PONTIFICIA UNIVERSIDAD JAVERIANA - CALI

A Theory to Reason About Distributed Information

By

Sergio Steven Ramírez Rico

DISSERTATION

Submitted in partial fulfillment of the requirements for the PhD

degree in Engineering and Applied Sciences

March, 2021

Doctoral committee:
Dr. Frank Valencia, Director
Dr. Camilo Rueda, co-Director

In memory of my godfather Juvenal Orjuela, who would be proud.

Abstract

This dissertation addresses the issue of reasoning about groups' information in spatially distributed multi-agent systems. The approach is provided as a generalization of spatial constraint systems (scs) which properly represents systems with partial information (constraints) distributed in spaces granted to the system's agents. The scs framework represents spaces as join-preserving functions that allow reasoning about spatial information and knowledge. Intuitively, information stored or residing within an agent's space can be seen as local to the agent or as information that it considers to be true. However, scs does not provide a mechanism to represent and analyze distributed information of groups of agents. Being able to characterize such information is relevant in distributed environments as it corresponds to information distributed among the members of a group, though none of them necessarily having it. Indeed, distributed information can be used, for instance, to analyze or predict changes in the information of a group when an agent is added/removed to/from it, to prevent potentially risky or unwanted evolutions of the system.

This work develops the theory of scs to formalize and analyze information of (possibly infinite) groups of agents. We shall equip scs with adapted join-preserving functions to represent the information of some group I acting itself as an agent. Intuitively, these functions represent the information that belongs to a (virtual) space formed with the agents of I.

Specifically, the main contributions of this work are: (i) the characterization of distributed information of groups as specific join-preserving functions, (ii) formalization of compositional properties of such functions to specify group's information in terms of that of its subgroups, (iii) definition of certain conditions under which the information of an infinite group of agents can be represented in terms of a finite subgroup of those agents, (iv) developing algorithms to compute distributed information and applications in geometry and Mathematical Morphology, and (v) a specification in rewriting logic to represent scs that allows verifying properties such as fault-tolerance and knowledge inference.

Resumen

Esta disertación se enfoca en el análisis de información grupal en sistemas multiagentes espacialmente distribuidos. Se propone una generalización de los sistemas espaciales de restricciones (SCS por sus siglas en inglés) los cuales representan adecuadamente sistemas con información parcial (restricciones) distribuida en espacios proporcionados a los agentes del sistema. En el marco SCS los espacios se representan como funciones que preservan el operator join, que permiten razonar sobre información espacial y conocimiento. Intuitivamente, la información almacenada o que reside en los espacios de los agentes puede verse como local para el agente o como información que este considera verdadera. Sin embargo, SCS no proporciona un mecanismo para representar y analizar la información distribuida de grupos de agentes. Tener la capacidad de caracterizar dicha información es relevante en ambientes distribuidos ya que esta corresponde a información distribuida entre los miembros de un grupo, aunque ninguno de ellos necesariamente la posea. Además, la información distribuida puede ser usada, por ejemplo, para analizar o predecir cambios en la información de un grupo cuando se agrega o remueve un agente, con el propósito de prevenir evoluciones del sistema potencialmente peligrosas o no deseadas.

Este trabajo desarrolla la teoría de SCS para formalizar y analizar información de grupos de agentes (que pueden ser infinitos). Equiparemos SCS con funciones adaptadas, que preservan el operador join, para representar la información de algún grupo I que se entiende como un simple agente. Intuitivamente, estas funciones representan la información que pertenece a un espacio (virtual) que se forma con los agentes de I.

Específicamente, las principales contribuciones de este trabajo son: (i) caracterización de la información distribuida de grupos como funciones particulares que preservan el operador join, (ii) formalización de propiedades composicionales de dichas funciones para especificar la información de grupos en términos de la información de sus subgrupos, (iii) definición de condiciones específicas bajo las cuales la información de un grupo infinito de agentes puede ser representada en términos de un subgrupo finito de dichos agentes, (iv) desarrollo de algoritmos para calcular información distribuida y aplicaciones en geometría y morfología matemática, y (v) una especificación formal en lógica de reescritura para representar SCS que permite la verificación de propiedades tales como tolerancia a fallas e inferencia de conocimiento.

Acknowledgments

After four years of interesting courses, many days of research, some nights without sleep, and a bike accident, I have finished my Ph.D. I would have not achieved it without the support and advice of many people throughout this experience.

I am especially indebted to my supervisors Frank Valencia and Camilo Rueda for trusting me and for their constant support in every stage and aspect of my training. They are admirable researchers, inspiring professors, and faithful friends. I feel really fortunate to have met them and to have the opportunity to work with them. Also, I am grateful to Camilo Rocha for his support and confidence in me and for introducing me to Frank and Camilo Rueda. I admire the three of them for their professionalism and personal values. I say for certain that they are role models for me.

I would like to express my gratitude to Pontificia Universidad Javeriana-Cali and the CLASSIC project of the Minciencias organization for funding my training and research. I also thank team Comète at Inria-École Polytechnique and Catuscia Palamidessi for receiving me as an intern.

I thank my dear friends Alicia Rosales, Andrea Ordoñez, Andrea Ramírez, Angélica Padilla, Carlos Pinzón, David Bravo, Esteban Pachón, Federica Granese, Ganesh Del Grosso, Hernán Carvajal, Jairo Castrellón, Juan Camilo Campos, Juan Carlos Romero, Marco Romanelli, Miguel Romero, Mitzi González, Nicolás López, Oscar González, and Santiago Quintero. Together we have many coffee and beer stories.

To conclude, I am grateful to my parents Gloria Rico and Jorge Ramírez for their unbreakable trust in me and their love, and to my brother Oscar Ramírez for his particular daily humor. Also, I am indebted to my family in Cali that supports me throughout these years. Mostly, I thank my girlfriend Angélica González for her love, friendship, patience, and support. She has been there, through the good and the bad.

Contents

Introduction

Spatial constraint systems (scs) are structures to model spatially distributed information in multi-agent systems. This dissertation studies scs as a framework for reasoning about groups' information in distributed systems. The *thesis* of this dissertation is that scs are suitable to specify and analyze information that is distributed among a possibly infinite group of agents. In this setting, distributed information corresponds to information that not necessarily some member of a group possesses, but that together they are able to deduce.

Distributed systems have established a wide research area with different challenges that have been studied for more than 30 years (some include [14, 23, 41, 42, 43, 44, 53, 61, 83, 84]). Systems involving agents interacting with each other by sharing information and resources have been used to model and solve different problems and issues, such as cloud computing, collaborative editing, shared resources, e-commerce, and social networks. In these systems, the agents have granted spaces to store, e.g., data, facts, beliefs, resources, and even other spaces. Particularly, agents' information has been shown to be important for the evolution of a system, for example, autonomous systems or communication protocols evolve according to the information that they have and receive during execution. Roughly speaking, the information that agents have stored in their spaces represents what they know or believe, and naturally such information governs their actions in the system.

Moreover, in the realm of knowledge analysis, reasoning about groups' information is an important issue. As an example, consider the situation in Figure 0.1 where two projections of a cylinder are generated from two different directions. Now, assume that there are two independent agents, one looking at the square and the other one looking at the circle. If we think of the projections as the information that each agent has or knows, i.e., their local information, and we ask them individually about the object used to produce the image, they may answer that it should be a cube or a sphere. Each answer would depend on which plane the agents are looking at, indeed, neither of them would know by his/her own which is the actual object that produces the image. Nevertheless, by combining the local information of the agents, they should be able to

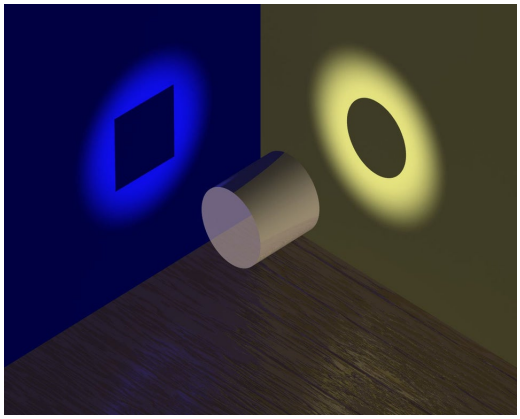


Figure 0.1: Different perceptions.

infer the real shape of the solid - in other words, the knowledge that the object is a cylinder is *distributed* among them. This notion of knowledge is referred to as *distributed knowledge*: it is information that none of the members of a group might possess, but together they are able to deduce.

Distributed information is central to this dissertation and has been relevant to analyze groups' information and understand their behavior. For instance, wisdom of the crowd describes the situation where a group, based on its information and experience, is more likely to make a good decision rather than its individuals. Another example is collective intelligence, which refers to a shared decision process, that can relate not only humans but animals; e.g., a colony of ants is capable of finding shelter and food by performing coordinated actions as a group. Formalizations of cognitive information and behavior of groups are useful for decision-making, consensus processes, group knowledge representations, biological process, robotics, and so on. Therefore, being able to reason about the (distributed) information of a group is of great importance not only in computer science but also in many other disciplines.

In general, local information and groups' information are inherent to systems that involve spatially distributed agents. As in the aforementioned examples, such information can be *spatial*, i.e., information that is stored within a space; or it can be *epistemic*, i.e., something that is known or believed by an agent or a group of agents. This two notions of information raise issues such as mobility between spaces and, appropriate specification of information to capture its epistemic nature. As a response to this matter, *Spatial Constraint Systems* (scs) [51] have been proposed to reason about spatial and epistemic information in concurrent systems. This framework is based on Constraint Systems (cs) [10, 77], that are structures from Concurrent Constraint Programming (CCP) [77] —a class of (concurrent) programming languages for computing with partial information (constraints) taking care of consistency and entailment.

Usually, cs are formalized as complete lattices whose elements represent partial information (constraints)

and their order represents entailment. In CCP, processes interact with a single store by querying and posting information. In scs, the theory of CCP is generalized for systems with spatial distribution of information. Namely, scs enhance cs with a family of certain functions (called space functions), to specify spatial an epistemic information. Different efforts have shown scs to be expressive enough to specify mobility of information and processes between spaces, to formalize knowledge, and to study applications in modal logic with other than spatial and epistemic modalities [32, 34, 35, 36]. However, scs does not provide a theory for reasoning about distributed information nor the mechanisms to compute it.

Given the proliferation of distributed systems and the complex dynamics of their environments and of the interactions between their users, it is of the utmost importance to equip these systems with appropriate modeling structures and with an expressive theory to reason about their behavior. This dissertation proposes a theory to reason about distributed information with infinitely many agents, based on the epistemic notion of *distributed knowledge* from [43]: It corresponds to knowledge "distributed" among the members of a group, without any of them necessarily having it. We shall generalize the theory of scs to formalize distributed information and their properties. Specifically, this approach aims at modeling groups by thinking of them as individuals, we will show that with the proper combination of the agents' information, represented by means of space functions, we can provide a specific space function to represent group's information. Furthermore, we present different characterizations of distributed information and, efficient algorithms to compute it based on notions from lattice theory. Additionally, we present an application in Mathematical Morphology [9], a theory for the analysis of geometrical structures, where we explore the meaning of distributed information and designed efficient algorithms adapted to such theory.

We emphasize that, in multi-agent systems, the number of agents could be unknown in advance and new agents can be added to the system in unpredictable ways. Therefore, it is often convenient to model the group of agents as an infinite set. In fact, in models from economics and epistemic logic [44, 46], groups of agents have been represented as infinite, even uncountable, sets. Accordingly, in this work groups of agents are allowed to be infinite.

Furthermore, in pursue of the general analysis of spatial representation of information, we also present a specification of CCP in rewriting logic including spatial and real-time behavior. This setting allows users to run processes within computational spaces where partial information (constraints) is represented by quantifier-free formulas defined on shared variables of the system. Entailment of constraints is supported by rewriting logic modulo SMT [69].

The development of this dissertation raises different theoretical challenges. Concerning the definition of distributed information, we must enhance space functions from the current version of scs to support in-

finitely many agents and compositional properties of distributed information. Also, the underlying cs must be enriched with notions from lattice theory to compute distributed information as a specific combination of constraints. This enhancement will allow the development of particular algorithms when the cs is distributive. Furthermore, for arbitrary lattices, the algorithms are supported by the fact that the lattice of all space functions inherits the completeness from the complete lattice that specifies the cs. Additionally, the study case on mathematical morphology is a pleasant application of scs that exemplifies its versatility and the accurateness of the proposed theory for distributed information. We will show that each added property faithfully fulfills some of the previously discussed issues. Finally, the real-time specification of spatial CCP in rewriting logic demands a proper implementation of the scheduler that will manage the execution of the processes in the system and, the construction of a real-time rewrite theory to model the constraint system, mobility between spaces, agents' local information, and time requirements.

Context

This dissertation is based on constraint systems and its extension for spatial reasoning, spatial constraint systems. Below we present both of them as a base of the present work.

Constraint Systems

Constraint systems (cs)[1] are algebraic structures for the semantics of process calculi from concurrent constraint programming (CCP) [78]. In this work we shall study cs as semantic structures for distributed information of a group of agents.

A cs is typically formalized as a complete lattice $(\mathbf{C}, \sqsubseteq)$ [10]. The elements of $\mathbf{C}$ represent partial information and we shall think of them as being *assertions*. They are traditionally referred to as *constraints* since they naturally express partial information (e.g., $x > 42$). The relation $\sqsubseteq$ corresponds to information order; $c \sqsubseteq d$, often written $d \sqsupseteq c$, means that c can be derived from d, that d represents at least as much information as c, or that if we assume that d holds true then c also holds true. The join $\sqcup$, the meet $\sqcap$, the bottom *true*, and the top *false* of the lattice correspond to conjunction, disjunction, the empty information and, the join of all (possibly inconsistent) information, respectively.

[1] For simplicity we use *cs* for both *constraint system* and its plural form.

Spatial Constraint Systems

In [51], the authors present the SCCP process calculus to generalize the theory of CCP for systems with spatial distribution of information. They introduce spatial functions to specify spatial and epistemic information. A spatial operator specifies a process, or local store of information, that resides within the space of a given agent. An epistemic construction specifies that the information computed by a process will be known to a given agent.

The notion of computational space and the epistemic notion of belief in SCCP are represented as a family of bottom and join-preserving maps $\mathfrak{s}_i : \mathbf{C} \to \mathbf{C}$ called *space functions*. A cs equipped with space functions is called a *spatial constraint system* (scs). From a *computational point of view*, $\mathfrak{s}_i(c)$ can be interpreted as an assertion specifying that c resides within the space of agent i. Thus, given a constraint $s = \mathfrak{s}_i(c) \sqcup \mathfrak{s}_j(d) \sqcup e$ we may think of c and d as holding within the spaces of agents i and j, respectively. Similarly, $\mathfrak{s}_i(\mathfrak{s}_j(c))$ can be viewed as a hierarchical spatial specification stating that c holds within the space the agent i attributes to agent j. From an *epistemic point of view*, $\mathfrak{s}_i(c)$ specifies that i considers c to be true. An alternative epistemic view is that i interprets c as $\mathfrak{s}_i(c)$. All these interpretations convey the idea of c being local or subjective to agent i.

Previous efforts have focused on mobility of information or processes between spaces. The authors of [32] investigate algebraic operations in scs that help to provide semantic foundations to reason about extrusion and epistemic phenomena. They extend the theory of scs with a family of functions $\mathfrak{e}_i : \mathbf{C} \to \mathbf{C}$, called *extrusion functions*, to specify information that is extruded from inside the space of a given agent. As such, it is possible to specify, e.g., $\mathfrak{s}_i(c \sqcup \mathfrak{e}_i(e)) = \mathfrak{s}_i(c) \sqcup e$, where an element e is extruded from the space of agent i. They also illustrate spatial and epistemic behaviors such as *program mobility* and *intentional lies*.

Contributions and Organization

The main focus of interest of the present work is the developing of the theory of spatial constraint systems to reason about the distributed information of potentially infinite groups. In contrast, we also study the problem of computing distributed information in some fundamental constraint systems (lattices). We provide algorithms to compute distributed information using different techniques derived from the theory of this work.

On the other hand, we present a specification of the spatial CCP process calculus using rewriting logic. We shall formalize partial information, CCP processes and agents that can run such processes in different spaces while subject to real-time requirements (e.g., upper bounds in the execution time of a given opera-

tion), which can be specified with both discrete and dense linear time.

Specifically, the contributions of this dissertation are the following.

Chapter 2: Distributed information

In the SCCP process calculus, scs are used to specify the spatial distribution of information in configurations $\langle P, c\rangle$ where P is a process and c is a constraint, called *the store*, representing the current partial information. For instance, a reduction $\langle\, P, \mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(b)\,\rangle \longrightarrow \langle\, Q, \mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(b \sqcup c)\,\rangle$ means that P, with a in the space of agent 1 and b in the space of agent 2, can evolve to Q while adding c to the space of agent 2.

Given the above reduction, assume that e is some piece of information resulting from the combination (join) of a, b and c, i.e., $e = a \sqcup b \sqcup c$, but strictly above the join of any two of them. We are then in the situation where neither agent has e in their spaces, but as a group they could potentially have e by combining their information. Intuitively, e is *distributed* in the spaces of the group $I = \{1,2\}$. Furthermore, being able to predict the information that agents may derive, as a group, is a relevant issue in multi-agent concurrent systems, particularly if e represents unwanted or conflicting information (e.g., $e = \mathit{false}$).

In this work we develop the theory of group space functions $\mathbb{D}_I : \mathbf{C} \to \mathbf{C}$ to reason about information distributed among the members of a potentially infinite group I. We shall refer to $\mathbb{D}_I$ as the *distributed space* of group I. In this theory $d \sqsupseteq \mathbb{D}_I(e)$ holds exactly when we can derive from d that e is distributed among the agents in I. For example, for $e = a \sqcup b \sqcup c$ given above, we will have $d = \mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(b \sqcup c) \sqsupseteq \mathbb{D}_{\{1,2\}}(e)$ meaning that from the information $\mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(b \sqcup c)$ we can derive that e is distributed among the group $I = \{1,2\}$. Furthermore, $\mathbb{D}_I(e) \sqsupseteq \mathbb{D}_J(e)$ holds whenever $I \subseteq J$, i.e., if e is distributed among a group I, it should also be distributed in a group that includes its agents.

Distributed information of infinite groups can be used to reason about multi-agent computations with unboundedly many agents. For example, a *computation* in SCCP is a possibly infinite reduction sequence γ of the form $\langle\, P_0, c_0\,\rangle \longrightarrow \langle\, P_1, c_1\,\rangle \longrightarrow \cdots$ with $c_0 \sqsubseteq c_1 \sqsubseteq \cdots$. The *result* of γ is $\bigsqcup_{n\geq 0} c_n$, the join of all the stores in the computation. In SCCP all fair computations from a configuration have the same result [51]. Thus, the *observable behaviour* of P with initial store c, written $\mathscr{O}(P,c)$, is defined as the result of any fair computation starting from $\langle P, c\rangle$. Now consider a setting where in addition to their sccp capabilities in [51], processes can also create new agents. Hence, unboundedly many agents, say agents $1, 2, \ldots$, may be created during an infinite computation. In this case, $\mathscr{O}(P,c) \sqsupseteq \mathbb{D}_{\mathbb{N}}(\mathit{false})$, where $\mathbb{N}$ is the set of natural numbers, would imply that some (finite or infinite) set of agents in any fair computation from $\langle P, c\rangle$ may reach contradictory local information among them. Notice that from the above-mentioned properties of distributed spaces, the existence of a finite set of agents $H \subseteq \mathbb{N}$ such that $\mathscr{O}(P,c) \sqsupseteq \mathbb{D}_H(\mathit{false})$ implies

$\mathscr{O}(P,c) \sqsupseteq \mathbb{D}_{\mathbb{N}}(\mathit{false})$. The converse of this implication will be called *group compactness* and we will provide meaningful sufficient conditions for it to hold.

Moreover, we present the particularities of $\mathbb{D}_I$ when the underlying cs is completely distributive. These results will be central to develop the algorithms to compute $\mathbb{D}_I$.

Chapter 3: Computing Distributed Space Functions

In this chapter we shall provide algorithms to compute the distributed information of a given group I, i.e., the distributed space function $\mathbb{D}_I$ that represents the collective information of I.

First, we study the cardinality of the set of all the space functions $\mathscr{S}(L)$ for some fundamental constraint systems (lattices). For the stereotypical non-distributive lattice $\mathbf{M}_n$ (the anti-chain of n elements extended with a top and a bottom element), we show that $|\mathscr{S}(\mathbf{M}_n)| = r_0^n + \cdots + r_n^n + r_1^{n+1} = n!\mathscr{L}_n(-1) + (n+1)^2$ where each term r_k^m in the inner summation represents the number of ways to place k non-attacking rooks on an $m \times m$ board and, $\mathscr{L}_n(x)$ is the Laguerre polynomial of degree n in x, given by the summation $\sum_{k=0}^{n} \binom{n}{k} \frac{(-1)^k}{k!} x^k$. We also present cardinality results for powerset and linear lattices that are part of the lattice theory folklore: The number of space functions is $n^{\log_2 n}$ for powerset lattices of size n and $\binom{2n}{n}$ for linear lattices of size $n+1$.

Furthermore, based on results from Chapter 2, we provide algorithms to compute the distributed space function $\mathbb{D}_I = \bigsqcap_{\mathscr{S}(L)} S$ given a group of agents I of size m with set of space functions $S = \{\mathfrak{s}_i \mid i \in I\} \subseteq \mathscr{S}(L)$ on a constraint system (lattice) L of size n. On the one hand, assuming L to be distributive, we present an algorithm that computes $\mathbb{D}_I$ in terms of the distributed space functions $\mathbb{D}_J$ and $\mathbb{D}_K$ of two subgroups $J, K \subseteq I$ such that $I = J \cup K$. On the other hand, with no assumption on L, we provide an algorithm that finds $\mathbb{D}_I$ by successive approximations starting with the function $\sigma_I(c) \stackrel{\text{def}}{=} \bigsqcap_L \{\mathfrak{s}_i(c) \mid i \in I\}$ for every $c \in L$. In principle, one may think that $\mathbb{D}_I = \sigma_I$, although we will show that it does not hold in general. We also show that $\mathbb{D}_I$ can be computed with worst-case time complexity in $O(n + m\log n)$ for powerset lattices, $O(mn^2)$ for lattices of sets, and $O(nm + n^3)$ for arbitrary lattices.

Finally, in Section 3.3 we investigate applications of the theory developed in this work to geometry and mathematical morphology (MM). Let us use A, B, C and X to denote sets in a vector space. In geometry, the *Minkowski addition* is given by $A \oplus B = \{a + b \mid a \in A, b \in B\}$ [80]. In MM, Minkowski addition is used to define one of its main operators: *dilation*. A dilation by a *structuring element* A can be seen as a function δ_A that transforms every input image X into the image $\delta_A(X) = A \oplus X$ [9]. We show that dilations are space functions and that the distributed space corresponding to these dilations is the dilation that arises from the intersection of their structuring elements: i.e., if $\mathfrak{s}_1 = \delta_A$ and $\mathfrak{s}_2 = \delta_B$ then $\mathbb{D}_{\{1,2\}} = \delta_{A \cap B}$. We also

discuss an epistemic interpretation of space functions in MM, intuitively, space functions capture the agents' perception of a given image.

Chapter 4: Specification in Rewriting Logic

The purpose of this chapter is to provide a specification of SCCP in rewriting logic [55]. It is a logic for concurrency that naturally deals with computational states and concurrent transitions between them. As such, rewriting logic is suitable to specify the actions of agents interacting with each other in the presence of spatial hierarchies for sharing information and knowledge. In addition to the SCCP capabilities presented in [51], we shall include time to represent, e.g., upper bounds in the execution time of processes.

As a novelty, the formal semantics of real-time SCCP is given in the form of a real-time rewriting logic semantics. It is fully executable in the Maude system [13] with the help of rewriting modulo SMT: partial information (i.e., constraints) in the specification is represented by quantifier-free formulas on the shared variables of the system that are under the control of SMT decision procedures. Essentially, the SMT solver will be used to verify entailment of constraints.

In this approach flat configurations of object-like terms encode the hierarchical structure of spaces, and equational and rewrite rules both axiomatize the concurrent computational steps of processes. Time attributes are associated to process-store interaction, as well as to process mobility in the space structure, by means of maps from agents to non-negative real quantities; these choices can be interpreted to denote, as previously mentioned, upper bounds in the execution time of the given operations. By using this specification, we shall symbolically analyze existential real-time reachability properties such as fault-tolerance and knowledge inference.

Organization

1. Chapter 1 presents background on lattice theory and spatial constraint systems.

2. Chapter 2 provides an algebraic theory for reasoning about possibly infinite groups of agents.

   a) Sections 2.1 and 2.2 define space projections and the set of all the space functions, resp.

   b) Section 2.3 characterizes the distributed space $\mathbb{D}_I$ as the *greatest* space function below the space functions that represent the spaces of the agents of a *possibly infinite* group I.

   c) In Section 2.4 we show that distributed spaces have an inherent *compositional* nature: The information of a group is determined by that of its subgroups.

d) Section 2.5 presents distributed spaces in Aumann and Kripke structures.

e) Section 2.6 provides an alternative characterization of a distributed space as the greatest function that satisfies certain basic properties.

f) Section 2.7 defines group projections.

g) Section 2.8 provides a *group compactness* result: Given an infinite group I, we identify a meaningful condition under which $c \sqsupseteq \mathbb{D}_I(e)$ implies $c \sqsupseteq \mathbb{D}_J(e)$ for some finite group $J \subseteq I$. Section 2.9 shows that without such condition the finite group J does not exist.

h) Section 2.10 provides a characterization of distributed spaces for distributive lattices: Given an infinite group I, $\mathbb{D}_I(c)$ can be viewed as the greatest information below all possible combinations of information in the spaces of the agents in I that derive c.

3. In Chapter 3 we provide algorithms to compute $\mathbb{D}_I$.

   a) Section 3.1 presents the cardinality of the set of all the space functions $\mathscr{S}(L)$ for some fundamental lattices.

   b) Section 3.2 provides the algorithms and their time complexity.

   c) Section 3.3 shows an application to mathematical morphology (MM).

4. Chapter 4 introduces the *real-time* SCCP calculus (called rtscc).

   a) Section 4.1 provides preliminaries on rewrite theory, real-time and SMT.

   b) Section 4.2 presents the rewriting logic semantics of rtscc.

   c) Section 4.3 illustrates the use of the rewriting logic semantics for reachability analysis.

Publications Associated to this Dissertation

The results in this document have been published in the following articles and technical reports.

Journal

Michell Guzmán, Sophia Knight, Santiago Quintero, Sergio Ramírez, Camilo Rueda, and Frank Valencia. Reasoning about distributed information with infinitely many agents. *Journal of Logical and Algebraic Methods in Programming*, 121:100674, 2021. doi:https://doi.org/10.1016/j.jlamp.2021.100674

Conferences (refereed)

1. Santiago Quintero, Sergio Ramírez, Camilo Rueda, and Frank Valencia. Counting and computing join-endomorphisms in lattices. In Uli Fahrenberg, Peter Jipsen, and Michael Winter, editors, *Relational and Algebraic Methods in Computer Science - 18th International Conference, RAMiCS 2020, Palaiseau, France, April 8-11, 2020, Proceedings*, volume 12062 of *Lecture Notes in Computer Science*, pages 253–269. Springer, 2020. doi:10.1007/978-3-030-43520-2_16.

2. Michell Guzmán, Sophia Knight, Santiago Quintero, Sergio Ramírez, Camilo Rueda, and Frank Valencia. Reasoning about distributed knowledge of groups with infinitely many agents. In Wan J. Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 29:1–29:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. URL: https://doi.org/10.4230/LIPIcs.CONCUR.2019.29.

3. Sergio Ramírez, Miguel Romero, Camilo Rocha, and Frank Valencia. Real-time rewriting logic semantics for spatial concurrent constraint programming. In Vlad Rusu, editor, *Rewriting Logic and Its Applications - 12th International Workshop, WRLA, Held as a Satellite Event of ETAPS, Thessaloniki, Greece, June 14-15, Proceedings*, volume 11152 of *Lecture Notes in Computer Science*, pages 226–244. Springer, 2018. doi:10.1007/978-3-319-99840-4_13.

Technical Reports

1. Michell Guzmán, Sophia Knight, Santiago Quintero, Sergio Ramírez, Camilo Rueda, and Frank Valencia. Algebraic structures from concurrent constraint programming calculi for distributed information in multi-agent systems, 2020. arXiv:2010.10667

2. Santiago Quintero, Sergio Ramírez, Camilo Rueda, and Frank D. Valencia. Counting and Computing Join-Endomorphisms in Lattices. In *RAMICS - 18th International Conference on Relational and Algebraic Methods in Computer Science*, LNCS, Paris, France, October 2020. Springer. URL: https://hal.archives-ouvertes.fr/hal-02422624

3. Miguel Romero, Sergio Ramírez, Camilo Rocha, and Frank Valencia. A rewriting logic approach to stochastic and spatial constraint system specification and verification, 2019. arXiv:1909.03819

Chapter 1

Preliminaries

The development of this dissertation relies on notions from order theory, lattice theory, and spatial constraint systems. We begin with some definitions and notations used throughout the document.

1.1. Order and Functions

This section presents the foundations and notation that we will use to formalize spatial constraint systems.

1.1.1. Definitions

Definition 1 (Partial Order [15])**.** Let P be a set. A *partial order* on P is a binary relation $\sqsubseteq$ on P that satisfies reflexivity, antisymmetry and transitivity. We use *poset* as a shorthand for partially ordered set; and $(P, \sqsubseteq)$ for the set P with an order relation $\sqsubseteq$.

The next example presents some well-known posets.

Example 1. The following are posets:

1. $(\mathbb{Z}, \leq)$: the integer numbers with the usual order.
2. $(\mathbb{Z}, \mathtt{div}(\_\,\_))$: the integer numbers where $a \sqsubseteq b = \mathtt{div}(a,b)$ holds if *a divides b*.
3. $(\mathscr{P}(S), \subseteq)$: the power set of a set S ordered by inclusion.

Next we introduce some particular elements in ordered sets.

Definition 2 ([15])**.** Let $(P, \sqsubseteq)$ be a poset and let $S \subseteq P$.

- An element $e \in S$ is the *greatest element* of S if $e' \sqsubseteq e$ for all $e' \in S$. If such e exists, we denote it by $\max(S)$. The *least element* of S ($\min(S)$) is defined by duality.
- An element $x \in P$ is an *upper bound* of S if $s \sqsubseteq x$ for all $s \in S$. If the set of all upper bounds of S has a least element, it is called the *least upper bound* of S.
- An element $x \in P$ is a *lower bound* of S if $x \sqsubseteq s$ for all $s \in S$. If the set of all lower bounds of S has a greatest element, it is called the *greatest lower bound* of S.

Notation 1 (Supremum and Infimum)**.** Let $(P, \sqsubseteq)$ be a poset and let $S \subseteq P$. We use $\bigsqcup_P S$ to denote the *least upper bound* (lub) (or *supremum* or *join*) of the elements in S, and $\bigsqcap_P S$ the *greatest lower bound* (glb) (or *infimum* or *meet*) of the elements in S. If $S = \{c, d\}$, $c \sqcup d$ and $c \sqcap d$ represent $\bigsqcup_P S$ and $\bigsqcap_P S$, respectively. If $S = \emptyset$, we denote $\bigsqcup_P S = \mathit{true}$ and $\bigsqcap_P S = \mathit{false}$.
We can always omit the index P from $\bigsqcup_P$ and $\bigsqcap_P$ when no confusion arises.

Some straightforward examples of the above definitions are the following.

Example 2. Consider the partial orders in Example 1.

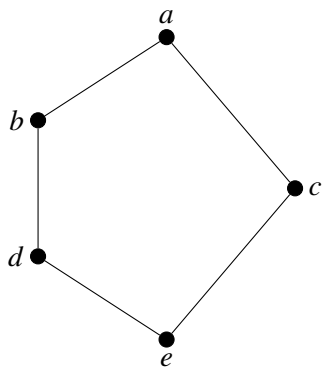
1. In $(\mathbb{Z}, \leq)$ if $S = \{n \mid n \leq 0\}$, then $\bigsqcup S = 0$. If $S = \{n \mid n \geq 1\}$, $\bigsqcap S = 1$.
2. In $(\mathbb{Z}, \mathtt{div}(\_\ \_))$, $\bigsqcup \mathbb{Z} = 0$ and $\bigsqcap \mathbb{Z} = 1$. Here, the join and meet operations are two well-known operations: $\sqcup$ is the *least common multiple* (lcm) and $\sqcap$ is the *greatest common divisor* (gcd).
3. In $(\mathscr{P}(\{1,2,3\}), \subseteq)$, we have $\sqcup = \cup$ and $\sqcap = \cap$. Thus, $\{1\} \sqcup \{2\} = \{1,2\}$ and $\{1\} \sqcap \{1,2,3\} = \{1\}$.

We now introduce complete lattices.

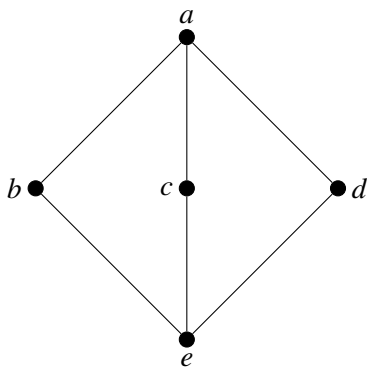
Definition 3 (Complete Lattice)**.** A non-empty poset $(P, \sqsubseteq)$ is said to be a *lattice* if for every $a, b \in P$, both $a \sqcup b$ and $a \sqcap b$ exist. If for every $S \subseteq P$, both $\bigsqcup_P S$ and $\bigsqcap_P S$ exist, then $(P, \sqsubseteq)$ is called a *complete lattice*.

Example 3. Lattices are ordered sets with two additional properties. Some examples include:

1. The posets in Example 1 are lattices. Moreover, $(\mathbb{Z}, \mathtt{div}(\_\ \_))$ and $(\mathscr{P}(\{1,2,3\}), \subseteq)$ are complete lattices. The poset $(\mathbb{Z}, \leq)$ fails to be a complete lattice: for instance, $\bigsqcup \mathbb{Z}$ and $\bigsqcap \mathbb{Z}$ do not exist in $\mathbb{Z}$.
2. Pictorially, Figure 1.1 shows two fundamental complete lattices known as $\mathbf{N}_5$ (Figure 1.1a) and $\mathbf{M}_3$ (Figure 1.1b). These lattices provide a fundamental result: A lattice is distributive if and only if it has not embedded either $\mathbf{N}_5$ or $\mathbf{M}_3$. Moreover, a lattice is modular —a less restrictive version of distributivity— if and only if it has not embedded $\mathbf{N}_5$. The proof of this statement can be found in [15].



(a) Non-modular lattice: $\mathbf{N}_5$.



(b) Non-distributive lattice: $\mathbf{M}_3$.

Figure 1.1: Two fundamental lattices.

Some basic properties are presented in the next definition.

Definition 4 (Properties)**.** Let $(P, \sqsubseteq)$ be a poset.

1. We say that $(P, \sqsubseteq)$ is *distributive* if for every $a, b, c \in P$, $a \sqcup (b \sqcap c) = (a \sqcup b) \sqcap (a \sqcup c)$.

2. A non-empty set $S \subseteq P$ is *directed* if for every pair of elements $a, b \in S$, there exists $c \in S$ such that $a \sqsubseteq c$ and $b \sqsubseteq c$. Equivalently, S is directed iff every finite subset of S has an upper bound in S.

3. An element $c \in P$ is *compact* if for any directed set D of P, $c \sqsubseteq \bigsqcup_P D$ implies $c \sqsubseteq d$ for some $d \in D$.

4. P is said to be a *completely distributive lattice* if it is a complete lattice and for any doubly indexed subset $\{x_{ij}\}_{i \in I, j \in J_i}$ of P

$$\bigsqcup_{i \in I} \left(\bigsqcap_{j \in J_i} x_{ij} \right) = \bigsqcap_{f \in F} \left(\bigsqcup_{i \in I} x_{i f(i)} \right)$$

   where F is the class of choice functions f choosing for each index $i \in I$ some index $f(i) \in J_i$.

The above properties can be checked for our previous posets.

Example 4. Consider the posets in Example 1.

1. All of them are distributive.

2. In $(\mathbb{Z}, \leq)$ any non-empty subset is directed.

3. Let S be a set. The directed sets in $(\mathscr{P}(S), \subseteq)$ are sets with a greatest element. Indeed, the elements of those directed sets are compact.

4. The lattice $\mathbf{M}_3$ in Figure 1.1b fails to be distributive: $b \sqcup (c \sqcap d) = b \sqcup e = b \neq a = a \sqcap a = (b \sqcup c) \sqcap (b \sqcup d)$. Lattice $\mathbf{N}_5$ in Figure 1.1a also fails: $b \sqcap (c \sqcup d) = b \sqcap a = b \neq d = e \sqcup d = (b \sqcap c) \sqcup (b \sqcap d)$.

This section closes with the definition of *self-maps* and some of its properties. We shall represent agents' spaces using this functions.

Definition 5 (Self-maps [26])**.** Let $(L, \sqsubseteq)$ be a complete lattice. A *self-map* on L is a function f from L to L. Let f be a self-map on L. We say that

1. f is *monotonic* if for every $a, b \in L$ such that $a \sqsubseteq b$, then $f(a) \sqsubseteq f(b)$.
2. f *preserves* the join of a set $S \subseteq L$ if $f(\bigsqcup S) = \bigsqcup \{f(c) \mid c \in S\}$.
3. f *preserves arbitrary joins* if it preserves the join of any arbitrary set on L.
4. f is *continuous* if it preserves the join of any directed set on L.

The following is a well-known fact about continuous functions.

Proposition 1 ([1])**.** *Let $(P, \sqsubseteq)$ be a poset where P is a countable set. Let f be a self-map that preserves the join of increasing chains, i.e., for every $S = \{c_1, c_2, \ldots\} \subseteq P$ such that $c_1 \sqsubseteq c_2 \sqsubseteq \cdots$, we have $f(\bigsqcup S) = \bigsqcup \{f(c) \mid c \in S\}$. Then f is continuous.*

1.2. Spatial Constraint Systems

In this section we recall the notion of constraint system and its spatial extension. In [51], the authors presented the novel spatial constraint systems where bottom and join-preserving functions are used to generalize constraint systems to represent (partial) information local to agents. Herein we require these functions to be also continuous, this condition will be necessary to deal with infinitely many agents. We also present some results about join-preserving functions that will be used in later sections.

1.2.1. Constraint systems

Constraint systems (cs) [78] are semantic structures to specify partial information. Following the traditional approach in [10], a cs can be formalized as a complete lattice $(\mathbf{C}, \sqsubseteq)$. The elements of $\mathbf{C}$ are called *constraints* and they represent (partial) information.

Definition 6 (Constraint Systems [10])**.** A *constraint system* (cs) is a complete lattice $(\mathbf{C}, \sqsubseteq)$. The elements of $\mathbf{C}$ are called *constraints*. The symbols $\sqcup$, *true* and *false* will be used to denote the join operation, the bottom, and the top element of $\mathbf{C}$.

The elements of a cs $(\mathbf{C}, \sqsubseteq)$, the *constraints*, represent (partial) information. A constraint c can be viewed as an *assertion*. The lattice order $\sqsubseteq$ is meant to capture entailment of information: $c \sqsubseteq d$, alternatively written $d \sqsupseteq c$, means that the assertion d represents at least as much information as c. We think of $d \sqsupseteq c$ as saying that d *entails* c or that c can be *derived* from d. The operator $\sqcup$ represents join of information; $c \sqcup d$ can be seen as an assertion stating that both c and d hold. We can think of $\sqcup$ as representing conjunction of assertions. The top element represents the join of all, possibly inconsistent, information, hence it is referred to as *false*. The bottom element *true* represents *empty information*. We say that c is *consistent* if $c \neq$ *false*, otherwise we say that c is *inconsistent*. Similarly, we say that c is consistent/inconsistent with d if $c \sqcup d$ is consistent/inconsistent.

The following example presents the Herbrand cs from [10]. It captures syntactic equality between terms based on a first-order alphabet.

Example 5 (Herbrand Constraint System [10])**.** The Herbrand cs captures *syntactic* equality between terms $t, t', \ldots$ built from a first-order alphabet $\mathscr{L}$ with variables $x, y, \ldots$, function symbols, and equality $=$. The constraints are (equivalence classes of) sets of equalities over the terms of $\mathscr{L}$, e.g., $\{x = t, y = t\}$ is a constraint. The relation $c \sqsubseteq d$ holds if the equalities in c follow from those in d, e.g., $\{x = y\} \sqsubseteq \{x = t, y = t\}$. The constraint *false* is the set of all term equalities in $\mathscr{L}$ and *true* is (the equivalence class of) the empty set. The compact elements are the (equivalence class of) finite sets of equalities. The join is the (equivalence class of) set union. Figure 1.2 depicts the Herbrand cs for variables x and y and, terms (constants) `a` and `b`.

In the above example constraints are sets of equations and thus the join of constraints corresponds to the equivalence class of *union* of their equations. We can also view a constraint c as a representation of a set of variable assignments [4]. For instance, a constraint $x > 42$ can be thought of as the set of assignments mapping x to a value greater than 42; i.e., the solutions or models of $x > 42$. In this case the join of constraints naturally corresponds to the *intersection* of their assignments, *false* as the empty set of assignments, and *true* as the set of all assignments. For example, the interpretation of the constraint $x > 42 \wedge x < 50$ corresponds to the intersection (join) of the interpretation of the constraints $x > 42$ and $x < 50$.

The following remark points out that the Herbrand cs is not distributive.

Remark 1. Consider the Herbrand cs in Example 5. Notice that it is not distributive. To see this consider the constraints $c = \{x = \mathtt{a}\}$, $d = \{x = \mathtt{a}, y = \mathtt{a}\}$ and $e = \{x = \mathtt{b}\}$ in Figure 1.2. We have $c \sqcup (d \sqcap e) = c \sqcup$ *true* $=$

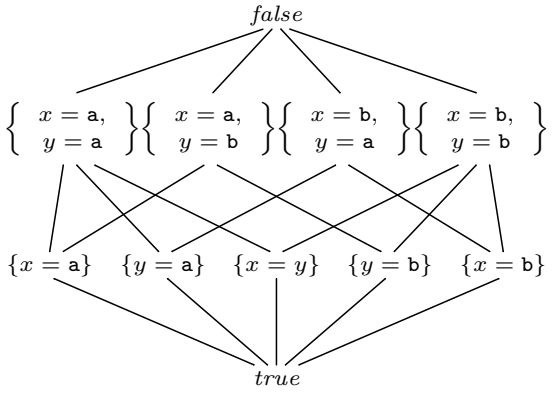


Figure 1.2: Herbrand cs for variables x and y and, constants a and b.

$c \neq (c \sqcup d) \sqcap (c \sqcup e) = d \sqcap \mathit{false} = d$. ■

Distributivity is ubiquitous in order theory and it plays a fundamental role in the results of this work. We consider three forms of distribution.

Definition 7 (Distributive Constraint Systems)**.** A cs $(\mathbf{C}, \sqsubseteq)$ is said to be *distributive (completely distributive)* iff it is a distributive (completely distributive) lattice. It is said to be a *constraint frame* iff its finite joins distribute over arbitrary meets. More precisely, $c \sqcup \bigsqcap S = \bigsqcap \{c \sqcup e \mid e \in S\}$ for every $c \in \mathbf{C}$ and $S \subseteq \mathbf{C}$.

Clearly every completely distributive cs is a constraint frame and every constraint frame is also distributive cs. For finite constraint systems all the three notions of distributivity are equivalent.

Constraint frames allow us to include the subtraction operator from co-Heyting Algebras, the dual of Heyting algebras [86]. The *subtraction operator* $d \ominus c$ in our setting corresponds to the *weakest constraint* one needs to join c with to derive d.

Definition 8 (Subtraction [86])**.** Let $(\mathbf{C}, \sqsubseteq)$ be a cs. Define $d \ominus c$ as $\bigsqcap \{e \in \mathbf{C} \mid c \sqcup e \sqsupseteq d\}$.

The following properties of subtraction correspond to standard logical properties if we interpret $\sqcup$ as conjunction, $\sqsupseteq$ as entailment, and $d \ominus c$ as the logical implication of d by c.

Proposition 2. *Let $(\mathbf{C}, \sqsubseteq)$ be a constraint frame. Then for every $c, d \in \mathbf{C}$ the following properties hold:*

1. $c \sqcup (d \ominus c) = c \sqcup d$,
2. $d \sqsupseteq (d \ominus c)$,
3. $d \ominus c = \mathit{true}$ *iff* $c \sqsupseteq d$.

Proof. Let $(\mathbf{C}, \sqsubseteq)$ be a constraint frame and $c, d \in \mathbf{C}$.

1. From Definitions 7 and 8, $c \sqcup (d \ominus c) = \bigsqcap\{c \sqcup e \mid e \in \mathbf{C} \text{ and } c \sqcup e \sqsupseteq d\}$. Let $A = \{c \sqcup e \mid e \in \mathbf{C} \text{ and } c \sqcup e \sqsupseteq d\}$. Since $c \sqcup d \in A$, then $c \sqcup d \sqsupseteq \bigsqcap A = c \sqcup (d \ominus c)$. For the other direction, notice that $d \sqsubseteq a$ for every $a \in A$, then $d \sqsubseteq \bigsqcap A$. It implies that $c \sqcup d \sqsubseteq c \sqcup \bigsqcap A = c \sqcup (d \ominus c)$.

   Thus $c \sqcup (d \ominus c) = c \sqcup d$, as wanted[1].

2. It follows directly from the definition of $d \ominus c$. Let $\Gamma = \{e \in \mathbf{C} \mid c \sqcup e \sqsupseteq d\}$. Since $d \in \Gamma$, we know that $\bigsqcap \Gamma \sqsubseteq d$. Then, $(d \ominus c) \sqsubseteq d$.

3. By using part (1), we can prove this statement as follows:

$$\begin{aligned} d \ominus c = \mathit{true} &\equiv c \sqcup (d \ominus c) = c \sqcup \mathit{true} \\ &\equiv c \sqcup d = c \\ &\equiv c \sqsupseteq d \end{aligned}$$

■

We conclude this section with a simple completely distributive cs.

Example 6 (Powerset Constraint System)**.** The power set of any set S ordered by inclusion $(\mathscr{P}(S), \subseteq)$ is a cs. In fact it is the stereotypical example of *completely distributive* cs. In this case $\sqsubseteq = \subseteq$, $\mathit{true} = \emptyset$, $\mathit{false} = S$, for every $A, B \subseteq S$, $A \sqcup B = A \cup B$, $A \sqcap B = A \cap B$, $B \ominus A = B \setminus A$.

1.2.2. Spatial Constraint Systems

The authors of [51] extended the notion of cs to account for distributed and multi-agent scenarios with a finite number of agents, each having their own space for local information and computations. The extended structures are called spatial cs (scs). We adapt scs to reason about possibly infinite groups of agents.

A scs is defined over a set (or group) of agents G. Each agent $i \in G$ has a *space* function $\mathfrak{s}_i : \mathbf{C} \longrightarrow \mathbf{C}$ satisfying some structural conditions. Recall that constraints can be viewed as assertions. Thus given $c \in \mathbf{C}$, we can then think of the constraint $\mathfrak{s}_i(c)$ as an assertion stating that c is a piece of information residing *within a space of agent i*. Some alternative *epistemic* interpretations of $\mathfrak{s}_i(c)$ is that it is an assertion stating that agent *i believes c*, that c holds within the space of agent i, or that agent *i interprets c* as $\mathfrak{s}_i(c)$. All these interpretations convey the idea that c is local or subjective to agent i.

[1] This statement is already proved in [32]. It is included here for the sake of completeness.

In [51] scs are used to specify the spatial distribution of information in configurations $\langle P, c\rangle$ where P is a process and c is a constraint. For instance, a reduction $\langle\, P, \mathfrak{s}_i(c) \sqcup \mathfrak{s}_j(d)\,\rangle \longrightarrow \langle\, Q, \mathfrak{s}_i(c) \sqcup \mathfrak{s}_j(d \sqcup e)\,\rangle$ means that P with c in the space of agent i and d in the space of agent j can evolve to Q while adding e to the space of agent j. This notion of space function is formalized in the next definition.

Definition 9 (Space Functions)**.** A *space function* over a constraint system $(\mathbf{C}, \sqsubseteq)$ is a *continuous* self-map $f : \mathbf{C} \longrightarrow \mathbf{C}$ such that for every $c, d \in \mathbf{C}$:

S.1 $f(\mathit{true}) = \mathit{true}$, and

S.2 $f(c \sqcup d) = f(c) \sqcup f(d)$.

We shall use $\mathscr{S}(\mathbf{C})$ to denote the *set of all space functions* over $\mathbf{C}$.

The assertion $f(c)$ can be viewed as saying that c is in the space represented by f. Property S.1 states that having an empty local space amounts to nothing. Property S.2 allows us to join and distribute the information in the space represented by f.

Remark 2 (Continuity). In [51] space functions were not required to be continuous. Nevertheless, we will argue later, in Remark 7, that continuity comes naturally in the intended phenomena we wish to capture: modeling information of possibly *infinite* groups. In fact, in [51] scs could only have finitely many agents. In this work we also extend scs to allow arbitrary, *possibly infinite*, sets of agents. ■

The continuity and preservation of finite joins by space functions will provide us with their preservation of arbitrary joins. The following proposition give us sufficient conditions for the existence of the join of an arbitrary set of a poset $(\mathbf{C}, \sqsubseteq)$. In [50] there is a sketch of its proof, here we present a detailed version of it.

Proposition 3 ([50])**.** *Let $(P, \sqsubseteq)$ be a poset. Suppose that $\bigsqcup F$ and $\bigsqcup D$ exist for every finite set $F \subseteq P$ and for every directed set $D \subseteq P$. Then $\bigsqcup A$ also exists for every $A \subseteq P$.*

Proof. Let $A \subseteq P$ be an arbitrary set and let $D = \{\bigsqcup F \mid F \subseteq A \text{ and } F \text{ is finite}\}$. First we prove that D is directed. For $F_1, F_2 \subseteq A$, both finite sets, the elements $\bigsqcup F_1, \bigsqcup F_2 \in D$ and the set $F_3 = F_1 \cup F_2 \subseteq A$ is finite. Then $\bigsqcup F_3 \in D$ and, both $\bigsqcup F_1 \sqsubseteq \bigsqcup F_3$ and $\bigsqcup F_2 \sqsubseteq \bigsqcup F_3$ hold.

To complete the proof we show that for every $c \in P$, c is an upper bound of D if and only if c is an upper bound of A.

For the "only if" direction, we prove its contrapositive. Let $c \in P$. If c is not an upper bound of A, there is an $a \in A$ such that $a \not\sqsubseteq c$. Notice that $\{a\} \subseteq A$ and thus $\bigsqcup\{a\} = a \in D$ but $a \not\sqsubseteq c$. Hence c is not an upper bound of D.

For the other direction, assume that $c \in \mathbf{C}$ is an upper bound of A. Let F be any finite subset of A. Since $e \sqsubseteq c$ for every $e \in F$, then $\bigsqcup F \sqsubseteq c$. Therefore, c is an upper bound of S.

We conclude with $\bigsqcup A = \bigsqcup S$. ■

The following proposition states two useful properties of space functions: monotonicity and preservation of arbitrary joins.

Proposition 4. *Let $f : \mathbf{C} \to \mathbf{C}$ be a function over a cs $(\mathbf{C}, \sqsubseteq)$. Then*

1. *If f is space function then f is monotonic.*
2. *f is space function if and only if it preserves arbitrary joins.*

Proof. Let $f : \mathbf{C} \to \mathbf{C}$ be a function over a cs $(\mathbf{C}, \sqsubseteq)$.

1. Suppose f is a space function and let $c, d \in \mathbf{C}$ such that $c \sqsubseteq d$. From axiom S.2 in Definition 9, $f(c \sqcup d) = f(c) \sqcup f(d)$. Since $c \sqcup d = d$, we know that $f(c \sqcup d) = f(d)$. Therefore $f(d) = f(c) \sqcup f(d)$, that implies $f(c) \sqsubseteq f(d)$.

2. The "if" direction is immediate. For the "only if" direction, assume that f is a space function. Let $A \subseteq \mathbf{C}$ be an arbitrary set and define $D = \{\bigsqcup F \mid F \subseteq A \text{ and } F \text{ is finite}\}$. We can prove $f(\bigsqcup A) = \bigsqcup \{f(a) \mid a \in A\}$ as follows

$$
\begin{aligned}
f\left(\bigsqcup A\right) &= f\left(\bigsqcup D\right) && (\textstyle\bigsqcup A = \bigsqcup D \text{ from Prop. 3}) \\
&= \bigsqcup \{f(d) \mid d \in D\} && (f \text{ is continuous, } D \text{ is directed}) \\
&= \bigsqcup \left\{ f\left(\bigsqcup F\right) \mid F \subseteq A \text{ and } F \text{ is finite} \right\} && (\text{Definition of } D) \\
&= \bigsqcup \left\{ \bigsqcup \{f(e) \mid e \in F\} \mid F \subseteq A \text{ and } F \text{ is finite} \right\} && (f \text{ preserves finite joins}) \\
&= \bigsqcup \{f(e) \mid e \in A\} && (\text{Simplifying})
\end{aligned}
$$

■

A *spatial constraint system* is a constraint system with a possibly infinite group of agents each one having a space function.

Definition 10 (Spatial Constraint Systems)**.** A *spatial constraint system (scs)* is a constraint system $(\mathbf{C}, \sqsubseteq)$ equipped with a possibly infinite tuple $\mathfrak{s} = (\mathfrak{s}_i)_{i \in G}$ of space functions from $\mathscr{S}(\mathbf{C})$.

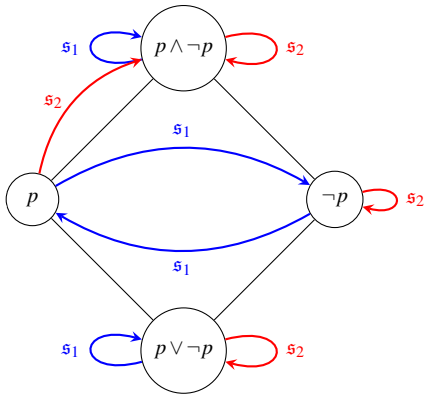


Figure 1.3: Cs given by the complete lattice $\mathbf{M}_2$ ordered by logical implication and space functions $\mathfrak{s}_1$, $\mathfrak{s}_2$.

We shall use $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$ to denote an scs with a tuple $(\mathfrak{s}_i)_{i\in G}$ of space functions. We refer to G and $\mathfrak{s}$ as the *group of agents* and *space tuple* of $\mathbf{C}$, resp., and to each $\mathfrak{s}_i$ as the *space function* in $\mathbf{C}$ of agent i. Subsets of G are also referred to as groups of agents (or sub-groups of G).

Remark 3 (Distributive Spatial Constraint Systems). A scs $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$ is said to be distributive (completely distributive) iff $(\mathbf{C}, \sqsubseteq)$ is a distributive (completely distributive) cs. ■

Let us illustrate a simple scs that will be used throughout the document.

Example 7. The scs $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i\in\{1,2\}})$ in Figure 1.3 is given by the complete lattice $\mathbf{M}_2$ and two agents. We have $\mathbf{C} = \{p \vee \neg p,\ p,\ \neg p,\ p \wedge \neg p\}$ and $c \sqsubseteq d$ holds if c is a logical consequence of d. The top element (false) is $p \wedge \neg p$, the bottom element (true) is $p \vee \neg p$, and the constraints p and $\neg p$ are incomparable with each other.

The set of agents is $\{1,2\}$ with space functions $\mathfrak{s}_1$ and $\mathfrak{s}_2$ as illustrated in Figure 1.3. The intuition is that the agent 2 sees no difference between p and *false* while agent 1 interprets $\neg p$ as p and vice versa.

More involved examples of scs include meaningful families of structures from logic and economics such as Kripke structures and Aumann structures (see Section 2.5 and [51]). We illustrate scs with infinite groups in the next chapter.

Chapter 2

Distributed Information

As we stated before, the central concept of this dissertation is *distributed information*. We aim to model and analyze the information that agents in a system can infer if we bring them together in a group. The purpose of this chapter is to introduce and formalize the notion of distributed information as well as the challenges to provide the theory to reason about it.

The seminal work on knowledge in distributed environments [43], proposes a hierarchy of states of group knowledge. Intuitively, the more knowledge the group has, the higher its rank in the hierarchy is. Thus, the authors present distributed knowledge as the *weakest* state of knowledge in the hierarchy, since the knowledge is distributed among the group's members, without any of them necessarily having it. In contrast, the *strongest* state of knowledge in the hierarchy is common knowledge, which corresponds to "public knowledge." In that approach, they focused on common knowledge and showed its importance in distributed systems, for example, to reach agreement. Regarding distributed knowledge, its importance resides in the capability to acquire new information, e.g., it could be the case that an agent has knowledge of a fact c, and another one has the knowledge that c implies a fact d. Notice that neither agent knows d, but together, they are able to infer it.

Consequently, we believe that a framework for modeling and analyzing information in distributed systems should be expressive enough to capture both common and distributed information. As such, we shall show that scs are suitable for that purpose. Previously, the authors of [51] defined *global information* in scs as a join-based construction of nested spaces w.r.t a group I that captures the notion of information that is globally known by the members of I, i.e., it captures the notion of common knowledge. In this work, we enhance and generalize the theory of scs to reason about group knowledge, in particular we characterize *distributed information* of a group I as the greatest space function, called $\mathbb{D}_I$, below all the space functions of the agents

in I, i.e., the greatest lower bound (a meet-based construction) of the set of all the space functions of the agents in I. Thus, we will equip scs with the theory to capture the notion of distributed knowledge.

Furthermore, we shall provide compositional properties of $\mathbb{D}_I$, also an alternative characterization of it as the greatest function that satisfies certain basic structural properties. Then, we present a compactness result that specifies sufficient conditions under which the distributed information of an infinite group can also be distributed in a finite group; and finally, characterizations of $\mathbb{D}_I$ when the underlying constraint system is completely distributive.

We begin with the definition of space projections, the representation of the information that an agent or group has w.r.t a given constraint.

2.1. Space Projections

In this section we will characterize the notion of collective information of a group of agents. Roughly speaking, the distributed (or collective) information of a group I is the join of each piece of information that resides in the space of an agent $i \in I$. For each constraint c, the distributed information of I w.r.t c is the information distributed among the members of I that can be derived from c. We wish to formalize whether a given constraint e can be derived from the collective information of the group I w.r.t c.

The following example, which we will use throughout this chapter, illustrates the above intuition.

Example 8. Consider a scs $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$ where $G = \mathbb{N}$ and $(\mathbf{C}, \sqsubseteq)$ is a constraint frame. Let $c = \mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(b \ominus a) \sqcup \mathfrak{s}_3(e \ominus b)$. The constraint c specifies the situation where $a, b \ominus a$ and $e \ominus b$ are in the spaces of agent 1, 2 and 3, respectively. Neither agent necessarily holds e in their space w.r.t c. Nevertheless, the information e can be derived from the collective information of the three agents w.r.t c, since from Proposition 2 we have $a \sqcup (b \ominus a) \sqcup (e \ominus b) \sqsupseteq e$. Let us now consider an example with infinitely many agents.

Infinitely Many Agents. Suppose that there exists an element $e' \in \mathbf{C}$ and an increasing chain $a_0 \sqsubseteq a_1 \sqsubseteq \cdots$ such that $\bigsqcup_{i\in\mathbb{N}} a_i \sqsupseteq e'$ and $e' \not\sqsubseteq a_i$ for every $i \in \mathbb{N}$ (i.e., e' is not *compact*, see Definition 4). Let $c' \stackrel{\text{def}}{=} \bigsqcup_{i\in\mathbb{N}} \mathfrak{s}_i(a_i)$. Notice that for no agent $i \in \mathbb{N}$ holds (or can derive) e' in their space since $e' \not\sqsubseteq a_i$. Yet, from our assumption, e' can be derived from the collective information w.r.t c' of all the agents in $\mathbb{N}$, i.e., $\bigsqcup_{i\in\mathbb{N}} a_i \sqsupseteq e'$.

The above example may suggest that distributed information can be obtained by joining individual local information derived from c. Such information can be characterized as the i-projection of agent i w.r.t c.

Definition 11 (Agent and Join Projections)**.** Let $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$ be a scs. Given $i \in G$, the *i-agent projection* of $c \in \mathbf{C}$ is defined as $\pi_i(c) \stackrel{\text{def}}{=} \bigsqcup\{e \mid c \sqsupseteq \mathfrak{s}_i(e)\}$. We say that e is *i-agent derivable* from c if and only if $\pi_i(c) \sqsupseteq e$. Given $I \subseteq G$ the *I-join projection* of a group I of c is defined as $\pi_I(c) \stackrel{\text{def}}{=} \bigsqcup\{\pi_i(c) \mid i \in I\}$. Similarly, we say that e is *I-join derivable* from c if and only if $\pi_I(c) \sqsupseteq e$.

The i-agent projection of $i \in G$ of c naturally represents the join of all the information that agent i has in c. The I-join projection of group I joins individual i-agent projections of c for $i \in I$. This projection can be used as a sound mechanism for reasoning about distributed information: If e is I-join derivable from c then it follows from the distributed information of I w.r.t c.

Example 9. Let $c = \mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(b \ominus a) \sqcup \mathfrak{s}_3(e \ominus b)$ as in Example 8. We have $\pi_1(c) \sqsupseteq a$, $\pi_2(c) \sqsupseteq (b \ominus a)$ and $\pi_3(c) \sqsupseteq (e \ominus b)$. Indeed, e is I-join derivable from c since $\pi_{\{1,2,3\}}(c) = \pi_1(c) \sqcup \pi_2(c) \sqcup \pi_3(c) \sqsupseteq e$. Similarly, we conclude that e' is I-join derivable from c' in Example 8 since $\pi_{\mathbb{N}}(c') = \bigsqcup_{i\in\mathbb{N}} \pi_i(c) \sqsupseteq \bigsqcup_{i\in\mathbb{N}} a_i \sqsupseteq e'$.

Nevertheless, I-join projections do not provide a complete mechanism for reasoning about distributed information as illustrated below.

Example 10. Let $d \stackrel{\text{def}}{=} \mathfrak{s}_1(b) \sqcap \mathfrak{s}_2(b)$. Recall that we think of $\sqcup$ and $\sqcap$ as conjunction and disjunction of assertions: d specifies that b is present in the space of agent 1 or in the space of agent 2 though not exactly in which one. Thus from d we should be able to conclude that b belongs to the space of *some* agent in $\{1,2\}$. Nevertheless, b is not necessarily I-join derivable from d since from $\pi_{\{1,2\}}(d) = \pi_1(d) \sqcup \pi_2(d)$ we cannot, in general, derive b. To see this consider the scs in Figure 2.1, taking $b = \neg p$. We have $\pi_{\{1,2\}}(d) = \pi_1(d) \sqcup \pi_2(d) = \mathit{true} \sqcup \mathit{true} = \mathit{true} \not\sqsupseteq b$. One can generalize this example to infinitely many agents.

Infinite Many Agents. Consider the scs in Example 8 and let $d' \stackrel{\text{def}}{=} \bigsqcap_{i\in\mathbb{N}} \mathfrak{s}_i(b')$. We should be able to conclude from d' that b' is in the space of *some* agent in $\mathbb{N}$ but, in general, b' is not $\mathbb{N}$-join derivable from d'.

2.2. Function Space

We now introduce a new partial order induced by a cs $\mathbf{C}$: The set of space functions ordered point-wise. Recall that $\mathscr{S}(\mathbf{C})$ denotes the set of all space functions over $\mathbf{C}$ (Definition 9). For notational convenience, we shall use $(f_I)_{I\subseteq G}$ to denote the tuple $(f_I)_{I\in\mathscr{P}(G)}$ of elements of $\mathscr{S}(\mathbf{C})$.

Definition 12 (Function Order)**.** Let $(\mathbf{C}, \sqsubseteq)$ be a cs. Given $f, g : \mathbf{C} \longrightarrow \mathbf{C}$ define $f \sqsubseteq_{\mathscr{S}} g$ iff $f(c) \sqsubseteq g(c)$ for every $c \in \mathbf{C}$.

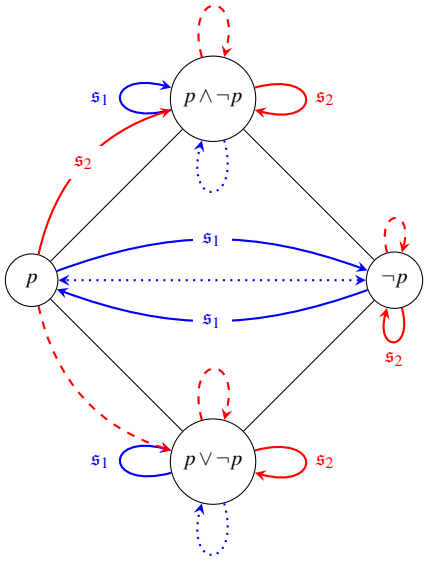


Figure 2.1: Projections π_1 (dotted blue) and π_2 (dashed red) for space functions $\mathfrak{s}_1$ and $\mathfrak{s}_2$.

An important design aspect of our structure is that the set of space functions $\mathscr{S}(\mathbf{C})$ can be made into a complete lattice.

Lemma 1 ([29])**.** *Let $(\mathbf{C}, \sqsubseteq)$ be a cs. Then $(\mathscr{S}(\mathbf{C}), \sqsubseteq_{\mathscr{S}})$ is a complete lattice.*

Proof. Let $S \subseteq \mathscr{S}(\mathbf{C})$ be a subset of space functions. Let $F = \bigsqcup_{\mathscr{S}(\mathbf{C})} S$ and for every constraint $c \in \mathbf{C}$, define $F(c)$ as $\bigsqcup_{\mathbf{C}} \{f(c) \mid f \in S\}$. First, we show that $F \in \mathscr{S}(\mathbf{C})$, i.e., it is a space function.

- $F(\mathit{true}) = \bigsqcup_{\mathbf{C}} \{f(\mathit{true}) \mid f \in S\} = \bigsqcup_{\mathbf{C}} \{\mathit{true}\} = \mathit{true}$.
- $F(c \sqcup d) = F(c) \sqcup F(d)$.

$$\begin{aligned} F(c \sqcup d) &= \bigsqcup\nolimits_{\mathbf{C}} \{f(c \sqcup d) \mid f \in S\} \\ &= \bigsqcup\nolimits_{\mathbf{C}} \{f(c) \sqcup f(d) \mid f \in S\} \\ &= \left(\bigsqcup\nolimits_{\mathbf{C}} \{f(c) \mid f \in S\}\right) \sqcup \left(\bigsqcup\nolimits_{\mathbf{C}} \{f(d) \mid f \in S\}\right) \\ &= F(c) \sqcup F(d) \end{aligned}$$

- Continuity: $F(\bigsqcup_{\mathbf{C}} D) = \bigsqcup_{\mathbf{C}} \{F(d) \mid d \in D\}$ for any directed set D on $\mathbf{C}$.

  Let D be a directed set. From definition of F and by the continuity of each space function $f \in S$, we

have

$$\begin{aligned} F\left(\bigsqcup_{\mathbf{c}} D\right) &= \bigsqcup_{\mathbf{c}} \left\{ f\left(\bigsqcup_{\mathbf{c}} D\right) \mid f \in S \right\} \\ &= \bigsqcup_{\mathbf{c}} \left\{ \bigsqcup_{\mathbf{c}} \{f(d) \mid d \in D\} \mid f \in S \right\} \\ &= \bigsqcup_{\mathbf{c}} \left\{ \bigsqcup_{\mathbf{c}} \{f(d) \mid f \in S\} \mid d \in D \right\} \\ &= \bigsqcup_{\mathbf{c}} \{F(d) \mid d \in D\} \end{aligned}$$

as required.

Now, we prove that F is the least upper bound of S. By definition, F is an upper bound of S: for every $f \in S$, $f \sqsubseteq_{\mathscr{S}} F$ holds. To complete the proof, take any other space function, say $g \in \mathscr{S}(\mathbf{C})$ s.t. $f \sqsubseteq_{\mathscr{S}} g$ for every $f \in S$. Then, for all $f \in S$ we have

$$\begin{aligned} f \sqsubseteq_{\mathscr{S}} g &\equiv \forall c{\in}\mathbf{C}\,(f(c) \sqsubseteq g(c)) \\ &\equiv \forall c{\in}\mathbf{C}\left(\bigsqcup_{\mathbf{c}} \{f(c) \mid f \in S\} \sqsubseteq g(c)\right) \\ &\equiv \forall c{\in}\mathbf{C}\,(F(c) \sqsubseteq g(c)) \\ &\equiv F \sqsubseteq_{\mathscr{S}} g \end{aligned}$$

Therefore, $\mathscr{S}(\mathbf{C})$ is a complete lattice. ■

In order to understand the notion of distributed spaces, it is convenient to give the following intuition first.

Remark 4. Suppose that f and g are space functions in $\mathscr{S}(\mathbf{C})$. Intuitively, $f(c)$ means that c belongs to the space represented by f. By definition, $f \sqsubseteq_{\mathscr{S}} g$ means that $f(c) \sqsubseteq g(c)$ for every $c \in \mathbf{C}$. Intuitively, from the assertion that c is in the space represented by g we can derive that c is also in the space represented by f. This can be interpreted as saying that the space represented by g is included in the space represented by f; in other words *the bigger the space, the smaller the function that represents it*. Thus every c in g is also in f, hence f is a bigger space. ■

Following the above intuition, the order relation $\sqsubseteq_{\mathscr{S}}$ of $\mathscr{S}(\mathbf{C})$ represents (reverse) space inclusion and the join and meet operations in $\mathscr{S}(\mathbf{C})$ represent intersection and union of spaces. The biggest and the smallest spaces are represented by the bottom and the top elements of the lattice $\mathscr{S}(\mathbf{C})$, here called $\lambda_{\perp}$ and

$\lambda_\top$, respectively, and defined as follows.

Definition 13 (Top and Bottom Spaces)**.** Let $\mathscr{S}(\mathbf{C})$ be the lattice of space functions. Define $\lambda_\bot$ and $\lambda_\top$ in $\mathscr{S}(\mathbf{C})$ as follows: $\lambda_\bot(c) \stackrel{\text{def}}{=} \mathit{true}$ for every $c \in \mathbf{C}$; and $\lambda_\top(c) \stackrel{\text{def}}{=} \mathit{true}$ if $c = \mathit{true}$ and $\lambda_\top(c) \stackrel{\text{def}}{=} \mathit{false}$ if $c \neq \mathit{true}$.

In the next section we use the properties of $(\mathscr{S}(\mathbf{C}), \sqsubseteq_\mathscr{S})$ to formalize distributed spaces of a group I as the greatest space function below every space function $\mathfrak{s}_i$ with $i \in I$.

2.3. Distributed Spaces as Maximum Spaces.

As we showed in Section 2.1, we need to provide scs with the proper mechanisms to model the distributed information of a group. The next remark specifies our context.

Remark 5 (The Objective). We have just illustrated in Example 10 that the I-join projection of c, $\pi_I(c)$, the join of individual projections, may not project all distributed information of a group I. To solve this problem we develop the notion of I-group projection of c, written $\Pi_I(c)$, by taking inspiration from the relation between space functions and i-agent projections. For that purpose, we will define a space function $\mathbb{D}_I$, called the *distributed space* of group I, that can be thought of as a virtual space including all the information that can be in the space of every member of I. We will then define an I-projection, Π_I, in terms of $\mathbb{D}_I$ much like π_i is defined in terms of $\mathfrak{s}_i$. ■

The distributed space $\mathbb{D}_I$ of a group I can be viewed as the function that represents the smallest space that includes all the local information of the agents in I. From Remark 4, $\mathbb{D}_I$ should be the *greatest space function* below the space functions of the agents in I. The existence of such a function follows from completeness of $(\mathscr{S}(\mathbf{C}), \sqsubseteq_\mathscr{S})$ stated in Lemma 1.

Definition 14 (Distributed Spaces)**.** Let $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$ be a scs. The *distributed spaces* of $\mathbf{C}$ are given by $\mathbb{D} = (\mathbb{D}_I)_{I\subseteq G}$ where

$$\mathbb{D}_I \stackrel{\text{def}}{=} \max\{\, f \in \mathscr{S}(\mathbf{C}) \mid f \sqsubseteq_\mathscr{S} \mathfrak{s}_i \text{ for every } i \in I \,\}.$$

We shall say that *e is distributed among* $I \subseteq G$ w.r.t c if and only if $c \sqsupseteq \mathbb{D}_I(e)$. We shall refer to each $\mathbb{D}_I$ as the *(distributed) space* of the group I.

Remark 6. From Lemma 1, $\mathbb{D}_I = \bigsqcup_{\mathscr{S}(\mathbf{C})}\{\, f \in \mathscr{S}(\mathbf{C}) \mid f \sqsubseteq_\mathscr{S} \mathfrak{s}_i \text{ for each } i \in I \,\} = \bigsqcap_{\mathscr{S}(\mathbf{C})}\{\mathfrak{s}_i \mid i \in I\}$ where $\bigsqcup_{\mathscr{S}(\mathbf{C})}$ and $\bigsqcap_{\mathscr{S}(\mathbf{C})}$ are the join and meet in the complete lattice $(\mathscr{S}(\mathbf{C}), \sqsubseteq_\mathscr{S})$. ■

Let us consider a concrete example.

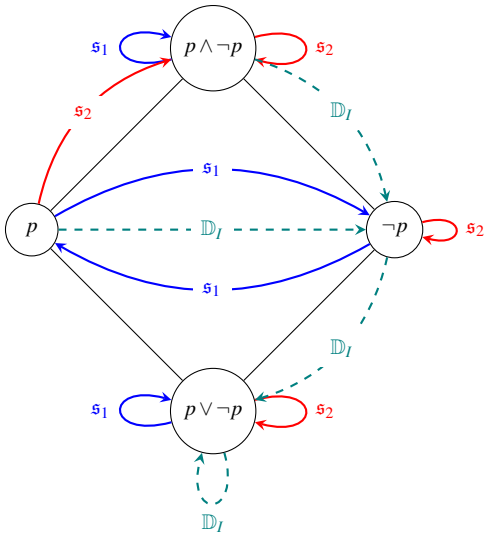


Figure 2.2: $\mathbb{D}_I$ as in Definition 14 with $I = \{1,2\}$ given $\mathfrak{s}_1$ and $\mathfrak{s}_2$.

Example 11. Figure 2.2 illustrates a scs with space functions $\mathfrak{s}_1$ and $\mathfrak{s}_2$, and their distributed space $\mathbb{D}_{\{1,2\}}$. The reader can verify that $\mathbb{D}_{\{1,2\}}$ is indeed the greatest function such that $\mathbb{D}_{\{1,2\}} \sqsubseteq_{\mathscr{S}} \mathfrak{s}_1$ and $\mathbb{D}_{\{1,2\}} \sqsubseteq_{\mathscr{S}} \mathfrak{s}_2$. Notice that $\mathfrak{s}_1(p) \sqcup \mathfrak{s}_2(\neg p) \sqsupseteq \mathbb{D}_{\{1,2\}}(p \sqcup \neg p) = \mathbb{D}_{\{1,2\}}(\mathit{false})$ meaning that if agents 1 and 2 had p and $\neg p$ in their corresponding spaces, as a group they could derive an inconsistency.

In [51], shared information of a group I specifies the fact that a given constraint e resides within the space of every agent in a group I. In contrast, distributed information specifies that e is distributed among the members of I. The next example illustrates this difference.

Example 12. Let $I = \{1,\ldots,n\}$ and let $\mathfrak{s}_I(e) \stackrel{\text{def}}{=} \bigsqcup_{i \in I} \mathfrak{s}_i(e)$ as in [51]. Here $\mathfrak{s}_I(e)$ specifies that e is shared by the agents in I: e is present in the space of i for every $i \in I$. Instead, $\mathbb{D}_I(e)$ means that the information e is distributed among the spaces of the members in I. To see the difference, let $I = \{1,2\}$ and consider the scs in Figure 2.2. Let $c = \mathfrak{s}_1(p) \sqcup \mathfrak{s}_2(\neg p)$. Notice that $p \wedge \neg p$ is distributed in I w.r.t c: $c \sqsupseteq \mathbb{D}_I(p \wedge \neg p)$. However, $p \wedge \neg p$ is not shared information of I w.r.t c, namely, $c \not\sqsupseteq \mathfrak{s}_I(p \wedge \neg p)$.

2.4. Compositionality of Distributed Spaces

Distributed spaces have pleasant compositional properties. They capture the intuition that the *distributed information* of a group I can be obtained from the the distributive information of its subgroups.

Proposition 5. *Let $(\mathbb{D}_I)_{I \subseteq G}$ be the distributed spaces of a scs $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$. Suppose that $K, J \subseteq I \subseteq G$. Then:*

1. $\mathbb{D}_I = \lambda_\top$, *if* $I = \emptyset$.

2. $\mathbb{D}_I = \mathfrak{s}_i$, *if* $I = \{i\}$.

3. $\mathbb{D}_J(a) \sqcup \mathbb{D}_K(b) \sqsupseteq \mathbb{D}_I(a \sqcup b)$.

4. $\mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a) \sqsupseteq \mathbb{D}_I(c)$ *if* $(\mathbf{C}, \sqsubseteq)$ *is a constraint frame.*

Proof. 1. It follows directly from Definition 13 and Definition 14.

2. Let $I = \{i\}$, from Definition 14, $\mathbb{D}_I = \max\{f \in \mathscr{S}(\mathbf{C}) \mid f \sqsubseteq_{\mathscr{S}} \mathfrak{s}_i\} = \mathfrak{s}_i$.

3. Assume $K, J \subseteq I$. From Definition 14 we conclude $\mathbb{D}_I \sqsubseteq_{\mathscr{S}} \mathbb{D}_J$ and $\mathbb{D}_I \sqsubseteq_{\mathscr{S}} \mathbb{D}_K$. Thus $\mathbb{D}_J(a) \sqsupseteq \mathbb{D}_I(a)$, $\mathbb{D}_K(b) \sqsupseteq \mathbb{D}_I(b)$ and therefore $\mathbb{D}_J(a) \sqcup \mathbb{D}_K(b) \sqsupseteq \mathbb{D}_I(a) \sqcup \mathbb{D}_I(b)$. Since $\mathbb{D}_I$ is a space function, $\mathbb{D}_I(a) \sqcup \mathbb{D}_I(b) = \mathbb{D}_I(a \sqcup b)$, then we obtain $\mathbb{D}_J(a) \sqcup \mathbb{D}_K(b) \sqsupseteq \mathbb{D}_I(a \sqcup b)$ as wanted.

4. It follows from part (3) with $a = a$ and $b = c \ominus a$, and Proposition 2.

■

Recall that $\lambda_\top$ corresponds to the empty space (see Definition 13). The first property realizes the intuition that the empty subgroup $I = \emptyset$ *does not* have any information whatsoever distributed w.r.t a consistent constraint c: If $c \sqsupseteq \mathbb{D}_\emptyset(e)$ and $c \neq$ *false* then $e =$ *true*. Intuitively, the second property says that the function $\mathbb{D}_I$ for the group of one agent must be the agent's space function. The third property states that a group can join the information of its subgroups. The last property uses subtraction, hence the constraint frame condition, to express that by joining the information a and $c \ominus a$ of their subgroups, the group I can obtain c.

Let us illustrate how to derive information of a group from smaller ones using Proposition 5.

Example 13. Let $c = \mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(b \ominus a) \sqcup \mathfrak{s}_3(e \ominus b)$ as in Example 8. We want to prove that e is distributed among $I = \{1,2,3\}$ w.r.t c, i.e., $c \sqsupseteq \mathbb{D}_{\{1,2,3\}}(e)$. Using Properties (2) and (4) in Proposition 5 we obtain $c \sqsupseteq \mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(b \ominus a) = \mathbb{D}_{\{1\}}(a) \sqcup \mathbb{D}_{\{2\}}(b \ominus a) \sqsupseteq \mathbb{D}_{\{1,2\}}(b)$, and then $c \sqsupseteq \mathbb{D}_{\{1,2\}}(b) \sqcup \mathfrak{s}_3(e \ominus b) = \mathbb{D}_{\{1,2\}}(b) \sqcup \mathbb{D}_{\{3\}}(e \ominus b) \sqsupseteq \mathbb{D}_{\{1,2,3\}}(e)$ as wanted.

Remark 7 (Continuity and Infinitely Many Agents). The case with infinitely many agents in Example 8 illustrates well why we require our spaces to be continuous in the presence of possibly infinite groups of agents. Clearly for the constraint c' we have: $c' = \bigsqcup_{i \in \mathbb{N}} \mathfrak{s}_i(a_i) \sqsupseteq \bigsqcup_{i \in \mathbb{N}} \mathbb{D}_{\mathbb{N}}(a_i)$. By continuity, $\bigsqcup_{i \in \mathbb{N}} \mathbb{D}_{\mathbb{N}}(a_i) = \mathbb{D}_{\mathbb{N}}(\bigsqcup_{i \in \mathbb{N}} a_i)$ which indeed captures the idea that each a_i is in the distributed space $\mathbb{D}_{\mathbb{N}}$. ■

2.5. Distributed Spaces in Aumann and Kripke Structures

We now consider two important structures for modeling knowledge. Aumann structures are used in mathematical economics for group epistemic reasoning. Kripke structures can be used to interpret modal, doxastic and epistemic logics. We shall prove that the notion of distributed knowledge in each of these structures is an instance of a distributed space.

Aumann Spatial Constraint Systems

Aumann structures [44] are an *event-based* approach to modelling knowledge. An Aumann structure is a tuple $\mathscr{A} = (S, \mathscr{P}_1, \ldots, \mathscr{P}_n)$ where S is a set of states and each $\mathscr{P}_i$ is a partition on S for agent i. The sets of each partition $\mathscr{P}_i$ are called *information sets*. If two states t and u are in the same information set for agent i, it means that in state t agent i considers state u possible, and vice versa. An *event* in an Aumann structure is any subset of S. Event e holds at state t if $t \in e$. The set $\mathscr{P}_i(s)$ denotes the information set of $\mathscr{P}_i$ containing s. The event of *agent i knowing e* is defined as

$$\mathsf{K}_i(e) = \{s \in S \mid \mathscr{P}_i(s) \subseteq e\},$$

and the *distributed knowledge of an event e among the agents in a group I* is defined as

$$\mathsf{D}_I(e) = \left\{ s \in S \;\middle|\; \bigcap_{i \in I} \mathscr{P}_i(s) \subseteq e \right\}.$$

An Aumann structure can be seen as a spatial constraint system $\mathbf{C}(\mathscr{A})$ with events as constraints, i.e., $\mathbf{C} = \{e \mid e \text{ is an event in } \mathscr{A}\}$, and for every $e_1, e_2 \in \mathbf{C}$, $e_1 \sqsubseteq e_2$ iff $e_2 \subseteq e_1$. The operators join ($\sqcup$) and meet ($\sqcap$) are intersection ($\cap$) and union ($\cup$) of events, respectively; *true* $= S$ and *false* $= \emptyset$. The space functions are the knowledge operators, i.e., $\mathfrak{s}_i(c) = \mathsf{K}_i(c)$.

The next proposition states that in fact distributed knowledge and distributed information coincide. To prove this result, we take the liberty of using Theorem 4 which we introduce later in Section 2.10.

Proposition 6 (Distributed Spaces in Aumman Structures)**.** *Let $\mathscr{A} = (S, \mathscr{P}_1, \ldots, \mathscr{P}_n)$ be an Aumann structure and let $\mathbf{C}(\mathscr{A})$ be its induced scs. Then $\mathbb{D}_I = \mathsf{D}_I$ for every $I \subseteq \{1, \ldots, n\}$.*

Proof. Let $I \subseteq G$. We shall prove that (I) $\mathsf{D}_I \in \mathscr{S}(\mathbf{C})$, (II) $\mathbb{D}_I(c) \sqsupseteq \mathsf{D}_I(c)$ and, (III) $\mathbb{D}_I(c) \sqsubseteq \mathsf{D}_I(c)$.

(I) Recall that in $\mathbf{C}(\mathscr{A})$ joins are intersections. From Proposition 4 (2), it suffices to show that given

an arbitrary $\bigcap_j A_j \in \mathbf{C}$, $\mathsf{D}_I\big(\bigcap_j A_j\big) = \bigcap_j \mathsf{D}_I(A_j)$. Indeed, we have $\mathsf{D}_I\big(\bigcap_j A_j\big) = \{s \mid \bigcap_{i\in I} \mathscr{P}_i(s) \subseteq \bigcap_j A_j\} = \{s \mid \bigcap_{i\in I} \mathscr{P}_i(s) \subseteq A_j \text{ for every } j\} = \bigcap_j \{s \mid \bigcap_{i\in I} \mathscr{P}_i(s) \subseteq A_j\} = \bigcap_j \mathsf{D}_I(A_j)$ as wanted.

(II) By definition of K_i and D_I, we know that for every $c \in \mathbf{C}$, $\mathsf{K}_i(c) \subseteq \mathsf{D}_I(c)$. This implies $\mathsf{D}_I(c) \sqsubseteq \mathsf{K}_i(c)$. Then $\mathsf{D}_I(c)$ is a lower bound in $\mathscr{S}(\mathbf{C})$ of the set of all the $\mathsf{K}_i(c)$. Thus, by Remark 6 $\mathbb{D}_I(c) \sqsupseteq \mathsf{D}_I(c)$ for every $c \in \mathbf{C}$.

(III) Now, let $t \in \mathsf{D}_I(c)$. Then, by definition of D_I, we have $\bigcap_{i\in I} \mathscr{P}_i(t) \subseteq c$. From Theorem 4,

$$\mathbb{D}_I(c) = \bigcup \left\{ \bigcap_{i\in I} \mathsf{K}_i(e_i) \;\Big|\; \bigcap_{i\in I} e_i \subseteq c \right\}.$$

Take $e_i = \mathscr{P}_i(t)$ for every $i \in I$. By assumption, $\bigcap_{i\in I} e_i \subseteq c$ and, by definition of K_i, we know that $t \in \mathsf{K}_i(e_i)$. Therefore, $t \in \bigcap_{i\in I} \mathsf{K}_i(e_i)$, i.e., $t \in \mathbb{D}_I(c)$ for every $c \in \mathbf{C}$. Thus $\mathbb{D}_I(c) \sqsubseteq \mathsf{D}_I(c)$ for every $c \in \mathbf{C}$, as wanted.

■

The next example shows that distributed knowledge in epistemic logic is also an instance of distributed information.

Kripke Spatial Constraint Systems [51]

An n-agent Kripke structure (KS) [23] M over a set of atomic propositions Φ is a tuple $M = (S, \pi, \mathscr{R}_1, \ldots, \mathscr{R}_n)$ where S is a nonempty set of states, $\pi : S \to (\Phi \to \{0,1\})$ is an interpretation that associates with each state a truth assignment to the primitive propositions in Φ, and $\mathscr{R}_i$ is a binary relation on S. The states of a KS are often referred to as *worlds*. Each $\mathscr{R}_i$ is referred to as the *possibility* relation for agent i: $(s,t) \in \mathscr{R}_i$ is meant to capture that agent i considers world t possible given its information in world s. The interpretation function π tells us what primitive propositions are true at a given world: p holds at state s iff $\pi(s)(p) = 1$. A *pointed KS* is a pair (M,s) where M is a KS and s, called the *actual world*, is a state of M.

Let $\mathscr{S}_n(\Phi)$ be a non-empty set of n-agent KS over Φ and let Δ be the set of all pointed Kripke structures (M,s) such that $M \in \mathscr{S}_n(\Phi)$. A Kripke n-scs for $\mathscr{S}_n(\Phi)$ is defined as the tuple $\mathbf{K}(\mathscr{S}_n(\Phi)) = (\mathbf{C}, \sqsubseteq, K_1, \ldots, K_n)$ where $\mathbf{C} = \mathscr{P}(\Delta)$, and for every $X, Y \in \mathbf{C} : X \sqsubseteq Y$ iff $Y \subseteq X$, and

$$K_i(X) \stackrel{\text{def}}{=} \{(M,s) \in \Delta \mid \forall t : (s,t) \in \mathscr{R}_i \text{ implies } (M,t) \in X\}$$

for every agent $i \in \{1,\ldots,n\}$. The Kripke n-scs $\mathbf{K}(\mathscr{S}_n(\Phi))$ is a scs where (spatial) constraints are sets of pointed KS ordered by $\supseteq$ and $\mathfrak{s}_i = K_i$ for every agent i. The $\sqcup$ is set intersection, the top element *false* is $\emptyset$, and the bottom *true* is the set Δ of all pointed Kripke structures (M,s) with $M \in \mathscr{S}_n(\Phi)$.

Distributed knowledge in logic is expressed with the formula $D_I(\phi)$ whose intended meaning is that the knowledge ϕ is distributed among the members of I. Its semantics is expressed in terms of its Kripke models; i.e., the models of $D_I(\phi)$ are given by $\mathscr{D}_I(X)$ defined next where X are the models of ϕ [23]. The function $\mathscr{D}_I : \mathbf{C} \to \mathbf{C}$ is defined as

$$\mathscr{D}_I(X) \stackrel{\text{def}}{=} \left\{ (M,s) \in \Delta \;\middle|\; \forall t : (s,t) \in \bigcap_{i\in I} \mathscr{R}_i \text{ implies } (M,t) \in X \right\}.$$

The following proposition states that distributed spaces capture the notion of distributed knowledge, i.e., $\mathbb{D}_I = \mathscr{D}_I$. As in the previous proposition, we use Theorem 4 from Section 2.10 to prove this result.

Proposition 7 (Distributed Spaces in Kripke Structures)**.** *Let* $\mathbf{K}(\mathscr{S}_n(\Phi)) = (\mathbf{C}, \sqsubseteq, K_1, \ldots, K_n)$ *be a Kripke n-scs. Then* $\mathbb{D}_I = \mathscr{D}_I$ *for every* $I \subseteq \{1,\ldots,n\}$.

Proof. Let $I \subseteq G$. We shall prove that (I) $\mathscr{D}_I \in \mathscr{S}(\mathbf{C})$, (II) $\mathbb{D}_I(c) \sqsupseteq \mathscr{D}_I(c)$ and, (III) $\mathbb{D}_I(c) \sqsubseteq \mathscr{D}_I(c)$.

(I) Recall that in $\mathbf{K}(\mathscr{S}_n(\Phi))$ joins are intersections. From Proposition 4 (2), it suffices to show that given an arbitrary $\bigcap_j A_j \in \mathbf{C}$, $\mathscr{D}_I\big(\bigcap_j A_j\big) = \bigcap_j \mathscr{D}_I(A_j)$.

Indeed, we have $\mathscr{D}_I\big(\bigcap_j A_j\big) = \{(M,s) \in \Delta \mid \forall t : (s,t) \in \bigcap_{i\in I} \mathscr{R}_i \text{ implies } (M,t) \in \bigcap_j A_j\} = \{(M,s) \in \Delta \mid \forall t : (s,t) \in \bigcap_{i\in I} \mathscr{R}_i \text{ implies } (M,t) \in A_j \text{ for every } j\} = \bigcap_j \{(M,s) \in \Delta \mid \forall t : (s,t) \in \bigcap_{i\in I} \mathscr{R}_i \text{ implies } (M,t) \in A_j\} = \bigcap_j \mathscr{D}_I(A_j)$ as wanted.

(II) Recall that in $\mathbf{K}(\mathscr{S}_n(\Phi))$, for every $i \in \{1,\ldots,n\}$, $\mathfrak{s}_i = K_i$. By definition of K_i and $\mathscr{D}_I$, we know that for every $X \in \mathbf{C}$, $K_i(X) \subseteq \mathscr{D}_I(X)$. This implies $\mathscr{D}_I(X) \sqsubseteq K_i(X)$. Then $\mathscr{D}_I(X)$ is a lower bound in $\mathscr{S}(\mathbf{C})$ of the set of all the $K_i(X)$. Thus, by Remark 6 $\mathbb{D}_I(X) \sqsupseteq \mathscr{D}_I(X)$ for every $X \in \mathbf{C}$.

(III) Now, let $(M,s) \in \mathscr{D}_I(X)$. Then, by definition of $\mathscr{D}_I$, we have for all t, $(s,t) \in \bigcap_{i\in I} \mathscr{R}_i$ implies $(M,t) \in X$. From Theorem 4,

$$\mathbb{D}_I(X) = \bigcup \left\{ \bigcap_{i\in I} K_i(X_i) \;\middle|\; \bigcap_{i\in I} X_i \subseteq X \right\}.$$

Take $X_i = \{(M,t) \mid (s,t) \in \bigcap_{i\in I} \mathscr{R}_i\}$ for every $i \in I$. It is clear that $\bigcap_{i\in I} X_i \subseteq X$. Notice that by definition of K_i, we know that $(M,s) \in K_i(X_i)$. Therefore, $(M,s) \in \bigcap_{i\in I} \mathsf{K}_i(X_i)$, i.e., $(M,s) \in \mathbb{D}_I(X)$ for every $X \in \mathbf{C}$. Thus $\mathbb{D}_I(X) \sqsubseteq \mathscr{D}_I(X)$ for every $X \in \mathbf{C}$, as wanted.

■

In Proposition 5 we listed some useful properties about $(\mathbb{D}_I)_{I\subseteq G}$. In the next section we shall see that $(\mathbb{D}_I)_{I\subseteq G}$ is the greatest solution of three basic properties.

2.6. Distributed Spaces as Group Distributions Candidates.

We now wish to single out a few fundamental properties on tuples of self-maps that can be used to characterize distributed spaces.

Definition 15 (Distribution Candidates)**.** Let $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$ a scs. A *group distribution candidate* (gdc) of $\mathbf{C}$ is a tuple $\mathbb{d} = (\mathbb{d}_I)_{I\subseteq G}$ of self-maps on $\mathbf{C}$ such that for each $I, J \subseteq G$:

D.1 $\mathbb{d}_I$ is a space function in $\mathbf{C}$,

D.2 $\mathbb{d}_I = \mathfrak{s}_i$, if $I = \{i\}$,

D.3 $\mathbb{d}_I \sqsupseteq_{\mathscr{S}} \mathbb{d}_J$ if $I \subseteq J$.

Property D.1 requires each $\mathbb{d}_I$ to be a space function. This is trivially met for $\mathbb{d}_I = \mathbb{D}_I$. Property D.2 says that the function $\mathbb{d}_I$ for a group of one agent must be the agent's space function. Clearly, $\mathbb{d}_{\{i\}} = \mathbb{D}_{\{i\}}$ satisfies D.2; indeed the distributed space of a single agent is their own space. Finally, Property D.3 states that $\mathbb{d}_I(c) \sqsupseteq \mathbb{d}_J(c)$, if $I \subseteq J$. This is also trivially satisfied if we take $\mathbb{d}_I = \mathbb{D}_I$ and $\mathbb{d}_J = \mathbb{D}_J$. Indeed, if a group I has some distributed information c then any group J, that includes I, should also have c. This realizes the intuition in Remark 4: The bigger the group, the bigger the space and thus the smaller the space function that represents it.

Properties D.1-D.3, however, do not determine $\mathbb{D}$ uniquely. In fact, there could be infinitely-many tuples of space functions that satisfy them. For example, if we were to chose $\mathbb{d}_\emptyset = \lambda_\top$, $\mathbb{d}_{\{i\}} = \mathfrak{s}_i$ for every $i \in G$, and $\mathbb{d}_I = \lambda_\bot$ whenever $|I| > 1$ then D.1-D.3 would be trivially met. But these space functions would not capture our intended meaning of distributed spaces, e.g., we would have $\mathit{true} \sqsupseteq \mathbb{d}_I(e)$ for every constraint e, thus implying that any e would be distributed in the empty information *true* amongst the agents in $I \neq \emptyset$. Nevertheless, we prove that $(\mathbb{D}_I)_{I\subseteq G}$ is the greatest solution satisfying D.1-D.3.

Theorem 1 (Max gdc)**.** *Let $(\mathbb{D}_I)_{I\subseteq G}$ be the distributed spaces of $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$. Then*

1. *$(\mathbb{D}_I)_{I\subseteq G}$ is a gdc of $\mathbf{C}$.*
2. *If $(\mathbb{d}_I)_{I\subseteq G}$ is a gdc of $\mathbf{C}$ then $\mathbb{d}_I \sqsubseteq_{\mathscr{S}} \mathbb{D}_I$ for each $I \subseteq G$.*

Proof. Let $(\mathbb{D}_I)_{I\subseteq G}$ be the distributed spaces of $\mathbf{C}$.

1. We need to prove that $(\mathbb{D}_I)_{I\subseteq G}$ satisfies properties D.1-D.3 in Definition 15.

   Property D.1 follows from definition of $\mathbb{D}_I$ (see Definition 14). Property D.2 is proven in Proposition 5 part (2). For property D.3, let $I,J\subseteq G$ such that $I\subseteq J$. Notice that $\{f\in\mathscr{S}(\mathbf{C})\mid f\sqsubseteq_{\mathscr{S}}\mathfrak{s}_i \text{ for every } i\in I\}\subseteq\{f\in\mathscr{S}(\mathbf{C})\mid f\sqsubseteq_{\mathscr{S}}\mathfrak{s}_j \text{ for every } j\in J\}$. Then $\mathbb{D}_J\sqsubseteq_{\mathscr{S}}\mathbb{D}_I$.

2. Since $(\mathbb{d}_I)_{I\subseteq G}$ is a gdc, we have $\mathbb{d}_I\sqsubseteq_{\mathscr{S}}\mathbb{d}_{\{i\}}=\mathfrak{s}_i$, for every $i\in I$. Then $\mathbb{d}_I\in\{f\in\mathscr{S}(\mathbf{C})\mid f\sqsubseteq_{\mathscr{S}}\mathfrak{s}_i \text{ for every } i\in I\}$ which implies $\mathbb{d}_I\sqsubseteq_{\mathscr{S}}\mathbb{D}_I$.

■

Therefore, Theorem 1 tells us that distributed spaces could have been equivalently defined as the *greatest space functions* satisfying properties D.1-D.3. We shall use the characterization of distributed spaces in Theorem 1 in the proofs of Proposition 8 and Theorem 5 in the next sections. Let us first illustrate the use of such properties in the following example.

Example 14. Let $(\mathbf{C},\sqsubseteq,(\mathfrak{s}_i)_{i\in G})$ and $c=\mathfrak{s}_1(a)\sqcup\mathfrak{s}_2(b\ominus a)\sqcup\mathfrak{s}_3(e\ominus b)$ as in Example 8. Here we shall show that e can be derived from the distributed information among $I=\{1,2,3\}$.
We want to prove $c\sqsupseteq\mathbb{D}_I(e)$ for $I=\{1,2,3\}$. Since $\mathbf{C}$ is a constraint frame, by Definition 15 and Proposition 2 we have

$$\begin{aligned}
c&=\mathbb{D}_{\{1\}}(a)\sqcup\mathbb{D}_{\{2\}}(b\ominus a)\sqcup\mathbb{D}_{\{3\}}(e\ominus b) && \text{(Def. 15 D.2)}\\
&\sqsupseteq\mathbb{D}_I(a)\sqcup\mathbb{D}_I(b\ominus a)\sqcup\mathbb{D}_I(e\ominus b) && \text{(Def. 15 D.3)}\\
&=\mathbb{D}_I(a\sqcup(b\ominus a)\sqcup(e\ominus b)) && \text{(Def. 15 D.1)}\\
&\sqsupseteq\mathbb{D}_I(e). && \text{(Prop. 2)}
\end{aligned}$$

Thus $c\sqsupseteq\mathbb{D}_I(e)$ as wanted.

Infinitely Many Agents. Recall the case with infinitely many agents from Example 8. Similarly, we can show that e' is distributed in $\mathbb{N}$ w.r.t c':

$$c'=\bigsqcup_{i\in\mathbb{N}}\mathbb{D}_{\{i\}}(a_i)\sqsupseteq\bigsqcup_{i\in\mathbb{N}}\mathbb{D}_I(a_i)=\mathbb{D}_I\left(\bigsqcup_{i\in\mathbb{N}}a_i\right)\sqsupseteq\mathbb{D}_I(e').$$

Now consider our counter-example in Example 10 where we have $d=\mathfrak{s}_1(b)\sqcap\mathfrak{s}_2(b)$. Recall that d specifies that b resides either within the space of agent 1 or in the space of agent 2, although we do not know

exactly in which one. Here we wish to prove that b is distributed in the group $\{1,2\}$, i.e., b can be derived from d as being in a space of a member of $\{1,2\}$. We want to prove $d \sqsupseteq \mathbb{D}_I(b)$ for $I = \{1,2\}$:

$$\begin{aligned} d = \mathfrak{s}_1(b) \sqcap \mathfrak{s}_2(b) &= \mathbb{D}_{\{1\}}(b) \sqcap \mathbb{D}_{\{2\}}(b) && \text{(Def. 15 D.2)} \\ &\sqsupseteq \mathbb{D}_{\{1,2\}}(b) \sqcap \mathbb{D}_{\{1,2\}}(b) && \text{(Def. 15 D.3)} \\ &= \mathbb{D}_{\{1,2\}}(b) && \text{(Property of } \sqcap) \end{aligned}$$

Then $d \sqsupseteq \mathbb{D}_{\{1,2\}}(b)$ as wanted.

Infinitely Many Agents. If we have $d' \stackrel{\text{def}}{=} \bigsqcap_{i\in\mathbb{N}} \mathfrak{s}_i(b')$ we can show that b' is distributed in the group $\mathbb{N}$ w.r.t d': $d' = \bigsqcap_{i\in\mathbb{N}} \mathbb{D}_{\{i\}}(b') \sqsupseteq \bigsqcap_{i\in\mathbb{N}} \mathbb{D}_{\mathbb{N}}(b') = \mathbb{D}_{\mathbb{N}}(b')$. ■

The above example shows the capability of $\mathbb{D}_I$ to express the distributed information of a group I in different scenarios. In the later sections we shall provide properties of $\mathbb{D}_I$ and its characterization for completely distributive lattices.

2.7. Group Projections

As promised at the beginning of Section 2.3 in Remark 5, we now give a definition of *Group Projection*. The function $\Pi_I(c)$ extracts exactly all information that the group I may have distributed w.r.t c.

Definition 16 (Group Projection)**.** Let $(\mathbb{D}_I)_{I\subseteq G}$ be the distributed spaces of a scs $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$. Given the set $I \subseteq G$, the I-group projection of $c \in \mathbf{C}$ is defined as $\Pi_I(c) \stackrel{\text{def}}{=} \bigsqcup\{e \mid c \sqsupseteq \mathbb{D}_I(e)\}$. We say that e is I-group derivable from c if and only if $\Pi_I(c) \sqsupseteq e$.

Much like space functions and agent projections, group projections and distributed spaces also form a pleasant correspondence: a *Galois connection* [15].

Proposition 8. *Let $(\mathbb{D}_I)_{I\subseteq G}$ be the distributed spaces of a scs $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$. For every $c, e \in \mathbf{C}$,*

1. *$c \sqsupseteq \mathbb{D}_I(e)$ if and only if $\Pi_I(c) \sqsupseteq e$.*
2. *$\Pi_I(c) \sqsupseteq \Pi_J(c)$ if $J \subseteq I$.*
3. *$\Pi_I(c) \sqsupseteq \pi_I(c)$.*

Proof. Let $(\mathbb{D}_I)_{I\subseteq G}$ be the distributed spaces of $\mathbf{C}$ and let $c, e \in \mathbf{C}$.

1. Let $S = \{d \mid c \sqsupseteq \mathbb{D}_I(d)\}$. First, assume that $c \sqsupseteq \mathbb{D}_I(e)$. Since $e \in S$, by Definition 16, $\Pi_I(c) = \bigsqcup S \sqsupseteq e$. Second, assume $\Pi_I(c) \sqsupseteq e$. Then by monotonicity, $\mathbb{D}_I(\Pi_I(c)) \sqsupseteq \mathbb{D}_I(e)$. From continuity of $\mathbb{D}_I$, we know that $\mathbb{D}_I(\Pi_I(c)) = \mathbb{D}_I(\bigsqcup S) = \bigsqcup\{\mathbb{D}_I(d) \mid d \in S\}$ and by definition of S, for every $d \in S$, we have $c \sqsupseteq \mathbb{D}_I(d)$, then $c \sqsupseteq \bigsqcup\{\mathbb{D}_I(d) \mid d \in S\}$. Therefore, $c \sqsupseteq \mathbb{D}_I(e)$.

2. Given that $(\mathbb{D}_I)_{I \subseteq G}$ is a gdc (see Theorem 1), if $J \subseteq I$, then $\mathbb{D}_J \sqsupseteq_{\mathscr{D}} \mathbb{D}_I$. Hence $\{d \mid c \sqsupseteq \mathbb{D}_J(d)\} \subseteq \{d \mid c \sqsupseteq \mathbb{D}_I(d)\}$ and thus $\Pi_I(c) \sqsupseteq \Pi_J(c)$ for every $c \in \mathbf{C}$.

3. By part (2), for every $\{i\} \subseteq I$ and every $c \in \mathbf{C}$, we have $\Pi_I(c) \sqsupseteq \Pi_{\{i\}}(c)$. It implies, $\Pi_I(c) \sqsupseteq \bigsqcup_{i \in I} \Pi_{\{i\}}(c)$, for every $c \in \mathbf{C}$. Then $\bigsqcup_{i \in I} \Pi_{\{i\}}(c) = \bigsqcup_{i \in I}\{\bigsqcup\{d \mid c \sqsupseteq \mathbb{D}_{\{i\}}(d)\}\} = \bigsqcup_{i \in I}\{\bigsqcup\{d \mid c \sqsupseteq \mathfrak{s}_i(d)\}\} = \bigsqcup_{i \in I}\{\pi_i(c)\} = \pi_I(c)$. Therefore, $\Pi_I(c) \sqsupseteq \pi_I(c)$, for every $c \in \mathbf{C}$.

■

The first property in Proposition 8, a Galois connection, states that we can conclude from c that e is in the distributed space of I exactly when e is I-group derivable from c. The second states that the bigger the group, the bigger the projection. The last property says that whatever is I-join derivable is I-group derivable; although the opposite is not true as shown in Example 10.

2.8. Group Compactness.

Suppose that an *infinite* group of agents I can derive e from c (i.e., $c \sqsupseteq \mathbb{D}_I(e)$). A legitimate question is whether there exists a *finite* sub-group J of agents from I that can also derive e from c. The following theorem provides a positive answer to this question given that e is a compact element (see Definition 4) and I-join derivable from c.

Theorem 2 (Group Compactness)**.** *Let $(\mathbb{D}_I)_{I \subseteq G}$ be the distributed spaces of a scs $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$. Suppose that $c \sqsupseteq \mathbb{D}_I(e)$. If e is compact and I-join derivable from c then there exists a finite set $J \subseteq I$ such that $c \sqsupseteq \mathbb{D}_J(e)$.*

Proof. Suppose that $c \sqsupseteq \mathbb{D}_I(e)$. If I is finite then take $J = I$. If I is not finite, since e is I-join derivable from c we have $\pi_I(c) = \bigsqcup S \sqsupseteq e$ where $S = \{\pi_i(c) \mid i \in I\}$.

Define $D_I = \{\pi_J(c) \mid J \subseteq I \text{ and } J \text{ is finite}\}$. Take any $\pi_H(c), \pi_K(c) \in D_I$. Since H and K are finite, their union $K \cup H$ must also be finite and included in I. Hence $\pi_{H \cup K}(c) \in D_I$. Therefore, D_I is a directed set.

Since $S = \{\pi_i(c) \mid i \in I\} = \{\pi_{\{i\}}(c) \mid i \in I\}$ is included in D_I, we obtain $\bigsqcup D_I \sqsupseteq \bigsqcup S \sqsupseteq e$. But e is compact and D_I directed hence there must be $\pi_J(c) \in D_I$, with J a finite set, such that $\pi_J(c) \sqsupseteq e$. From Proposition 8 (3) and Proposition 8 (1), we conclude $c \sqsupseteq \mathbb{D}_J(e)$ as wanted. ■

Let us illustrate Theorem 2 with our recurrent example.

Example 15. Consider a scs $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$ where $G = \mathbb{N}$ and $\mathbf{C}$ is a countable set. Let $d = \bigsqcup_{i\in\mathbb{N}} \mathfrak{s}_i(a_i)$ for some increasing chain $a_0 \sqsubseteq a_1 \sqsubseteq \cdots$, and $b \in \mathbf{C}$ such that $b \sqsubseteq \bigsqcup_{i\in\mathbb{N}} a_i$.

Notice that $d \sqsupseteq \mathbb{D}_{\mathbb{N}}(b)$ and $\pi_{\mathbb{N}}(d) \sqsupseteq b$. Hence b is $\mathbb{N}$-join derivable from d. If b is compact, by Theorem 2 there must be a finite subset $J \subseteq \mathbb{N}$ such that $d \sqsupseteq \mathbb{D}_J(b)$.

2.9. Group Compactness without *I*-join derivability

Let us assume $c \sqsupseteq \mathbb{D}_I(e)$ as in Theorem 2. By Proposition 8 (1), we know that e is I-group derivable from c but not necessarily I-join derivable from c. The problem in establishing group compactness in the *absence of I-join derivability* has to do with d' in the infinite case in Example 10. We have $d' = \bigsqcap_{i\in\mathbb{N}} \mathfrak{s}_i(b')$. Notice that we cannot guarantee that b' is $\mathbb{N}$-join derivable from d' ($\pi_{\mathbb{N}}(d') \sqsupseteq b'$). One can verify that $d' \sqsupseteq \mathbb{D}_{\mathbb{N}}(b')$, i.e., b' resides in the space of agent i for some $i \in \mathbb{N}$. Then, b' is I-group derivable from d' ($\Pi_{\mathbb{N}}(d') \sqsupseteq b'$). Nevertheless we cannot guarantee the existence of a finite $J \subset \mathbb{N}$ such that that $d' \sqsupseteq \mathbb{D}_J(b')$. In fact, the existence of such a J cannot be guaranteed even if e (b' in Example 10) is compact as stated in the next theorem.

Theorem 3 (Non-Compactness)**.** *There exists a scs $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$ with distributed spaces $(\mathbb{D}_I)_{I\subseteq G}$ such that for some $c, e \in \mathbf{C}$ and $I \subseteq G$: (1) e is compact, (2) $c \sqsupseteq \mathbb{D}_I(e)$ but (3) there is no finite subset $J \subseteq I$ with $c \sqsupseteq \mathbb{D}_J(e)$.*

Proof. Consider the scs $(\mathbf{C}, \leq, (\mathfrak{s}_n)_{n\in\mathbb{N}})$ (Figure 2.3) defined by

$$\mathbf{C} = \left\{0, \frac{1}{2}\right\} \cup \left\{\frac{1}{2} + \frac{1}{2n} \;\middle|\; n \geq 1\right\} \quad \text{and} \quad \mathfrak{s}_n(x) = \begin{cases} \frac{1}{2} + \frac{1}{2n} & x \geq \frac{1}{2} \\ 0 & x < \frac{1}{2} \end{cases}$$

where $\mathfrak{s}_n$ is a self-map on $\mathbf{C}$ for every $n \geq 1$. For $n = 0$, $\mathfrak{s}_n(x) = 0$ for every $x \in \mathbf{C}$.

First we prove that for every $n \in \mathbb{N}$, $\mathfrak{s}_n$ is a space function. From Definition 9 we prove:

a. $\mathfrak{s}_n$ preserves 0 and binary joins.

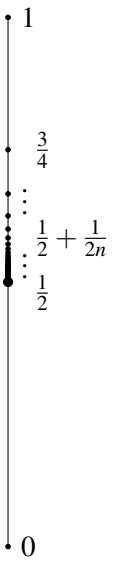


Figure 2.3: Constraint system $(\mathbf{C},\leq)$ with $\mathbf{C}=\{0,\frac{1}{2}\}\cup\{\frac{1}{2}+\frac{1}{2n}\mid n\geq 1\}$.

Recall that for every $n\in\mathbb{N}$, $\mathfrak{s}_n(0)=0$. If $x<\frac{1}{2}$ and $y<\frac{1}{2}$, then $\mathfrak{s}_n(x\sqcup y)=0=\mathfrak{s}_n(x)\sqcup\mathfrak{s}_n(y)$. If either $x\geq\frac{1}{2}$ or $y\geq\frac{1}{2}$ hold, then $\mathfrak{s}_n(x\sqcup y)=\frac{1}{2}+\frac{1}{2n}=\mathfrak{s}_n(x)\sqcup\mathfrak{s}_n(y)$.

b. $\mathfrak{s}_n$ is continuous.

For $n=0$, it is immediate that $\mathfrak{s}_n$ is continuous.

Since $\mathbf{C}$ is countable, from Proposition 1 it suffices to prove that $\mathfrak{s}_n$ preserves the join of increasing chains. For any $n\geq 1$, consider any increasing chain $x_1\leq x_2\leq\cdots$ in $(\mathbf{C},\leq)$. To prove $\bigsqcup_{i\geq 1}\mathfrak{s}_n(x_i)=\mathfrak{s}_n\left(\bigsqcup_{i\geq 1}x_i\right)$.

Let $J,K\subseteq\mathbb{N}\setminus\{0\}$ be two index sets such that: if $x_i<\frac{1}{2}$ then $i\in J$ and if $x_i\geq\frac{1}{2}$ then $i\in K$. Notice that $J\cap K=\emptyset$ and $J\cup K=\mathbb{N}\setminus\{0\}$. Then using this and part (a), we have $\mathfrak{s}_n\left(\bigsqcup_{i\geq 1}x_i\right)=\mathfrak{s}_n\left(\left(\bigsqcup_{j\in J}x_j\right)\sqcup\left(\bigsqcup_{k\in K}x_k\right)\right)=\mathfrak{s}_n\left(\bigsqcup_{j\in J}x_j\right)\sqcup\mathfrak{s}_n\left(\bigsqcup_{k\in K}x_k\right)$. Also, $\bigsqcup_{j\in J}x_j=0$ and $\bigsqcup_{k\in K}x_k\geq\frac{1}{2}$, therefore $\mathfrak{s}_n\left(\bigsqcup_{j\in J}x_j\right)\sqcup\mathfrak{s}_n\left(\bigsqcup_{k\in K}x_k\right)=0\sqcup(\frac{1}{2}+\frac{1}{2n})=\bigsqcup_{j\in J}0\sqcup\bigsqcup_{k\in K}(\frac{1}{2}+\frac{1}{2n})=\bigsqcup_{j\in J}\mathfrak{s}_n(x_j)\sqcup\bigsqcup_{k\in K}\mathfrak{s}_n(x_k)=\bigsqcup_{i\geq 1}\mathfrak{s}_n(x_i)$. Then $\mathfrak{s}_n\left(\bigsqcup_{i\geq 1}x_i\right)=\bigsqcup_{i\geq 1}\mathfrak{s}_n(x_i)$ as wanted.

To complete the proof we now show that given the above scs $(\mathbf{C},\leq,(\mathfrak{s}_n)_{n\in\mathbb{N}})$, for $I=\mathbb{N}\setminus\{0\}$ and $c=e=\frac{1}{2}$: (1) e is compact, (2) $c\geq\mathbb{D}_I(e)$ but (3) $c\not\geq\mathbb{D}_N(e)$ for any finite set $N\subseteq I$.

1. $\frac{1}{2}$ is compact. This follows directly from the definition of our constraint system $(\mathbf{C},\leq)$.

2. $\frac{1}{2}\geq\mathbb{D}_I\left(\frac{1}{2}\right)$. By definition of $\mathbb{D}_I$, $\mathbb{D}_I\left(\frac{1}{2}\right)\leq\mathfrak{s}_n\left(\frac{1}{2}\right)$ for every $n\in I$. Then $\mathbb{D}_I\left(\frac{1}{2}\right)\leq\bigsqcap_{n\geq 1}\mathfrak{s}_n\left(\frac{1}{2}\right)$. Since for every $n\geq 1$, $\mathfrak{s}_n\left(\frac{1}{2}\right)=\frac{1}{2}+\frac{1}{2n}\geq\frac{1}{2}$ then $\bigsqcap_{n\geq 1}\mathfrak{s}_n\left(\frac{1}{2}\right)=\frac{1}{2}$. Thus, $\frac{1}{2}\geq\mathbb{D}_I\left(\frac{1}{2}\right)$.

3. $\frac{1}{2} \not\geq \mathbb{D}_N\left(\frac{1}{2}\right)$ for any finite set $N \subseteq I$. Notice that, for any finite set $N \subseteq I$, $\mathbb{D}_N\left(\frac{1}{2}\right) = \mathfrak{s}_m\left(\frac{1}{2}\right) > \frac{1}{2}$, where $m = \max(N)$. Hence, $\frac{1}{2} \not\geq \mathbb{D}_N\left(\frac{1}{2}\right)$.

■

2.10. Distributed Spaces in Completely Distributive Lattices

In this section we present another characterization of distributed spaces for distributive lattices: For a group I, $\mathbb{D}_I(c)$ can be understood as the greatest information below all possible combinations of information in the spaces of the agents in I that derive c. We also provide compositionality properties capturing the intuition that just like *distributed information* of a group I is the collective information from all its members, it is also the collective information of its subgroups. We shall argue that the following results can be used to produce algorithms to efficiently compute $\mathbb{D}_I(c)$ for finite constraint systems.

We recall J-tuples, a general form of tuples that allows for an arbitrary index set J.

Definition 17 ([58])**.** Let J be an index set. Given a set X, a J-tuple of elements of X is a function $\mathbf{x} : J \longrightarrow X$. If $j \in J$, we denote $\mathbf{x}(j)$ by x_j and refer to it as the j-th coordinate of $\mathbf{x}$. The function $\mathbf{x}$ is denoted itself by $(x_j)_{j\in J}$. We use X^J to denote the set of all J-tuples.

The next theorem is one of main results of this dissertation. It establishes that for completely distributed lattices, $\mathbb{D}_K(c)$ is the greatest information below all possible combinations of information in the spaces of the agents in K that derive c.

Theorem 4. *Let $(\mathbb{D}_I)_{I\subseteq G}$ be the distributed spaces of a scs $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$. Suppose that $(\mathbf{C}, \sqsubseteq)$ is completely distributive. Let $\overline{\mathbb{D}}_K : \mathbf{C} \longrightarrow \mathbf{C}$, with $K \subseteq G$, be the function defined as follows:*

$$\overline{\mathbb{D}}_K(c) \stackrel{\text{def}}{=} \bigsqcap \left\{ \bigsqcup_{k\in K} \mathfrak{s}_k(a_k) \mid (a_k)_{k\in K} \in \mathbf{C}^K \text{ and } \bigsqcup_{k\in K} a_k \sqsupseteq c \right\}.$$

Then $\mathbb{D}_K = \overline{\mathbb{D}}_K$.

The above theorem is presented in [33] for finite cs. Here we generalize this for completely distributive lattices.

For the sake of the presentation, we give the proof of the above theorem in Section 2.11. Nevertheless, we would like to mention that the central and non-obvious property used in the proof is that of $\overline{\mathbb{D}}_K$ being a *continuous function*. The distributivity of $(\mathbf{C}, \sqsubseteq)$ is crucial for this. In fact without it the equality $\mathbb{D}_K = \overline{\mathbb{D}}_K$ does not necessarily hold as shown by the following counter-example.

Example 16. Consider the lattice $\mathbf{M}_3$, which is not (completely) distributive, and the space functions $\mathfrak{s}_1$ and $\mathfrak{s}_2$ in Figure 2.4. We obtain $\overline{\mathbb{D}}_I(b \sqcup c) = \overline{\mathbb{D}}_I(e) = a$ and $\overline{\mathbb{D}}_I(b) \sqcup \overline{\mathbb{D}}_I(c) = b \sqcup a = b$. Then, $\overline{\mathbb{D}}_I(b \sqcup c) \neq \overline{\mathbb{D}}_I(b) \sqcup \overline{\mathbb{D}}_I(c)$, i.e., $\overline{\mathbb{D}}_I$ is not a space function.

We can use Theorem 4 to prove the following properties characterizing the information of a group from that of its subgroups.

Theorem 5. *Let $(\mathbb{D}_I)_{I \subseteq G}$ be the distributed spaces of a scs $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$. Suppose that $(\mathbf{C}, \sqsubseteq)$ is completely distributive. Let $I, J, K \subseteq G$ be such that $I = J \cup K$. Then the following equalities hold:*

1. $\mathbb{D}_I(c) = \bigsqcap \{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(b) \mid a, b \in \mathbf{C} \text{ and } a \sqcup b \sqsupseteq c\}$.
2. $\mathbb{D}_I(c) = \bigsqcap \{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a) \mid a \in \mathbf{C}\}$.
3. $\mathbb{D}_I(c) = \bigsqcap \{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a) \mid a \in \mathbf{C} \text{ and } a \sqsubseteq c\}$.

We find it convenient to give the proof of Theorem 5 in Section 2.11. The properties in this theorem bear witness to the inherent compositional nature of our notion of distributed space. The first property in Theorem 5 essentially reformulates Theorem 4 in terms of subgroups rather than agents. It can be proven by replacing $\mathbb{D}_J(a)$ and $\mathbb{D}_K(b)$ by $\overline{\mathbb{D}}_J(a)$ and $\overline{\mathbb{D}}_K(b)$, defined in Theorem 4 and using distributivity of joins over meets. The second and third properties in Theorem 5 are pleasant simplifications of the first one using co-Heyting subtraction. These properties realize the intuition that by joining the information a and $c \ominus a$ of their subgroups, the group I can obtain c.

We now conclude this chapter with the proofs of Theorem 4 and Theorem 5.

2.11. Proofs of Section 2.10

This section is devoted to the proofs of the compositionality properties of distributed spaces given in Section 2.10. To simplify our notation, we define sets of J-tuples whose join derive a given constraint c.

Definition 18. Let $(\mathbf{C}, \sqsubseteq)$ be a cs and J some index set. For every $c \in \mathbf{C}$, let $T_c^J = \{(a_j)_{j \in J} \in \mathbf{C}^J \mid \bigsqcup_{j \in J} a_j \sqsupseteq c\}$. For simplicity, we use T_c instead of T_c^J when no confusion arises.

2.11.1. Proof of Theorem 4

The function $\overline{\mathbb{D}}_K$ is defined in Theorem 4. Here we find it convenient to use the following simplified version.

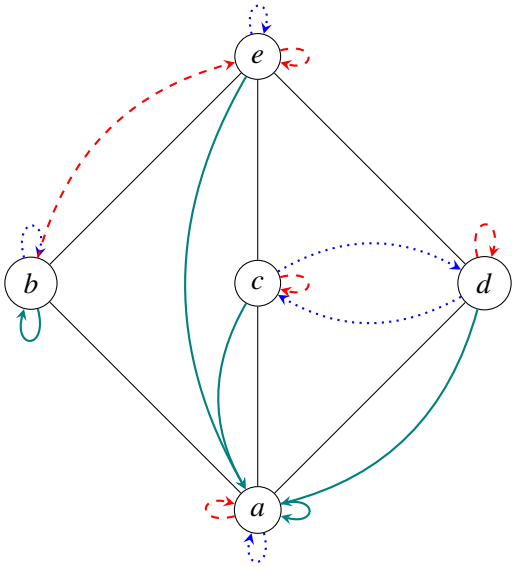


Figure 2.4: Group $I = \{1,2\}$, $\mathfrak{s}_1$ (dotted blue) and $\mathfrak{s}_2$ (dashed red) are space functions, $\overline{\mathbb{D}}_I$ (normal teal) in Theorem 4, is not a space function of the non-distributive lattice $\mathbf{M}_3$: $\overline{\mathbb{D}}_I(b) \sqcup \overline{\mathbb{D}}_I(c) = b \neq a = \overline{\mathbb{D}}_I(b \sqcup c)$.

Proposition 9. $\overline{\mathbb{D}}_K(c) = \bigsqcap \{\bigsqcup_{k\in K} \mathfrak{s}_k(a_k) \mid (a_k)_{k\in K} \in T_c\}$.

The following is an immediate consequence from the definition of $\overline{\mathbb{D}}_K$.

Proposition 10. *The function $\overline{\mathbb{D}}_K$ is monotonic.*

Proof. Let $c \sqsupseteq d$. We have $\{\bigsqcup_{k\in K} \mathfrak{s}_k(a_k) \mid (a_k)_{k\in K} \in T_c\} \subseteq \{\bigsqcup_{k\in K} \mathfrak{s}_k(a_k) \mid (a_k)_{k\in K} \in T_d\}$. Thus $\overline{\mathbb{D}}_K(c) \sqsupseteq \overline{\mathbb{D}}_K(d)$. ■

Next lemma states that $\overline{\mathbb{D}}_K$ is a space function. As pointed out in Section 2.10, the proof of continuity of $\overline{\mathbb{D}}_K$ uses the assumption of $(\mathbf{C}, \sqsubseteq)$ being completely distributive.

Lemma 2. *Let $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$ be a scs. Suppose that $(\mathbf{C}, \sqsubseteq)$ is completely distributive. Then, for any $K \subseteq G$, $\overline{\mathbb{D}}_K$ is a space function.*

Proof. Let $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$ be a scs and assume $(\mathbf{C}, \sqsubseteq)$ to be completely distributive. To show that $\overline{\mathbb{D}}_K$ is a space function we prove: (i) it satisfies S.1 and S.2 in Definition 9, and (ii) it is continuous.

(I) $\overline{\mathbb{D}}_K$ *satisfies* S.1 *and* S.2.

For S.1, one can verify that $\overline{\mathbb{D}}_K(\mathit{true}) = \mathit{true}$.

To prove S.2, it suffices to show that $\overline{\mathbb{D}}_K(c \sqcup d) \sqsubseteq \overline{\mathbb{D}}_K(c) \sqcup \overline{\mathbb{D}}_K(d)$. The other direction follows by monotonicity (Proposition 10). Consider the following derivation:

$$
\begin{aligned}
&\overline{\mathbb{D}}_K(c) \sqcup \overline{\mathbb{D}}_K(d) \\
&= \langle \text{Definition of } \overline{\mathbb{D}}_K \rangle \\
&\quad \overline{\mathbb{D}}_K(c) \sqcup \bigsqcap \left\{ \bigsqcup_{k\in K} \mathfrak{s}_k(b_k) \mid (b_k)_{k\in K} \in T_d \right\} \\
&= \langle \sqcup \text{ distributes over } \sqcap \rangle \\
&\quad \bigsqcap \left\{ \overline{\mathbb{D}}_K(c) \sqcup \bigsqcup_{k\in K} \mathfrak{s}_k(b_k) \mid (b_k)_{k\in K} \in T_d \right\} \\
&= \langle \text{Definition of } \overline{\mathbb{D}}_K \rangle \\
&\quad \bigsqcap \left\{ \bigsqcap \left\{ \bigsqcup_{k\in K} \mathfrak{s}_k(a_k) \mid (a_k)_{k\in K} \in T_c \right\} \sqcup \bigsqcup_{k\in K} \mathfrak{s}_k(b_k) \mid (b_k)_{k\in K} \in T_d \right\} \\
&= \langle \sqcup \text{ distributes over } \sqcap \rangle \\
&\quad \bigsqcap \left\{ \bigsqcap \left\{ \bigsqcup_{k\in K} \mathfrak{s}_k(a_k) \sqcup \bigsqcup_{k\in K} \mathfrak{s}_k(b_k) \mid (a_k)_{k\in K} \in T_c \right\} \mid (b_k)_{k\in K} \in T_d \right\} \\
&= \langle \text{Associativity of } \sqcap \rangle \\
&\quad \bigsqcap \left\{ \bigsqcup_{k\in K} \mathfrak{s}_k(a_k) \sqcup \bigsqcup_{k\in K} \mathfrak{s}_k(b_k) \mid (a_k)_{k\in K} \in T_c \text{ and } (b_k)_{k\in K} \in T_d \right\} \\
&= \langle \text{Associativity of } \sqcup \rangle \\
&\quad \bigsqcap \left\{ \bigsqcup_{k\in K} (\mathfrak{s}_k(a_k) \sqcup \mathfrak{s}_k(b_k)) \mid (a_k)_{k\in K} \in T_c \text{ and } (b_k)_{k\in K} \in T_d \right\} \\
&= \langle \mathfrak{s}_k \text{ is a space function} \rangle \\
&\quad \bigsqcap \left\{ \bigsqcup_{k\in K} \mathfrak{s}_k(a_k \sqcup b_k) \mid (a_k)_{k\in K} \in T_c \text{ and } (b_k)_{k\in K} \in T_d \right\} \\
&\sqsupseteq \langle c_k{=}a_k{\sqcup}b_k;\ (\bigsqcup_{k\in K} c_k = \bigsqcup_{k\in K}(a_k{\sqcup}b_k) \sqsupseteq c{\sqcup}d) \text{ implies } (c_k)_{k\in K}{\in}T_{c\sqcup d} \rangle \\
&\quad \bigsqcap \left\{ \bigsqcup_{k\in K} \mathfrak{s}_i(c_k) \mid (c_k)_{k\in K} \in T_{c\sqcup d} \right\} \\
&= \langle \text{Definition of } \overline{\mathbb{D}}_K \rangle \\
&\quad \overline{\mathbb{D}}_K(c \sqcup d)
\end{aligned}
$$

(II) $\overline{\mathbb{D}}_K$ *is continuous.*

Let D be any directed set on $\mathbf{C}$, we will prove $\overline{\mathbb{D}}_K(\bigsqcup D) = \bigsqcup\{\overline{\mathbb{D}}_K(d) \mid d \in D\}$. We proceed with $\overline{\mathbb{D}}_K(\bigsqcup D) \sqsubseteq \bigsqcup\left\{\overline{\mathbb{D}}_K(d) \mid d \in D\right\}$. The other direction follows by monotonicity.

By definition of $\overline{\mathbb{D}}_K$, $\bigsqcup\left\{\overline{\mathbb{D}}_K(d) \mid d \in D\right\} = \bigsqcup\{\bigsqcap\{\bigsqcup_{k\in K}\mathfrak{s}_k(a_k) \mid (a_k)_{k\in K} \in T_d\} \mid d \in D\}$. Since $(\mathbf{C}, \sqsubseteq)$ is completely distributive (see Definition 4), for the subset $\{\bigsqcup_{k\in K}\mathfrak{s}_k(a_k)\}_{d\in D,(a_k)_{k\in K}\in T_d}$ of $\mathbf{C}$, we have

$$\bigsqcup_{d\in D}\left\{\bigsqcap_{(a_k)_{k\in K}\in T_d}\left\{\bigsqcup_{k\in K}\mathfrak{s}_k(a_k)\right\}\right\} = \bigsqcap_{f\in F}\left\{\bigsqcup_{d\in D}\left\{\bigsqcup_{k\in K}\mathfrak{s}_k(f_k(d))\right\}\right\}$$

where F is the class of *choice functions* f choosing for each $d \in D$ some index $f(d) \in T_d$. Recall that $f_k(d)$ is the k-th element of K-tuple $f(d)$. We can rewrite the right-hand side of the above equality using $\sqcup$ properties and the fact that $\mathfrak{s}_k$ preserves arbitrary joins (see Proposition 4). Then we obtain

$$\bigsqcup_{d\in D}\left\{\bigsqcap_{(a_k)_{k\in K}\in T_d}\left\{\bigsqcup_{k\in K}\mathfrak{s}_k(a_k)\right\}\right\} = \bigsqcap_{f\in F}\left\{\bigsqcup_{k\in K}\mathfrak{s}_k\left(\bigsqcup_{d\in D}f_k(d)\right)\right\}.$$

We now show that for every $f \in F$, $(\bigsqcup_{d\in D} f_k(d))_{k\in K} \in T_{\sqcup D}$. Notice that for every $d \in D$, $\bigsqcup_{k\in K} f_k(d) \sqsupseteq d$. We have

$$\bigsqcup_{k\in K}\left(\bigsqcup_{d\in D}f_k(d)\right) = \bigsqcup_{d\in D}\left(\bigsqcup_{k\in K}f_k(d)\right) \sqsupseteq \bigsqcup_{d\in D} d = \bigsqcup D.$$

Therefore $(\bigsqcup_{d\in D} f_k(d))_{k\in K} \in T_{\sqcup D}$ (see Definition 18).

Then, for every $f \in F$, the element $\bigsqcup_{k\in K}\mathfrak{s}_k(\bigsqcup_{d\in D} f_k(d)) \in \{\bigsqcup_{k\in K}\mathfrak{s}_k(a_k) \mid (a_k)_{k\in K} \in T_{\sqcup D}\}$, this implies

$$\left\{\bigsqcup_{k\in K}\mathfrak{s}_k\left(\bigsqcup_{d\in D}f_k(d)\right) \;\middle|\; f \in F\right\} \subseteq \left\{\bigsqcup_{k\in K}\mathfrak{s}_k(a_k) \;\middle|\; (a_k)_{k\in K} \in T_{\sqcup D}\right\}.$$

Consequently,

$$\bigsqcap_{f\in F}\left\{\bigsqcup_{k\in K}\mathfrak{s}_k\left(\bigsqcup_{d\in D}f_k(d)\right)\right\} \sqsupseteq \bigsqcap\left\{\bigsqcup_{k\in K}\mathfrak{s}_k(a_k) \;\middle|\; (a_k)_{k\in K} \in T_{\sqcup D}\right\} = \overline{\mathbb{D}}_K\left(\bigsqcup D\right).$$

Thus, we conclude $\overline{\mathbb{D}}_K$ is continuous.

■

Finally we prove Theorem 4. Its statement now is simplified due to Proposition 9.

Theorem (4). Let $(\mathbb{D}_I)_{I\subseteq G}$ be the distributed spaces of a scs $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$. Suppose that $(\mathbf{C}, \sqsubseteq)$ is a completely distributive lattice. Then $\mathbb{D}_K = \overline{\mathbb{D}}_K$.

Proof. Let $(\mathbb{D}_I)_{I\subseteq G}$ be the distributed spaces of a scs $(\mathbf{C},\sqsubseteq,(\mathfrak{s}_i)_{i\in G})$ and assume $(\mathbf{C},\sqsubseteq)$ to be completely distributive.

We divide the proof in two parts: I. $\overline{\mathbb{D}}_K \sqsubseteq_{\mathscr{S}} \mathbb{D}_K$ and II. $\mathbb{D}_K \sqsubseteq_{\mathscr{S}} \overline{\mathbb{D}}_K$.

I. $\overline{\mathbb{D}}_K \sqsubseteq_{\mathscr{S}} \mathbb{D}_K$.

Recall that from Definition 14, $\mathbb{D}_K = \max\{h \in \mathscr{S}(\mathbf{C}) \mid h \sqsubseteq_{\mathscr{S}} \mathfrak{s}_k \text{ for all } k \in K\}$. Thus we have to prove: (i) $\overline{\mathbb{D}}_K \in \mathscr{S}(\mathbf{C})$ and (ii) $\overline{\mathbb{D}}_K \sqsubseteq_{\mathscr{S}} \mathfrak{s}_k$ for every $k \in K$.

From Lemma 2 we have (i). For part (ii), let $c \in \mathbf{C}$ and $S = \{\bigsqcup_{k\in K} \mathfrak{s}_k(a_k) \mid (a_k)_{k\in K} \in T_c\}$. From definition of $\overline{\mathbb{D}}_K$, for every $k \in K$, the element $\mathfrak{s}_k(c) = \mathfrak{s}_k(c) \sqcup \bigsqcup_{j\in K\setminus\{k\}} \mathfrak{s}_j(\mathit{true}) \in S$. Then for every $c \in \mathbf{C}$, $\overline{\mathbb{D}}_K(c) = \bigsqcap S \sqsubseteq \mathfrak{s}_k(c)$. Therefore for every $k \in K$, $\overline{\mathbb{D}}_K \sqsubseteq_{\mathscr{S}} \mathfrak{s}_k$. Then, $\overline{\mathbb{D}}_K \sqsubseteq_{\mathscr{S}} \mathbb{D}_K$ holds.

II. $\mathbb{D}_K \sqsubseteq_{\mathscr{S}} \overline{\mathbb{D}}_K$.

Let $c \in \mathbf{C}$ and $S = \{\bigsqcup_{k\in K} \mathfrak{s}_k(a_k) \mid (a_k)_{k\in K} \in T_c\}$. Notice that, for any $(a_k)_{k\in K} \in T_c$, $\bigsqcup_{k\in K} \mathbb{D}_K(a_k) \sqsubseteq \bigsqcup_{k\in K} \mathfrak{s}_k(a_k)$. Since $\mathbb{D}_K$ is a space function and monotonic, we know that $\bigsqcup_{k\in K} \mathbb{D}_K(a_k) = \mathbb{D}_K(\bigsqcup_{k\in K} a_k) \sqsupseteq \mathbb{D}_K(c)$. Thus, for every $(a_k)_{k\in K} \in T_c$, we have $\mathbb{D}_K(c) \sqsubseteq \bigsqcup_{k\in K} \mathfrak{s}_k(a_k)$, i.e., $\mathbb{D}_K(c)$ is a lower bound of S. Then for every $c \in \mathbf{C}$, $\mathbb{D}_K(c) \sqsubseteq \bigsqcap S = \overline{\mathbb{D}}_K(c)$. Therefore $\mathbb{D}_K \sqsubseteq_{\mathscr{S}} \overline{\mathbb{D}}_K$ as wanted.

Thus, from (I) and (II), we conclude Theorem 4, i.e., $\mathbb{D}_K = \overline{\mathbb{D}}_K$. ■

2.11.2. Proof of Theorem 5

We now prove the compositional properties of distributed spaces introduced in Theorem 5.

Theorem (5). Let $(\mathbb{D}_I)_{I\subseteq G}$ be the distributed spaces of a scs $(\mathbf{C},\sqsubseteq,(\mathfrak{s}_i)_{i\in G})$. Suppose that $(\mathbf{C},\sqsubseteq)$ is completely distributive and let $I,J,K \subseteq G$ be such that $I = J \cup K$. Then

1. $\mathbb{D}_I(c) = \bigsqcap\{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(b) \mid a,b \in \mathbf{C} \text{ and } a \sqcup b \sqsupseteq c\}$.
2. $\mathbb{D}_I(c) = \bigsqcap\{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a) \mid a \in \mathbf{C}\}$.
3. $\mathbb{D}_I(c) = \bigsqcap\{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a) \mid a \in \mathbf{C} \text{ and } a \sqsubseteq c\}$.

We present the proof of each item separately.

Proof of Theorem 5 (1). Let $(\mathbb{D}_I)_{I\subseteq G}$ be the distributed spaces of a scs $(\mathbf{C},\sqsubseteq,(\mathfrak{s}_i)_{i\in G})$ where $(\mathbf{C},\sqsubseteq)$ is completely distributive. Let $I,J,K \subseteq G$ be such that $I = J \cup K$.

- $\mathbb{D}_I(c) \sqsubseteq \bigsqcap\{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(b) \mid a,b \in \mathbf{C} \text{ and } a \sqcup b \sqsupseteq c\}$.

  Let $a,b,c \in \mathbf{C}$ such that $a \sqcup b \sqsupseteq c$. Since $(\mathbb{D}_I)_{I \subseteq G}$ is a gdc (see Theorem 1) and, both $J \subseteq I$ and $K \subseteq I$ hold, for every $e \in \mathbf{C}$, we have $\mathbb{D}_I(e) \sqsubseteq \mathbb{D}_J(e)$ and $\mathbb{D}_I(e) \sqsubseteq \mathbb{D}_K(e)$. Then $\mathbb{D}_I(a) \sqcup \mathbb{D}_I(b) \sqsubseteq \mathbb{D}_J(a) \sqcup \mathbb{D}_K(b)$. From the fact that $\mathbb{D}_I$ is a space function and hence monotonic we conclude $\mathbb{D}_I(c) \sqsubseteq \mathbb{D}_I(a) \sqcup \mathbb{D}_I(b)$. Therefore $\mathbb{D}_I(c) \sqsubseteq \mathbb{D}_J(a) \sqcup \mathbb{D}_K(b)$. Thus, $\mathbb{D}_I(c) \sqsubseteq \bigsqcap\{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(b) \mid a,b \in \mathbf{C} \text{ and } a \sqcup b \sqsupseteq c\}$.

- $\mathbb{D}_I(c) \sqsupseteq \bigsqcap\{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(b) \mid a,b \in \mathbf{C} \text{ and } a \sqcup b \sqsupseteq c\}$.

  Let $c \in \mathbf{C}$ and $S = \{\bigsqcup_{i \in I} \mathfrak{s}_i(c_i) \mid (c_i)_{i \in I} \in T_c^I\}$. We first show the following claim:

  Claim. For every $\bigsqcup_{i \in I} \mathfrak{s}_i(c_i) \in S$, there are some $a,b \in \mathbf{C}$ such that (i) $a \sqcup b \sqsupseteq c$ and (ii) $\bigsqcup_{i \in I} \mathfrak{s}_i(c_i) \sqsupseteq \mathbb{D}_J(a) \sqcup \mathbb{D}_K(b)$.

  Let $\bigsqcup_{i \in I} \mathfrak{s}_i(c_i) \in S$. Since $I = J \cup K$, we have $\bigsqcup_{i \in I} \mathfrak{s}_i(c_i) = \bigsqcup_{j \in J} \mathfrak{s}_j(c_j) \sqcup \bigsqcup_{k \in K} \mathfrak{s}_k(c_k)$. Let $a = \bigsqcup_{j \in J} c_j$ and $b = \bigsqcup_{k \in K} c_k$. From Theorem 4, we have

  $$\mathbb{D}_J(a) = \bigsqcap \left\{ \bigsqcup_{j \in J} \mathfrak{s}_j(a_j) \;\middle|\; (a_j)_{j \in J} \in T_a^J \right\} \text{ and } \mathbb{D}_K(b) = \bigsqcap \left\{ \bigsqcup_{k \in K} \mathfrak{s}_k(b_k) \;\middle|\; (b_k)_{k \in K} \in T_b^K \right\}.$$

  Given that $(\mathbf{C}, \sqsubseteq)$ is completely distributive and by associativity of $\sqcap$, $\mathbb{D}_J(a) \sqcup \mathbb{D}_K(b) = \bigsqcap R$ where $R = \left\{ \bigsqcup_{j \in J} \mathfrak{s}_j(a_j) \sqcup \bigsqcup_{k \in K} \mathfrak{s}_k(b_k) \;\middle|\; (a_j)_{j \in J} \in T_a^J \text{ and } (b_k)_{k \in K} \in T_b^K \right\}$. Clearly $a \sqcup b \sqsupseteq c$ and $\bigsqcup_{i \in I} \mathfrak{s}_i(c_i) \in R$. Then $\bigsqcup_{i \in I} \mathfrak{s}_i(c_i) \sqsupseteq \bigsqcap R = \mathbb{D}_J(a) \sqcup \mathbb{D}_K(b)$. This shows (i) and (ii).

  From the above claim and Theorem 4, we obtain $\mathbb{D}_I(c) = \bigsqcap S \sqsupseteq \bigsqcap\{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(b) \mid a,b \in \mathbf{C} \text{ and } a \sqcup b \sqsupseteq c\}$ as wanted.

■

Proof of Theorem 5 (2). Let $(\mathbb{D}_I)_{I \subseteq G}$ be the distributed spaces of a scs $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$ where $(\mathbf{C}, \sqsubseteq)$ is completely distributive. Let $I,J,K \subseteq G$ be such that $I = J \cup K$.

Let $a,c \in \mathbf{C}$. Recall that $c \ominus a$ represents the least element $e \in \mathbf{C}$ such that $a \sqcup e \sqsupseteq c$. Take any $b \in \mathbf{C}$ such that $a \sqcup b \sqsupseteq c$. Then $b \sqsupseteq c \ominus a$ and since space functions are monotonic $\mathbb{D}_J(a) \sqcup \mathbb{D}_K(b) \sqsupseteq \mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a)$. From this it follows that $\bigsqcap(S \cup \{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a)\ , \mathbb{D}_J(a) \sqcup \mathbb{D}_K(b)\}) = \bigsqcap(S \cup \{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a)\})$ for any $S \subseteq \mathbf{C}$.

From Theorem 5 (1) and the above argument, we have

$$\begin{aligned}\mathbb{D}_I(c) &= \bigsqcap \{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(b) \mid a \sqcup b \sqsupseteq c\} \\ &= \bigsqcap (\{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(b) \mid a \sqcup b \sqsupseteq c\} \cup \{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a) \mid a \in \mathbf{C}\}) \\ &= \bigsqcap \{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a) \mid a \in \mathbf{C}\}.\end{aligned}$$

Thus $\mathbb{D}_I(c) = \bigsqcap \{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a) \mid a \in \mathbf{C}\}$. ■

Proof of Theorem 5 (3). Let $(\mathbb{D}_I)_{I \subseteq G}$ be the distributed spaces of a scs $(\mathbf{C}, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$ where $(\mathbf{C}, \sqsubseteq)$ is completely distributive. Let $I, J, K \subseteq G$ be such that $I = J \cup K$.

Let $c \in \mathbf{C}$ and take any $a' \not\sqsupseteq c$. It suffices to find $a \in \mathbf{C}$ such that $a \sqsubseteq c$ and $\mathbb{D}_J(a') \sqcup \mathbb{D}_K(c \ominus a') \sqsupseteq \mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a)$ since then $\bigsqcap (S \cup \{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a)\ , \mathbb{D}_J(a') \sqcup \mathbb{D}_K(c \ominus a')\}) = \bigsqcap (S \cup \{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a)\})$ for any $S \subseteq \mathbf{C}$.

Given $a' \not\sqsupseteq c$ either (a) $a' \sqsupset c$ or (b) a' and c are incomparable w.r.t $\sqsubseteq$, written $a' \parallel c$.

- Suppose (a) holds. Then take $a = c$ thus $c \ominus a = true$. By monotonicity we have $\mathbb{D}_J(a') \sqcup \mathbb{D}_K(c \ominus a') \sqsupseteq \mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a)$ as wanted.

- Suppose (b) holds, i.e., $a' \parallel c$. Notice that $c \ominus a' \sqsubseteq c$. By cases, assume $c \ominus a' = c$. Then we can take $a = true$, and thus $c \ominus a = c = c \ominus a'$. By monotonicity we have $\mathbb{D}_J(a') \sqcup \mathbb{D}_K(c \ominus a') \sqsupseteq \mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a)$ as wanted. Now suppose $c \ominus a' \sqsubset c$ holds. We can build a poset $P = (\{a' \sqcup c,\ a',\ c,\ c \ominus a',\ a' \sqcap (c \ominus a')\ \}, \sqsubseteq)$ which is a non-distributive sub-lattice of $(\mathbf{C}, \sqsubseteq)$, isomorphic to a lattice known as $\mathbf{N}_5$ (see Figure 2.5). But this contradicts $(\mathbf{C}, \sqsubseteq)$ to be distributive (see [15]).

From Theorem 5 (2) and the above argument, we have

$$\begin{aligned}\mathbb{D}_I(c) &= \bigsqcap \{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a) \mid a \in \mathbf{C}\} \\ &= \bigsqcap (\{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a) \mid a \sqsubseteq c\} \cup \{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a) \mid a \not\sqsubseteq c\}) \\ &= \bigsqcap \{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a) \mid a \sqsubseteq c\}.\end{aligned}$$

Thus, $\mathbb{D}_I(c) = \bigsqcap \{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a) \mid a \sqsubseteq c\}$. ■

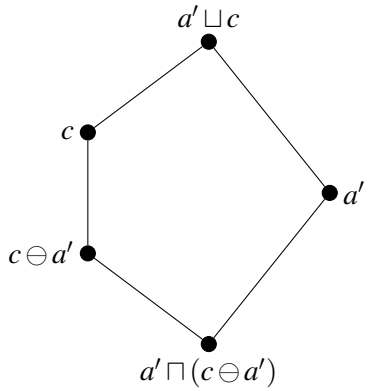


Figure 2.5: Poset $(\{a' \sqcup c, a', c, c \ominus a', a' \sqcap (c \ominus a')\}, \sqsubseteq)$ isomorphic to lattice $\mathbf{N}_5$.

2.12. Summary

In this chapter we presented the main technical results of this dissertation. We have formalized and developed the theory of the collective information of a group of agents I as the space function $\mathbb{D}_I$. Intuitively, the space function $\mathbb{D}_I$ represents the smallest space that includes all the local information of the agents in I.

We first constructed the complete lattice $(\mathscr{S}(\mathbf{C}), \sqsubseteq_{\mathscr{S}})$ (Lemma 1) where $\mathscr{S}(\mathbf{C})$ is the set of all space functions defined on the complete lattice $(\mathbf{C}, \sqsubseteq)$. We then defined $\mathbb{D}_I$ as the greatest space function in $\mathscr{S}(\mathbf{C})$ below the space functions of agents in I (Definition 14) and presented some of its basic compositional properties (Proposition 5). We showed that $\mathbb{D}_I$ could also be alternative defined as the greatest group distribution candidate (gdc) (Theorem 1). We illustrated in Section 2.5 that $\mathbb{D}_I$ can be interpreted as distributed knowledge in Aumann structures, a representative model for epistemic group reasoning. Furthermore, we showed that distributed knowledge in epistemic logic is also an instance of distributed information.

We also defined agent, join and group projections (Definitions 11 and 16). Group (agent) projection of a given $c \in \mathbf{C}$ represents the join of all the information that the group (agent) has in c. Join projections are the join of individual agent projections. We stated that group projections and distributed spaces form a Galois connection (Proposition 8). We then provided a *group compactness* result: Given an *infinite* group I, we identified *join-derivability* (Definition 11) as a condition under which $c \sqsupseteq \mathbb{D}_I(e)$ implies $c \sqsupseteq \mathbb{D}_J(e)$ for some finite group $J \subseteq I$ (Theorem 2). We then showed that without this condition we cannot guarantee the existence of such finite set $J \subseteq I$ (Theorem 3).

Finally we showed that if $\mathbf{C}$ is completely-distributive, $\mathbb{D}_I(c)$ can be characterized as the greatest information below all possible combinations of information in the spaces of the agents in I that derive c (Theorem 4) and, more succinctly, as the combination of the information of its subgroups that derive c (Theorem 5). In Section 2.11 we proved both theorems.

Chapter 3

Computing Distributed Space Functions

In the previous chapter we presented the theory of distributed space functions. We established that the distributed information of a group I of agents can be defined as the greatest space function, called $\mathbb{D}_I$, below all the space functions of the agents in I. Indeed, from Remark 6 we know that $\mathbb{D}_I$ can be computed as $\bigsqcap_{\mathscr{S}(\mathbf{C})} S$ where $S = \{\mathfrak{s}_i \mid i \in I\}$. Moreover, we have provided mechanisms to compute $\mathbb{D}_I$ in completely distributive lattices as the information distributed in I's subgroups. Such results suggest a divide-and-conquer fashion for computing distributed information: *The distributed information of a group can be computed as the combination of that of its subgroups*.

In this chapter we shall use those results to provide algorithms to compute the distributed information of a given group. In what follows, we assume a finite constraint system $(L, \sqsubseteq)$; under this finiteness assumption, space functions are just those functions that preserve the join of any finite set on L. In this context, space functions are commonly called *join-endomorphisms*.

Remark 8. Given a natural number $n \geq 1$, we use $\mathbf{n}$ to denote the poset $(\{1,\ldots,n\}, \sqsubseteq)$ with the linear order $x \sqsubseteq y$ iff $x \leq y$. The poset $\bar{\mathbf{n}}$ is the set $\{1,\ldots,n\}$ with the discrete order $x \sqsubseteq y$ iff $x = y$. Given a poset L, we use $L_\perp$ for the poset that results from adding a bottom element to L. The poset $L^\top$ is similarly defined. The lattice $\mathbf{2}^n$ is the n-fold Cartesian product of $\mathbf{2}$ ordered coordinate-wise. We define $\mathbf{M}_n$ as the lattice $(\bar{\mathbf{n}}_\perp)^\top$. A *lattice of sets* is a set of sets ordered by inclusion and closed under finite unions and intersections. A *powerset lattice* is a lattice of sets that includes all the subsets of its top element.

First, given a cs $(L, \sqsubseteq)$ where L is finite, we compute the cardinality of the set of all space functions $\mathscr{S}(L)$ defined on L.

3.1. The Size of the Function Space

The main result of this section is Theorem 6 that states the size of $\mathscr{S}(\mathbf{M}_n)$. Propositions 11 and 12 present the size of $\mathscr{S}(L)$ for the cases when L is a powerset lattice and a total order, respectively. These propositions are well-known in the lattice theory community [8, 48, 74]. However, we include our original proofs of them.

For the sake of the presentation, in this section we simplify the notation of space functions by omitting the index that represents the agent.

3.1.1. Distributive Lattices

We begin with lattices isomorphic to $\mathbf{2}^n$. They include *finite boolean algebras* and *powerset* lattices [15]. The size of these lattices are easy to infer from the observation that the join-preserving functions on them are determined by their action on the lattices' atoms.

Proposition 11. *Suppose that $m \geq 0$. Let L be any lattice isomorphic to the product lattice $\mathbf{2}^m$. Then $|\mathscr{S}(L)| = n^{\log_2 n}$ where $n = 2^m$ is the size of L.*

Proof. Take any set S of size m. Consider the powerset lattice $L = \mathscr{P}(S)$ ordered by inclusion. We have $n = |\mathscr{P}(S)| = 2^m$. We shall show that $|\mathscr{S}(L)| = n^{\log_2 n}$. Since $\mathscr{P}(S)$ is isomorphic to $\mathbf{2}^m$, the result follows from the fact that isomorphic lattices have the same number of space functions.

Let $\mathscr{F}$ be the family of functions $f : \mathscr{P}(S) \to \mathscr{P}(S)$ that satisfy (a) $f(T) = \bigcup_{t \in T} f(\{t\})$ if $|T| > 1$ and (b) $f(\emptyset) = \emptyset$. The equality $|\mathscr{S}(L)| = n^{\log_2 n}$ follows from the following claim: (1) $\mathscr{F} = \mathscr{S}(L)$ and (2) $|\mathscr{F}| = n^{\log_2 n}$.

To prove (1) one can verify that if $f \in \mathscr{F}$ then f is a space function where $\sqcup$ is $\cup$ and $\bot$ is the $\emptyset$. Hence $f \in \mathscr{S}(L)$. On the other hand, if $f \notin \mathscr{F}$ then either $f(T) \neq \bigcup_{t \in T} f(\{t\})$ for some $T \subseteq S$ or $f(\emptyset) \neq \emptyset$. But since $\sqcup = \cup$ and $\bot = \emptyset$, we have $T = \bigsqcup_{t \in T} \{t\}$ but $f(T) \neq \bigsqcup_{t \in T} f(\{t\})$ or $f(\bot) \neq \bot$. Hence $f \notin \mathscr{S}(L)$.

To prove (2) notice that given $f \in \mathscr{F}$, for each $T \subseteq S$ if $|T| > 1$ then the value $f(T)$ is determined by the values of f applied to each singleton $\{t\} \subseteq S$, and if $|T| = 0$ the value $f(T)$ is fixed to $\emptyset$. The set $\mathscr{P}(S)$ has $\log_2 n = m$ singletons. Since there is no restriction on how each $f \in \mathscr{F}$ should map singletons, $|\mathscr{F}| = n^{\log_2 n}$ as wanted. ■

Thus, powerset lattices and boolean algebras have a super-polynomial, sub-exponential number of space functions. Nevertheless, linear order lattices allow for an exponential number of space functions given by the *central binomial coefficient*. The following proposition can be proved alternatively from the observation

that the space functions over a linear order are also monotonic functions. In fact, this result appears in [8] and has been studied in [48, 74].

Proposition 12. *Suppose that $n \geq 0$. Let L be any lattice isomorphic to the linear order lattice $\mathbf{n}_\perp$. Then $|\mathscr{S}(L)| = \binom{2n}{n}$.*

Proof. Let $\mathscr{M}_\perp(L)$ be the set of monotonic functions from L to L that preserve $\perp$. We claim that $\mathscr{M}_\perp(L) = \mathscr{S}(L)$. The inclusion $\mathscr{S}(L) \subseteq \mathscr{M}_\perp(L)$ follows from Proposition 4 and the fact that space functions preserve bottoms. For $\mathscr{M}_\perp(L) \subseteq \mathscr{S}(L)$, take $f \in \mathscr{M}_\perp(L)$. By definition $f(\perp) = \perp$. Take any $a, b \in L$. So either $a \sqsubseteq b$ or $b \sqsubseteq a$. If $a \sqsubseteq b$ then $f(a \sqcup b) = f(b)$ and by monotonicity of f, $f(b) = f(a) \sqcup f(b)$. Similarly if $b \sqsubseteq a$ then $f(a \sqcup b) = f(a) = f(a) \sqcup f(b)$. We conclude that $f \in \mathscr{S}(L)$.

Now, for every $f \in \mathscr{M}_\perp(L)$ we have $f(\perp) = \perp$, then $|\mathscr{S}(L)| = |\mathscr{M}_\perp(L)| = |\mathscr{M}(L \setminus \{\perp\} \to L)|$ where $\mathscr{M}(L \setminus \{\perp\} \to L)$ is the set of monotonic functions from $L \setminus \{\perp\}$ to L. Thus, to complete the proof it suffices to show $|\mathscr{M}(L \setminus \{\perp\} \to L)| = \binom{2n}{n}$.

Notice that $|L| = n+1$. Consider the equation $\sum_{i=1}^{n+1} X_i = n$ where the variable X_i takes a value between 0 and n. Let $Sol(n)$ be the set of all solutions to this equation. We can show that $|Sol(n)| = |\mathscr{M}(L \setminus \{\perp\} \to L)|$ by providing the following bijection $\sigma : \mathscr{M}(L \setminus \{\perp\} \to L) \to Sol(n)$. The function σ associates each $f \in \mathscr{M}(L \setminus \{\perp\} \to L)$ with a solution $\sigma(f)$ assigning to every X_i the number of consecutive values from $L \setminus \{\perp\}$ mapped by f to the i-th value of L.

From combinatorics we know that for any pair of positive integers n and k, the number of k-tuples of non-negative integers whose sum is n equals $\binom{n+k-1}{n}$ [24]. For $k = n+1$ these tuples correspond exactly to the solutions in $Sol(n)$. Therefore we have shown $|\mathscr{S}(L)| = |\mathscr{M}_\perp(L)| = |\mathscr{M}(L \setminus \{\perp\} \to L)| = |Sol(n)| = \binom{2n}{n}$ as wanted. ■

By using properties of the binomial coefficient, we can infer that $\frac{4^n}{2\sqrt{n}} \leq \binom{2n}{n} \leq 4^n$ for $n \geq 1$. Together with Proposition 12, this gives us explicit exponential lower and upper bounds for $|\mathscr{S}(L)|$ when L is a linear lattice.

3.1.2. Non-distributive Case

The number of space functions for some non-distributive lattices of a given size can be much bigger than that for those distributive lattices of the same size in the previous section. We will characterize this number for an archetypal non-distributive lattice ($\mathbf{M}_n$) in terms of Laguerre (and rook) polynomials.

Laguerre polynomials are solutions to Laguerre's second-order linear differential equation $xy'' + (1-x)y' + ny = 0$ where y' and y'' are the first and second derivatives of an unknown function y of the variable x,

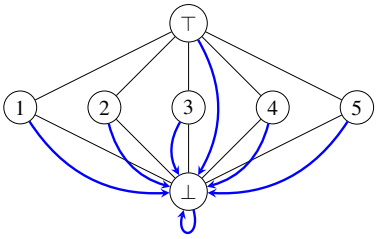
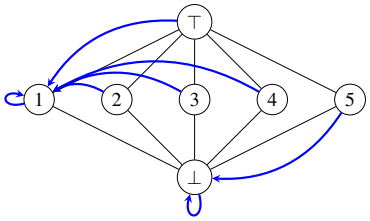
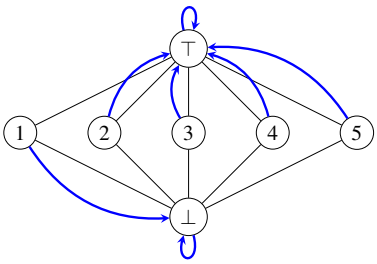
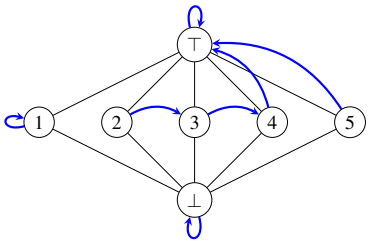
 Let $\mathscr{F}_1$ be the family of functions f that for all $e \in \mathbf{M}_n, f(e) = \bot$.	 Let $\mathscr{F}_2$ be the family of bottom preserving functions f such that for some $e, e' \in I$: (a) $f(\top) = e$, (b) $f(e') = \bot$ or $f(e') = e$, and (c) $f(e'') = e$ for all $e'' \in I \setminus \{e'\}$.
 Let $\mathscr{F}_3$ be the family of top and bottom preserving functions f such that for some $e \in I$: (a) $f(e) = \bot$, and (b) $f(e') = \top$ for all $e' \in I \setminus \{e\}$.	 Let $\mathscr{F}_4$ be the family of top and bottom preserving functions f that for some $J \subseteq I$: (a) $f(e) = \top$ for every $e \in J$, (b) $f\restriction_{I \setminus J}$ is injective, and (c) $\mathrm{Img}(f\restriction_{I \setminus J}) \subseteq I$.

Table 3.1: Families $\mathscr{F}_1, \ldots, \mathscr{F}_4$ of space functions of $\mathbf{M}_n$. $I = \{1, \ldots, n\}$. $f\restriction_A$ is the restriction of f to a subset A of its domain. $\mathrm{Img}(f)$ is the image of f. A function from each $\mathscr{F}_i$ for $\mathbf{M}_5$ is depicted with blue arrows.

and n is a non-negative integer. The Laguerre polynomial of degree n in x, $\mathscr{L}_n(x)$ is given by the summation $\sum_{k=0}^{n} \binom{n}{k} \frac{(-1)^k}{k!} x^k$.

The lattice $\mathbf{M}_n$ is non-distributive for any $n \geq 3$. The size of $\mathscr{S}(\mathbf{M}_n)$ can be succinctly expressed as follows.

Theorem 6. $|\mathscr{S}(\mathbf{M}_n)| = (n+1)^2 + n!\mathscr{L}_n(-1)$.

Proof. Let $\mathscr{F} = \bigcup_{i=1}^{4} \mathscr{F}_i$ where the mutually exclusive $\mathscr{F}_i$'s are defined in Table 3.1, and $I = \{1, \ldots, n\}$. The proof is divided in two parts: (I) $\mathscr{F} = \mathscr{S}(\mathbf{M}_n)$ and (II) $|\mathscr{F}| = (n+1)^2 + n!\mathscr{L}_n(-1)$.

Part (I) For $\mathscr{F} \subseteq \mathscr{S}(\mathbf{M}_n)$, it is easy to verify that each $f \in \mathscr{F}$ is a space function by following the cases described in Table 3.1.

For $\mathscr{S}(\mathbf{M}_n) \subseteq \mathscr{F}$ we show that for any function f from $\mathbf{M}_n$ to $\mathbf{M}_n$ if $f \notin \mathscr{F}$, then $f \notin \mathscr{S}(\mathbf{M}_n)$. Immediately, if $f(\bot) \neq \bot$ then $f \notin \mathscr{S}(\mathbf{M}_n)$.

Suppose $f(\bot) = \bot$. Let J, K, H be disjoint possibly empty sets such that $I = J \cup K \cup H$ and let $j = |J|$, $k = |K|$ and $h = |H|$. The sets J, K, H represent the elements of I mapped by f to $\top$, to elements of I, and to $\bot$, respectively. More precisely, $\mathrm{Img}(f\restriction_J) = \{\top\}$, $\mathrm{Img}(f\restriction_K) \subseteq I$ and $\mathrm{Img}(f\restriction_H) = \{\bot\}$. Furthermore, for every f there are three possibilities to map $\top$, namely, (1) $f(\top) = \bot$, (2) $f(\top) \in I$ or (3) $f(\top) = \top$. For each case, we show that $f \notin \mathscr{S}(\mathbf{M}_n)$.

1. *Case* $f(\top) = \bot$.

   Since $f \notin \mathscr{F}_1$ there is an $e \in I$ such that $f(e) \neq \bot$. We have $e \sqsubseteq \top$ but $f(e) \not\sqsubseteq f(\top)$. Then f is not monotonic. From Proposition 4 we conclude $f \notin \mathscr{S}(\mathbf{M}_n)$.

2. *Case* $f(\top) \in I$.

   Let K_1, K_2 be disjoint possibly empty sets such that $K_1 \cup K_2 = K$, $\mathrm{Img}(f\restriction_{K_1}) = \{f(\top)\}$ and $\mathrm{Img}(f\restriction_{K_2}) \neq \{f(\top)\}$. Notice that if $j > 0$ or $|K_2| > 0$, f is non-monotonic and then $f \notin \mathscr{S}(\mathbf{M}_n)$.

   Now, for $j = 0$ and $K_2 = \emptyset$. Since $\mathrm{Img}(f\restriction_K) = \{f(\top)\}$ and $f \notin \mathscr{F}_2$ then $h > 1$. Therefore there must be $e_1, e_2 \in H$ such that $f(e_1) = f(e_2) = \bot$. This implies $f(e_1 \sqcup e_2) = f(\top) \neq \bot = f(e_1) \sqcup f(e_2)$, therefore $f \notin \mathscr{S}(\mathbf{M}_n)$.

3. *Case* $f(\top) = \top$.

   a) Suppose $k = 0$. Notice that $f \notin \mathscr{F}_3$ and $f \notin \mathscr{F}_4$ hence $h \neq 1$ and $h \neq 0$. Thus $h > 1$ implies that there are at least two $e_1, e_2 \in H$ such that $f(e_1) = f(e_2) = \bot$. But then $f(e_1 \sqcup e_2) = f(\top) = \top \neq \bot = f(e_1) \sqcup f(e_2)$, hence $f \notin \mathscr{S}(\mathbf{M}_n)$.

   b) Suppose $k > 0$. Assume $h = 0$. Notice that $K = I \setminus J$ and $\mathrm{Img}(f\restriction_K) \subseteq I$. Since f is a $\bot$ and $\top$ preserving function and it satisfies conditions (a) and (c) of $\mathscr{F}_4$ but $f \notin \mathscr{F}_4$, then f must violate condition (b). Thus $f\restriction_K$ is not injective. Then there are $a, b \in K$ such that $a \neq b$ but $f(a) = f(b)$. Then $f(a) \sqcup f(b) \neq \top = f(a \sqcup b)$. Consequently, $f \notin \mathscr{S}(\mathbf{M}_n)$.

   Assume $h > 0$. There must be $e_1, e_2, e_3 \in I$ such that $f(e_1) = \bot$ and $f(e_2) = e_3$. Notice that $f(e_1) \sqcup f(e_2) = e_3 \neq \top = f(\top) = f(e_1 \sqcup e_2)$. Therefore, $f \notin \mathscr{S}(\mathbf{M}_n)$.

Part (II) We prove that $|\mathscr{F}| = \sum_{i=1}^{4} |\mathscr{F}_i| = (n+1)^2 + n!\mathscr{L}_n(-1)$. Recall that $n = |I|$. It is easy to prove that $|\mathscr{F}_1| = 1$, $|\mathscr{F}_2| = n^2 + n$ and $|\mathscr{F}_3| = n$.

1. $|\mathscr{F}_1| = 1$.

   There is only one function mapping every element in $\mathbf{M}_n$ to $\bot$.

2. $|\mathscr{F}_2| = n^2 + n$.

   Since $\top$ is mapped to an element of I, there are n possibilities to choose such element. If there is an element of I mapped to $\bot$, for each one of the previous n options there are also n possibilities to choose an element of I to be mapped to $\bot$. Then, in this case there are n^2 functions. If no element of I is mapped to $\bot$, then there are n additional functions.

3. $|\mathscr{F}_3| = n$.

   One of the elements of I is mapped to $\bot$. All the other elements of I are mapped to $\top$. Then, there are n functions that can be defined in $\mathscr{F}_3$.

4. $|\mathscr{F}_4| = n!\mathscr{L}_n(-1)$.

   Let $f \in \mathscr{F}_4$ and let $J \subseteq I$ be a possibly empty set such that $\mathrm{Img}(f\restriction_J) = \{\top\}$ and $\mathrm{Img}(f\restriction_{I\setminus J}) \subseteq I$, where $f\restriction_{I\setminus J}$ is an injective function. We shall call $j = |J|$.

   For each of the $\binom{n}{j}$ possibilities for J, the elements of $I \setminus J$ are to be mapped to I by the injective function $f\restriction_{I\setminus J}$. The number of functions $f\restriction_{I\setminus J}$ is $\frac{n!}{j!}$. Therefore, $|\mathscr{F}_4| = \sum_{j=0}^{n} \binom{n}{j} \frac{n!}{j!}$. This sum equals $n!\mathscr{L}_n(-1)$ which in turn is equal to $\mathscr{R}_n(1)$.

It follows that $|\mathscr{F}| = \sum_{i=1}^{4} |\mathscr{F}_i| = (n+1)^2 + n!\mathscr{L}_n(-1)$, as wanted. ■

In combinatorics rook polynomials are generating functions of the number of ways to place non-attacking rooks on a board. A *rook polynomial* (for square boards) $\mathscr{R}_n(x)$ has the form $\sum_{k=0}^{n} x^k r(k,n)$ where the (rook) coefficient $r(k,n)$ represents the number of ways to place k non-attacking rooks on an $n \times n$ chessboard. For instance, $r(0,n) = 1$, $r(1,n) = n^2$ and $r(n,n) = n!$. In general $r(k,n) = \binom{n}{k}^2 k!$.
Rook polynomials are related to Laguerre polynomials by the equation $\mathscr{R}_n(x) = n!x^n\mathscr{L}_n(-x^{-1})$. Therefore, as a direct consequence of the above theorem, we can also characterize $|\mathscr{S}(\mathbf{M}_n)|$ in combinatorial terms as the following sum of rook coefficients.

Corollary 1. *Given the rook coefficient $r(k,n)$, let $r'(n+1,n) = r(1,n+1)$ and $r'(k,n) = r(k,n)$ if $k \leq n$. Then $|\mathscr{S}(\mathbf{M}_n)| = \sum_{k=0}^{n+1} r'(k,n)$.*

Proof. From Theorem 6, we have $|\mathscr{S}(\mathbf{M}_n)| = (n+1)^2 + n!\mathscr{L}_n(-1)$. By definition of rook polynomials, $n!\mathscr{L}_n(-1) = \mathscr{R}_n(1) = \sum_{k=0}^{n} r(k,n)$. Then, $|\mathscr{S}(\mathbf{M}_n)| = (n+1)^2 + \sum_{k=0}^{n} r(k,n)$. Also, notice that by definition

of $r(k,n)$ and r', $(n+1)^2 = \binom{n+1}{1}^2 1! = r(1,n+1) = r'(n+1,n)$. Then we can rewrite $(n+1)^2 + \sum_{k=0}^{n} r(k,n)$ as $r'(n+1,n) + \sum_{k=0}^{n} r'(k,n)$. Thus $|\mathscr{S}(\mathbf{M}_n)| = \sum_{k=0}^{n+1} r'(k,n)$. ■

We conclude this section with another pleasant correspondence between the space functions in $\mathscr{S}(\mathbf{M}_n)$ and $\mathscr{R}_n(x)$. Let $f : L \longrightarrow L$ be a function over a lattice $(L, \sqsubseteq)$. We say that f is *non-reducing* in L iff it does not map any value to a smaller one; i.e., there is no $e \in L$ such that $f(e) \sqsubset e$. The number of space functions that are non-reducing in $\mathbf{M}_n$ is exactly the value of the rook polynomial $\mathscr{R}_n(x)$ for $x = 1$.

Corollary 2. $\mathscr{R}_n(1) = |\{\ f \in \mathscr{S}(\mathbf{M}_n) \mid f \textit{ is non-reducing in } \mathbf{M}_n\ \}|$.

Proof. Let $A = \{\ f \in \mathscr{S}(\mathbf{M}_n) \mid f \text{ is non-reducing in } \mathbf{M}_n\ \}$. We will prove that $|A| = \mathscr{R}_n(1)$. Let $\mathscr{F} = \bigcup_{i=1}^{4} \mathscr{F}_i$ where the mutually exclusive $\mathscr{F}_i$'s are defined in Table 3.1. In the proof of Theorem 6 we show that $\mathscr{S}(\mathbf{M}_n) = \mathscr{F}$ and that $\mathscr{R}_n(1) = |\mathscr{F}_4|$. Notice that every function in $\mathscr{F}_4$ is non-reducing and every function in $\mathscr{F} \setminus \mathscr{F}_4$ is not non-reducing. Hence $A = \mathscr{F}_4$, thus $|A| = \mathscr{R}_n(1)$. ■

3.2. Algorithms

In this section we shall provide efficient algorithms to compute $\mathbb{D}_I$. Since $\mathbb{D}_I = \bigsqcap_{\mathscr{S}(L)}\{\mathfrak{s}_i \mid i \in I\}$ (Remark 6), computing distributed spaces can be seen as the following maximization problem:

> *Given a finite scs $(L, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$ and $S = \{\mathfrak{s}_i \mid i \in I\} \subseteq \mathscr{S}(L)$ with $I \subseteq G$, find $\bigsqcap_{\mathscr{S}(L)} S$, i.e., the greatest space function in the lattice $\mathscr{S}(L)$ below every $\mathfrak{s}_i \in S$.*

Finding $\mathbb{D}_I$ may not be immediate. For instance, see $\mathbb{D}_I$ in Figure 2.2 for a small lattice of four elements and two space functions. A *naive approach* is to compute $\mathbb{D}_I$ by taking $\sigma_I(c) \stackrel{\text{def}}{=} \bigsqcap_L \{\mathfrak{s}_i(c) \mid i \in I\}$ for each $c \in L$. However, this does not work since σ_I is not necessarily a space function as shown in Figure 3.1.

A *brute force* solution to compute $\mathbb{D}_I$ can be obtained by generating the set $S' = \{g \mid g \in \mathscr{S}(L) \text{ and } g \sqsubseteq_{\mathscr{S}} \mathfrak{s}_i \text{ for all } i \in I\}$ and taking its join. This approach works since $\bigsqcup_{\mathscr{S}(L)} S' = \mathbb{D}_I$ but as shown in Section 3.1, the size of $\mathscr{S}(L)$ can be super-polynomial for distributive lattices and exponential in general.

Nevertheless, one can use lattice properties to compute $\mathbb{D}_I$ efficiently. For distributive lattices, we use the inherent compositional nature of $\mathbb{D}_I$ and, for arbitrary lattices, we present an algorithm that uses the function σ_I in the naive approach to computing $\mathbb{D}_I$ by approximating it from above.

We will give the time complexities in terms of the number of basic binary lattice operations (i.e., meets, joins and subtractions) performed during execution.

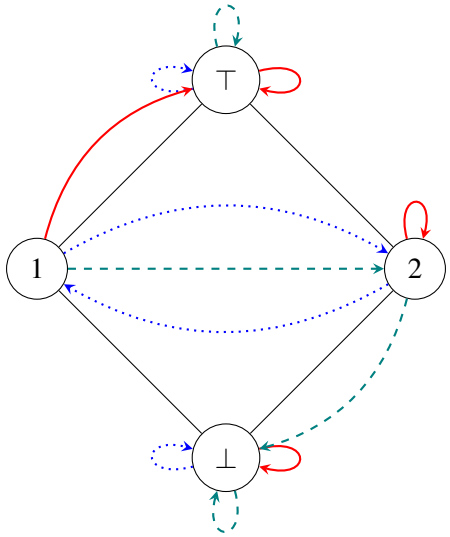


Figure 3.1: Space functions $\mathfrak{s}_1$ (dotted blue) and $\mathfrak{s}_2$ (normal red), defined on lattice $\mathbf{M}_2$. For $I = \{1,2\}$, the function $\sigma_I(c) \stackrel{\text{def}}{=} \bigsqcap_L \{\mathfrak{s}_i(c) \mid i \in I\}$ (dashed teal) is not a space function: $\sigma_I(1 \sqcup 2) \neq \sigma_I(1) \sqcup \sigma_I(2)$.

3.2.1. Meet of Space Functions in Distributive Lattices

Here we present some results derived from Theorem 4 and Theorem 5 in Section 2.10. We shall use such results for computing the space function $\mathbb{D}_I$ in a finite distributive lattice L. In what follows we assume $n = |L|$ and $m = |I|$.

Corollary 3. *Let $(L, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$ be a finite distributive scs and $I \subseteq G$. Then $\mathbb{D}_I = \overline{\mathbb{D}}_I$, where $\overline{\mathbb{D}}_I(c) = \bigsqcap_L \{\bigsqcup_{i\in I} \mathfrak{s}_i(a_i) \mid (a_i)_{i\in I} \in L^I$ and $\bigsqcup_{i\in I} a_i \sqsupseteq c\}$.*

The above corollary is an immediate result from Theorem 4. As we mention before, it basically says that $\mathbb{D}_I(c)$ is the greatest element in L below all possible applications of the functions $\mathfrak{s}_i$, with $i \in I$, to elements whose join is greater or equal to c.

Naive Algorithm A_1**.** One could use Corollary 3 directly in the obvious way to provide an algorithm for $\mathbb{D}_I$ by computing $\overline{\mathbb{D}}_I$: i.e., computing the meet of elements of the form $\bigsqcup_{i\in I} \mathfrak{s}_i(a_i)$ for every tuple $(a_i)_{i\in I} \in L^I$ such that $\bigsqcup_{i\in I} a_i \sqsupseteq c$. For each $c \in L$, $\overline{\mathbb{D}}_I(c)$ checks n^m tuples $(a_i)_{i\in I} \in L^I$, each one with a cost in $O(m)$. Thus A_1 can compute $\mathbb{D}_I$ by performing $O(n \times n^m \times m) = O(mn^{m+1})$ binary lattice operations.

Nevertheless, we can use Corollary 3 to provide a recursive characterization of $\mathbb{D}_I$ that can be used in a divide-and-conquer algorithm with lower time complexity. The next corollary follows from Theorem 5.

Corollary 4. *Let $(L, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$ be a finite distributive scs and $I = J \cup K \subseteq G$. Then $\mathbb{D}_I(c) = \bigsqcap_L \{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(b) \mid a, b \in L$ and $a \sqcup b \sqsupseteq c\}$.*

The above proposition bears witness to the compositional nature of $\mathbb{D}_I$.

Naive Algorithm A_2**.** We can use Corollary 4 to compute $\mathbb{D}_I$ with the following recursive procedure: Take any partition $\{J,K\}$ of I such that the absolute value of $|J|-|K|$ is at most 1. Then compute the meet of all $\mathbb{D}_J(a) \sqcup \mathbb{D}_K(b)$ for every $a,b \in L$ such that $a \sqcup b \sqsupseteq c$. Then given $c \in L$, the time complexity of a *naive* implementation of the above procedure can be obtained as the solution of the equation $T(m) = n^2(1+2T(m/2))$ and $T(1) = 1$ which is in $O(mn^{2\log_2 m})$. Therefore, $\mathbb{D}_I$ can be computed in $O(mn^{1+2\log_2 m})$.

The time complexity of the naive algorithm A_2 is better than that of A_1. However, by using a simple memoization technique to avoid repeating recursive calls and the following observations one can compute $\mathbb{D}_I$ in a much lower time complexity order.

3.2.2. Using Subtraction and Down-sets to characterize $\mathbb{D}_I$

In what follows we show that $\mathbb{D}_I$ can be computed in $O(mn^2)$ for distributive lattices and, in particular, in $O(n+m\log n)$ for powerset lattices —given in terms of the number of basic binary lattice operations (i.e., meets, joins and subtractions) performed during execution. To achieve this we use the subtraction operator and the notion of down set.

Subtraction Operator. Notice that in Corollary 4 we are considering *all* pairs $a,b \in L$ such that $a \sqcup b \sqsupseteq c$. However, because of the monotonicity of space functions, it suffices to take, for each $a \in L$, just *the least* b such that $a \sqcup b \sqsupseteq c$. In finite distributive lattices, and more generally in co-Heyting algebras [26], the *subtraction* operator $c \ominus a$ in Definition 8 gives us exactly such a least element.

Down-sets. Besides using just $c \ominus a$ instead of all b's such that $a \sqcup b \sqsupseteq c$, we can use a further simplification: Rather than including every $a \in L$, we only need to consider every a in the *down-set* of c. Recall that the down-set of c is defined as $\downarrow c = \{e \in L \mid e \sqsubseteq c\}$. This additional simplification is justified using properties of distributive lattices to show that for any $a' \in L$, such that $a' \not\sqsubseteq c$, there exists $a \sqsubseteq c$ such that $\mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a) \sqsubseteq \mathbb{D}_J(a') \sqcup \mathbb{D}_K(c \ominus a')$.

The above observations lead us to the next corollary that follows from Theorem 5.

Corollary 5. *Let* $(L, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$ *be a finite distributive scs and* $I = J \cup K \subseteq G$*. Then* $\mathbb{D}_I(c) = \bigsqcap_L \{\mathbb{D}_J(a) \sqcup \mathbb{D}_K(c \ominus a) \mid a \in \downarrow c\}$.

By using the above result we can derive a recursive algorithm that, given a finite distributive lattice L and $I \subseteq G$, computes $\mathbb{D}_I$ in worst-case time complexity $O(mn^2)$ where $m = |I|$ and $n = |L|$. We show this

algorithm next.

3.2.3. Algorithms for Distributive Lattices

We first describe the algorithm DMEETAPP that computes the value $\mathbb{D}_I(c)$. We then describe the algorithm DMEET that computes the function $\mathbb{D}_I$ by calling DMEETAPP in a particular order to avoid repeating computations. We use the following definition to specify the calling order.

Definition 19. A *binary partition tree (bpt)* of a finite set $S \neq \emptyset$ is a binary tree such that (a) its root is S, (b) if $|S| = 1$ then its root is a leaf, and (c) if $|S| > 1$ it has a left and a right subtree, themselves bpts of S_1 and S_2, resp., for a partition $\{S_1, S_2\}$ of S.

Let Δ be a bpt of S. We use $\Delta(S')$ for the subtree of Δ rooted at $S' \subseteq S$, if it exists. We use $\langle S, \Delta_1, \Delta_2 \rangle$ for the bpt of S with Δ_1 and Δ_2 as its left and right subtrees.

The following proposition is an immediate consequence of the previous definition.

Proposition 13. *The size (number of nodes) of any bpt of S is $2m-1$ where $m = |S|$.*

DMEETAPP(Δ, c). Let $(L, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$ be a finite distributive scs and $I \subseteq G$. Let $\Delta = \langle S, \Delta_1, \Delta_2 \rangle$ be a bpt of $S \subseteq \mathscr{S}(L)$ where L is a distributive lattice and $S = \{\mathfrak{s}_i \mid i \in I\}$. The recursive program DMEETAPP(Δ, c) defined in Algorithm 1 computes $\mathbb{D}_I(c)$. It uses a global lookup table T for storing the results of calls to DMEETAPP. Initially each entry of T stores a `null` value not included in L. Since S is the union of the roots of Δ_1 and Δ_2, the correctness of DMEETAPP(Δ, c) follows from Corollary 5. Termination follows from the fact that L is finite and the bpts Δ_1 and Δ_2 in the recursive calls are strictly smaller than Δ.

Algorithm 1 DMEETAPP(Δ, c) returns $\mathbb{D}_I(c)$, given a finite distributive scs $(L, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$ and $S = \{\mathfrak{s}_i \mid i \in I\}$ with $I \subseteq G$. Δ is a bpt of S and T is a global lookup table.

```
1: procedure DMEETAPP(Δ, c)                                   ▷ Δ = ⟨S, Δ1, Δ2⟩
2:     if IsNull(T[S, c]) then
3:         if S = {s_i} then
4:             T[S, c] ← s_i(c)
5:         else
6:             T[S, c] ← ⨅_L {DMEETAPP(Δ1, a) ⊔ DMEETAPP(Δ2, c ⊖ a) | a ∈ ↓c}.
```

Let us consider an execution of DMEETAPP(Δ, c). From Proposition 2, it follows that $c \ominus a \in \downarrow c$. Then for each recursive call DMEETAPP(Δ', a') performed by an execution of DMEETAPP(Δ, c) we have $a' \in \downarrow c$. This and the fact that T is initialized with `null` values not in L lead us the following simple observation.

Observation 7. Let $\Delta = \langle S, \Delta_1, \Delta_2 \rangle$ with Δ_1 and Δ_2 rooted at S_1 and S_2, resp. Assume that $T[S_1, a'], T[S_2, a'] \in L$ for every $a' \in {\downarrow}c$. Then the number of binary lattice operations (meets, joins, subtractions) performed by DMEETAPP(Δ, c) is in $O(|{\downarrow}c\,|)$.

DMEET(L, S, P)**.** The values of $\mathbb{D}_I(c)$ for each $c \in P \subseteq L$ are computed by the program in Algorithm 2 as follows. To satisfy the assumption in Observation 7, it visits each node S' of Δ in *post-order* (i.e., before visiting a node it first visits its children). For each subtree $\Delta(S')$ of Δ, it calls DMEETAPP$(\Delta(S'), c)$ for every $c \in P$ in *increasing order* with respect to the order of L: before calling DMEETAPP$(\Delta(S'), c)$ it calls first DMEETAPP$(\Delta(S'), c')$ for each $c' \in (P \cap {\downarrow}c) \setminus \{c\}$. The correctness of the call DMEET(L, S, P) follows from that of DMEETAPP(Δ, c).

Algorithm 2 DMEET(L, S, P). Given a finite distributive scs $(L, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$, $P \subseteq L$ and $S = \{\mathfrak{s}_i \mid i \in I\}$ with $I \subseteq G$, compute $T[S, c] = \mathbb{D}_I(c)$ for each $c \in P$. Δ is a bpt of S and T is a global lookup table.

```
1: T[S', a] ← null                                  ▷ for each a ∈ P and each node S' of Δ
2: for each S' in a post-order traversal sequence of Δ do   ▷ visit each S' of Δ in post-order
3:     for each c ∈ P in increasing order do        ▷ visit each c ∈ P in increasing order w.r.t L
4:         DMEETAPP(Δ(S'), c)
```

Complexity for Distributive Lattices. Assume that L is a distributive lattice of size n and that S is a subset of $\mathscr{S}(L)$ of size m. The above-mentioned traversals of Δ and P ensure that the assumption in Observation 7 is satisfied by each call of the form DMEETAPP$(\Delta(S'), c)$ performed during the execution of DMEET(L, S, L). From Proposition 13 we know that the number of iterations of the outer **for** is $2m - 1$. Clearly $|{\downarrow}c\,|$ and $|P|$ are both in $O(n)$. Thus, given S' we conclude from Observation 7 that the total number of operations from all calls of the form DMEETAPP$(\Delta(S'), c)$, executed in the inner **for**, is in $O(n^2)$. The worst-case time complexity of DMEET(L, S, L) is then in $O(mn^2)$.

Complexity for Powerset Lattices. Assume that L is a powerset lattice. We can compute $\mathbb{D}_I$ in $O(n + m \log n)$ as follows. First call DMEET(L, S, P) where $P = J(L) \cup \{\bot\}$ and $J(L)$ is the set of all the singleton sets (*join-irreducible* elements [15]) of L. Since $|J(L)| = \log_2 n$ and $|{\downarrow}c\,| = 2$ for every $c \in J(L)$, DMEET(L, S, P) can be performed in $O(m \log n)$. This produces $T[S, c] = \mathbb{D}_I(c)$ for each $c \in P$. To compute $T[S, e] = \mathbb{D}_I(e)$ for each $e \in L \setminus P$ in a total time of $O(n)$, visit each such an e in increasing order and set $T[S, e] = T[S, a] \sqcup T[S, b]$ for some $a, b \in {\downarrow}e \setminus \{e\}$ such that $e = a \sqcup b$. Since $e \notin P$ there must be a, b satisfying the above condition.

We now present algorithms to compute $\mathbb{D}_I$ in arbitrary lattices.

3.2.4. Algorithms for Arbitrary Lattices

The previous algorithm may fail to produce the $\mathbb{D}_I$ for non-distributive finite lattices. Nonetheless, for any arbitrary finite lattice L, $\mathbb{D}_I$ can be computed by successive approximations, starting with some self-map known to be smaller than each $\mathfrak{s}_i$ with $i \in I$ and greater than $\mathbb{D}_I$. Assume a self-map $\sigma : L \to L$ such that $\sigma \sqsupseteq_{\mathscr{I}} \mathbb{D}_I$ and, for all $i \in I$, $\sigma \sqsubseteq_{\mathscr{I}} \mathfrak{s}_i$. A good starting point is $\sigma(u) = \bigsqcap\{\mathfrak{s}_i(u) \mid i \in I\}$, for all $u \in L$. By definition of $\sqcap$, σ is the biggest function under all functions $\mathfrak{s}_i$ with $i \in I$, hence $\sigma \sqsupseteq_{\mathscr{I}} \mathbb{D}_I$. The program GMEET in Algorithm 3 computes decreasing upper bounds of $\mathbb{D}_I$ by correcting σ values not conforming to the following *space function property:* $\sigma(u) \sqcup \sigma(v) = \sigma(u \sqcup v)$. The correction decreases σ and maintains the invariant $\sigma \sqsupseteq_{\mathscr{I}} \mathbb{D}_I$, as stated in Theorem 8.

GMEET. The procedure in Algorithm 3 loops through all the pairs $u, v \in L$ while there is some pair satisfying cases (1) or (2) in Theorem 8 for the current σ. When there is, it updates σ as stated in the theorem. At the end of the loop, σ satisfies the join preservation property for all the pairs $u, v \in L$. By the invariant $\sigma \sqsupseteq_{\mathscr{I}} \mathbb{D}_I$, this means $\sigma = \mathbb{D}_I$.

Algorithm 3 GMEET finds $\sigma = \mathbb{D}_I$ given a finite scs $(L, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$ and $I \subseteq G$.

```
1: σ(u) ← ⨅{s_i(u) | i ∈ I}                                   ▷ for all u ∈ L
2: while u, v ∈ L ∧ σ(u) ⊔ σ(v) ≠ σ(u ⊔ v) do
3:     if σ(u) ⊔ σ(v) ⊏ σ(u ⊔ v) then                          ▷ case (1)
4:         σ(u ⊔ v) ← σ(u) ⊔ σ(v)
5:     else                                                   ▷ case (2)
6:         σ(u) ← σ(u) ⊓ σ(u ⊔ v)
7:         σ(v) ← σ(v) ⊓ σ(u ⊔ v)
```

Theorem 8. *Let $(L, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$ be a finite scs, $I \subseteq G$, $u, v \in L$ and $\sigma : L \to L$. Assume $\sigma \sqsupseteq_{\mathscr{I}} \mathbb{D}_I$ holds, and consider the following updates:*

1. *when $\sigma(u) \sqcup \sigma(v) \sqsubset \sigma(u \sqcup v)$, assign $\sigma(u \sqcup v) \leftarrow \sigma(u) \sqcup \sigma(v)$*
2. *when $\sigma(u) \sqcup \sigma(v) \not\sqsubseteq \sigma(u \sqcup v)$, assign $\sigma(u) \leftarrow \sigma(u) \sqcap \sigma(u \sqcup v)$ and also $\sigma(v) \leftarrow \sigma(v) \sqcap \sigma(u \sqcup v)$*

Let σ' be the function resulting after the update. Then, (1) $\sigma' \sqsubset_{\mathscr{I}} \sigma$ and (2) $\sigma' \sqsupseteq_{\mathscr{I}} \mathbb{D}_I$.

Proof. We present the proof by cases on the updates.

- For update (1).

  Given the condition, the assignment obviously decreases $\sigma(u \sqcup v)$, so $\sigma' \sqsubset_{\mathscr{I}} \sigma$.

To prove that $\sigma' \sqsupseteq_{\mathscr{I}} \mathbb{D}_I$, recall that $\sigma \sqsupseteq_{\mathscr{I}} \mathbb{D}_I$, then $\sigma(u) \sqsupseteq \mathbb{D}_I(u)$ and $\sigma(v) \sqsupseteq \mathbb{D}_I(v)$. Therefore, $\sigma'(u \sqcup v) = \sigma(u) \sqcup \sigma(v) \sqsupseteq \mathbb{D}_I(u) \sqcup \mathbb{D}_I(v) = \mathbb{D}_I(u \sqcup v)$.

- For update (2).

  The assignments either decrease $\sigma(u)$ or $\sigma(v)$ (or both). To see why, assume the opposite. We know that $\sigma(u) = \sigma(u) \sqcap \sigma(u \sqcup v)$ and $\sigma(v) = \sigma(v) \sqcap \sigma(u \sqcup v)$ imply $\sigma(u) \sqsubseteq \sigma(u \sqcup v)$ and $\sigma(v) \sqsubseteq \sigma(u \sqcup v)$. Therefore, $\sigma(u) \sqcup \sigma(v) \sqsubseteq \sigma(u \sqcup v)$, contradicting the condition for update 2.

  Now, we prove that assignments in update 2 also preserve the invariant $\sigma \sqsupseteq_{\mathscr{I}} \mathbb{D}_I$.

  Assume $\sigma(u) \sqcap \sigma(u \sqcup v) \sqsubset \sigma(u)$ (otherwise the invariant holds trivially). By the invariant hypothesis for σ before the assignment, we have that $\sigma(u) \sqsupseteq \mathbb{D}_I(u)$ and $\sigma(u \sqcup v) \sqsupseteq \mathbb{D}_I(u \sqcup v)$. Therefore,

$$\begin{aligned}
\sigma'(u) = \sigma(u) \sqcap \sigma(u \sqcup v) &\sqsupseteq \mathbb{D}_I(u) \sqcap \mathbb{D}_I(u \sqcup v) \\
&= \mathbb{D}_I(u) \sqcap (\mathbb{D}_I(u) \sqcup \mathbb{D}_I(v)) \\
&= \mathbb{D}_I(u) \sqcup (\mathbb{D}_I(u) \sqcap \mathbb{D}_I(v)) \\
&= \mathbb{D}_I(u)
\end{aligned}$$

  The proof for $\sigma'(v)$ is analogous.

■

As for the previous algorithms, the worst-time time complexity will be expressed in terms of the binary lattice operations performed during execution.

Complexity for GMEET. Assume a fixed set I of size m. The complexity of the initialization (Line 1) of GMEET is $O(nm)$ with $n = |L|$. The value of σ for a given $w \in L$ can be updated (decreased) at most n times. Thus, there are at most n^2 updates of σ for all values of L. Finding a $w = u \sqcup v$ where $\sigma(w)$ needs an update because $\sigma(u) \sqcup \sigma(v) \neq \sigma(u \sqcup v)$ (test of the loop, Line 2) takes $O(n^2)$. Hence, the worst time complexity of the loop is in $O(n^4)$.

The time complexity of GMEET can be reduced significantly by using appropriate data structures. We present this enhancement next in GMEET+.

GMEET+. Essentially, Algorithm 4 uses different sets to separate lattice pairs (u, v) with respect to the current σ. We have support (correct) pairs sets $\texttt{Sup}_w = \{(u,v) \mid w = u \sqcup v \wedge \sigma(u) \sqcup \sigma(v) = \sigma(w)\}$, conflicts sets $\texttt{Con}_w = \{(u,v) \mid w = u \sqcup v \wedge \sigma(u) \sqcup \sigma(v) \sqsubset \sigma(w)\}$ and failures sets $\texttt{Fail}_w = \{(u,v) \mid w = u \sqcup v \wedge \sigma(u) \sqcup$

Algorithm 4 GMEET+ finds $\sigma = \mathbb{D}_I$ given a finite scs $(L, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$ and $I \subseteq G$.

1: $\sigma(u) \leftarrow \bigsqcap\{\mathfrak{s}_i(u) \mid i \in I\}$ ▷ for all $u \in L$
2: Initialize $\mathtt{Sup}_w, \mathtt{Con}_w, \mathtt{Fail}_w$, for all w
3: **while** $w \in L$ such that $(u,v) \in \mathtt{Con}_w$ **do** ▷ some conflict set not empty
4: $\quad \mathtt{Con}_w \leftarrow \mathtt{Con}_w \backslash \{(u,v)\}$
5: $\quad \sigma(w) \leftarrow \sigma(u) \sqcup \sigma(v)$
6: $\quad \mathtt{Fail}_w \leftarrow \mathtt{Fail}_w \cup \mathtt{Sup}_w$ ▷ all pairs previously in $\mathtt{Sup}_w$ are now failures
7: $\quad \mathtt{Sup}_w \leftarrow \{(u,v)\}$
8: $\quad$ CHECKSUPPORTS(w) ▷ for $u \in L$, verify property $\mathtt{Sup}_{w\sqcup u}$
9: $\quad$ **while** $z \in L$ such that $(x,y) \in \mathtt{Fail}_z$ **do** ▷ some failures set not empty
10: $\quad\quad \mathtt{Fail}_z \leftarrow \mathtt{Fail}_z \backslash \{(x,y)\}$
11: $\quad\quad$ **if** $\sigma(x) \neq \sigma(x) \sqcap \sigma(z)$ **then**
12: $\quad\quad\quad \sigma(x) \leftarrow \sigma(x) \sqcap \sigma(z)$ ▷ $\sigma(x)$ decreases
13: $\quad\quad\quad \mathtt{Fail}_x \leftarrow \mathtt{Fail}_x \cup \mathtt{Sup}_x$ ▷ all pairs in $\mathtt{Sup}_x$ are now failures
14: $\quad\quad\quad \mathtt{Sup}_x \leftarrow \emptyset$
15: $\quad\quad\quad$ CHECKSUPPORTS(x) ▷ for $u \in L$, verify property $\mathtt{Sup}_{x\sqcup u}$
16: $\quad\quad$ **if** $\sigma(y) \neq \sigma(y) \sqcap \sigma(z)$ **then**
17: $\quad\quad\quad \sigma(y) \leftarrow \sigma(y) \sqcap \sigma(z)$ ▷ $\sigma(y)$ decreases
18: $\quad\quad\quad \mathtt{Fail}_y \leftarrow \mathtt{Fail}_y \cup \mathtt{Sup}_y$ ▷ all pairs in $\mathtt{Sup}_y$ are now failures
19: $\quad\quad\quad \mathtt{Sup}_y \leftarrow \emptyset$
20: $\quad\quad\quad$ CHECKSUPPORTS(y) ▷ for $u \in L$, verify property $\mathtt{Sup}_{y\sqcup u}$
21: $\quad\quad$ **if** $\sigma(x) \sqcup \sigma(y) = \sigma(z)$ **then**
22: $\quad\quad\quad \mathtt{Sup}_z \leftarrow \mathtt{Sup}_z \cup \{(x,y)\}$ ▷ (x,y) is now correct
23: $\quad\quad$ **else**
24: $\quad\quad\quad \mathtt{Con}_z \leftarrow \mathtt{Con}_z \cup \{(x,y)\}$ ▷ (x,y) is now a conflict

$\sigma(v) \not\sqsubseteq \sigma(w)\}$. The values of σ are updated as stated in Theorem 8 and so maintains the invariant $\sigma \sqsupseteq_{\mathscr{S}} \mathbb{D}_I$. An additional invariant is that, for all w, sets $\mathtt{Sup}_w, \mathtt{Con}_w, \mathtt{Fail}_w$ are pairwise disjoint. When the outer loop finishes sets $\mathtt{Con}_w$ and $\mathtt{Fail}_w$ are empty (for all w) and thus every (u,v) belongs to $\mathtt{Sup}_{u\sqcup v}$, i.e. the resulting $\sigma = \mathbb{D}_I$.

The auxiliary procedure CHECKSUPPORTS(u) identifies all pairs of the form $(u,x) \in \mathtt{Sup}_{u\sqcup x}$ that may no longer satisfy the space function property $\sigma(u) \sqcup \sigma(x) = \sigma(u \sqcup x)$ because of an update to $\sigma(u)$. When this happens, it adds (u,x) to the appropriate conflicts or failures set. The time complexity of the algorithm depends on the set operations computed for each $w \in L$ chosen, either in the *conflicts* $\mathtt{Con}_w$ set or in the *failures* $\mathtt{Fail}_w$ set. When a w is selected (for some (u,v) such that $u \sqcup v = w$) the following holds: (1) at least one of $\sigma(w), \sigma(u), \sigma(v)$ is decreased, (2) some fix k number of elements are removed from or added to a set, (3) a union of two *disjoint* sets is computed, and (4) new support sets of w, u or v are calculated.

Complexity for GMEET+. With an appropriate implementation, operations (1)-(2) take $O(1)$, and also operation (3), since sets are disjoint. Operation (4) clearly takes $O(n)$. In each loop of the (outer or inner)

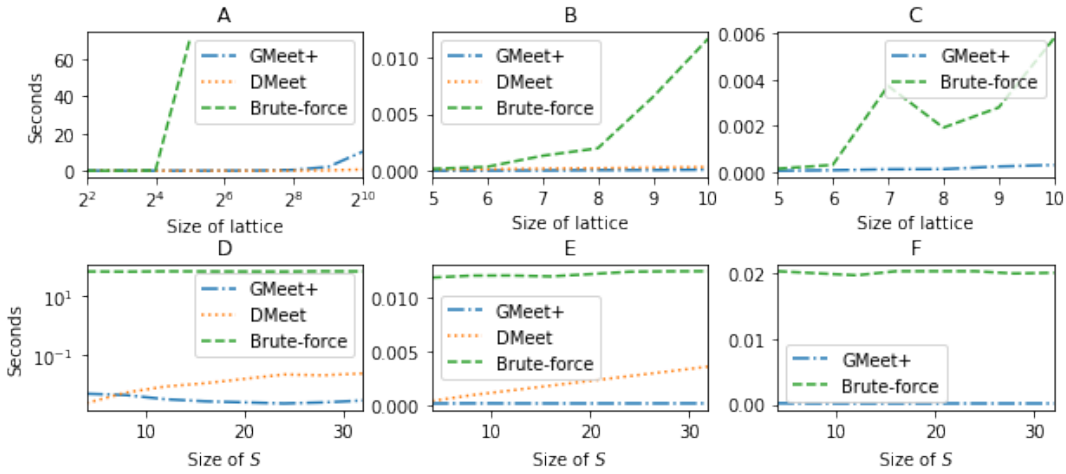


Figure 3.2: Average performance time of GMEET+, DMEET and BRUTE-FORCE. Plots A and D use $\mathbf{2}^n$ lattices, B and E distributive lattices, and C and F arbitrary (possibly non-distributive) lattices. Plots A-C have a fixed number of space functions and plots D-F have a fixed lattice size.

cycles of the algorithm, at least one σ reduction is computed. Furthermore, for each reduction of σ, $O(n)$ operations are performed. The maximum possible number of $\sigma(w)$ reductions, for a given w, is equal to the length d of the longest strictly decreasing chain in the lattice. The total number of possible σ reductions is thus equal to nd. The total number of operations of the algorithm is then $O(n^2 d)$. In general, d could be (at most) equal to n, therefore, after initialization, worst case complexity is $O(n^3)$. The initialization (Lines 1-2) takes $O(nm)+O(n^2)$, where $m = |I|$. Worst time complexity is thus $O(mn+n^3)$. For powerset lattices, $d = \log_2 n$, thus worst time complexity in this case is $O(mn+n^2 \log_2 n)$.

3.2.5. Experimental Results

Consider Figure 3.2. In plots 3.2.A-C, the horizontal axis is the size of the lattice. In plots 3.2.D-F, the horizontal axis is the size of S. Curves in images 3.2.A-C plot, for each algorithm, the average execution time of 100 runs (10 for 3.2.A) with random sets $I \subseteq G$ of size 4. Images 3.2.D-F, show the mean execution time of each algorithm for 100 runs (10 for 3.2.D) varying the number of space functions ($|I| = 4j$, $1 \leq j \leq 8$). The lattice size is fixed: $|L| = 10$ for 3.2.E and 3.2.F, and $|L| = 2^5$ for 3.2.D. In all cases the lattices were randomly generated, and the parameters selected to showcase the difference between each algorithm with a sensible overall execution time. For a given scs $(L, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$ and $I \subseteq G$, the brute-force algorithm explores the whole space $\mathscr{S}(L)$ to find all the space function below each $\mathfrak{s}_i$ with $i \in I$ and then computes the greatest of them. In particular, the measured spike in plot 3.2.C corresponds to the random lattice of seven elements with the size of $\mathscr{S}(L)$ being bigger than in the other experiments in the same figure. In our experiments we observed that for a fixed I, as the size of the lattice increases, DMEET outperforms GMEET+. This

is noticeable in lattices $\mathbf{2}^n$ (see 3.2.A). Similarly, for a fixed lattice, as the size of I increases GMEET+ outperforms DMEET. GMEET+ performance can actually improve with a higher number of space functions (see 3.2.D) since the initial σ is usually smaller in this case.

3.3. Applications: Minkowski Addition and Mathematical Morphology

In this section we shall show that some fundamental operations from Mathematical Morphology (MM) have a counterpart in the theory we developed in the previous chapter. In particular we shall show that *distributed spaces*, the central notion of this dissertation, have a natural interpretation in MM. Furthermore, we shall use our results on distributed information to provide new constructions and results for MM.

3.3.1. Preliminaries

To present the results in this section uniformly we shall use a fundamental structure from algebra called *module*. We recall the definition of module as well as that of group and ring.

Definition 20 (Group [20])**.** Let G be a set and $\star$ a binary operation on G. A *group* is an ordered pair $(G,\star)$ such that: $\star$ is associative, there exists an element $e \in G$, called an *identity* of G, that for all $a \in G$, $a\star e = e\star a = a$; for each $a \in G$, there is an element $(-a) \in G$, called an *inverse* of a, such that $a\star(-a) = (-a)\star a = e$.
The group $(G,\star)$ is called *abelian* if $\star$ is commutative.

Definition 21 (Ring [20])**.** A *ring* R is a set equipped with two binary operations $+$ and $\times$ such that: $(R,+)$ is an abelian group, $\times$ is associative, and $\times$ distributes over $+$.
A ring R is said to be *commutative* if $\times$ is commutative. Also, R is said to have an *identity* if there is an element $1 \in R$ such that, for all $a \in R$, $1 \times a = a \times 1 = a$.
For simplicity we shall use ab instead of $a \times b$ for $a, b \in R$.

Definition 22 (Module [20])**.** Let R be a ring, M be a set and $+$ be a binary operation on M. We say that M is a left *R-module* or a *left module over R* if:

- $(M,+)$ is an abelian group,
- $\mathbf{u}+\mathbf{v} \in M$ and $r\mathbf{u} \in M$ whenever $\mathbf{u},\mathbf{v} \in M$ and $r \in R$.
- For every $\mathbf{u},\mathbf{v} \in M$ and every $r,s \in R$,

- $r(s\mathbf{u}) = (rs)\mathbf{u}$,
- $r(\mathbf{u}+\mathbf{v}) = r\mathbf{u}+r\mathbf{v}$,
- $(r+s)\mathbf{u} = r\mathbf{u}+s\mathbf{u}$.

If R has an identity, say 1, the axiom $1\mathbf{u} = \mathbf{u}$ is included.

A module M over a *ring* R is a generalization of the notion of *vector space*. In a vector space the ring R needs to be a *field* [5]. We shall take the liberty of referring to the elements of M and R as *vectors* and *scalars*, resp. As the reader may notice, the former will be written in boldface to distinguish them from the latter. As usual, we shall refer to the binary operation $+$ as addition.

We shall use the following basic properties of modules. The additive identity for any module M and the additive inverse for every $\mathbf{u} \in M$ are unique. If R is a field then the R-module M is a *vector space* over R. If M is a vector space over R then 1 is the only multiplicative identity for M.

The following examples of modules are fundamental in Mathematical Morphology. One of them is not a vector space; it justifies using modules rather than vector spaces as the underlying structure.

Example 17. The set R^n of all n-tuples of elements of a ring R can be made into an R-module. Given $\mathbf{u} = (p_1,\ldots,p_n) \in R^n, \mathbf{v} = (q_1,\ldots,q_n) \in R^n$ and $r \in R$, define $\mathbf{u}+\mathbf{v} = (p_1+q_1,\ldots,p_n+q_n)$ and $r\cdot\mathbf{v} = (rp_1,\ldots,rp_n)$. The module additive identity $\mathbf{0}$ is $(0,\ldots,0) \in R^n$, where 0 is the ring additive identity, and the module inverse additive $-\mathbf{u}$ is $(-p_1,\ldots,-p_n)$ where $-p_i$ is the ring additive inverse of p_i.
The *Euclidean n-dimensional space* $\mathbb{R}^n$ is obtained by taking R as the set of reals numbers $\mathbb{R}$ in the above example. Since $\mathbb{R}$ is also a field, $\mathbb{R}^n$ is also a vector space. The *n-dimensional grid* $\mathbb{Z}^n$ is obtained by taking R as the set of integers $\mathbb{Z}$. This is an example of a module that it is not a vector space since the ring $\mathbb{Z}$ is not a field.

3.3.2. Minkowski Addition

In geometry, vector addition is extended to addition of sets of vectors in an operation known as *Minkowski addition*. From now on we shall omit mentioning the ring of the module when it is unimportant or clear from the context.

Definition 23 (Minkowski Sum [80])**.** Let M be a module and $A, B \subseteq M$. The *Minkowski addition* of A and B is defined thus $A \oplus B = \{\mathbf{u}+\mathbf{v} \mid \mathbf{u} \in A \text{ and } \mathbf{v} \in B\}$.

It is easy to see that $\oplus$ is associative and commutative, it has $\{\mathbf{0}\}$ and $\emptyset$ as identity and absorbent elements, resp., and that it distributes over set union.

Proposition 14 ([80])**.** *Let M be a module. Then $(\mathscr{P}(M),\oplus)$ is a* commutative monoid with zero element $\emptyset$ *and identity $\{\mathbf{0}\}$. Furthermore, $X\oplus(A\cup B)=(X\oplus A)\cup(X\oplus B)$ for every $X,A,B\subseteq M$.*

Regarding $\oplus$ and set intersection, it is known that the distribution law $A\oplus(B\cap C)=(A\cap B)\oplus(A\cap C)$ holds for convex sets [80] —a set of points such that, given any two points in the set, the line segment joining them lies entirely within it. Nevertheless, in general, $\oplus$ does not distribute over set intersection as illustrated next.

Example 18. Take the Euclidean one-dimensional vector space $\mathbb{R}$ and let $X=\{0,1\}$, $A=\{1\}$ and $B=\{2\}$. One can verify that $\emptyset=X\oplus(A\cap B)\neq(X\oplus A)\cap(X\oplus B)=\{2\}$.

However, as part of the application results in this section we shall establish a pleasant new equation for $X\oplus(A\cap B)$: For every module M and for all $X,A,B\subseteq M$ we have

$$X\oplus(A\cap B)=\bigcap_{Y\subseteq X}(Y\oplus A)\cup((X\setminus Y)\oplus B). \tag{3.1}$$

The Minkowski sum has been applied in mathematical morphology as well as in collision detection, robot motion planing, aggregation theory [81, 59, 87]. In this section we focus on applications to mathematical morphology.

3.3.3. Mathematical Morphology

Mathematical morphology (MM) is a theory developed for the analysis of geometric structures [82] . It is founded upon, among others, set theory, lattice theory, geometry, topology and probability. Basically, this theory considers an arbitrary space M where its objects are transformed by two fundamental operations: *dilation* and *erosion*.

In [9] dilations and erosions are typically defined in terms of Minkowski additions over the modules $\mathbb{R}^n$ or $\mathbb{Z}^n$ given in Example 17. Here we generalize the definition in [9] to arbitrary modules.

Definition 24 (Dilations and Erosions in Modules)**.** Let M be a module. A *dilation* by $S\subseteq M$ is a function $\delta_S:\mathscr{P}(M)\longrightarrow\mathscr{P}(M)$ given by $\delta_S(X)=X\oplus S=\bigcup_{\mathbf{u}\in S}X\oplus\{\mathbf{u}\}$. An *erosion* by $S\subseteq M$ is a function $\varepsilon_S:\mathscr{P}(M)$ $\longrightarrow\mathscr{P}(M)$ given by $\varepsilon_S(X)=X\ominus S$ where $X\ominus S\stackrel{\text{def}}{=}\bigcap_{\mathbf{u}\in S}X\oplus\{-\mathbf{u}\}$.

In MM, a binary image X is typically represented as a subset of the module $M=\mathbb{Z}^2$ where a pixel is activated if its corresponding coordinate (or position vector) is in X. The *translation* of a vector $\mathbf{u}$ by a vector $\mathbf{v}$ is given by $\mathbf{u}+\mathbf{v}$. The dilation $\delta_S(X)$ describes the interaction of X with another image S

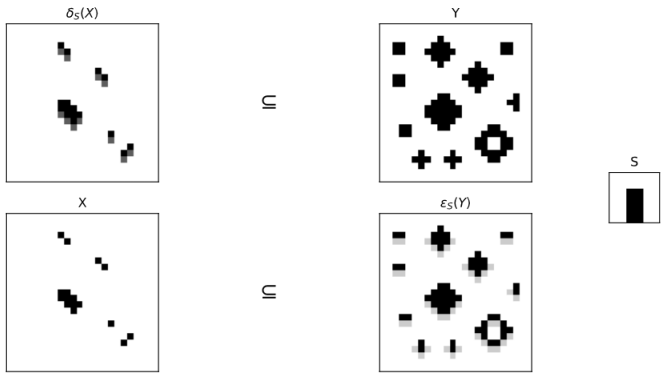


Figure 3.3: Example of Galois connection between dilations and erosions: $\delta_S(X) \subseteq Y$ iff $X \subseteq \varepsilon_S(Y)$. Here X and Y are images, S is the structuring element (centered at the origin), $\delta_S(X)$ is a dilation and $\varepsilon_S(Y)$ is an erosion. Pixels added/removed by dilation/erosion are depicted in gray.

referred to as a *structuring element* and typically assumed to include the center, i.e., $\mathbf{0} = (0,0) \in S$. The dilated image $\delta_S(X)$ "inflates" the original one by including X and adding the pixels from the translation of every $\mathbf{v}$ in X by each $\mathbf{u}$ in S. Intuitively, $\delta_S(X)$ can be viewed as redrawing the image X with the brush S [9]. This is illustrated in Figure 3.3 where an image X is dilated by the structuring element $S = \{(0,0),(0,-1)\}$, and in Figure 3.4 where an image X is dilated by two different structuring elements $A = \{(1,1),(0,0),(1,0),(-1,-1),(0,-1)\}$ and $B = \{(-1,1),(-1,0),(0,0),(0,-1),(1,-1)\}$.

Furthermore, the erosion $\varepsilon_S(X)$ in $\mathbb{R}^n$ or $\mathbb{Z}^n$ can be defined in terms of the translations of S that are contained in X. The next proposition states this result for modules.

Proposition 15. *Let M be a module and $S,X \subseteq M$. Then $\varepsilon_S(X) = \{\mathbf{u} \in M \mid S \oplus \{\mathbf{u}\} \subseteq X\}$.*

Proof. Let M be a module and $S,X \subseteq M$. From Definition 24, we will prove that $\bigcap_{\mathbf{u}\in S} X \oplus \{-\mathbf{u}\} = \{\mathbf{u} \in M \mid S \oplus \{\mathbf{u}\} \subseteq X\}$. Take any $\mathbf{v} \in \bigcap_{\mathbf{u}\in S} X \oplus \{-\mathbf{u}\}$, then for every $\mathbf{u} \in S$, $\mathbf{v} = \mathbf{w} + (-\mathbf{u})$ for some $\mathbf{w} \in X$. Since for every $\mathbf{u} \in S$, $\mathbf{u} + \mathbf{v} \in S \oplus \{\mathbf{v}\}$ and $\mathbf{u} + \mathbf{v} = \mathbf{w} \in X$ (see Proposition 14), we have $S \oplus \{\mathbf{v}\} \subseteq X$. Therefore, $\mathbf{v} \in \{\mathbf{u} \in M \mid S \oplus \{\mathbf{u}\} \subseteq X\}$.

Now, let $\mathbf{v} \in \{\mathbf{u} \in M \mid S \oplus \{\mathbf{u}\} \subseteq X\}$, then $S \oplus \{\mathbf{v}\} \subseteq X$. Notice that for any $\mathbf{w} \in S$, $\mathbf{w} + \mathbf{v} \in S \oplus \{\mathbf{v}\} \subseteq X$ and therefore $\mathbf{v} = (\mathbf{w} + \mathbf{v}) + (-\mathbf{w}) \in X \oplus \{-\mathbf{w}\}$ for every $\mathbf{w} \in S$. Then $\mathbf{v} \in \bigcap_{\mathbf{u}\in S} X \oplus \{-\mathbf{u}\}$. ■

Assuming that S includes the center, the above proposition tells us that an erosion $\varepsilon_S(X)$ reduces the image X by erasing the pixels in X whose translation by some element of S is not within X. This can also be easily seen from the fact that erosions satisfy the equation $\varepsilon_S(X) = \{\mathbf{u} \in M \mid \text{ for each } \mathbf{v} \in S : \mathbf{u} + \mathbf{v} \in X\}$ which follows directly from Proposition 15. Figure 3.3 illustrates the erosion of an image Y by the structuring element S.

3.3.4. Dilations as Space Functions

We now state that the power set with the usual order and the set of all dilations over it form a *(completely distributive) spatial constraint system*; i.e., an scs whose underlying lattice is completely distributive.

Theorem 9. *Let M be a module. Then $(\mathscr{P}(M), \subseteq, (\delta_S)_{S\subseteq M})$ is a completely distributive scs.*

Proof. The power set of any set ordered by inclusion is a completely distributive lattice with join $\sqcup = \cup$ and meet $\sqcap = \cap$ [15]. Hence $(\mathscr{P}(M), \subseteq)$ is a completely distributive cs.

It remains to prove that for any $S \in \mathscr{P}(M)$, the dilation δ_S is a space function. From Proposition 4 part (2) it suffices to show that $\delta_S(\bigcup_i A_i) = \bigcup_i \delta_S(A_i)$ for every arbitrary union $\bigcup_i A_i \in \mathscr{P}(M)$. This follows from the following equations:

$$\begin{aligned}
\delta_S\left(\bigcup_i A_i\right) &= \left\{\mathbf{x}+\mathbf{e} \mid \mathbf{x} \in \bigcup_i A_i \text{ and } \mathbf{e} \in S\right\} \\
&= \{\mathbf{x}+\mathbf{e} \mid \mathbf{x} \in A_i \text{ for some } i \text{ and } \mathbf{e} \in S\} \\
&= \{\mathbf{x}+\mathbf{e} \mid \mathbf{x}+\mathbf{e} \in \delta_S(A_i) \text{ for some } i\} \\
&= \bigcup_i \delta_S(A_i)
\end{aligned}$$

Thus, $(\mathscr{P}(M), \subseteq, (\delta_S)_{S\subseteq M})$ is a completely distributive scs.

■

The above theorem states that dilations are space functions. Erosions, on the other hand, are space projections (see Definition 11).

Proposition 16. *Let M be a module. For every $S \subseteq M$, the function ε_S is the S-projection in the scs $(\mathscr{P}(M), \subseteq, (\delta_S)_{S\subseteq M})$.*

Proof. Let M be a module and $S \subseteq M$. To prove that ε_S is an S-projection we show that dilations and erosions form a Galois connection, i.e., for every $X, Y \subseteq \mathscr{P}(M)$, $\delta_S(X) \subseteq Y$ iff $X \subseteq \varepsilon_S(Y)$. It is known that a Galois connection determines each function uniquely. Therefore from Proposition 8 (1) it follows that the erosion ε_S must then be a projection.

Pick arbitrary $X, Y, S \subseteq \mathscr{P}(M)$. We have $\delta_S(X) = \bigcup_{\mathbf{v}\in S} X \oplus \{\mathbf{v}\} \subseteq Y$ iff for every $\mathbf{u} \in S$, $X \oplus \{\mathbf{u}\} \subseteq Y$.

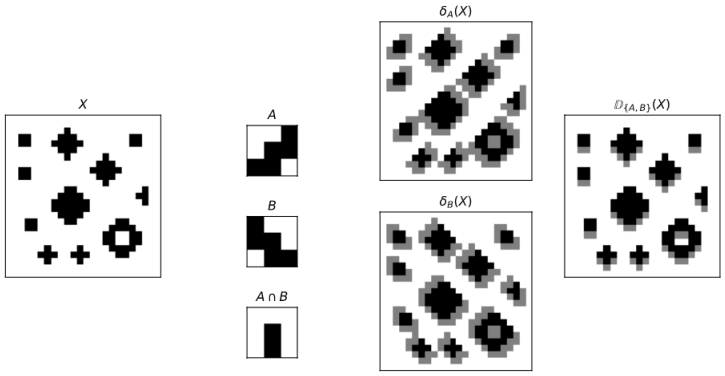


Figure 3.4: From left to right, image X, structuring elements A, B and $A \cap B$, dilations $\delta_A(X)$ and $\delta_B(X)$, and dilation $\mathbb{D}_{\{A,B\}}(X)$. Structuring elements are centered at the origin. Pixels added by dilation are depicted in gray.

Furthermore, with the help of the monoid laws for $\oplus$ (Proposition 14) we can show that for every $\mathbf{u} \in S$:

$$
\begin{aligned}
& X \oplus \{\mathbf{u}\} \subseteq Y \\
\textit{iff}\ & \langle \text{Property of } \subseteq \rangle \\
& (X \oplus \{\mathbf{u}\}) \cup Y = Y \\
\textit{iff}\ & \langle Z{=}Z{\oplus}\{\mathbf{0}\} \text{ and } \{\mathbf{0}\}{=}\{-\mathbf{u}\}{\oplus}\{\mathbf{u}\} \rangle \\
& (X \oplus \{\mathbf{u}\}) \cup (Y \oplus \{-\mathbf{u}\} \oplus \{\mathbf{u}\}) = Y \\
\textit{iff}\ & \langle X{\oplus}(A{\cup}B){=}(X{\oplus}A){\cup}(X{\oplus}B) \rangle \\
& (X \cup (Y \oplus \{-\mathbf{u}\})) \oplus \{\mathbf{u}\} = Y \\
\textit{iff}\ & \langle \text{Adding } \{-\mathbf{u}\} \text{ by } \oplus \rangle \\
& (X \cup (Y \oplus \{-\mathbf{u}\})) \oplus \{\mathbf{u}\} \oplus \{-\mathbf{u}\} = Y \oplus \{-\mathbf{u}\} \\
\textit{iff}\ & \langle \{-\mathbf{u}\}{\oplus}\{\mathbf{u}\}{=}\{\mathbf{0}\} \text{ and } Z{\oplus}\{\mathbf{0}\}{=}Z \rangle \\
& X \cup (Y \oplus \{-\mathbf{u}\}) = Y \oplus \{-\mathbf{u}\} \\
\textit{iff}\ & \langle \text{Property of } \subseteq \rangle \\
& X \subseteq Y \oplus \{-\mathbf{u}\}.
\end{aligned}
$$

Clearly for every $\mathbf{u} \in S$, $X \subseteq Y \oplus \{-\mathbf{u}\}$ iff $X \subseteq \bigcap_{\mathbf{v} \in S} Y \oplus \{-\mathbf{v}\} = \varepsilon_S(Y)$. We have then established that $\delta_S(X) \subseteq Y$ iff $X \subseteq \varepsilon_S(Y)$ as wanted. ■

In the proof of the above proposition, we show that dilations and erosions form a Galois connection.

Despite this is a known fact in the MM community for $\mathbb{R}^n$ or $\mathbb{Z}^n$ space [72], here we proved a more general version of it for modules.

Remark 9 (An Epistemic Interpretation of MM operations). Since we have shown that dilation is a space function, and erosion is a space projection, we can now think of these functions in terms of information available to an agent. Thus, for example in Figure 3.3, when Y is the actual state, $\varepsilon_S(Y)$ shows what S perceives. The interpretation of δ_S is more complicated: in fact, we interpret X as the information available to, or perceived by, agent S, when $\delta_S(X)$ is the real situation.

We notice that with the interpretation of dilations as visual perception, an agent with perfect vision corresponds to the structuring element which is a single pixel at the origin, and if structuring elements $S_1 \subseteq S_2$, then S_2 represents vision which is more blurry than the vision represented by S_1, as seen in Figure 3.5. Thus, the agent only perceives some of the real information, and some parts of the actual state of the world are hidden from the agent. Specifically, we can consider this as an agent with blurred vision who does not perceive some of the edges of objects. In the instance of Figure 3.5, there are two agents, agent 1, corresponding to structuring element S_1, with slightly blurred vision, and agent 2 with corresponding structuring element S_2, with extremely blurred vision.

Neither agent can perceive the edges of the objects. So, when $\delta_{S_1}(X)$ is the true state of affairs, S_1 perceives the information in X; in effect, losing one pixel at the edge of each object. On the other hand, when $\delta_{S_2}(X)$ is the real situation, S_2 perceives it as X, because this agent loses two pixels from the edge of every object. Thus, given an agent S, we may think of $\delta_S(X)$ as the situation where agent S has information X, because X is what the agent perceives when $\delta_S(X)$ is the real situation.

In contrast, in Figure 3.5 we note that if X represents the real situation, then agent 1 perceives $\varepsilon_{S_1}(X)$, losing one pixel at the edge of each object. Similarly, agent 2 perceives $\varepsilon_{S_2}(X)$ losing two pixels at the edge of each object.

In Figure 3.3, the agent's vision is blurred in the vertical direction: they only perceive a pixel if it also has another pixel below it, but they are unable to perceive the bottom edge of any object. ■

One may wonder if every space function over $(\mathscr{P}(M), \subseteq)$ is a dilation δ_S for some $S \subseteq M$. Proposition 18 answers this question negatively. First, we need to introduce a new family of functions over modules.

Definition 25 (Scale Function)**.** Let M be a module over a ring R. Given $r \in R$, a scale by r is a function $s_r : \mathscr{P}(M) \longrightarrow \mathscr{P}(M)$ defined as $s_r(X) = \{r\mathbf{u} \mid \mathbf{u} \in X\}$.

It is easy to see that s_r is a space function over $(\mathscr{P}(M), \subseteq)$.

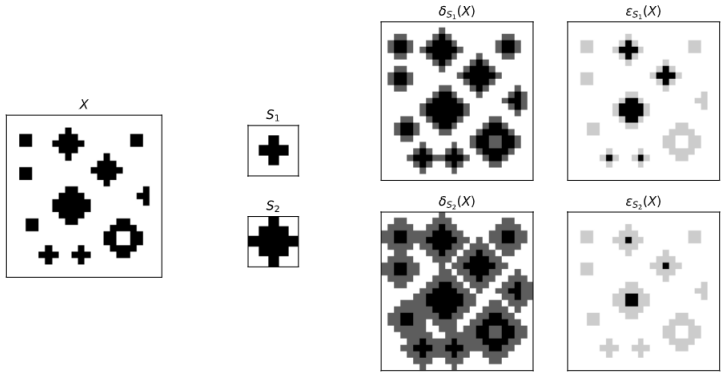


Figure 3.5: From left to right, image X, structuring elements S_1 and S_2, dilations $\delta_{S_1}(X)$ and $\delta_{S_2}(X)$, and erosions $\varepsilon_{S_1}(X)$ and $\varepsilon_{S_2}(X)$. Structuring elements are centered at the origin. Pixels added/removed by dilation/erosion are depicted in dark/light grey.

Proposition 17. *Let M be a module over a ring R. Then for every $r \in R$, s_r is a space function over $(\mathscr{P}(M), \subseteq)$*

Proof. Let M be a module over a ring R. From Proposition 4 (2), it suffices to show that given $r \in R$ and an arbitrary $\bigcup_i A_i \in \mathscr{P}(M)$, $s_r(\bigcup_i A_i) = \bigcup_i s_r(A_i)$. Indeed, we have $s_r(\bigcup_i A_i) = \{r\mathbf{u} \mid \mathbf{u} \in \bigcup_i A_i\} = \{r\mathbf{u} \mid \mathbf{u} \in A_i \text{ for some } i\} = \{r\mathbf{u} \mid r\mathbf{u} \in s_r(A_i) \text{ for some } i\} = \bigcup_i s_r(A_i)$ as wanted. ■

The following proposition gives us a necessary and sufficient condition for a scale function to be a dilation. In particular, it tells us that we can have infinitely many space functions that *are not dilations* by some structuring element if the underlying module is, for example, the Euclidean vector space $\mathbb{R}^n$ or the grid $\mathbb{Z}^n$ (see Example 17).

Proposition 18. *Let M be a module over a ring R. Then for each $r \in R$, $s_r = \delta_S$ for some $S \subseteq M$ if and only if r is a multiplicative identity for M.*

Proof. Suppose that r is a multiplicative identity for M. Take $S = \{\mathbf{0}\}$. Clearly $s_r = \delta_{\{\mathbf{0}\}}$. For the other direction we proceed by contradiction. Let us suppose that r is not a multiplicative identity for M but that there exists S such that $s_r = \delta_S$. By applying both space functions to $\{\mathbf{0}\}$, we obtain $s_r(\{\mathbf{0}\}) = \{r\mathbf{0}\} = \{\mathbf{0}\} = \delta_S(\{\mathbf{0}\})$. Then for every $\mathbf{v} \in S$, $\mathbf{0} + \mathbf{v} = \mathbf{0}$, hence $\mathbf{v} = \mathbf{0}$. Thus $S = \{\mathbf{0}\}$. It follows that for every $\mathbf{u} \in M$, $s_r(\{\mathbf{u}\}) = \{r\mathbf{u}\} = \delta_{\{\mathbf{0}\}}(\{\mathbf{u}\}) = \{\mathbf{u}\}$. This implies that for every $\mathbf{u} \in M$, $r\mathbf{u} = \mathbf{u}$, thus r is a multiplicative identity for M, a contradiction.

■

3.3.5. The Distributed Spaces and Dilations

We have shown that dilations are space functions while erosions are space projections. However, the main construction of this work is that of *distributed spaces*: The greatest space function below a given set of space functions. The problem we shall address is the following: Given two dilations δ_A and δ_B, *find* the greatest space function $\mathbb{D}_{\{A,B\}}$ below them. Let us consider some issues regarding this question.

Recall that join and meet operations of the power set $(\mathscr{P}(M),\subseteq)$ are set union and intersection. Notice that simply taking the point-wise greatest lower bound does not work, i.e., in general, the equation $\mathbb{D}_{\{A,B\}}(X) = \delta_A(X) \cap \delta_B(X)$ *does not hold*.

Example 19. Consider Example 18 with the Euclidean one-dimensional vector space $M = \mathbb{R}$, $X = \{0,1\}$, $A = \{1\}$ and $B = \{2\}$. Let $f(Y) = \delta_A(Y) \cap \delta_B(Y)$ for every $Y \subseteq M$. One can verify that $f(\{0\} \cup \{1\}) = \{2\} \neq \emptyset = f(\{0\}) \cup f(\{1\})$, hence f is not even a space function.

Furthermore, Proposition 18 tells us that there are space functions over $(\mathscr{P}(M),\subseteq)$ that are not dilations. Thus, in principle it is not clear if $\mathbb{D}_{\{A,B\}}$ is itself a dilation and if it is, we would like to identify what its structuring element should be.
The main result of this section, given next, addresses these issues.

Theorem 10 (Distributed Spaces as Dilations)**.** *Suppose that M is a module. Let $(\mathbb{D}_S)_{S\subseteq M}$ be the distributed spaces of the scs $(\mathscr{P}(M),\subseteq,(\delta_S)_{S\subseteq M})$. Then $\mathbb{D}_{\{A,B\}} = \delta_{A\cap B}$ for every $A,B \subseteq M$.*

Therefore, the greatest space function below two dilations is a dilation by the intersection of their structuring elements. According to our intuition about dilations, the theorem tells us that given δ_A and δ_B, the dilation $\mathbb{D}_{\{A,B\}}$ applied to an image X can be intuitively described as the image obtained by re-drawing X with (the brush) $A \cap B$. This is illustrated in Figure 3.4 by showing $\mathbb{D}_{\{A,B\}}$ applied to an image X. The result of $\mathbb{D}_{\{A,B\}}(X)$ was computed using the $O(n^2)$ procedure in Section 3.2.3.

We now devote the final part of this section to illustrate the application of the theory developed in previous sections to prove the above result.

3.3.6. Application: Proof of Theorem 10

We wish to prove that $\mathbb{D}_{\{A,B\}} = \delta_{A\cap B}$ for $A,B \subseteq M$. From Definition 24, we have $\delta_{A\cap B}(X) = X \oplus (A \cap B)$ and, from Corollary 5, we have $\mathbb{D}_{\{A,B\}}(X) = \bigcap_{Y\subseteq X}(Y \oplus A) \cup ((X \setminus Y) \oplus B)$. Therefore, it suffices to show that

$$X \oplus (A \cap B) = \bigcap_{Y \subseteq X} (Y \oplus A) \cup ((X \setminus Y) \oplus B)$$

for all $X \subseteq M$. Recall that the above equality is the distributivity equation for the Minkowski addition discussed in Equation 3.1. It is important also to recall the equation $X \oplus (A \cap B) = (X \oplus A) \cap (X \oplus B)$ *does not hold* in general. Nevertheless, it does if X is a singleton set as shown next.

Let us consider the *singleton case*: Suppose that X is an arbitrary set of the form $\{\mathbf{v}\}$. We obtain the following equations:

$$\begin{aligned}
\mathbb{D}_{\{A,B\}}(\{\mathbf{v}\}) &= \bigcap_{Y \subseteq X} (Y \oplus A) \cup ((X \setminus Y) \oplus B) \\
&= ((\emptyset \oplus A) \cup (\{\mathbf{v}\} \oplus B)) \cap ((\{\mathbf{v}\} \oplus A) \cup (\emptyset \oplus B)) \\
&= (\{\mathbf{v}\} \oplus B) \cap (\{\mathbf{v}\} \oplus A) \\
&= \{\mathbf{v} + \mathbf{w} \mid \mathbf{w} \in B\} \cap \{\mathbf{v} + \mathbf{w} \mid \mathbf{w} \in A\} \\
&= \{\mathbf{v} + \mathbf{w} \mid \mathbf{w} \in A \cap B\} \\
&= \delta_{A \cap B}(\{\mathbf{v}\}).
\end{aligned}$$

Now for the general case we will use the *continuity of space functions*. Suppose that X is an arbitrary set. From Definition 14 and Theorem 9 we know that $\mathbb{D}_{\{A,B\}}$ and $\delta_{A \cap B}$ are *space functions*. Furthermore from Proposition 4 it follows that space functions preserve arbitrary joins. Thus, with the help of this preservation of arbitrary joins (unions) and the singleton case above, we can obtain the desired result in a simple way; namely $\mathbb{D}_{\{A,B\}}(X) =$

$$\mathbb{D}_{\{A,B\}}\left(\bigcup_{\mathbf{v} \in X} \{\mathbf{v}\}\right) = \bigcup_{\mathbf{v} \in X} \mathbb{D}_{\{A,B\}}(\{\mathbf{v}\}) = \bigcup_{\mathbf{v} \in X} \delta_{A \cap B}(\{\mathbf{v}\}) = \delta_{A \cap B}\left(\bigcup_{\mathbf{v} \in X} \{\mathbf{v}\}\right) = \delta_{A \cap B}(X).$$

■

3.4. Summary

In this chapter we presented algorithms to compute distributed space functions $\mathbb{D}_I$ given a finite scs $(L, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$.

In Section 3.1, we characterized the size of the set of all space functions $\mathscr{S}(L)$. Namely, we established the cardinality $(n+1)^2 + n! \mathscr{L}_n(-1)$ for the lattice $\mathbf{M}_n$, $n^{\log_2 n}$ for power sets (boolean algebras) and $\binom{2n}{n}$ for linear orders.

The algorithms in Section 3.2 are based on theorems from Section 2.10. We computed the value $\mathbb{D}_I(c)$ in distributive lattices using Algorithm 1, and the whole function $\mathbb{D}_I$ using Algorithm 2 that is based on the

previous one. For arbitrary lattices we presented Algorithm 3 and Algorithm 4 that compute $\mathbb{D}_I$ approximating it from above. Algorithm 4 is the enhanced version of Algorithm 3. For these algorithms we computed their worst-time complexity and we presented some performance results.

Also, we provided an application of the theory developed in Chapter 2 to geometry and Mathematical Morphology. First, we recalled the notion of Minkowski addition of sets in vector spaces as an operation in the algebraic structure of modules (Section 3.3.1) and showed some of its basic properties (Sections 3.3.2 and 3.3.3). We then proved that given a module M, the structure $(\mathscr{P}(M), \subseteq, (\delta_S)_{S \subseteq M})$ is an scs where dilations are space functions (Section 3.3.4). Furthermore, we showed that erosions are space projections (Proposition 16). Since space functions and projections can also be viewed as the information a given agent sees, we then gave a natural epistemic interpretation of these MM operations as an agent's perception of a given image (Remark 9).

In Section 3.3.5 we proved that given two dilations δ_A and δ_B, the *distributed space function* of the group $\{A, B\}$ (i.e., $\mathbb{D}_{\{A,B\}}$) corresponds to the dilation $\delta_{A \cap B}$. Finally, in Section 3.3.6 we used the theory developed in previous sections to prove Theorem 10 and a novel law for $X \oplus (A \cap B)$: i.e., $X \oplus (A \cap B) = \bigcap_{Y \subseteq X} (Y \oplus A) \cup ((X \setminus Y) \oplus B)$.

Chapter 4

Specification of SCCP in Rewriting Logic

In Chapter 2 we have studied the theory of distributed information of a possibly infinite group. In Chapter 3 we provided algorithms to compute the distributed space function $\mathbb{D}_I$ for different lattices. In this chapter we study an specification of SCCP in rewriting logic, that is a logic for concurrency. Roughly speaking, scs specify the theory to represent spatial information in CCP; thus we shall provide a language to specify and analyze spatial CCP for modeling spatial behavior.

Different extensions of CCP have been proposed to capture aspects such as mobility [27, 12, 32], stochastic behavior [30], and temporal behavior [76, 75, 16, 31, 60] for timed and reactive computations, where processes can be constrained also by unit delays and time-out conditions.

However, all of them have a centralized notion of store (set of constraints), which implies that they are unsuitable to model systems where information and processes can be spatially distributed among certain groups of agents. An example of such systems is SCCP [51], that enhanced and generalized the theory of CCP for systems with spatial distribution of information. These systems allow computational hierarchical spaces that can be granted to agents. In SCCP, each space may have CCP processes and other (sub) spaces, processes can post and query information in their given space (i.e., locally), and may as well move from one space to another [32].

In this chapter we shall focus in an extension of SCCP to represent timed processes. We shall present the formal semantics of real-time SCCP in the form of a real-time rewriting logic semantics that is executable in the Maude system [13] with the help of rewriting modulo SMT: partial information (i.e., constraints) in the specification is represented by quantifier-free formulas on the shared variables of the system that are under the control of SMT decision procedures.

4.1. Preliminaries

The development of this chapter relies on notions of process calculi, concurrent constraint programming, and rewriting logic; all within the general scope of concurrency theory. Concurrency theory is the field of theoretical computer science concerned with the fundamental aspects of systems consisting of multiple computing agents that interact among each other. This covers a vast variety of systems including reactive systems.

4.1.1. Order-sorted Rewriting Logic

Rewriting logic [55] is a general semantic framework that unifies a wide range of models of concurrency. Language specifications can be executed in Maude [13], a high-performance rewriting logic implementation and benefit from a wide set of formal analysis tools available to it, such as an LTL model checker and an inductive theorem prover.

Definition 26 (Rewriting Logic)**.** A *rewriting logic specification* or *rewrite theory* is a tuple $\mathscr{R} = (\Sigma, E \uplus B, R)$ where:

- $(\Sigma, E \uplus B)$ is an order-sorted equational theory with $\Sigma = (S, \leq, F)$ a signature with finite poset of sorts $(S, \leq)$ and a set of function symbols F typed with sorts in S; E is a set of Σ-equations, which are universally quantified Horn clauses with atoms that are Σ-equations $t = u$ with t, u terms of the same sort; B is a set of structural axioms, disjoint from the set of equations E, (e.g., associativity, commutativity, identity) such that there exists a matching algorithm modulo B producing a finite number of B-matching substitutions or failing otherwise; and
- R a set of universally quantified conditional rewrite rules of the form

$$t \to u \textbf{ if } \bigwedge_i \phi_i$$

  where t, u are Σ-terms of the same sort and each ϕ_i is a Σ-equality.

Given $X = \{X_s\}_{s \in S}$, an S-indexed family of disjoint variable sets with each X_s countably infinite, the *set of terms of sort s* and the *set of ground terms of sort s* are denoted, resp., by $T_\Sigma(X)_s$ and $T_{\Sigma,s}$; similarly, $T_\Sigma(X)$ and T_Σ denote, resp., the set of terms and the set of ground terms. The expressions $\mathscr{T}_\Sigma(X)$ and $\mathscr{T}_\Sigma$ denote the corresponding order-sorted Σ-term algebras. All order-sorted signatures are assumed *preregular* [28], i.e., each Σ-term t has a unique *least sort* $ls(t) \in S$ s.t. $t \in T_\Sigma(X)_{ls(t)}$. It is also assumed that Σ has *nonempty*

sorts, i.e., $T_{\Sigma,s} \neq \emptyset$ for each $s \in S$. Many-sorted equational logic is the special case of order-sorted equational logic when the subsort relation $\leq$ is restricted to be the identity relation over the sorts.

Remark 10 (Induced Relations). An equational theory $\mathscr{E} = (\Sigma, E \uplus B)$ induces the *congruence relation* $=_{\mathscr{E}}$ on $T_\Sigma(X)$ defined for $t, u \in T_\Sigma(X)$ by $t =_{\mathscr{E}} u$ if and only if $\mathscr{E} \vdash t = u$, which denotes $\mathscr{E}$-provability by the deduction rules for order-sorted equational logic in [56]. The expressions $\mathscr{T}_{\mathscr{E}}(X)$ and $\mathscr{T}_{\mathscr{E}}$ (also written $\mathscr{T}_{\Sigma/E\uplus B}(X)$ and $\mathscr{T}_{\Sigma/E\uplus B}$) denote the quotient algebras induced by $=_{\mathscr{E}}$ on the term algebras $\mathscr{T}_\Sigma(X)$ and $\mathscr{T}_\Sigma$, resp.; $\mathscr{T}_{\Sigma/E\uplus B}$ is called the *initial algebra* of $(\Sigma, E \uplus B)$.

A rewrite theory $\mathscr{R} = (\Sigma, E \uplus B, R)$ induces a *rewrite relation* $\rightarrow_{\mathscr{R}}$ on $T_\Sigma(X)$ (sometimes denoted also as $\rightarrow_{R/E\uplus B}$) defined for every $t, u \in T_\Sigma(X)$ by $t \rightarrow_{\mathscr{R}} u$ if and only if there is a rule $(l \rightarrow r \textbf{ if } \phi) \in R$ and a substitution $\theta : X \rightarrow T_\Sigma(X)$ satisfying $t =_{E\uplus B} l\theta$, $u =_{E\uplus B} r\theta$, and $E \vdash \phi\theta$. The tuple $\mathscr{T}_{\mathscr{R}} = (\mathscr{T}_{\Sigma/E\uplus B}, \rightarrow^*_{\mathscr{R}})$ is called the *initial reachability model of* $\mathscr{R}$ [11]. ■

A *topmost rewrite theory* is a tuple $\mathscr{R} = (\Sigma, E \uplus B, R)$ with top sort *State*, i.e., no operator in Σ has *State* as argument sort and each rule $(l \rightarrow r \textbf{ if } \phi) \in R$ satisfies $l, r \in T_\Sigma(X)_{State}$ and $l \notin X$.

Definition 27 (Rewriting Logic Semantics)**.** Given a language $\mathscr{L}$, the rewriting logic semantics of $\mathscr{L}$ is a rewrite theory $\mathscr{R}_{\mathscr{L}} = (\Sigma_{\mathscr{L}}, E_{\mathscr{L}} \uplus B_{\mathscr{L}}, R_{\mathscr{L}})$ where $\rightarrow_{\mathscr{R}_{\mathscr{L}}}$ provides a step-by-step formal description of $\mathscr{L}$'s *observable* run-to-completion mechanisms.

In the above definition, $\Sigma_{\mathscr{L}}$ specifies the syntax of $\mathscr{L}$, the types and operators to represent, e.g., agents, processes and the store. Besides, $E_{\mathscr{L}} \uplus B_{\mathscr{L}}$ and $R_{\mathscr{L}}$ give definitions for the deterministic and concurrent features of $\mathscr{L}$, resp.

The conceptual distinction between equations and rules in $\mathscr{R}_{\mathscr{L}}$ has important consequences that are captured by rewriting logic's *abstraction dial* [57]. Setting the level of abstraction in which all the interleaving behavior of evaluations in $\mathscr{L}$ is observable, corresponds to the special case in which the dial is turned down to its minimum position by having $E_{\mathscr{L}} \uplus B_{\mathscr{L}} = \emptyset$. The abstraction dial can also be turned up to its maximal position as the special case in which $R_{\mathscr{L}} = \emptyset$, thus obtaining an equational semantics of $\mathscr{L}$ without observable transitions. The rewriting logic semantics presented in this work is *faithful* in the sense that such an abstraction dial is set at a position that exactly captures the interleaving behavior of the concurrency model.

Real time. *Time sampling strategies* offer alternatives to assign the time that a rewrite step needs to be applied. For instance, the maximal time sampling strategy advances time by the maximum possible time elapse and tries to advance time by a user-given time value in tick rules having other forms. There are two different kinds of tick rule applications that the maximal strategy can treat: (i) ticks from states from which

time can only advance up to a certain maximal time, and (ii) ticks from states from which time can advance by any amount. Here, the tick rule is the second one and the maximal time sampling strategy handles it by advancing time by a user-given time value.

A *time-robust* system is one where from any given state time can advance either by: (i) any amount, (ii) any amount up to (and including) a specific instant in time, or (iii) not at all. Advancing time is not affected unless in a specific state time is advanced all the way to the specific bound in time given in (ii). An instantaneous rewrite rule can only be applied at specific times, namely, when the system has advanced time by the maximal possible amount.

A time-robust system may have *Zeno paths*, those are paths where the sum of the durations of an infinite number of tick steps is bounded. It is necesary to differentiate between Zeno paths forced by the specification and Zeno paths that are due to bad choices in the tick increments. The intuition in the second type of Zeno behavior does not reflect realistic behaviors in the system and therefore is not simulated by the maximal time sampling strategy. The paths of the system that do not exhibit this unrealistic kind of Zeno behavior are called *timed fair paths*.

For systems satisfying time-robustness and tick-stabilizing properties unbounded and time-bounded LTL (excluding the *next* operator) model checking using the maximum time elapsed strategy is complete [64].

4.1.2. SMT Solving

Satisfiability Modulo Theories (SMT) studies methods for checking satisfiability of first-order formulas in specific models. The SMT problem is a decision problem for logical formulas with respect to combinations of background theories expressed in classical first-order logic with equality. An SMT instance is a formula ϕ (typically quantifier free, but not necessarily) in first-order logic and a model $\mathscr{T}$, with the goal of determining if ϕ is satisfiable in $\mathscr{T}$.

In this work, the representation of the constraint system is based on SMT solving technology. Given a many-sorted equational theory $\mathscr{E}_0 = (\Sigma_0, E_0)$ and a set of variables $X_0 \subseteq X$ over the sorts in Σ_0, the formulas under consideration are in the set $QF_{\Sigma_0}(X_0)$ of quantifier-free Σ_0-formulas: each formula being a Boolean combination of Σ_0-equation with variables in X_0 (i.e., atoms). The terms in $T_{\mathscr{E}_0}$ are called *built-ins* and represent the portion of the specification that will be handled by the SMT solver (i.e., semantic data types). In this setting, an SMT instance is a formula $\phi \in QF_{\Sigma_0}(X_0)$ and the initial algebra $\mathscr{T}_{\mathscr{E}_0^+}$, where $\mathscr{E}_0^+$ is a *decidable extension* of $\mathscr{E}_0$ such that

$$\phi \text{ is satisfiable in } \mathscr{T}_{\mathscr{E}_0^+} \quad \text{iff} \quad (\exists \sigma : X_0 \to T_{\Sigma_0})\ \mathscr{T}_{\mathscr{E}_0} \models \phi\sigma.$$

Many decidable theories $\mathscr{E}_0^+$ of interest are supported by SMT solvers satisfying this requirement (see [69] for details). In this work, the Maude alpha 118 release, which integrates Yices2 [21] and CVC4 [7], is used for reachability analysis with SMT constraints.

Maude [13]. It is a language and system based on rewriting logic. It supports order-sorted equational and rewrite theory specifications in *functional* and *system* modules, resp. Maude provides support for real-time rewrite theories [64] and built-ins as proposed in the rewriting modulo SMT approach [69].

4.2. Rewriting Logic Semantics

This section introduces the rtscc real-time rewriting logic semantics in the form of a real-time rewrite theory $\mathscr{R}$. The syntax of rtscc is given in the Maude functional module `SCCP-SYNTAX` that includes the notation for commands (processes), agents and time. All the constructors for system states are given in the system module `SCCP-STATE` and the transitions between states are defined by rewriting rules in the module `SCCP`.

4.2.1. System States

The tree-like structure of the hierarchical spaces is represented as a flat configuration of object-like terms encoding the state of execution of the agents. The hierarchical relationships among spaces are specified by common prefixes as part of an agent's name. In an observable state, each agent's space is represented by a set of object-like terms: some encoding the state of execution of all its processes and exactly one object representing its local store. The object-based system is represented using Maude's predefined module `CONFIGURATION` imported in the system module `SCCP-STATE`. The object and class identifiers are:

```
sort Sys .
subsorts Nat Aid < Oid .
ops agent process Simulation :    -> Cid .
op {_}         : Configuration    -> Sys .
```

The system states are represented by the topsort `Sys` with argument a configuration of objects containing the setup of each one of the agents in the system. A `Configuration` is a multiset of objects with set union denoted by juxtaposition and identity `none`. There are two types of object identifiers: agent identifiers (`Aid`) to represent agents and their hierarchical structure, and natural numbers (`Nat`) for some additional identification used internally in $\mathscr{R}$. There are three types of class identifiers (`Cid`): agents, processes, and a simulation object. The latter specifies the attributes required for the real-time simulation of the system, such as the global time and the scheduler.

Each agent has one attribute, its store, and each process has two attributes: an universal identifier (used internally for execution purposes) and the command (i.e., process) it is executing:

```
op store :_    : Boolean          -> Attribute [ctor] .
op Uid :_      : Nat              -> Attribute [ctor] .
op command :_  : SCCPCmd          -> Attribute [ctor] .
```

The syntax of commands is presented in Section 4.2.2. Section 4.2.3 explains how quantifier free formulas are used to represent constraints (as `Boolean`) and the entailment relation is encoded with the help of SMT-based decisions procedures.

Finally, the attributes of the simulation object include the global time (attribute `gtime`); the priority queue of the system commands to be processed, ordered by time-to-execution (attribute `pqueue`); the collection of pending commands, i.e., ask commands that are waiting for its guarding constraint to become activate (attribute `pend`); the counter for assigning the next internal identifier when spawning a new process (attribute `nextID`); a flag that is on whenever a tick rule needs to be applied (attribute `flg`); and a collection of maps containing the time it takes to process certain commands relative to the space where they are executed (attributes `MAsk`, `MTell`, `MSp`, and `MExt`). The sort `Time`, as it is often the case in Real-time Maude [63], can be used to represent either discrete or dense linear time, while `Ttime` is the name of the Maude view that is used to instantiate parameterized sorts with time:

```
op gtime :_    : Time             -> Attribute [ctor] .
op pqueue :_   : Heap{Tuple}      -> Attribute [ctor] .
op pend :_     : Heap{Tuple}      -> Attribute [ctor] .
op nextID :_   : Nat              -> Attribute [ctor] .
op flg :_      : Bool             -> Attribute [ctor] .
op MAsk :_     : Map{Aid, Ttime}  -> Attribute [ctor] .
op MTell :_    : Map{Aid, Ttime}  -> Attribute [ctor] .
op MSP :_      : Map{Aid, Ttime}  -> Attribute [ctor] .
op MExt :_     : Map{Aid, Ttime}  -> Attribute [ctor] .
```

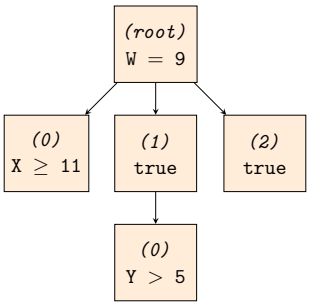
As mentioned before, the qualified identifiers of agents are used to encode the hierarchical structure of spaces (sort `Aid`). The root of any tree is denoted by constant `root` and any other qualified name corresponds to a dot-separated list natural numbers (sort `Nat`), organized from left to right:

```
op root : -> Aid .
op _._  : Nat Aid -> Aid .
```

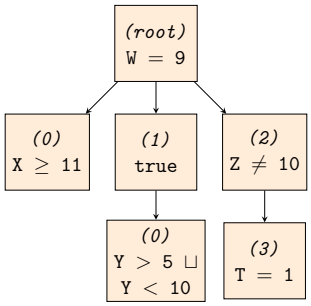
Example 20. In this syntax, the hierarchical structure depicted in Figure 4.1a can be specified as:

```
< root : agent | store : (W:Integer === (9).Integer) >
< 0 . root : agent | store : (X:Integer >= 11) >
< 1 . root : agent | store : true >
< 2 . root : agent | store : true >
< 0 . 1 . root : agent | store : (Y:Integer > 5) >
```

where, for instance, the agent with identifier `0.root` has local store $X \geq 11$.



(a) Initial state of the system.



(b) Final state of the system.

Figure 4.1: Hierarchical organization of spaces.

4.2.2. Commands

The following EBNF-like notation defines the process-like syntax of commands:

$$P ::= \mathbf{0} \mid \mathbf{tell}(c) \mid \mathbf{ask}(c) \to P \mid P \parallel P \mid [P]_i \mid \uparrow_i (P)$$

where c is a constraint and i an agent identifier. The $\mathbf{tell}(c)$ command posts the constraint c to the local store (once a constraint is added, it cannot be removed from the store so that the store grows monotonically). The command $\mathbf{ask}(c) \to P$ queries if c can be deduced from the information in the local store; if so, the process behaves like P, otherwise, it remains blocked until more information is added to the store. A basic CCP process like-language usually adds *parallel composition* ($P \parallel Q$) combining processes P and Q concurrently. The command $[P]_i$ indicates that command P must be executed inside the agent i's space: any information that P produces is available to other commands that execute within the same space. The command $\uparrow_i (P)$ denotes that P is to be run outside the space of agent i and the information posted by P is going to be stored in the parent of agent i. The `SCCP-SYNTAX` module includes the syntax of commands in `rtscc`:

```
op 0       :                  -> SCCPCmd .
op tell    : Boolean          -> SCCPCmd .
op ask_->_ : Boolean SCCPCmd -> SCCPCmd .
op _||_    : SCCPCmd SCCPCmd -> SCCPCmd [assoc comm gather (e E) ] .
op _in_    : SCCPCmd Nat     -> SCCPCmd .
op _out_   : SCCPCmd Nat     -> SCCPCmd .
```

4.2.3. Time Scaffolding

The real-time behavior in $\mathscr{R}$ associates timing behavior to those commands that interact with stores (i.e., tell and ask commands) and to commands that involve mobility among the space structure of the system (i.e., $[\_]_\_$ and $\uparrow\_(\_)$). More precisely, tell and ask commands can take time when posting and querying, resp., from a store. Moving the execution of a command inside an agent and extruding from a space can also take

up time. Such times are given by the time maps `MTell` (for *tell*), `MAsk` (for *ask*), `MSp` (for $\lfloor\_\rfloor\_$), and `MExt` (for $\uparrow\_(\_)$), and can be consulted using the `getTimeCmd` function. For example, `MTell(i)` denotes the time it takes to execute a tell command inside the agent's i space.

```
op fTime : Map{Aid, Ttime} Aid -> Time .
eq fTime(M:Map{Aid, Ttime}, L1)
 = if $hasMapping(M, L1) then M[L1] else 0 fi .

op getTimeCmd : Attribute Attribute Attribute Attribute SCCPCmd Aid -> Time .
eq getTimeCmd(MTell: MT, MAsk: MA, MIn: MI, MOut: MO, tell(B1), L1)
 = fTime(MT, L1) .
eq getTimeCmd(MTell: MT, MAsk: MA, MIn: MI, MOut: MO, C1 in I1, L1)
 = fTime(MI, L1) .
eq getTimeCmd(MTell: MT, MAsk: MA, MIn: MI, MOut: MO, C1 out I1, L1)
 = fTime(MO, L1) .
eq getTimeCmd(MTell: MT, MAsk: MA, MIn: MI, MOut: MO, C1, L1)
 = 0 [owise] .
```

The run-to-completion time of commands is simulated with the help of a leftist heap that keeps track of all the active commands that are waiting for the global timer to advance. One motivation to use leftist heaps is that insertion, removal, and querying are defined without the need of structural axioms, which may result in performance gains during execution.

***Leftist Heap* [62]** This structure is a heap-ordered binary tree that satisfies the *leftist property: The rank of any left child (i.e., the length of its rightmost path to a leaf) is at least as large as the rank of its right sibling.* Each entry in the heap is a pair (i,t) where i is an unique identifier of a process and t the time it needs to start executing. At the beginning, all the processes belong to the heap and they are ordered with respect to their execution time. A process is executed when its execution time is the minimum time of all the processes that are pending to complete their transitions. The leftist heap is implemented as a parameterized container in the functional module `LEFTIST-HEAP{X::STRICT-TOTAL-ORDER}`, with admissible parameters only being strict total orders:

```
sort Heap{X} NeHeap{X} .
subsort NeHeap{X} < Heap{X} .
op empty : -> Heap{X} [ctor] .
op T(_,_,_,_) : Nat X$Elt Heap{X} Heap{X} -> NeHeap{X} [ctor] .
op isEmpty : Heap{X} -> Bool .
eq isEmpty(empty)       = true .
eq isEmpty(T(Ra,E,L,R)) = false .
op rank    : Heap{X} -> Nat .
eq rank(empty)       = 0 .
eq rank(T(Ra,E,L,R)) = Ra .
op makeT   : X$Elt Heap{X} Heap{X} -> NeHeap{X} .
eq makeT(E,L,R)
 = if rank(L) >= rank(R)
   then T(rank(R) + 1,E,L,R)
   else T(rank(L) + 1,E,R,L) fi .
```

Heaps are constructed from the constant `empty` and the `T(_,_,_,_)` function symbol: Its first argument is the rank of the tree (sort `Nat`), the second one the label (sort `X$Elt`), and the third and fourth ones the left

and right children (sort `Heap{X}`), resp. In the semantics of rtscc, the sort `X$Elt` is instantiated with the sort of pairs of the form (I,T), where I is an internal process identifier and T is the run-completion time of such a process. Auxiliary operations include `isEmpty`, `rank`, and `makeT`, which are used to verify whether a heap is empty, compute the rank of a given heap, and create a heap out of two heaps, resp. Other key operations on leftist heaps are the merging of two heaps (function `merge`), inserting an element in a heap (function `insert`), removing an element (function `deleteMin`) and, finding the element at the top of a non-empty heap (function `findMin`).

```
op merge     : Heap{X} Heap{X} -> Heap{X} .
eq merge(empty, L) = L .
eq merge(L, empty) = L .
eq merge(T(Ra,E,L,R),T(Ra',E',L',R'))
 = if (E < E' or E == E')
   then makeT(E,L,merge(R,T(Ra',E',L',R')))
   else makeT(E',L',merge(T(Ra,E,L,R),R')) fi .
op insert    : X$Elt Heap{X} -> NeHeap{X} .
eq insert(E,L) = merge(T(1,E,empty,empty),L) .
op deleteMin : NeHeap{X} -> Heap{X} .
eq deleteMin(T(Ra,E,L,R)) = merge(L,R) .
op findMin   : NeHeap{X} -> X$Elt .
eq findMin(T(Ra,E,L,R)) = E .
```

4.2.4. The Constraint System

In this rewriting logic semantics, the sort `Boolean` (available in the current version of Maude from the `INTEGER` module) defines the data type used to represent rtscc's constraints. Terms of sort `Boolean` are quantifier-free formulas built from variables ranging over the Booleans and integer numbers, and the usual function symbols. The current version of Maude is integrated with the CVC4 [7] and Yices2 [21] SMT solvers, which can be queried via the meta-level. In this semantics, queries to the SMT solvers are encapsulated by functions `check-sat` and `check-unsat`:

```
op check-sat : Boolean -> Bool .
eq check-sat(B) = metaCheck(['INTEGER], upTerm(B)) .
op check-unsat : Boolean -> Bool .
eq check-unsat(B) = not(check-sat(B)) .
```

The function invocation `check-sat`(B) returns true only if B is satisfiable. Otherwise, it returns false if it is unsatisfiable or undefined if the SMT solver cannot decide. Note that function `check-unsat`(B) returns true only if B is unsatisfiable. Therefore, the rewriting logic semantics of rtscc instantiates the constraint system $(\mathbf{C},\sqsubseteq)$ by having quantifier-free formulas, modulo the semantic equivalence in $\mathscr{T}_{\mathscr{E}_0^+}$ (i.e., the model implemented in the SMT solver extending the initial model $\mathscr{T}_{\mathscr{E}_0}$), as the constraints $\mathbf{C}$ and semantic validity relative to $\mathscr{T}_{\mathscr{E}_0^+}$ as the entailment relation $\sqsubseteq$. More precisely, if Γ is a finite set of terms of sort `Boolean` (representing all the constraints in a given store) and ϕ is a term of sort `Boolean` (constraint), by defining

$\psi = \bigwedge_{\gamma \in \Gamma} \gamma$, the following equivalence holds[1]:

$$\psi \sqsupseteq \phi \quad \text{iff} \quad \mathscr{T}_{\mathscr{E}_0^+} \models \psi \Rightarrow \phi \quad \text{iff} \quad \texttt{check-unsat}\,(\psi \wedge \neg\phi)\,.$$

In order to make a direct relation between the entailment relation $\sqsubseteq$ and the Maude syntax, the operator `entails` is defined as follows:

```
op entails : Boolean Boolean -> Bool .
eq entails(C1:Boolean, C2:Boolean)
 = check-unsat(C1:Boolean and not(C2:Boolean)) .
```

4.2.5. System Transitions

The tick rule models time elapse in the system [63]:

```
crl [tick] :
  { X < I : Simulation | pqueue : P, gtime : T, flg : true,
            pend : P0, Atts > }
 => { X < I : Simulation | pqueue : merge(delta(deleteMin(P),T0),P0),
              gtime : (T plus T0), flg : false, pend : empty, Atts > }
 if T0 := p2(findMin(P)) .
```

where the auxiliary operation `delta` reduce `T0` units the execution time of every command in the heap `P`:

```
op delta : Heap{Tuple} Time -> Heap{Tuple} .
eq delta(empty,T') = empty .
eq delta(T(N,((I,T)),P,P0),T')
 = T(N,((I,T monus T')),delta(P,T'),delta(P0,T')) .
```

When the `[tick]` rule is fired, the global time is incremented in `T0` units, where `T0` is the minimum time present in the priority queue `P`, which is modified by removing the process with the minimum execution time. It also adds the pending commands to the priority queue. The pending commands are *ask* commands that have already been activated for execution, although they have not been able to execute because their guards have not been met by the state of their corresponding local stores. The tick rule puts all these pending process back in the main queue, so that their guards can be checked again and be executed or put back in the pending queue. Figure 4.2 depicts the possible transitions an *ask* command can take between being in the priority queue, in the pending queue, and finally executing. The rules `[ask]` and `[delay]` are introduced below.

The invisible transitions of the semantics are specified with the help of equational rules. Namely, one for removing a `0` command from a configuration and another one to join the contents of two stores of the same space (i.e., two stores with the same `Aid`). The latter type of transition is important because when a new process is spawned in an agent's space, a store with the empty constraint (i.e., `true`) is created for

[1] Recall that $\sqcup$ and $\sqsubseteq$ correspond to conjunction and logical implication, resp.

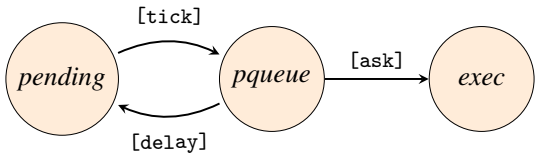


Figure 4.2: Possible transitions for ask commands.

that space. If such a space existed before, then the idea is that the newly created store is subsumed by the existing one. Note that neither of the invisible transitions takes time, i.e., they are really instantaneous, and they axiomatize structural properties of commands.

```
eq { < L0: process | command: 0, Atts > X }
 = { none X } .
eq < L0: agent | store: B0 > < L0: agent | store: B1 >
 = < L0: agent | store: (B0 and B1) > .
```

The following six rules capture the concurrent observable behavior in $\mathscr{R}$:

```
crl [tell]:
    { < L0: agent | store: B0 >
      < L0: process | UID: I0, command: tell (B1) >
      < I: Simulation | pqueue: H, flg: false, pend: P, Atts > X }
 => { < L0: agent | store: (B0 and B1) >
      < I: Simulation | pqueue: H, flg: true, pend: P, Atts > X }
 if I0 == p1(findMin(H)) .

crl [parallel]:
    { < L0: process | UID: I0, command: (C0 || C1) >
      < I: Simulation | pqueue: H, nextID: N, flg: false, pend: P,
        MTell: MT, MAsk: MA, MIn: MI, MOut: MO, Atts > X }
 => { < L0: process | UID: N, command: C0 >
      < L0: process | UID: (N + 1), command: C1 >
      < I: Simulation | pqueue: H, nextID: (N + 2), flg: true,
        pend: H0, MTell: MT, MAsk: MA, MIn: MI, MOut: MO, Atts > X }
 if I0 == p1(findMin(H))
 /\ H0 := insert(((N, getTimeCmd(MTell: MT, MAsk: MA, MIn: MI, MOut: MO, C0, L0))),
          insert(((N + 1, getTimeCmd(MTell: MT, MAsk: MA, MIn: MI, MOut: MO, C1, L0))), P)) .

crl [space]:
    { < L0: process | UID: I0, command: (C0 in N0) >
      < I: Simulation | pqueue: H, nextID: N, flg: false, pend: P,
        MTell: MT, MAsk: MA, MIn: MI, MOut: MO, Atts > X }
 => { < N0 . L0: agent | store: true >
      < N0 . L0: process | UID: N, command: C0 >
      < I: Simulation | pqueue: H, flg: true, pend: H0, nextID: (N+1),
        MTell: MT, MAsk: MA, MIn: MI, MOut: MO, Atts > X }
 if I0 == p1(findMin(H))
 /\ H0:= insert(((N, getTimeCmd(MTell: MT, MAsk: MA, MIn: MI, MOut: MO, C0, L0))), P) .

crl [extrussion]:
    { < N0 . L0: process | UID: I0, command: (C0 out N0) >
      < I: Simulation | pqueue: H, nextID: N, flg: false, pend: P,
        MTell: MT, MAsk: MA, MIn: MI, MOut: MO, Atts > X }
 => { < L0: process | UID: N, command: C0 >
      < I: Simulation | pqueue: H, flg: true, pend: H0, nextID: (N+1),
        MTell: MT, MAsk: MA, MIn: MI, MOut: MO, Atts > X }
 if I0 == p1(findMin(H))
 /\ H0:= insert(((N, getTimeCmd(MTell: MT, MAsk: MA, MIn: MI, MOut: MO, C0, N0 . L0))), P) .

crl [ask]:
    { < L0: agent | store: B0 >
```

```
      < L0: process | UID: I0, command: (ask B1 -> C1) >
      < I: Simulation | pqueue: H, flg: false, pend: P, nextID: N,
        MTell: MT, MAsk: MA, MIn: MI, MOut: MO, Atts > X }
 => { < L0: agent | store: B0 >
      < L0: process | UID: N, command: C1 >
      < I: Simulation | pqueue: H, flg: true, pend: H0, nextID: (N+1),
        MTell: MT, MAsk: MA, MIn: MI, MOut: MO, Atts > X }
 if I0 == p1(findMin(H))
 /\ entails(B0,B1)
 /\ H0:= insert(((N,getTimeCmd(MTell: MT, MAsk: MA, MIn: MI, MOut: MO, C1, L0) plus alpha(MA, L0))), P) .

crl [delay]:
    { < L0: agent | store: B0 >
      < L0: process | UID: I0, command: (ask B1 -> C1) >
      < I: Simulation | pqueue: H, pend: P, Atts > X }
 => { < L0: agent | store: B0 >
      < L0: process | UID: I0, command: (ask B1 -> C1) >
      < I: Simulation | pqueue: deleteMin(H),
                        pend: insert(((I0,T1)),P),Atts > X }
 if I0 == p1(findMin(H))
 /\ not(entails(B0,B1))
 /\ T1:= (p2(findMin(H))) .
```

The `[tell]` rule implements the execution semantics of a `tell` command by posting the given constraint in the local store and by removing such a command from the configuration. The `[parallel]` rule implements the semantics for parallel composition of process by spawning the two process in the current space. Rule `[space]` creates a new agent's space denoted by `N0.L0`, with an empty store (i.e., `true`), beside the execution of program `C0` within the new agent's space. The `[extrusion]` rule executes `C0` in the parent's space `L0`. The `[ask]` rule executes a command `C1` when the guard `B1` in `ask B1 -> C1` holds: that is, when `B1` is a semantic consequence of the contents `B0` of the current local store. Notice that the semantic consequence relation of the constraint system is queried by asking the SMT solver. The `[delay]` rule represents the negative answer for the ask rule: whenever `B0` does not entail `B1` or `B1` is not a semantic consequence of the contents `B0` of the current store. The `[delay]` rule moves an ask command into the pending heap, where it will remain until the `tick` rule executes again.

Remark 11 (Time-robustness). The real-time rewrite theory presented in this work is *time-robust* (see Section 4.1.1), namely: (i) in any given state, time can advance either any amount up to a specific instant in time or not at all; and (ii) instantaneous rules (i.e., those that are not the tick rule and take zero time) can only be applied when the system has advanced the maximal possible amount of time before any timed action can become enabled. Under these two assumptions and by using the maximal time sampling strategy, unbounded and time-bounded search and model checking are sound and complete with respect to *timed fair paths* [64]. They exclude paths with an infinite sequence of tick steps where, at each step, time could have advanced to time r (the duration of the first step in a path) or beyond, but with a total path duration less than r. Also are excluded those 'unfair' paths containing an infinite and consecutive sequence of 0-time ticks over a state on which an instantaneous rule can be applied. Note that a time-robust system may have

Zeno paths, where the sum of the durations of an infinite number of tick steps is bounded. By restricting the computations to time-bounded prefixes only a finite set of states can be reached from an initial state, so that the target real-time specification does not exhibit any Zeno behavior and temporal properties can be model checked. ■

4.3. Reachability Analysis

The goal of this section is to explain how the rewriting logic semantics $\mathscr{R}$ of rtscc and rewriting modulo SMT can be used as an automatic mechanism for solving existential reachability goals in the initial model $\mathscr{T}_{\mathscr{R}}$. This approach can be useful for symbolically proving or disproving real-time safety properties of $\mathscr{T}_{\mathscr{R}}$. The approach presented in this section mainly relies on Maude's `search` command, which explores reachable states looking for a violation of the specified property and if found returns a witness of the failure.

The examples presented in this section use the following time functions for the processes:

```
MAsk  : root |-> 1/20, (0 . root) |-> 1/10, (0 . 1 . root) |-> 3/20,
        (1 . root) |-> 1/10, (2 . root) |-> 1/10
MTell : root |-> 1/10, (0 . root) |-> 3/20, (0 . 1 . root) |-> 1/5,
        (1 . root) |-> 3/20, (2 . root) |-> 3/20
MSP   : root |-> 1/2, (0 . root) |-> 7/10, (0 . 1 . root) |-> 4/5,
        (1 . root) |-> 13/20, (2 . root) |-> 3/5
MExt  : root |-> 1/2, (0 . root) |-> 13/20, (0 . 1 . root) |-> 1,
        (1 . root) |-> 1/2, (2 . root) |-> 3/5
```

For example, querying the store at `root`'s space takes 1/20 time units.

Fault tolerance. This property ensures a system to continue operating properly in the event of the failure; *consistency* means that a local failure does not propagate to the entire system. In $\mathscr{R}$, this means that if a store becomes inconsistent, it is not the case that such an inconsistency spreads to the entire system. Of course, inconsistencies can appear in other stores due to some unrelated reasons.

Searching an inconsistent store can be easily implemented with the help of $\mathscr{R}$ and Maude's `search` command. The answer of this command in the positive would mean that from some initial state, there is a state in which a store becomes inconsistent at some point of execution within a given time interval. Taking advantage of $\mathscr{R}$ and the rewriting modulo SMT approach, also is possible to know when a store is inconsistent. As an example, consider the initial configuration in Figure 4.1a and the following `search` command with a the time interval $[0, 1.5)$:

```
search in SCCP :
    { < root : agent | store : (W:Integer === (9).Integer) >
      < 0 . root : agent | store : (X:Integer >= 11) >
      < 0 . 1 . root : agent | store : (Y:Integer > 5) >
      < 1 . root: agent | store: true > < 2 . root: agent | store: true >
      < root : process | UID : 1, command : ((((ask X:Integer > 2 ->
```

```
        (tell(Y:Integer < 10) in 0 in 1 out 0)) in 0)
        || ((tell(Z:Integer =/== (10).Integer)
        || (tell(T:Integer === 1) in 3)) in 2))
        || (tell(X:Integer <= 10) in 0)) >
      < 1 : Simulation | gtime : 0,pqueue : T(1,((1,0)),empty,empty),
        pend : empty,nextID : 19, flg : false, ... (time maps)}
=>* { < 1 : Simulation | gtime : T:Time, Atts:AttributeSet >
      < A:Aid : agent | store : B0:Boolean > C:Configuration }
    such that check-unsat(B0:Boolean) and T < 3/2 .
```

Note that a store is inconsistent if it is unsatisfiable, thereby checking whether a store is inconsistent is accomplished with the function `check-unsat`. The aforementioned command searches for an inconsistent store during the first 1.5 units of time of the system's execution. This command does not find an inconsistent store between the first 1.5 units of time in any of the 56 reachable states. The configuration of the system up to this point is depicted in Figure 4.1b.

However, it could be the case that a store results inconsistent during execution. For instance, the addition of the process `tell(X <= 10) in 0` in the previous command leads to an inconsistency in the space `0 . root`. The output for the `search` command is the following:

```
Solution 1 (state 159)
states: 160  rewrites: 16666 in 876ms cpu (875ms real)
        (19025 rewrites/second)
C:Configuration --> < root:agent | store:(W:Integer === (9).Integer) >
  < 0 . 1 . root : agent | store : (Y:Integer > 5) >
  < 1 . root:agent | store:true > < 2 . root:agent | store:true >
  < 2 . root:process | UID: 28,
                       command: tell(Z:Integer =/== (10).Integer) >
  < 2 . root:process | UID: 27,
                       command: (tell(T:Integer === (1).Integer) in 3) >
  < 0 . root:process | UID : 24,
                       command: (tell(Y:Integer < 10) in 0 in 1 out 0) >
A:Aid --> 0 . root
B0 --> X:Integer >= (11).Integer and X:Integer <= (10).Integer
Atts --> pqueue : T(2,(25,1/20),T(1,(24,4/5),empty,empty),T(1,(28,1/10),
  T(1,(27,3/5),empty,empty),empty)),pend : empty,nextID : 29,
  flg : true, ... (time maps)
T --> 1/2
```

There are 238 reachable states (from the initial state) and 74 of them have an inconsistent store between the first 1.5 units of time. The first inconsistency appears in 0.5 time units, and the last one in 1.3 times units. Notice that, the system continues evolving even though there is an inconsistency.

Knowledge inference. It refers to acquiring new knowledge from existing facts. In the setting of $\mathscr{R}$, this means that from a given initial state an agent, at some point, has gained enough information to infer new facts. A positive answer to such a query, means that from some initial state, at some moment during execution, there is at least one agent that has gained enough information to infer the given facts. As an example, consider the system in Figure 4.1a and the following `search` command:

```
search in SCCP :
    { < root : agent | store : (W:Integer === (9).Integer) >
```

```
      < 0 . root : agent | store : (X:Integer >= 11) >
      < 0 . 1 . root : agent | store : (Y:Integer > 5) >
      < 1 . root : agent | store : true >
      < 2 . root : agent | store : true >
      < root : process | UID : 1, command : (((ask X:Integer > 2 ->
        (tell(Y:Integer < 10) in 0 in 1 out 0)) in 0)
        || ((tell(Z:Integer =/== (10).Integer)
        || (tell(T:Integer === 1) in 3)) in 2)) >
      < 1 : Simulation | gtime : 0,pqueue : T(1,((1,0)),empty,empty),
        pend : empty, nextID : 19, flg : false, ... (time maps) > }
=>* { < 1 : Simulation | gtime : T:Time, Atts:AttributeSet >
      < A:Aid : agent | store : B0:Boolean > C:Configuration }
    such that entails(B0:Boolean, Y:Integer < 15) and T:Time > 0 and T:Time < 2 .
```

It checks if there is a state, reachable from the given initial state, in which some store logically implies $Y < 15$ in the time interval $(0, 2)$. This command does not find a store with enough information in such time interval in any of the 56 reachable states. However, if the time interval in the command is changed to $(2, 3)$ the query finds two solutions:

```
Solution 1 (state 54)
states: 55  rewrites: 7466 in 360ms cpu (358ms real)
                      (20738 rewrites/second)
C:Configuration --> < root:agent | store:(W:Integer === (9).Integer) >
  < 0 . root : agent | store : (X:Integer >= 11) >
  < 1 . root : agent | store : true >
  < 2 . root : agent | store : (Z:Integer =/== (10).Integer) >
  < 3 . 2 . root : agent | store : (T:Integer === (1).Integer) >
A:Aid --> 0 . 1 . root
B0:Boolean --> Y:Integer > (5).Integer and Y:Integer < (10).Integer
Atts:AttributeSet --> pqueue : T(1,(29,1/10),empty,empty),pend : empty,
  nextID : 30,flg : true, ... (time maps)
T:Time --> 5/2

Solution 2 (state 55)
states: 56  rewrites: 7601 in 368ms cpu (366ms real)
                      (20654 rewrites/second)
C:Configuration --> < root:agent | store:(W:Integer === (9).Integer) >
  < 0 . root : agent | store : (X:Integer >= 11) >
  < 1 . root : agent | store : true >
  < 2 . root : agent | store : (Z:Integer =/== (10).Integer) >
  < 3 . 2 . root : agent | store : (T:Integer === (1).Integer) >
A:Aid --> 0 . 1 . root
B0:Boolean --> Y:Integer > (5).Integer and Y:Integer < (10).Integer
Atts:AttributeSet --> pqueue : empty,pend : empty,nextID : 30,
  flg : false, ... (time maps)
T:Time --> 13/5
...
```

4.4. Summary

This chapter presented the real-time rewriting logic semantics for spatial concurrent constraint programming (rtscc) and its implementation in the Maude system. In this setting, time attributes are associated to process-store interaction, as well as to process mobility in the space structure.

We began this chapter with the neccesary background in rewriting logic, real-time specifications and SMT solving (Section 4.1). In Section 4.2 we presented the rewriting logic semantics for rtscc. Namely, we

provided the states (Section 4.2.1), commands (Section 4.2.2) and transitions of the system (Section 4.2.5). The real-time modeling is supported for both discrete and dense linear time. The scheduler is represented as a leftist heap which provides straighforward methods to manage the processes to be executed (Section 4.2.3). The constraint system is materialized with the help of rewriting modulo SMT representing the constraints by quantifier-free formulas on the shared variables of the system that are under the control of SMT decision procedures (Section 4.2.4).

Finally, reachability analysis in presented in Section 4.3. Some examples of existential properties such as fault tolerance and knowledge inference are included.

Related Work

We present individually the related work of each chapter.

Chapter 2: Distributed information. The closest related work is that of [51] (and its extended version [52]) which introduces spatial constraint systems (scs) for the semantics of a spatial ccp language. Their work is confined to a finite number of agents and to reasoning about agents individually rather than as groups. We added the continuity requirement to the space functions of [51] to be able to reason about possibly infinite groups. In [32, 35, 39] scs are used to reason about beliefs, lies and other epistemic utterances but also restricted to a finite number of agents and individual, rather than group, behavior of agents.

This work is inspired by the epistemic concept of distributed knowledge [23]. Knowledge in distributed systems was discussed in [42], based on interpreting distributed systems using Hintikka's notion of possible worlds. In this definition of distributed knowledge, the system designer ascribes knowledge to processors (agents) in each global state (a processor's local state). In [43] the authors present a general framework to formalize the knowledge of a group of agents, in particular the notion of distributed knowledge. The authors consider distributed knowledge as knowledge that is distributed among the agents belonging to a given group, without any individual agent necessarily having this knowledge.

In [44] the authors study knowledge and common knowledge in situations with infinitely many agents. The authors highlight the importance of reasoning about infinitely many agents in situations where the number of agents is not known in advance. Their work does not address distributed knowledge but points out potential technical difficulties in their future work. In the realm of Economics and game theory, models of infinitely many agents are used to discover mass phenomena that do not necessarily occur in the case of a fixed finite number of agents [17, 45, 47, 54]. Also infinite sets of agents are used in game theory [6, 25]. For example for games played with two teams, we may want to specify that everyone in a team knows that everyone in the other team knows a given proposition, regardless of the team size. This could be naturally specified with infinitely many agents.

There are other group phenomena that are closely related to the group phenomena here studied. In partic-

ular, group polarization [22, 2], from social sciences, and group improvisation [73], from computer music. Group polarization refers to the natural tendency of a group to make more extreme decisions than their individuals. Group improvisation involves constraining musical pattern variation choices of a participant according to choices made by others in the group. We plan to study these phenomena in future work.

Chapter 3: Computing Distributed Space Functions. In [40] a bit-vector representation of a lattice is discussed. This work gives algorithms of logarithmic (in the size of the lattice) complexity to compute join and meet operations. These results count bit-vector operations. From [8] we know that $\mathscr{S}(L)$ is isomorphic to the downset of $(P \times P^{op})$, where P is the set of join-prime elements of L, and that this, in turn, is isomorphic to the set of order-preserving functions from $(P \times P^{op})$ to $\mathbf{2}$. Thus, for the purpose of computing $\mathbb{D}_I$, we get bounds $O(m \log_2(2^{(n^2)}) = O(mn^2)$ for set lattices and $O(m(\log_2 n)^2)$ for powerset lattices where $n = |L|$ and $m = |I|$. This, however, assumes a bit-vector representation of a lattice isomorphic to $\mathscr{S}(L)$. Computing this representation takes time and space proportional to the size of $\mathscr{S}(L)$ [40] which could be exponential as stated in Section 3.1.

The lattice $\mathscr{S}(L)$ have been studied in [29]. The authors showed that a finite lattice L is distributive iff $\mathscr{S}(L)$ is distributive. A lower bound of $2^{2n/3}$ for the number of monotonic self-maps of any finite poset L is given in [19]. Nevertheless to the best of our knowledge, no other authors have studied the problem of determining the size $\mathscr{S}(L)$ nor algorithms for computing $\mathbb{D}_I$.

Complete lattices have been used as a framework to define morphological operators specifically to study grey-level images [72]. In this context, dilations and erosions are defined as operators that preserve arbitrary suprema and infima, resp. This proposal is a generalization of what we studied in Section 3.3 where we present dilations and erosions by some *structuring element*. As a novelty, we proposed the scs $(\mathscr{P}(M), \subseteq, (\delta_S)_{S \subseteq M})$ where M is a module and $(\delta_S)_{S \subseteq M}$ are dilations defined on the cs $(\mathscr{P}(M), \subseteq)$. It allowed us to apply our theoretical results to prove MM properties, e.g., that dilations and erosions form a Galois connection (Proposition 16). Also, we provided the interpretation of distributed information for images. Namely, we showed that given two dilations δ_A and δ_B, the greatest dilation below them is exactly $\mathbb{D}_{\{A,B\}}$ which in turn equals $\delta_{A \cap B}$ (Theorem 10). As a future work, we plan to explore these results for grey-scale images.

Chapter 4: Specification in Rewriting Logic. An extension of concurrent constraint programming in [76] presents a timed asynchronous computation model and propose an implementation using loop-free deterministic finite automata, a declarative framework for reactive systems where time is represented as discrete time

units. More recently, J. A. Pérez and C. Rueda [65] propose an operational semantics based on probabilistic automaton, extending the work in [76], with probabilistic and non-deterministic choices for processes. The inclusion of stochastic information for processes proposed by J. Aranda et al. in [3] associates to each computation a random variable determining its time duration: given a set of competing actions, the fastest action is executed, that is, the one with the shortest duration. Also, G. Sarria and C. Rueda [79] present a real-time extension of CCP with application to music interaction.

In the realm of rewriting logic, Degano et al. [18] provide a rewriting logic semantics for Milner's CCS with interleaving behavior. Additionally, a set of axioms is defined for a logical characterization of the concurrency of CCS processes. In [85], the authors use rewriting logic to represent the semantics of CCS and a modal logic for describing local capabilities of CCS processes. In particular, they study how to make executable the SOS semantics of CSS and present a fully executable specification of the semantics. More recently, M. Romero and C. Rocha [71] have proposed a symbolic rewriting logic semantics of the spatial modality of CCP with extrusion.

Conclusions

This dissertation has provided both theoretical and practical results. In the following, we present the main achievements of this work.

Chapter 2: Distributed information. We have introduced an algebraic theory for reasoning about possibly infinite groups of agents. We have defined the distributed information of a group as the greatest space function below all the space functions that represent the spaces of the agents in the group (Section 2.3). This allows us to represent such distributed information as a virtual space owned by the group. We stated that group projections and distributed spaces form a Galois connection (Proposition 8), that is, there is an equivalent relation between the distributed information of a group w.r.t a given constraint and the information that the group has w.r.t such constraint. Additionally, we proved (Section 2.5) that distributed information can be interpreted as distributed knowledge in Aumann and Kripke structures, which are representative models for epistemic (group) reasoning.

We pursued the study of conditions under which a piece of information derived by the combined local information of an infinite group of agents could be derived by some finite subgroup of those agents. We showed that this is the case when the information inferred by the group from a given supplied piece of information, is itself compact, and derivable from the combination of what each agent can infer from it in the local space (Theorem 2). We further showed, however, that compactness does not hold in general without those conditions (Theorem 3).

Furthermore, we showed a fundamental result that provides a way to compute, for completely distributive lattices, the greatest information that can infer some other given piece of information, and is below all possible combinations of local informations deriving that piece in the spaces of some given group of agents (Theorem 4). We had also stated some properties relating the information of a group of agents w.r.t the information of subgroups of those agents (Theorem 5).

Chapter 3: Computing Distributed Space Functions. We have shown that given a lattice L of size n and a set $\{\mathfrak{s}_i \mid i \in I\} \subseteq \mathscr{S}(L)$ of size m, $\mathbb{D}_I$ can be computed in the worst-case in $O(n + m\log n)$ binary lattice operations for powerset lattices, $O(mn^2)$ for lattices of sets, and $O(nm + n^3)$ for arbitrary lattices. We illustrated the experimental performance of the algorithms and an application in mathematical morphology.

We have stated the cardinality of the set of space functions $\mathscr{S}(L)$ for significant families of lattices. To the best of our knowledge, it is a novelty to establish the cardinality $(n+1)^2 + n!\mathscr{L}_n(-1)$ for the lattice $\mathbf{M}_n$. The cardinalities $n^{\log_2 n}$ for power sets (boolean algebras) and $\binom{2n}{n}$ for linear orders can also be found in the lattice literature [8, 48, 74]. Here we presented our original proofs for these statements.

We have provided efficient algorithms for computing the distributed space $\mathbb{D}_I$ of a given group I. In general, these results are important, as algebraic structures consisting of a lattice and space functions (join-endomorphisms) are very common in mathematics and computer science.

Finally, we used the developed theory (Chapter 2) to investigate applications in mathematical morphology (MM). In this domain, two fundamental operations, dilation and erosion, provide ways to perform geometric transformations, in particular within the realm of image processing based on so-called structuring elements. We considered these MM operations, generalized with Minkowski addition over modules, and used the theory to derive some interesting distribution properties. We also gave the interpretation of maps in group spatial constraint systems as MM operations over structuring elements and showed that the maximum map under two given group distribution maps (seen as dilations) corresponds to the dilation over the intersection of their structuring elements. In so doing we provided a proof that erosion and dilations, for structuring elements that are modules, form a Galois connection (Proposition 16). This allowed us to prove that the operation of erosion corresponds to our defined operation of projection of information into the spaces of the structural element (Definition 16). We also discussed an interpretation of dilations and erosions as an epistemic agent's perception of a given image (Remark 9).

Chapter 4: Specification in Rewriting Logic. We have presented a real-time rewriting logic semantics for spatial concurrent constraint programming (rtscc) that is fully executable in the Maude system. This setting allows the specification of the main characteristics of SCCP such as spatial information, hierarchical spaces, partial information and mobility. The configurations that we defined to represent the system states can be thought of as an extension of the basic SCCP configurations $\langle P, c\rangle$ where P is a process and c is a constraint. We included real-time behavior where the scheduler is constructed by using a leftist heap which is a data structure that provides useful definitions of operations for insertion, removal, and querying. We found it convenient for the implementation and it may result in performance gains during execution.

Besides, other models of rtscc are spatially-distributed multi-agent reactive systems that may have different computing capabilities and be subject to real-time requirements. In this approach, time attributes are associated to process-store interaction, as well as to process mobility in the space structure, by means of maps from agents to non-negative real quantities. Details about the underlying constraint system have been given as materialized with the help of rewriting modulo SMT. Furthermore, examples of reachability analysis performed on this semantics have been given to illustrate certain aspects of the timing behavior of agents distributed across hierarchical spaces, such as fault-tolerance and knowledge inference.

Future Work

This dissertation develops the theory of distributed information, algorithms to compute it, and a specification of sccp in rewriting logic, we present future work on each issue separately.

Chapter 2: Distributed information. Some of the proofs of the main results of this work rely on the completeness of the lattice of the underlying constraint system (Theorem 4). As future work we would like to generalise our results to the directed-complete partial orders, the central semantic structure of domain theory [26]. In fact, Scott-continuity is a central concept of our theory, hence this research direction seem promising.

We plan to extend our work with a process calculus with dynamic creation/removal of agents. The idea is to investigate situations where we could prove that it is not possible to reach a state where $\mathbb{D}_A(e)$ holds, with A as the set of all possible agents and e is some sensible information. This could be interpreted as saying that the agents will never be able to derive e by pooling or joining their own information. Along the same lines, for meaningful scs such as Kripke scs with infinitely many agents, we would like to study, the decision and complexity problem of whether $c \sqsupseteq \mathbb{D}_I(e)$ given c, e and infinite set I under the conditions of our compactness result (Theorem 2). Notice that our compactness result, does not provide us with a bound on the size of the finite subset $J \subset I$ such that $d \sqsupseteq \mathbb{D}_I(e)$.

Chapter 3: Computing Distributed Space Functions. As future work we plan to explore in detail the applications of our work in mathematical morphology for gray-scale images and computer music [73]. Furthermore, in the same spirit of [49] we have developed algorithms to generate distributive and arbitrary lattices. In our experiments, we observed that for every generated lattice L of size n, $n^{\log_2 n} \leq |\mathscr{S}(L)| \leq (n+1)^2 + n!\mathscr{L}_n(-1)$ holds, and in particular, if the generated lattice was distributive, $n^{\log_2 n} \leq |\mathscr{S}(L)| \leq \binom{2n}{n}$ holds. We plan to establish if these inequalities hold for every finite lattice.

Also, we plan to enhance our algorithms using notions from lattice theory, such as join-irreducible elements and modular lattices. The idea is that we can define space functions (join-endomorphisms) by

taking care just on the join-irreducible elements of a lattice. Regarding modular lattices, we plan to propose new algorithms taking advantage of the properties of these lattices that are less restrictive than distributive lattices.

Chapter 4: Specification in Rewriting Logic. The real-time rewriting logic semantics for rtscc presented in this work can be generalized to specify distributed information of groups. In due course, we plan to explore case studies on other key aspects of spatially distributed concurrent processes such as privacy and also include probability to represent, for example, random inclusion of processes during execution.

Other direction to pursue is a more general and fully symbolic rewriting logic semantics for rtscc where time information can also be modeled as shared variables under the control of the SMT decision procedures. In such a setting, interesting properties of real-time systems such as missed deadlines and deadlocks could be fully analyzed, e.g., for infinitely many initial states in a system.

Bibliography

[1] Samson Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors. *Handbook of Logic in Computer Science (Vol. 3): Semantic Structures*. Oxford University Press, Inc., USA, 1995.

[2] Mário S. Alvim, Sophia Knight, and Frank D. Valencia. Toward a Formal Model for Group Polarization in Social Networks. In Mário S. Alvim, Konstantinos Chatzikokolakis, Carlos Olarte, and Frank Valencia, editors, *The Art of Modelling Computational Systems: A Journey from Logic and Concurrency to Security and Privacy - Essays Dedicated to Catuscia Palamidessi on the Occasion of Her 60th Birthday*, volume 11760 of *Lecture Notes in Computer Science*, pages 419–441. Springer, November 2019. URL: `https://hal.archives-ouvertes.fr/hal-02410747`.

[3] Jesús Aranda, Jorge A. Pérez, Camilo Rueda, and Frank D. Valencia. Stochastic Behavior and Explicit Discrete Time in Concurrent Constraint Programming. In *Logic Programming*, volume 5366, pages 682–686. Springer, 2008. `doi:10.1007/978-3-540-89982-2_57`.

[4] Andrés Aristizábal, Filippo Bonchi, Catuscia Palamidessi, Luis Pino, and D. Valencia, Frank. Deriving labels and bisimilarity for concurrent constraint programming. In *Proceedings of the 14th International Conference on Foundations of Software Science and Computation Structures, FOSSACS 2011*, LNCS, pages 138–152. Springer, 2011.

[5] R.B. Ash. *Basic Abstract Algebra: For Graduate Students and Advanced Undergraduates*. Dover books in science and mathematics. Dover Publications, 2007. URL: `https://books.google.com.co/books?id=kCAPckPaHTQC`.

[6] Robert J. Aumann. Agreeing to disagree. *The Annals of Statistics*, 4(6):1236–1239, 1976. URL: `http://www.jstor.org/stable/2958591`.

[7] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *Computer Aided Verification*, volume 6806, pages 171–177. Springer, 2011. doi:10.1007/978-3-642-22110-1_14.

[8] G. Birkhoff. *Lattice Theory*. Number v. 25,pt. 2 in American Mathematical Society colloquium publications. American Mathematical Society, 1967.

[9] Isabelle Bloch, Henk Heijmans, and Christian Ronse. Mathematical morphology. In Marco Aiello, Ian Pratt-Hartmann, and Johan Van Benthem, editors, *Handbook of Spatial Logics*, pages 857–944. Springer Netherlands, 2007.

[10] Frank S. Boer, Alessandra Di Pierro, and Catuscia Palamidessi. Nondeterminism and infinite computations in constraint programming. *Theoretical Computer Science*, pages 37–78, 1995.

[11] Roberto Bruni and José Meseguer. Semantic foundations for generalized rewrite theories. *Theoretical Computer Science*, 360(1-3):386–414, August 2006. doi:10.1016/j.tcs.2006.04.012.

[12] Davide Chiarugi, Moreno Falaschi, Diana Hermith, Roberto Marangoni, and Carlos Olarte. Stochastic modelling of non markovian dynamics in biochemical reactions. In Ignacio Rojas and Francisco M. Ortuño Guzman, editors, *International Work-Conference on Bioinformatics and Biomedical Engineering, IWBBIO 2013, Granada, Spain, March 18-20, 2013. Proceedings*, pages 537–544. Copicentro Editorial, 2013. URL: http://iwbbio.ugr.es/papers/iwbbio_088.pdf.

[13] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. *All about Maude - A High-Performance Logical Framework: How to Specify, Program, and Verify Systems in Rewriting Logic*. Number 4350 in Lecture Notes in Computer Science. Springer, 2007.

[14] Rodica Condurache, Riccardo De Masellis, and Valentin Goranko. Dynamic multi-agent systems: Conceptual framework, automata-based modelling and verification. In Matteo Baldoni, Mehdi Dastani, Beishui Liao, Yuko Sakurai, and Rym Zalila Wenkstern, editors, *PRIMA 2019: Principles and Practice of Multi-Agent Systems*, pages 106–122, Cham, 2019. Springer International Publishing.

[15] Brian A Davey and Hilary A Priestley. *Introduction to lattices and order*. Cambridge university press, 2nd edition, 2002.

[16] F. de Boer, M. Gabbrielli, and M. C. Meo. A timed concurrent constraint language. *Information and Computation*, 161:45–83, 2000.

[17] Gerard Debreu and Herbert Scarf. A limit theorem on the core of an economy. *International Economic Review*, 4(3):235–246, 1963. URL: http://www.jstor.org/stable/2525306.

[18] Pierpaolo Degano, Fabio Gadducci, and Corrado Priami. A causal semantics for CCS via rewriting logic. *Theoretical Computer Science*, 275(1-2):259–282, March 2002. doi:10.1016/S0304-3975(01)00165-7.

[19] Dwight Duffus, Vojtech Rodl, Bill Sands, and Robert Woodrow. Enumeration of order preserving maps. *Order*, 9(1):15–29, 1992.

[20] D.S. Dummit and R.M. Foote. *Abstract Algebra*. Wiley, 2003. URL: https://books.google.com.co/books?id=KJDBQgAACAAJ.

[21] Bruno Dutertre. Yices 2.2. In *Computer Aided Verification*, volume 8559, pages 737–744. Springer, 2014. doi:10.1007/978-3-319-08867-9_49.

[22] Joan-María Esteban and Debraj Ray. On the measurement of polarization. *Econometrica*, 62(4):819–851, 1994.

[23] Ronald Fagin, Joseph Y Halpern, Yoram Moses, and Moshe Y Vardi. *Reasoning about knowledge*. MIT press Cambridge, 4th edition, 1995.

[24] W. Feller. *An introduction to probability theory and its applications*. Wiley series in probability and mathematical statistics: Probability and mathematical statistics. Wiley, 1971.

[25] John Geanakoplos. Chapter 40 common knowledge. In *Handbook of Game Theory with Economic Applications*, volume 2 of *Handbook of Game Theory with Economic Applications*, pages 1437 – 1496. Elsevier, 1994. URL: http://www.sciencedirect.com/science/article/pii/S1574000505800724.

[26] Gerhard Gierz, Karl Heinrich Hofmann, Klaus Keimel, Jimmie D. Lawson, Michael Mislove, and Dana S. Scott. *Continuous lattices and domains*. Cambridge University Press, 2003.

[27] D. Gilbert and C. Palamidessi. Concurrent constraint programming with process mobility. In *Proc. of the Conference on Computational Logic - CL 2000*, Lecture Notes in Artificial Intelligence, pages 463–477. Springer-Verlag, 2000.

[28] Joseph A. Goguen and José Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, November 1992. doi:10.1016/0304-3975(92)90302-V.

[29] George Grätzer and E. Schmidt. On the lattice of all join-endomorphisms of a lattice. *Proceedings of The American Mathematical Society - PROC AMER MATH SOC*, 9:722–722, 1958.

[30] V. Gupta, R. Jagadeesan, and P. Panangaden. Stochastic processes as concurrent constraint programs. In *Symposium on Principles of Programming Languages*, pages 189–202, 1999. URL: citeseer.nj.nec.com/gupta98stochastic.html.

[31] V. Gupta, R. Jagadeesan, and V.A. Saraswat. Computing with continuous change. *Science of Computer Programming*, 30(1):3 – 49, 1998. Concurrent Constraint Programming. URL: http://www.sciencedirect.com/science/article/pii/S0167642397000063.

[32] Michell Guzman, Stefan Haar, Salim Perchy, Camilo Rueda, and Frank D. Valencia. Belief, knowledge, lies and other utterances in an algebra for space and extrusion. *Journal of Logical and Algebraic Methods in Programming*, 86(1):107 – 133, 2017. URL: http://www.sciencedirect.com/science/article/pii/S2352220816301080.

[33] Michell Guzmán, Sophia Knight, Santiago Quintero, Sergio Ramírez, Camilo Rueda, and Frank Valencia. Reasoning about distributed knowledge of groups with infinitely many agents. In Wan J. Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 29:1–29:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. URL: https://doi.org/10.4230/LIPIcs.CONCUR.2019.29.

[34] Michell Guzman, Salim Perchy, Camilo Rueda, and Frank Valencia. Deriving Inverse Operators for Modal Logic. In *Theoretical Aspects of Computing – ICTAC 2016*, volume 9965 of *Lecture Notes in Computer Science*, pages 214–232. Springer, 2016. URL: https://hal.inria.fr/hal-01328188.

[35] Michell Guzmán, Salim Perchy, Camilo Rueda, and Frank D. Valencia. Characterizing right inverses for spatial constraint systems with applications to modal logic. *Theoretical Computer Science*, 744:56–77, October 2018. doi:10.1016/j.tcs.2018.05.022.

[36] Michell Guzmán and Frank D. Valencia. On the expressiveness of spatial constraint systems. In Manuel Carro, Andy King, Neda Saeedloei, and Marina De Vos, editors, *Technical Communications*

of the 32nd International Conference on Logic Programming, ICLP 2016 TCs, October 16-21, 2016, New York City, USA, volume 52 of *OASICS*, pages 16:1–16:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. URL: https://doi.org/10.4230/OASIcs.ICLP.2016.16.

[37] Michell Guzmán, Sophia Knight, Santiago Quintero, Sergio Ramírez, Camilo Rueda, and Frank Valencia. Algebraic structures from concurrent constraint programming calculi for distributed information in multi-agent systems, 2020. arXiv:2010.10667.

[38] Michell Guzmán, Sophia Knight, Santiago Quintero, Sergio Ramírez, Camilo Rueda, and Frank Valencia. Reasoning about distributed information with infinitely many agents. *Journal of Logical and Algebraic Methods in Programming*, 121:100674, 2021. doi:https://doi.org/10.1016/j.jlamp.2021.100674.

[39] Stefan Haar, Salim Perchy, Camilo Rueda, and Frank Valencia. An Algebraic View of Space/Belief and Extrusion/Utterance for Concurrency/Epistemic Logic. In *17th International Symposium on Principles and Practice of Declarative Programming (PPDP 2015)*, pages 161–172. ACM SIGPLAN, 2015. URL: https://hal.inria.fr/hal-01256984.

[40] Michel Habib and Lhouari Nourine. Tree structure for distributive lattices and its applications. *Theoretical Computer Science*, 165(2):391–405, 1996.

[41] Anne Hakansson and Ronald Hartung. *Agent and Multi-Agent Systems in Distributed Systems-Digital Economy and E-Commerce*. Springer, 2013.

[42] Joseph Y Halpern. Using reasoning about knowledge to analyze distributed systems. *Annual review of computer science*, 2(1):37–68, 1987.

[43] Joseph Y Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM (JACM)*, 37(3):549–587, 1990.

[44] Joseph Y Halpern and Richard A Shore. Reasoning about common knowledge with infinitely many agents. *Information and Computation*, 191(1):1–40, 2004.

[45] Wei He, Xiang Sun, and Yeneng Sun. Modeling infinitely many agents. *Theoretical Economics*, 12(2):771–815, 2017. URL: https://onlinelibrary.wiley.com/doi/abs/10.3982/TE1647.

[46] Werner Hildenbrand. On economies with many agents. *Journal of Economic Theory*, 2(2):161 – 188, 1970.

[47] Werner Hildenbrand. *Core and Equilibria of a Large Economy. (PSME-5)*. Princeton University Press, 1974. URL: http://www.jstor.org/stable/j.ctt13x0tjf.

[48] Peter Jipsen. Relation algebras, idempotent semirings and generalized bunched implication algebras. In *Relational and Algebraic Methods in Computer Science*, pages 144–158. Springer International Publishing, 2017.

[49] Peter Jipsen and Nathan Lawless. Generating all finite modular lattices of a given size. *Algebra universalis*, 74(3):253–264, 2015.

[50] P.T. Johnstone. *Stone Spaces*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1982. URL: https://books.google.com.co/books?id=CiWwoLNbpykC.

[51] Sophia Knight, Catuscia Palamidessi, Prakash Panangaden, and Frank D. Valencia. Spatial and Epistemic Modalities in Constraint-Based Process Calculi. In *CONCUR 2012 - 23rd International Conference on Concurrency Theory*, volume 7454 of *Lecture Notes in Computer Science*, pages 317–332. Springer, 2012. URL: https://hal.archives-ouvertes.fr/hal-00761116.

[52] Sophia Knight, Prakash Panangaden, and Frank Valencia. Computing with epistemic and spatial modalities. Submitted for journal publication, 2019.

[53] Yoann Kubera, Philippe Mathieu, and Sébastien Picault. Everything can be agent! In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, page 1547–1548, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[54] Richard McLean and Andrew Postlewaite. Core convergence with asymmetric information. *Games and Economic Behavior*, 50(1):58 – 78, 2005. Special Issue in Honor of David Schmeidler. URL: http://www.sciencedirect.com/science/article/pii/S089982560400096X.

[55] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, April 1992. doi:10.1016/0304-3975(92)90182-F.

[56] José Meseguer. Membership algebra as a logical framework for equational specification. In *Recent Trends in Algebraic Development Techniques*, volume 1376, pages 18–61. Springer, 1998. doi:10.1007/3-540-64299-4_26.

[57] José Meseguer and Grigore Roşu. The rewriting logic semantics project: A progress report. *Information and Computation*, 231:38–69, October 2013. doi:10.1016/j.ic.2013.08.004.

[58] James R Munkres. *Topology; a First Course [By] James R. Munkres*. Prentice-Hall, 1974.

[59] L. Najman and H. Talbot. *Mathematical Morphology: From Theory to Applications*. ISTE. Wiley, 2013. URL: https://books.google.com.co/books?id=9FUlX8YrRvMC.

[60] Mogens Nielsen, Catuscia Palamidessi, and Frank D. Valencia. Temporal concurrent constraint programming: Denotation, logic and applications. *Nordic Journal of Computing*, 9(1):145–188, 2002.

[61] Ann Nowé, Peter Vrancx, and Yann-Michaël De Hauwere. *Game Theory and Multi-agent Reinforcement Learning*, pages 441–470. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. URL: https://doi.org/10.1007/978-3-642-27645-3_14.

[62] Chris Okasaki. *Purely Functional Data Structures*. Cambridge University Press, Cambridge, 2003.

[63] Peter Csaba Ölveczky and José Meseguer. Specification of real-time and hybrid systems in rewriting logic. *Theoretical Computer Science*, 285(2):359–405, August 2002. doi:10.1016/S0304-3975(01)00363-2.

[64] Peter Csaba Ölveczky and José Meseguer. Abstraction and completeness for real-time maude. *Electr. Notes Theor. Comput. Sci.*, 176(4):5–27, 2007. doi:10.1016/j.entcs.2007.06.005.

[65] Jorge A. Pérez and Camilo Rueda. Non-determinism and Probabilities in Timed Concurrent Constraint Programming. In *Logic Programming*, volume 5366, pages 677–681. Springer, 2008. doi:10.1007/978-3-540-89982-2_56.

[66] Santiago Quintero, Sergio Ramírez, Camilo Rueda, and Frank Valencia. Counting and computing join-endomorphisms in lattices. In Uli Fahrenberg, Peter Jipsen, and Michael Winter, editors, *Relational and Algebraic Methods in Computer Science - 18th International Conference, RAMiCS 2020, Palaiseau, France, April 8-11, 2020, Proceedings*, volume 12062 of *Lecture Notes in Computer Science*, pages 253–269. Springer, 2020. doi:10.1007/978-3-030-43520-2_16.

[67] Santiago Quintero, Sergio Ramírez, Camilo Rueda, and Frank D. Valencia. Counting and Computing Join-Endomorphisms in Lattices. In *RAMICS - 18th International Conference on Relational and Algebraic Methods in Computer Science*, LNCS, Paris, France, October 2020. Springer. URL: https://hal.archives-ouvertes.fr/hal-02422624.

[68] Sergio Ramírez, Miguel Romero, Camilo Rocha, and Frank Valencia. Real-time rewriting logic semantics for spatial concurrent constraint programming. In Vlad Rusu, editor, *Rewriting Logic and Its Applications - 12th International Workshop, WRLA, Held as a Satellite Event of ETAPS, Thessaloniki, Greece, June 14-15, Proceedings*, volume 11152 of *Lecture Notes in Computer Science*, pages 226–244. Springer, 2018. doi:10.1007/978-3-319-99840-4_13.

[69] Camilo Rocha, José Meseguer, and César Muñoz. Rewriting modulo SMT and open system analysis. *Journal of Logical and Algebraic Methods in Programming*, 86(1):269–297, January 2017. doi:10.1016/j.jlamp.2016.10.001.

[70] Miguel Romero, Sergio Ramírez, Camilo Rocha, and Frank Valencia. A rewriting logic approach to stochastic and spatial constraint system specification and verification, 2019. arXiv:1909.03819.

[71] Miguel Romero and Camilo Rocha. Symbolic execution and reachability analysis using rewriting modulo SMT for spatial concurrent constraint systems with extrusion. In Aaron Dutle, César Muñoz, and Anthony Narkawicz, editors, *NASA Formal Methods*, pages 435–451. Springer, 2018.

[72] C. Ronse. Why mathematical morphology needs complete lattices. *Signal Processing*, 21(2):129–154, 1990.

[73] Camilo Rueda and Frank Valencia. On validity in modelization of musical problems by ccp. *Soft Computing*, 8(9):641–648, 2004. URL: https://doi.org/10.1007/s00500-004-0390-7.

[74] Luigi Santocanale. On Discrete Idempotent Paths. In *Combinatorics on Words*, volume 11682, pages 312–325. Springer, 2019.

[75] V. Saraswat, R. Jagadeesan, and V. Gupta. Default timed concurrent constraint programming. In *Proc. of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 272–285, January 1995.

[76] Vijay A. Saraswat, Radha Jagadeesan, and Vineet Gupta. Foundations of timed concurrent constraint programming. In *Proceedings of the Ninth Annual Symposium on Logic in Computer Science (LICS)*, pages 71–80. IEEE Computer Society, 1994. doi:10.1109/LICS.1994.316085.

[77] Vijay A. Saraswat and Martin Rinard. Concurrent constraint programming. In *17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 232–245. ACM Press, 1990. doi:10.1145/96709.96733.

[78] Vijay A. Saraswat, Martin Rinard, and Prakash Panangaden. The semantic foundations of concurrent constraint programming. In *Proceedings of the 18th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '91, pages 333–352. ACM, 1991. URL: http://doi.acm.org/10.1145/99583.99627.

[79] Gerardo Sarria and Camilo Rueda. Real-time concurrent constraint programming. In *34th Latinamerican Conference on Informatics (CLEI 2008)*, pages 379–391. CLEI, 2008.

[80] Rolf Schneider. *Convex Bodies: The Brunn–Minkowski Theory*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2 edition, 2013. doi:10.1017/CBO9781139003858.

[81] J. Serra and P. Soille. *Mathematical Morphology and Its Applications to Image Processing*. Computational Imaging and Vision. Springer Netherlands, 2012. URL: https://books.google.com.co/books?id=T82qCAAAQBAJ.

[82] Jean Serra. Introduction to mathematical morphology. *Computer Vision, Graphics, and Image Processing*, 35(3):283 – 305, 1986. URL: http://www.sciencedirect.com/science/article/pii/0734189X86900022.

[83] Maarten Steen and Andrew S. Tanenbaum. A brief introduction to distributed systems. *Computing*, 98(10):967–1009, October 2016. URL: https://doi.org/10.1007/s00607-016-0508-7.

[84] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice-Hall, Inc., USA, 2006.

[85] Alberto Verdejo and Narciso Martí-Oliet. Two Case Studies of Semantics Execution in Maude: CCS and LOTOS. *Formal Methods in System Design*, 27(1-2):113–172, September 2005. doi:10.1007/s10703-005-2254-x.

[86] Steven Vickers. *Topology via logic*. Cambridge University Press, 1st edition, 1996.

[87] Valentin Zelenyuk. Aggregation of scale efficiency. *European Journal of Operational Research*, 240(1):269 – 277, 2015. URL: http://www.sciencedirect.com/science/article/pii/S0377221714005414.

Index

Index of Symbols