. Eventually, DESMON considers these norms as unnecessary, since their necessity (which is 0.7) is below the consensus degree ($\alpha_{nec} = 0.9$). Thus, it deactivates each synthesised norm, keeping the normative system stable with no norms.

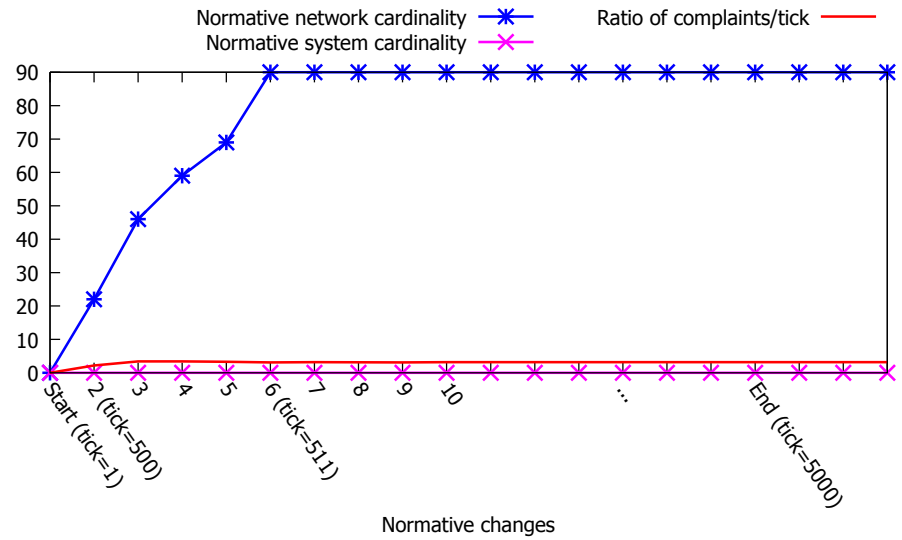

Figure 8.3: Prototypical execution of DESMON with a population of 70 moderates and 30 spammers, and $\alpha_{nec} = 0.9$.

	Consensus degree (α_{nec})				
Population	0.1	0.3	0.5	0.7	0.9
30M-70S (0.3 complain power)	30	X	X	X	X
50M-50S (0.5 complain power)	50	50	X	X	X
70M-30S (0.7 complain power)	70	70	70	X	X

Table 8.5: Number of norms that IRON converged to.

	Consensus degree (α_{nec})				
Population	0.1	0.3	0.5	0.7	0.9
30M-70S (0.3 complain power)	1	X	X	X	X
50M-50S (0.5 complain power)	1	1	X	X	X
70M-30S (0.7 complain power)	1	1	1	X	X

Table 8.6: Number of norms that IRON converged to.

Let us now compare DESMON's results with those obtained by IRON (Section 5.10.2) and SIMON (Section 6.7.2). Tables 8.5 and 8.6 depict these results. Observe that neither IRON nor SIMON could converge to a stable normative system when the complain power of a population was *below* the consensus degree ($\mathsf{C}_{mathitPw} < \alpha_{nec}$). There, IRON and SIMON were penalised by their high reactivity to conflicts. By contrast, DESMON could converge to a normative system with no norms. Here, DESMON benefited from its capability to deliberatively synthesise norms.

8.6 Conclusions

This chapter has tackled the synthesis of normative systems by considering different degrees of reactivity, thus answering research question R4. With this aim, it has introduced DESMON, a synthesis strategy that can perform norm synthesis with different degrees of reactivity to conflicts. Based on SIMON (introduced in Chapter 6), DESMON includes norms in the normative system only when it decides that they are necessary. Here, the difference between being more reactive or more deliberative relies on the amount of evidences about a norm's necessity that DESMON requires to make this decision. That is, the more evidences it requires, the more deliberative it is.

To demonstrate DESMON's performance, it has been empirically evaluated in the on-line community scenario, and it has been compared with IRON and SIMON. By being more deliberative, DESMON has been able to converge to a stable normative system in situations in which IRON and SIMON were not able to converge. This was accomplished without compromising effectiveness and compactness. To the best of our knowledge, DESMON is the first computational approach that can consider different degrees of reactivity during on-line norm synthesis, namely that can consider different amounts of information to synthesise norms.

Chapter 9

Conclusions and future work

This chapter draws the conclusions of this dissertation. Firstly, Section 9.1 summarises the contributions made during this thesis. Then, Section 9.2 draws some conclusions attained from this thesis. Finally, Section 9.3 presents some lines for future research.

9.1 Contributions

Norms have been widely used as a coordination mechanism in Multi-Agent Systems (MAS) [Dignum, 1999a, Boella et al., 2006, Shoham and Leyton-Brown, 2009]. In the literature, a wide extent of problems have been studied regarding the use of norms in MAS. Of these, a key problem is that of how to synthesise the norms that will successfully coordinate agents' interactions within a MAS. Solving this norm synthesis problem is specially challenging in the case of *open* MAS, in which agents may be developed by third parties and their behaviours may be unknown. These conditions make highly difficult to compute the norms that should govern an open MAS, since assessing what and how to regulate is not straightforward. The literature has provided means to synthesise norms for Multi-Agent Systems, yet no computational approach has been introduced to synthesise norms for a MAS at runtime (on-line), without involving agents in norm synthesis (namely, exogenously), and by considering multiple synthesis objectives (multi-objective synthesis).

Against this background, this dissertation has introduced the first computational framework to perform on-line, exogenous and multi-objective norm synthesis for open MAS. This framework provides: This framework is composed of:

1. A general, domain-independent *computational model* to perform on-line norm synthesis by observing a MAS at runtime, identifying situations that

are undesirable for coordination, and synthesising norms to avoid these situations. This model considers that norms are synthesised exogenously, and thus it does not require previous knowledge of agents' behaviours and their potential interactions.

2. An *abstract architecture* to implement a regulatory entity that observes a MAS at runtime, and executes a strategy that synthesises normative systems based on the perceptions collected from the MAS.

3. A *family of synthesis strategies* that implement the aforementioned norm synthesis model, and focus on achieving different synthesis objectives.

Overall, this family of synthesis strategies can be employed to synthesise normative systems that are *effective* in avoiding situations that are detrimental to the agents, while being as *compact* and *liberal* as possible. On the one hand, compactness is concerned with synthesising normative systems that reduce agents' efforts when reasoning about norms. On the other hand, liberality is concerned with preserving the agents' freedom to the greatest possible extent, without compromising effectiveness and compactness. Moreover, it can be employed to synthesise norms by considering different degrees of *reactivity*. Reactivity is related to the amount of information that is considered to perform norm synthesis. Briefly, being reactive allows to move in real time and to synthesise norms with little information, but cannot guarantee efficient norms. By contrast, being more deliberative is bound to generate more efficient norms, but it also requires more information and may incur in increasing time and computation costs. A key point is then to adjust the degree of reactivity of norm synthesis in order to generate successful normative systems without requiring too much time and computation costs.

Hereafter, a fine-grained analysis of the contributions of this dissertation is provided.

9.1.1 On-line norm synthesis for open MAS

The first research question of this thesis addressed the development of computational means to synthesise norms for open MAS at runtime without involving domain agents in norm synthesis (research question R1 in Section 1.2). To answer this question, Chapter 3 made the following contributions:

– **An abstract and domain-independent computational model for on-line norm synthesis.** Firstly, we characterised the norm synthesis problem at hand, and provided a formal framework that underpinned the rest of the work developed during this thesis. Thereafter, we introduced a domain-independent computational model to perform norm synthesis for open MAS. Within this model, norms are synthesised at runtime and without requiring domain agents' participation in the synthesis process. In such model, norm synthesis is driven by the main goal of yielding normative systems that effectively avoid *conflicts*, namely situations that are undesirable for multi-agent coordination. Particularly, this model is not aimed to avoid conflicts before they arise for the first

time. Instead, it is aimed at learning from arisen conflicts, creating norms that avoid these conflicts in the future. With this aim, our model makes two main assumptions:

- *Agents' actions are observable.* To create a norm, this model requires that agents' actions are observable, namely that they can be inferred by observing agents' activities.
- *A conflict is caused by one action performed by a single agent in the state previous to the conflict.* Given a conflict, this model considers that a norm can be created by first blaming a single agent for the conflict, and then prohibiting the (observable) action the agent performed in the transition to the conflict.

To assess a norm's performance, the model proposes its evaluation based on the conflicts that arise after agents fulfil and infringe it along time. On the one hand, a norm can be evaluated in terms of its effectiveness based on the conflicts that arise after agents fulfil it. In short, the more conflicts a norm avoids, the more effective. On the other hand, a norm can be evaluated in terms of its necessity by analysing the conflicts that arise after agents infringe it. The less conflicts arise after a norm is infringed, the less necessary the norm. Based on norms' performances, norms are refined to yield normative systems whose norms effectively avoid conflicts within a MAS as long as agents comply with their norms.

The introduced computational model is abstract and domain-independent. This allows to apply it to perform norm synthesis in different application domains by providing little domain-dependent information. Nevertheless, to perform norm synthesis in a specific domain, this model considers a set of domain-dependent elements, such as functions to perceive a particular scenario, detecting conflicts, and yielding agents' perceptions and actions. To the best of our knowledge, this is the first domain-independent computational model for on-line, exogenous norm synthesis that has been introduced in the literature.

– **An architecture to perform on-line norm synthesis in open MAS**. We introduced an architecture to implement a regulatory entity – the so-called *Norm Synthesis Machine* (NSM), which is in charge of observing a MAS at runtime and performing norm synthesis as described by the norm synthesis model. Such architecture considers that a NSM is composed of:

- A *normative network* to keep track of synthesised norms.
- A *collection of operators* over to the normative network.
- A *synthesis strategy* that is executed by a NSM to perform norm synthesis.

9.1.2 Synthesis of effective normative systems

We introduced a synthesis strategy, the so-called BASE, which can be ran by a NSM to perform norm synthesis. BASE's synthesis is guided by the goal of synthesising normative systems that are *effective* in avoiding conflicts, while avoiding

over-regulation. Particularly, BASE implements the norm synthesis model mentioned above. We empirically evaluated BASE's norm synthesis in agent-based simulations of a road traffic scenario, and analysed its performance. BASE was proven to be able to synthesise normative systems that effectively avoided cars' collisions at runtime, without involving them in norm synthesis. In this way, we also proved the validity of the norm synthesis model introduced above.

9.1.3 Synthesis of compact normative systems

The second research question of this dissertation was referred to the synthesis of *compact* normative systems (research question R2 in Section 1.2). Chapters 5 and 6 answered this question by introducing IRON and SIMON, two synthesis strategies that extend BASE by considering compactness as a synthesis objective. Both strategies pursue compactness by performing *norm generalisations*. Briefly, a norm generalisation represents a group of norms as a single, general norm, hence compacting a normative system. The main difference between IRON and SIMON relies on their approach to norm generalisation. IRON takes a *conservative* approach to norm generalisation that requires *full* evidence to generalise. Briefly, IRON synthesises a general norm only when it verifies that all the norms it represents have been synthesised. Moreover, it requires that these norms are effective and necessary to avoid conflicts. This allows IRON to achieve compactness without risking to include ineffective or unnecessary norms in the normative system. As a dual operation to generalisation, IRON is endowed with the computational capability to *specialise* norms, namely to backtrack norm generalisations. This allows IRON to undo generalisations that produced norms that do not perform well in regulating conflicts.

By contrast, SIMON takes an *optimistic* approach to norm generalisation: it generalises norms when there is a *minimum* amount of evidence to generalise. SIMON's generalisation is inspired in the anti-unification of terms as defined in [Armengol and Plaza, 2000], which consists in generalising a set of feature terms to their *least common subsumer* or *most specific generalisation*. Analogously, SIMON generalises pairs of norms to their most specific generalisation, namely the most specific norm that generalises both of them. In this way, SIMON can synthesise a general norm even though the norms it represents have not been synthesised yet, and there is no evidence about their performance. This allows SIMON to perform further generalisations than IRON, though it may lead to over-generalisations. Thus, SIMON is also endowed with the computational capability to revise and to backtrack (specialise) over-generalisations.

IRON's and SIMON's performance was empirically evaluated in two scenarios: a road traffic scenario, and an on-line community scenario. In this way, we not only could analyse the performance of both approaches, but also proved the domain-independence of the computational model for norm synthesis. In both scenarios, IRON and SIMON were capable of synthesising compact normative systems that are effective to avoid conflicts. In particular, SIMON was shown to significantly outperform IRON in terms of compactness. This became more evident in the on-line community scenario, in which SIMON was capable of syn-

thesising a single norm to regulate the behaviours of all the users in a community. By contrast, IRON synthesised multiple norms.

To the best of our knowledge, IRON and SIMON are the first on-line norm synthesis strategies that consider *compactness* as a synthesis objective. By using these strategies, a NSM can provide the agents within an open MAS with compact normative systems that, in addition to effectively coordinate their activities, require low computational efforts when reasoning about norms.

9.1.4 Synthesis of liberal normative systems

The third research question of this thesis addressed the synthesis of *liberal* normative systems (research question R3 in Section 1.2). To answer this question, Chapter 7 introduced LION, a strategy that extends SIMON by considering liberality as a synthesis objective. LION is capable of detecting and exploiting synergies between norms, thus detecting when two norms are substitutable or complementary. By discarding substitutable norms, LION increases a normative system's liberality, while it safeguards its effectiveness by preserving complementary norms. In this way LIONyields normative systems that constrain agents' behaviours just the necessary to avoid conflicts.

LION's performance was empirically evaluated and compared with SIMON's in the traffic scenario. There, LION proved to outperform SIMON in terms of liberality. On the one hand, SIMON synthesised normative systems that contained substitutable norms (i.e., left-hand side priority norms, and right-hand side priority norms). This came as a result of SIMON's incapability to detect synergies between norms. By contrast, LION was capable of detecting and exploiting synergies between norms, thus synthesising normative systems that contained either left-hand side priority norms, or right-hand side priority norms.

9.1.5 Deliberative synthesis of normative systems

The last research question of this dissertation referred to the synthesis of normative systems by considering different degrees of *reactivity* during norm synthesis (research question R4 in Section 1.2). Chapter 8 answered this question by introducing DESMON, a synthesis strategy that extends SIMON and allows to synthesise norms with different degrees of *reactivity* to conflicts. DESMON allows to adjust the amount of evidences about conflicts that are required to decide to include a new norm in a normative system. Thus, DESMON can synthesise norms by being reactive (namely, by considering little evidence to synthesise a norm), or by being more deliberative (that is, by considering more conflict evidences to decide whether a norm is really necessary).

We empirically studied DESMON's performance in the on-line community scenario. Moreover, we compared DESMON's norm synthesis with IRON's and SIMON's. By being more deliberative, DESMON was able to converge to a stable normative system in situations in which IRON and SIMON were not able to converge, without compromising effectiveness and compactness. There, DESMON benefitted from its capability to deliberate about a norm's necessity, that is, to

make sure that a norm is really necessary before including it in the normative system. By contrast, IRON and SIMON were penalised by their high reactivity to conflicts. Briefly, they reacted to conflicts by immediately including new norms in the normative system, which were eventually considered as unnecessary, and then discarded. Thereafter, further conflicts provoked IRON/SIMON to immediately re-include these norms in the normative system, even though they had been considered as unnecessary. This started a cycle of continuous norm deactivations/re-activations that made both IRON and SIMON unable to converge.

To the best of our knowledge, DESMON is the first on-line norm synthesis approach that can synthesise norms for open MAS by considering different degrees of reactivity to conflicts, that is, by adjusting the amount of information required to decide either to include new norms in the normative system, or not.

9.2 Lessons learned

Based on the summary above, we draw several main lessons that we learnt in this thesis, which we detail in what follows.

The key to on-line norm synthesis in open MAS is to observe agents' interactions

We have seen that norm synthesis can be performed by observing agents' activities while they interact within a MAS. The norm synthesis model introduced here has been proven to be appropriate of synthesising norms without involving domain agents in norm synthesis. In this way, such model is appropriate to synthesise norms for open MAS, whose agents cannot be assumed to be capable (or even willing) to synthesise their own norms.

Norm synthesis demands to consider multiple objectives

We have seen that synthesising normative systems that are effective to avoid conflicts is not enough. From the work reviewed in the literature, we have learned that norm synthesis should consider the computational efforts and the freedom of the agents for whom norms are synthesised. On the one hand, reducing the amount of norms the agents are provided with can help to reduce their computational efforts when reasoning about norms. This may allow not to overload agents' computational capabilities, thus enabling them to interact more efficiently. On the other hand, reducing the amount of constraints imposed on the agents may allow to regulate their behaviour while giving them as much freedom as possible while guaranteeing their successful coordination.

Norm relationships play a key role in norm synthesis

As we have seen, norms may have relationships between them. First, we have discussed that norms may have syntactic relationships that can be analytically

discovered. An example of these relationships is generalisation, which can be exploited to synthesise compact normative systems. Moreover, we have argued that norms may have semantic relationships that can be empirically detected. Of these, we have focused on substitutability and complementarity. On the one hand, substitutability relationships allow to detect norms that are unnecessary in presence of another norm. On the other hand, complementarity relationships allow to discover norms that are all necessary to avoid conflicts. Moreover, we have seen how an on-line norm synthesis strategy (LION) can take benefit of generalisation, substitutability and complementarity relationships to synthesise compact normative systems that avoid conflicts within a MAS while preserving agents' freedom as much as possible. Therefore, we conclude that norm relationships play a key role in norm synthesis.

The nature of conflicts drives the degree of reactivity required by norm synthesis

The synthesis strategies introduced in this dissertation have been empirically evaluated in two scenarios: the road traffic scenario, and the on-line community scenario. From these domains, we have observed that the different types of conflicts in different domains may require different degrees of reactivity to conflicts. For instance, in the road traffic scenario, the dramatic consequences of car collisions (e.g., human casualties) require to be highly reactive to conflicts, thus immediately regulating. In this way, once a collision arises, norms can be immediately synthesised to avoid future conflicts, even though there is no enough evidence to assess if they are really necessary.

In the on-line community scenario, IRON, SIMON and DESMON were employed to build a participatory regulatory mechanism that synthesised norms aligned with the users' preferences. There, users' complaints were considered as conflicts that triggered norm synthesis. In this way, norms were synthesised to regulate the type of situations the agents felt uncomfortable with. From the experiments we noted that, in our on-line community scenario, being more deliberative becomes a requirement. Briefly, the more evidence gathered about a norm's necessity before including it in the normative system, the surer one can be that the norm is "supported" by the overall community, namely that a sufficient amount of users believe that the norm is necessary.

9.3 Future work

While this dissertation has pursued to contribute to the synthesis of normative systems for open MAS, it also opens several paths for future work. In the following, some problems where further research is due are presented.

9.3.1 On the synthesis of norms

As detailed in Section 9.1.1, the norm synthesis model introduced in this thesis is based on the assumption that a conflict is caused by a *single action* performed by a *single agent* in the state previous to the conflict. As an example, in the road traffic scenario, a collision between two cars is considered to be caused by the action that one of the two the cars performed before colliding (e.g., *"go forward"*). Nevertheless, it may be the case that a conflict is caused by a combination of actions. For instance, a collision between two cars may unleash further collisions. Note that, in this case, these latter collisions:

1. Would not be caused by the last cars that collided. Instead, they would be caused by one of the two cars that caused the first collision.

2. Would not be caused in the state previous to the collision. Instead, they would be caused several states before, when the first collision arose.

Therefore, a more general, sophisticated approach would be considering that a conflict may be caused by a combination of actions along a series of states. This would enable our norm synthesis model to be appropriate to synthesise norms in domains with these particular characteristics.

9.3.2 On the relationships between norms

This dissertation has provided means to detect and to exploit three types of relationships between norms: generalisation, substitutability, and complementarity. As previously detailed, while generalisation relationships can be employed to synthesise compact normative systems, substitutability and complementarity relationships can be employed to synthesise liberal normative systems without compromising effectiveness. Overall, we have seen that exploiting norm relationships is a powerful means to yield compact and liberal normative systems. Therefore, it would pay off to investigate further relationships between norms. For instance, one may consider *exclusivity* between norms: two norms would be exclusive if they performed worse together than separately. By exploiting exclusivity, a norm synthesis strategy may yield normative systems that are *more effective* in avoiding conflicts.

Additionally, further improvements may be done. For instance, the LION synthesis strategy detects substitutability and complementarity between *pairs* of norms, but it is not capable of detecting such relationships between *groups* of norms. Therefore, it would pay off to endow our norm synthesis mechanism with the computational capability to detect such kind of non-pairwise relationships.

9.3.3 Configuring a norm synthesis strategy

The norm synthesis strategies provided in this dissertation are highly flexible and customizable. Overall, they provide means to: (1) adjust the amount of evidences considered when evaluating norms (from BASE to DESMON); (2) configure norm generalisation (SIMON, LION and DESMON); (3) adjust the amount

of evidences required to detect norms' semantic relationships (LION); and to (4) adjust the degree of reactivity to conflicts (DESMON)

Nevertheless, each synthesis strategy requires to be manually configured previously to be executed by a Norm Synthesis Machine, and its configuration cannot not automatically change at runtime unless a designer re-configures it. This may be a disadvantage if the conditions of a MAS are unknown at design time, or if these conditions change at runtime. Therefore, our synthesis strategies could be extended with means to automatically learn at runtime the parameters that would best fit to each particular domain. In this way, they would be able to re-configure themselves at runtime, thus adapting to changes of a MAS as it evolves.

9.3.4 Considering additional norm synthesis goals

In this thesis, we have considered three synthesis objectives: effectiveness, compactness, and liberality. Overall, these goals can be employed to synthesise normative systems that successfully avoid conflicts, while reducing agents' computational efforts when reasoning about norms, and preserving their freedom as much as possible. However, we may consider further synthesis objectives, such as synthesising normative systems that are *fair*. Fairness is a concept that has been employed in the field of common-pool resource allocation in open MAS [Pitt et al., 2012, Pitt and Diaconescu, 2014]. There, fairness is generally understood as allocating resources equitably. To measure fairness, they incorporate the concepts of distributive justice [Rescher, 1969]. Along these lines, the concepts of fairness and distributive justice could be incorporated in norm synthesis to synthesise normative systems that achieve coordination, while being compact, liberal, and "fair" to agents.

Bibliography

[Aamodt and Plaza, 1994] Aamodt, A. and Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–59.

[Åberg, 1998] Åberg, L. (1998). Traffic rules and traffic safety. *Safety science*, 29(3):205–215.

[Adler et al., 2005] Adler, J. L., Satapathy, G., Manikonda, V., Bowles, B., and Blue, V. J. (2005). A multi-agent approach to cooperative traffic management and route guidance. *Transportation Research Part B: Methodological*, 39(4):297–318.

[Ågotnes et al., 2007] Ågotnes, T., Van Der Hoek, W., Sierra, C., and Wooldridge, M. (2007). On the logic of normative systems. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, IJCAI '07, pages 1175–1180.

[Agotnes and Wooldridge, 2010] Agotnes, T. and Wooldridge, M. (2010). Optimal Social Laws. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 667–674.

[Ahlfeldt and Maennig, 2010] Ahlfeldt, G. M. and Maennig, W. (2010). Substitutability and complementarity of urban amenities: External effects of built heritage in berlin. *Real Estate Economics*, 38(2):285–323.

[Armengol and Plaza, 2000] Armengol, E. and Plaza, E. (2000). Bottom-up induction of feature terms. *Machine Learning*, 41(3):259–294.

[Artikis et al., 2009] Artikis, A., Kaponis, D., and Pitt, J. (2009). *Multi-Agent Systems: Semantics and Dynamics of Organisational Models*, chapter Dynamic Specifications of Norm-Governed Systems. V. Dignum, IGI Global.

[Axelrod, 1986] Axelrod, R. (1986). An evolutionary approach to norms. *American political science review*, 80(04):1095–1111.

[Axelrod et al., 2004] Axelrod, R., Hammond, R. A., and Grafen, A. (2004). Altruism via kin-selection strategies that rely on arbitrary tags with which they coevolve. *Evolution*, 58(8):1833–1838.

[Axelrod, 1997] Axelrod, R. M. (1997). *The complexity of cooperation: Agent-based models of competition and collaboration*. Princeton University Press.

[Bicchieri, 2006] Bicchieri, C. (2006). *The Grammar of Society: The Nature and Dynamics of Social Norms*. Cambridge University Press.

[Binmore, 2005] Binmore, K. (2005). *Natural Justice*. Oxford University Press.

[Boella and Lesmo, 2001] Boella, G. and Lesmo, L. (2001). Deliberate normative agents. In *Social order in MAS*. Kluwer.

[Boella and van der Torre, 2004] Boella, G. and van der Torre, L. (2004). Regulative and constitutive norms in normative multiagent systems. *Proceedings of KR'04*, pages 255–265.

[Boella and van der Torre, 2006] Boella, G. and van der Torre, L. (2006). An architecture of a normative system: counts-as conditionals, obligations and permissions. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 229–231. ACM.

[Boella and Van Der Torre, 2007] Boella, G. and Van Der Torre, L. (2007). Norm negotiation in multiagent systems. *International Journal of Cooperative Information Systems*, 16(01):97–122.

[Boella et al., 2006] Boella, G., Van Der Torre, L., and Verhagen, H. (2006). Introduction to normative multiagent systems. *Computational & Mathematical Organization Theory*, 12(2-3):71–79.

[Boella et al., 2008a] Boella, G., Van Der Torre, L., and Verhagen, H. (2008a). Introduction to the special issue on normative multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 17(1):1–10.

[Boella et al., 2008b] Boella, G., van der Torre, L., and Verhagen, H. (2008b). Ten challenges for normative multiagent systems. In *Dagstuhl Seminar Proceedings*, volume 8361. Citeseer.

[Bollinger, 2001] Bollinger, J. (2001). Bollinger on bollinger band.

[Bou et al., 2006] Bou, E., López-Sánchez, M., and Rodríguez-Aguilar, J. A. (2006). Adaptation of autonomic electronic institutions through norms and institutional agents. In *Engineering Societies in the Agents World*, Lecture Notes in Computer Science, page In press. Springer.

[Bou et al., 2007a] Bou, E., López-Sánchez, M., and Rodríguez-Aguilar, J. A. (2007a). Towards self-configuration in autonomic electronic institutions. In *COIN 2006 Workshops*, number LNAI 4386, pages 220–235. Springer.

[Bou et al., 2007b] Bou, E., Lpez-Snchez, M., and Rodrguez-Aguilar, J. A. (2007b). Self-adaptation in autonomic electronic institutions through case-based reasoning. In *Proceedings of MA4CS Workshop of ECCS'07*, page To appear as Lecture Notes in Computer Science.

[Broersen et al., 2001] Broersen, J., Dastani, M., Hulstijn, J., Huang, Z., and van der Torre, L. (2001). The boid architecture: conflicts between beliefs, obligations, intentions and desires. In *Proceedings of the fifth international conference on Autonomous agents*, pages 9–16. ACM.

[Bulling and Dastani, 2011] Bulling, N. and Dastani, M. (2011). Verifying normative behaviour via normative mechanism design. In *IJCAI*, volume 11, pages 103–108.

[Burmeister et al., 1997] Burmeister, B., Haddadi, A., and Matylis, G. (1997). Application of multi-agent systems in traffic and transportation. In *Software Engineering. IEE Proceedings-[see also Software, IEE Proceedings]*, volume 144, pages 51–60. IET.

[Campos et al., 2011] Campos, J., Esteva, M., López-Sánchez, M., Morales, J., and Salamó, M. (2011). Organisational adaptation of multi-agent systems in a peer-to-peer scenario. *Computing*, 91(2):169–215.

[Campos et al., 2009] Campos, J., López-Sánchez, M., and Esteva, M. (2009). Assistance layer, a step forward in Multi-Agent Systems Coordination Support. In *Autonomous Agents and Multiagent Systems*, pages 1301–1302.

[Campos et al., 2008] Campos, J., López-Sánchez, M., Rodríguez-Aguilar, J. A., and Esteva, M. (2008). Formalising situatedness and adaptation in electronic institutions. In *Coordination, Organizations, Institutions, and Norms in Agent Systems IV. Lecture Notes in Computer Science (LNAI)*, pages 103–117. Springer.

[Castelfranchi et al., 1998] Castelfranchi, C., Conte, R., and Paolucci, M. (1998). Normative reputation and the costs of compliance. *Journal of Artificial Societies and Social Simulation*, 1(3):3.

[Castelfranchi et al., 1999] Castelfranchi, C., Dignum, F., Jonker, C. M., and Treur, J. (1999). Deliberative normative agents: Principles and architecture. In *ATAL'99*, pages 364–378.

[Chalub et al., 2006] Chalub, F. A., Santos, F. C., and Pacheco, J. M. (2006). The evolution of norms. *Journal of theoretical biology*, 241(2):233–240.

[Chou, 1969] Chou, Y. (1969). *Statistical analysis: with business and economic applications*. Holt, Rinehart and Winston Quantitative Methods Series. Holt, Rinehart and Winston.

[Christelis and Rovatsos, 2009] Christelis, G. and Rovatsos, M. (2009). Automated norm synthesis in an agent-based planning enviroment. In *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 161–168.

[Coleman and Coleman, 1994] Coleman, J. S. and Coleman, J. S. (1994). *Foundations of social theory*. Harvard university press.

[Conte and Castelfranchi, 1995] Conte, R. and Castelfranchi, C. (1995). Understanding the functions of norms in social groups through simulation. *Artificial societies: the computer simulation of social life*.

[Conte et al., 1998] Conte, R., Castelfranchi, C., and Dignum, F. (1998). Autonomous norm acceptance. In *Proceedings of the 5th International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages*, pages 99–112. Springer-Verlag.

[Criado et al., 2010] Criado, N., Argente, E., and Botti, V. (2010). A bdi architecture for normative decision making. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 1383–1384. International Foundation for Autonomous Agents and Multiagent Systems.

[de Pinninck et al., 2007] de Pinninck, A. P., Sierra, C., and Schorlemmer, M. (2007). Friends no more: norm enforcement in multiagent systems. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 92. ACM.

[Dellarocas and Klein, 2000] Dellarocas, C. and Klein, M. (2000). Civil agent societies: Tools for inventing open agent-mediated electronic marketplaces. In Moukas, A., Ygge, F., and Sierra, C., editors, *Agent Mediated Electronic Commerce II*, volume 1788 of *Lecture Notes in Computer Science*, pages 24–39. Springer Berlin Heidelberg.

[Dignum, 1999a] Dignum, F. (1999a). Autonomous agents with norms. *Artif. Intell. Law*, 7(1):69–79.

[Dignum, 1999b] Dignum, F. (1999b). Social interactions of autonomous agents: Private and global views on communication. In *Formal Models of Agents*, pages 103–122. Springer.

[Dignum et al., 2000] Dignum, F., Morley, D., Sonenberg, E. A., and Cavedon, L. (2000). Towards socially sophisticated bdi agents. In *MultiAgent Systems, 2000. Proceedings. Fourth International Conference on*, pages 111–118. IEEE.

[Dignum, 2009] Dignum, V. (2009). *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models: Semantics and Dynamics of Organizational Models*. IGI Global.

[Dignum et al., 2004] Dignum, V., Dignum, F., and Meyer, J.-J. (2004). An agent-mediated approach to the support of knowledge sharing in organizations. *The Knowledge Engineering Review*, 19(02):147–174.

[Doniec et al., 2008] Doniec, A., Mandiau, R., Piechowiak, S., and Espié, S. (2008). A behavioral multi-agent model for road traffic simulation. *Engineering Applications of Artificial Intelligence*, 21(8):1443–1454.

[Dresner and Stone, 2008] Dresner, K. and Stone, P. (2008). A multiagent approach to autonomous intersection management. *Journal of Artificial Intelligence Research (JAIR)*, 31:591–656.

[Elster, 1989a] Elster, J. (1989a). *The cement of society: A survey of social order*. Cambridge University Press.

[Elster, 1989b] Elster, J. (1989b). Social norms and economic theory. *The Journal of Economic Perspectives*, pages 99–117.

[Epstein, 2001] Epstein, J. (2001). Learning to be thoughtless: Social norms and individual computation. *Computational Economics*, 18(1):9–24.

[Esteva et al., 2001] Esteva, M., Rodríguez-Aguilar, J. A., Sierra, C., Garcia, P., and Arcos, J. L. (2001). On the formal specifications of electronic institutions. In Dignum, F. and Sierra, C., editors, *AgentLink*, volume 1991 of *Lecture Notes in Computer Science*, pages 126–147. Springer.

[Fischer et al., 1996] Fischer, K., Mller, J. P., Heimig, I., and Scheer, A.-W. (1996). Intelligent agents in virtual enterprises. pages 205–223.

[Fitoussi and Tennenholtz, 2000] Fitoussi, D. and Tennenholtz, M. (2000). Choosing social laws for multi-agent systems: Minimality and simplicity. *Artificial Intelligence*, 119(1-2):61–101.

[Flentge et al., 2001] Flentge, F., Polani, D., and Uthmann, T. (2001). Modelling the emergence of possession norms using memes. *Journal of Artificial Societies and Social Simulation*, 4(4).

[Fox et al., 2000] Fox, M. S., Barbuceanu, M., and Teigen, R. (2000). Agent-oriented supply-chain management. *International Journal of Flexible Manufacturing Systems*, 12(2-3):165–188.

[France et al., 2003] France, J., Ghorbani, A., et al. (2003). A multiagent system for optimizing urban traffic. In *Intelligent Agent Technology, 2003. IAT 2003. IEEE/WIC International Conference on*, pages 411–414. IEEE.

[García-Camino et al., 2006] García-Camino, A., Rodríguez-Aguilar, J.-A., Sierra, C., and Vasconcelos, W. (2006). Norm-oriented programming of electronic institutions. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 670–672. ACM.

[García-Camino et al., 2009] García-Camino, A., Rodríguez-Aguilar, J. A., Sierra, C., and Vasconcelos, W. (2009). Constraint rule-based programming of norms for electronic institutions. *Autonomous Agents and Multi-Agent Systems*, 18(1):186–217.

[Georgeff and Lansky, 1987] Georgeff, M. P. and Lansky, A. L. (1987). Reactive reasoning and planning. In *AAAI*, volume 87, pages 677–682.

[Gibbs, 1965] Gibbs, J. P. (1965). Norms: The problem of definition and classification. *American Journal of Sociology*, pages 586–594.

[Goldman and Rosenschein, 1993] Goldman, C. V. and Rosenschein, J. S. (1993). Emergent coordination through the use of cooperative state-changing rules. In *Proceedings of the Twelfth International Workshop on Distributed Artificial Intelligence*, pages 171–185.

[Governatori and Rotolo, 2007] Governatori, G. and Rotolo, A. (2007). Bio logical agents: norms, beliefs, intentions in defeasible logic. In *Normative Multi-Agent Systems (NorMAS)*, number 07122, pages 1–34. LZI: Leibniz-Zentrum für Informatik.

[Griffiths and Luck, 2010] Griffiths, N. and Luck, M. (2010). Norm Emergence in Tag-Based Cooperation. In *Proceedings of COIN*.

[Griffiths and Luck, 2011] Griffiths, N. and Luck, M. (2011). Norm diversity and emergence in tag-based cooperation. In *Coordination, Organizations, Institutions, and Norms in Agent Systems VI*, pages 230–249. Springer.

[Guttman et al., 1998] Guttman, R. H., Moukas, A. G., and Maes, P. (1998). Agent-mediated electronic commerce: a survey. *The Knowledge Engineering Review*, 13(02):147–159.

[Hechter, 1988] Hechter, M. (1988). *Principles of group solidarity*, volume 11. Univ of California Press.

[Hewitt, 1986] Hewitt, C. (1986). Offices are open systems. *ACM Transactions on Information Systems (TOIS)*, 4(3):271–287.

[Hilpinen et al., 1971] Hilpinen, R., Kanger, S., Segerberg, K., and Hansson, B. (1971). *Deontic Logic: Introductory and Systematic Readings*. Reidel.

[Hinds and Lee, 2011] Hinds, D. and Lee, R. M. (2011). Assessing the social network health of virtual communities. In Hershey, E. I. R. M. A., editor, *Virtual Communities: Concepts, Methodologies, Tools and Applications*, chapter 49, pages 715–730. IGI Global.

[Huynh et al., 2006] Huynh, T. D., Jennings, N. R., and Shadbolt, N. R. (2006). An integrated trust and reputation model for open multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 13(2):119–154.

[Iocchi et al., 2001] Iocchi, L., Nardi, D., and Salerno, M. (2001). Reactivity and deliberation: a survey on multi-robot systems. In *Balancing reactivity and social deliberation in multi-agent systems*, pages 9–32. Springer.

[Jennings et al., 1997] Jennings, J. S., Whelan, G., and Evans, W. F. (1997). Cooperative search and rescue with a team of mobile robots. In *Advanced Robotics, 1997. ICAR'97. Proceedings., 8th International Conference on*, pages 193–200. IEEE.

[Jennings, 1993] Jennings, N. R. (1993). Commitments and conventions: The foundation of coordination in multi-agent systems. *The knowledge engineering review*, 8(03):223–250.

[Kanger, 1971] Kanger, S. (1971). New foundations for ethical theory. In Hilpinen, R., editor, *Deontic Logic: Introductory and Systematic Readings*, volume 33 of *Synthese Library*, pages 36–58. Springer Netherlands.

[Kitano et al., 1997] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. (1997). Robocup: The robot world cup initiative. In *Proceedings of the first international conference on Autonomous agents*, pages 340–347. ACM.

[Kitano et al., 1999] Kitano, H., Tadokoro, S., Noda, I., Matsubara, H., Takahashi, T., Shinjou, A., and Shimada, S. (1999). Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research. In *Systems, Man, and Cybernetics, 1999. IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on*, volume 6, pages 739–743. IEEE.

[Kittock, 1995] Kittock, J. (1995). Emergent conventions and the structure of multi–agent systems. In *Lectures in Complex systems: the proceedings of the 1993 Complex systems summer school, Santa Fe Institute Studies in the Sciences of Complexity Lecture Volume VI, Santa Fe Institute*, pages 507–521. Addison-Wesley.

[Kittock, 1993] Kittock, J. E. (1993). Emergent conventions and the structure of multi-agent systems. *L. Nadel and D. Stein, eds.*

[Klein et al., 2003] Klein, M., Rodriguez-Aguilar, J.-A., and Dellarocas, C. (2003). Using domain-independent exception handling services to enable robust open multi-agent systems: The case of agent death. *Autonomous Agents and Multi-Agent Systems*, 7(1-2):179–189.

[Knight, 1993] Knight, K. (1993). Are many reactive agents better than a few deliberative ones? In *IJCAI*, volume 93, pages 432–437.

[Koeppen et al., 2011] Koeppen, J., Lopez-Sanchez, M., Morales, J., and Esteva, M. (2011). Learning from experience to generate new regulations. In *Coordination, Organizations, Institutions, and Norms in Agent Systems VI*, volume 6541 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 337–356. Springer.

[Kollingbaum, 2005] Kollingbaum, M. J. (2005). *Norm-governed practical reasoning agents*. PhD thesis, University of Aberdeen.

[Lewis, 1969] Lewis, D. (1969). *Convention: A Philosophical Study*. Harvard University Press.

[López et al., 2002] López, F. L., Luck, M., and d'Inverno, M. (2002). Constraining autonomy through norms. In *AAMAS*, pages 674–681. ACM.

[López et al., 2006] López, F. L., Luck, M., and dInverno, M. (2006). A normative framework for agent-based systems. *Computational & Mathematical Organization Theory*, 12(2-3):227–250.

[Mavromichalis and Vouros, 2001] Mavromichalis, V. K. and Vouros, G. A. (2001). Balancing between reactivity and deliberation in the icagent framework. In *Balancing reactivity and social deliberation in multi-agent systems*, pages 53–75. Springer.

[McKenzie, 1977] McKenzie, G. (1977). Complementarity, substitutability, and independence. *Oxford Economic Papers*, 29(3):430–441.

[Meyer and Wieringa, 1993] Meyer, J.-J. C. and Wieringa, R. J., editors (1993). *Deontic logic in computer science: normative system specification.* John Wiley and Sons Ltd., Chichester, UK.

[Morales, 2015] Morales, J. (2010–2015). The norm synthesis machine, a library to synthesise norms for multi-agent systems. `https://github.com/NormSynthesis/NormSynthesisMachine`.

[Morales et al., 2011a] Morales, J., López-Sánchez, M., and Esteva, M. (2011a). Evaluation of an automated mechanism for generating new regulations. In *Proceedings of the 14th international conference on Advances in artificial intelligence: spanish association for artificial intelligence*, CAEPIA'11, pages 12–21, Berlin, Heidelberg. Springer-Verlag.

[Morales et al., 2011b] Morales, J., López-Sánchez, M., and Esteva, M. (2011b). Using experience to generate new regulations. In *IJCAI '11: Proceedings of the 22nd International Joint Conference in Artificial Intelligence*, pages 307–312, Barcelona, Spain. AAAI Press, USA.

[Morales et al., 2015a] Morales, J., López-Sánchez, M., Rodríguez-Aguilar, J. A., Vasconcelos, W., and Wooldridge, M. (2015a). On-line automated synthesis of compact normative systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 10(1):2:1–2:33.

[Morales et al., 2013a] Morales, J., Lopez-Sanchez, M., Rodriguez-Aguilar, J. A., Wooldridge, M., and Vasconcelos, W. (2013a). Automated synthesis of normative systems. In *AAMAS '13: Proceedings of the 12th international conference on autonomous agents and multiagent systems*, pages 483–490. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).

[Morales et al., 2014a] Morales, J., Lopez-Sanchez, M., Rodriguez-Aguilar, J. A., Wooldridge, M., and Vasconcelos, W. (2014a). Minimality and simplicity in the on-line automated synthesis of normative systems. In *AAMAS '14: Proceedings of the 13th international conference on autonomous agents and multiagent systems*, pages 109–116, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).

[Morales et al., 2015b] Morales, J., Lopez-Sanchez, M., Rodriguez-Aguilar, J. A., Wooldridge, M., and Vasconcelos, W. (2015b). Extending normlab to spur research on norm synthesis (demonstration). In Printing, S., editor, *AAMAS '15: Proceedings of the 14th international conference on autonomous agents and multiagent systems*, pages 1931–1932, Istanbul, Turkey. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).

[Morales et al., 2015c] Morales, J., Lopez-Sanchez, M., Rodriguez-Aguilar, J. A., Wooldridge, M., and Vasconcelos, W. (2015c). Synthesising liberal normative systems. In *AAMAS '15: Proceedings of the 14th international conference on autonomous agents and multiagent systems*, pages 433–441. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).

[Morales et al., 2013b] Morales, J., López-Sánchez, M., Rodríguez-Aguilar, J. A., Wooldridge, M., and Vasconcelos, W. W. (2013b). Iron: A machine for the automated synthesis of normative systems (demonstration). In *AAMAS '13: Proceedings of the 12th international conference on autonomous agents and multiagent systems*, pages 1389–1390, Saint Paul, Minnesota, USA. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).

[Morales et al., 2015d] Morales, J., Mendizabal, I., and Sánchez-Pinsach, D. (2010–2015d). Normlab, a framework to support research on norm synthesis. `https://github.com/NormSynthesis/NormLabSimulators`.

[Morales et al., 2015e] Morales, J., Mendizabal, I., Sánchez-Pinsach, D., López-Sánchez, M., and Rodríguez-Aguilar, J. A. (2015e). Using iron to build frictionless on-line communities. *AI Communications*, 28:55–71.

[Morales et al., 2014b] Morales, J., Mendizabal, I., Sanchez-Pinsach, D., Lopez-Sanchez, M., Wooldridge, M., and Vasconcelos, W. (2014b). Normlab: A framework to support research on norm synthesis. In *AAMAS '14: Proceedings of the 13th international conference on autonomous agents and multiagent systems*, pages 1697–1698. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).

[Nielsen, 2006] Nielsen, J. (2006). The 90-9-1 rule for participation inequality in social media and online communities. http://www.nngroup.com/articles/participation-inequality/.

[North, 1990] North, D. C. (1990). *Institutions, institutional change and economic performance*. Cambridge university press.

[Omicini, 2001] Omicini, A. (2001). Soda: Societies and infrastructures in the analysis and design of agent-based systems. In *Agent-oriented software engineering*, pages 185–193. Springer.

[Onn and Tennenholtz, 1997] Onn, S. and Tennenholtz, M. (1997). Determination of social laws for multi-agent mobilization. *Artificial Intelligence*, 95(1):155–167.

[Piskorski and Gorbatai, 2013] Piskorski, M. J. and Gorbatai, A. D. (2013). Testing colemans social-norm enforcement mechanism: Evidence from wikipedia. *Harvard Business School Strategy Unit Working Paper*, (11-055).

[Pitt and Diaconescu, 2014] Pitt, J. and Diaconescu, A. (2014). The algorithmic governance of common-pool resources. *From Bitcoin to Burning Man and Beyond: The Quest for Identity and Autonomy in a Digital Society*, pages 119–128.

[Pitt et al., 2012] Pitt, J., Schaumeier, J., Busquets, D., and Macbeth, S. (2012). Self-organising common-pool resource allocation and canons of distributive justice. In *Self-Adaptive and Self-Organizing Systems (SASO), 2012 IEEE Sixth International Conference on*, pages 119–128. IEEE.

[Pujol, 2006] Pujol, J. M. (2006). *Structure in artificial societies*. PhD thesis, PhD thesis, Software Department, Universitat Politénica de Catalunya.

[Rao and Georgeff, 1991] Rao, A. S. and Georgeff, M. P. (1991). Modeling rational agents within a bdi-architecture. *KR*, 91:473–484.

[Rao et al., 1995] Rao, A. S., Georgeff, M. P., et al. (1995). Bdi agents: From theory to practice. In *ICMAS*, volume 95, pages 312–319.

[Rescher, 1969] Rescher, N. (1969). Distributive justice: A constructive critique of the utilitarian theory of distribution.

[Riesbeck and Schank, 1989] Riesbeck, C. K. and Schank, R. C. (1989). *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, Hillsdale, NJ, US.

[Riolo et al., 2001] Riolo, R. L., Cohen, M. D., and Axelrod, R. (2001). Evolution of cooperation without reciprocity. *Nature*, 414(6862):441–443.

[Rosenschein, 1986] Rosenschein, J. S. (1986). Rational interaction: cooperation among intelligent agents.

[Rosenschein and Zlotkin, 1994] Rosenschein, J. S. and Zlotkin, G. (1994). *Rules of encounter: designing conventions for automated negotiation among computers*. MIT press.

[Saam and Harrer, 1999] Saam, N. J. and Harrer, A. (1999). Simulating norms, social inequality, and functional change in artificial societies. *Journal of Artificial Societies and Social Simulation*, 2(1):2.

[Sabater and Sierra, 2002] Sabater, J. and Sierra, C. (2002). Reputation and social network analysis in multi-agent systems. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 475–482. ACM.

[Sadri et al., 2006] Sadri, F., Stathis, K., and Toni, F. (2006). Normative kgp agents. *Computational & Mathematical Organization Theory*, 12(2-3):101–126.

[Salazar et al., 2010a] Salazar, N., Rodrguez-Aguilar, J. A., and Arcos, J. L. (2010a). Convention emergence through spreading mechanisms. In van der Hoek, W., Kaminka, G. A., Lespérance, Y., Luck, M., and Sen, S., editors, *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pages 1431–1432. IFAAMAS.

[Salazar et al., 2010b] Salazar, N., Rodriguez-Aguilar, J. A., and Arcos, J. L. (2010b). Robust coordination in large convention spaces. *AI Communications*, 23(4):357–372.

[Samuelson, 1974] Samuelson, P. A. (1974). Complementarity: An essay on the 40th anniversary of the hicks-allen revolution in demand theory. *Journal of Economic Literature*, pages 1255–1289.

[Savarimuthu et al., 2009] Savarimuthu, B. T. R., Cranefield, S., Boella, G., Noriega, P., Pigozzi, G., and Verhagen, H. (2009). A categorization of simulation works on norms. *Normative Multi-Agent Systems*, pages 39–58.

[Schelling, 2006] Schelling, T. C. (2006). *Micromotives and macrobehavior*. WW Norton & Company.

[Sen and Sen, 2010] Sen, O. and Sen, S. (2010). Effects of social network topology and options on norm emergence. In *Proceedings of COIN*, pages 211–222.

[Sen and Airiau, 2007a] Sen, S. and Airiau, S. (2007a). Emergence of norms through social learning. In *IJCAI*, pages 1507–1512.

[Sen and Airiau, 2007b] Sen, S. and Airiau, S. (2007b). Emergence of norms through social learning. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 1507–1512, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Shoham and Leyton-Brown, 2009] Shoham, Y. and Leyton-Brown, K. (2009). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, New York.

[Shoham and Tennenholtz, 1992a] Shoham, Y. and Tennenholtz, M. (1992a). Emergent conventions in multi-agent systems: Initial experimental results and observations (preliminary report). In Nebel, B., Rich, C., and Swartout, W. R., editors, *KR*, pages 225–231. Morgan Kaufmann.

[Shoham and Tennenholtz, 1992b] Shoham, Y. and Tennenholtz, M. (1992b). On the synthesis of useful social laws for artificial agent societies (preliminary report). In *AAAI*, pages 276–281.

[Shoham and Tennenholtz, 1995] Shoham, Y. and Tennenholtz, M. (1995). On social laws for artificial agent societies: off-line design. *Artificial Intelligence*, 73(1-2):231–252.

[Spivey and Abrial, 1992] Spivey, J. M. and Abrial, J. (1992). *The Z notation*. Prentice Hall Hemel Hempstead.

[Stoytchev and Arkin, 2001] Stoytchev, A. and Arkin, R. C. (2001). Combining deliberation, reactivity, and motivation in the context of a behavior-based robot architecture. In *Computational Intelligence in Robotics and Automation, 2001. Proceedings 2001 IEEE International Symposium on*, pages 290–295. IEEE.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

[Sycara, 1988] Sycara, K. P. (1988). Resolving goal conflicts via negotiation. In *AAAI*, volume 88, pages 245–250.

[Traulsen and Schuster, 2003] Traulsen, A. and Schuster, H. G. (2003). Minimal model for tag-based cooperation. *Physical Review E*, 68(4):046129.

[Tsvetovatyy et al., 1997] Tsvetovatyy, M., Gini, M., Mobasher, B., Ski, Z. W., and Ski, W. (1997). Magma an agent based virtual market for electronic commerce. *Applied Artificial Intelligence*, 11(6):501–523.

[Turner and Killian, 1957] Turner, R. H. and Killian, L. M. (1957). Collective behavior.

[van Der Hoek et al., 2007] van Der Hoek, W., Roberts, M., and Wooldridge, M. (2007). Social laws in alternating time: Effectiveness, feasibility, and synthesis. *Synthese*, 156(1):1–19.

[van der Torre and Tan, 1998] van der Torre, L. W. and Tan, Y.-H. (1998). Prohairetic deontic logic (pdl). In *Logics in Artificial Intelligence*, pages 77–91. Springer.

[Van Der Torre and Tan, 1999] Van Der Torre, L. W. and Tan, Y.-H. (1999). Diagnosis and decision making in normative reasoning. *Artificial Intelligence and Law*, 7(1):51–67.

[van Smoorenburg and van der Velden, 2000] van Smoorenburg, M. and van der Velden, R. (2000). The training of school-leavers: Complementarity or substitution? *Economics of Education Review*, 19(2):207 – 217.

[Villatoro et al., 2011] Villatoro, D., Sabater-Mir, J., and Sen, S. (2011). Social instruments for robust convention emergence. In *IJCAI*, pages 420–425.

[Von Wright, 1951] Von Wright, G. H. (1951). Deontic logic. *Mind*, pages 1–15.

[von Wright, 1963] von Wright, G. H. (1963). Norm and action; a logical enquiry. international library of philosophy and scientific method.

[Walker and Wooldridge, 1995] Walker, A. and Wooldridge, M. (1995). Understanding the emergence of conventions in multi-agent systems. In *International Conference on Multiagent Systems*, ICMAS'95, pages 384–389.

[Wang et al., 2013] Wang, X., Mueen, A., Ding, H., Trajcevski, G., Scheuermann, P., and Keogh, E. (2013). Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26(2):275–309.

[Ward et al., 2009] Ward, A. J., Brown, J. A., and Rodriguez, D. (2009). Governance bundles, firm performance, and the substitutability and complementarity of governance mechanisms. *Corporate Governance: An International Review*, 17(5):646–660.

[Wiering et al., 2000] Wiering, M. et al. (2000). Multi-agent reinforcement learning for traffic light control. In *ICML*, pages 1151–1158.

[Wooldridge et al., 2000] Wooldridge, M., Jennings, N. R., and Kinny, D. (2000). The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and multi-agent systems*, 3(3):285–312.