Università degli Studi di Padova

DIPARTIMENTO MATEMATICA

# Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea Magistrale in Informatica

# Solving systems of fixpoint equations: an algorithmic perspective


*Supervisor*
*Prof.* Paolo Baldan
University of Padua

*Co-supervisor*
*Postdoc* Tommaso Padoan
University of Padua

*Master Candidate*
Denis Mazzucato

I dedicate this to my parents and siblings, to all my friends and university/working colleagues that supported me during this coronavirus lockdown. In particular, I would like to thank Gianmarco, Joel, Luca, Marcello, and Stefano from TaralLago, without whom this thesis would have been completed two months earlier.

# Abstract

Systems of fixpoint equations over complete lattices are at the heart of many verification tasks and analysis techniques, such as model-checking of various kinds of specification logics. Some recent research showed that a game-theoretical characterization of the solution of systems of equations could be developed in terms of suitable parity games, which are referred to as powerset games. While, in principle, at least for finite lattices, the characterization allows one to determine the solution constructively, a naive application of the theory can be impractical due to the vast amount of moves one needs to play before singling out the solution. A key observation is that many of such moves are "redundant" and can be discarded when searching for the winner of the game. As a first step, the thesis formalizes this observation by introducing the notion of selection for powerset games, a subset of the moves sufficient to completely characterize the winner and showing that, under suitable conditions, a minimal selection exists. Furthermore, we introduced a logical form to efficiently represent these selections, in the form of so-called symbolic $\exists$-moves.

The game-theoretical characterization view opens the way to developing various algorithms, with either a global approach that determines the winner of each game's position or with a local approach that focuses on a specific position. Since various verification tasks boil down in checking whether a specific state enjoys some property, that in terms means establishing the winner of a specific position, we focus our efforts on local algorithms.

Specifically, building on a previous proposal we develop a local algorithm that works over complete lattices. It relies on a description of a set of fundamental operators over some finite generic lattices. It efficiently solves the verification task associated through symbolic $\exists$-moves used to minimize the number of positions explored to solve the game and retrieve the winner.

# Contents

# 1 Introduction

Systems of fixpoint equations over complete lattices frequently arise in verification problems. Some fundamental analyses, such as abstract interpretation in static analysis [Cousot and Cousot, 1977], boil down to systems of least fixpoint equations generated from the program's flow control graph. Moreover, systems of possibly mixed least and greatest fixpoint equations allow one to express several verification tasks uniformly. Notable examples come from the area of model-checking. Invariant/safety properties can be characterized as greatest fixpoints, while liveness/reachability properties are the least fixpoints. Specification modal logics, such as the $\mu$-calculus [Kozen, 1983], get most of their power from a profitable mixture between fixpoints. Consequently, a logic formula can be model checked on a given transition system by verifying whether a state of such system, often the initial one, is contained in the solution of the system of equations arising from the formula. Behavioral equivalences represent another area of interest since they can be typically characterized as solutions of greatest fixpoint equations. The best-known example in such an area is bisimilarity that can be seen as the greatest fixpoint of a suitable operator over the lattice of binary relations on states (see for example [Sangiorgi, 2011]).

The introduction of nested least and greatest fixpoint equation in the $\mu$-calculus increases its expressiveness, but it also increases the complexity of model-checking algorithms. Common approaches to the model-checking problems rely on encod-

ing in terms of parity games, see, e.g., [Bradfield and Walukiewicz, 2018; Emerson and Jutla, 1991; Stirling, 1995]. The seminal paper [Jurdziński, 2000] provides an algorithm for the solution of parity games, which is polynomial in the number of states and exponential in (half of) the alternation depth, recently improved to quasi-polynomial in [Calude et al., 2017]. A detailed discussion regarding the complexity of $\mu$-calculus model-checking can be found in [Bradfield and Walukiewicz, 2018].

Parity games belong to the class of two-player games. Players are usually referred to as the existential player ($\exists$) and the universal player ($\forall$). Since parity games are zero-sum games, only one player can have a winning strategy from each position. Thus, solving a parity game means determining which positions are winning for player $\exists$ and which are for player $\forall$. In Jurdziński's work, a crucial role is played by the notion of progress measure, which is used to characterize the parity game's solution. Intuitively, it witnesses the existence of a winning strategy for one of the two players. In [Hasuo et al., 2016; Baldan et al., 2018], the notion of progress measure has been extended to the solution of systems of equations in complete lattices. In particular, in [Baldan et al., 2018], a notion of parity game is introduced, referred to as the fixpoint game, which is shown to provide a sound and complete characterization of the solution of systems of fixpoint equations over continuous lattices. Of special interest for this thesis is a further generalization of this game, called the powerset game, which works on general complete lattices.

The game-theoretical characterization of the solution of systems of fixpoint equations allows one, in principle, at least for finite lattices, to devise an algorithm for solving the game and thus the associated verification problem. Although, a direct approach can be unfeasible, in practice, due to the massive number of moves that need to be explored before finding the winner. However, the exploration of some moves is useless. In fact, once a particular move is known to be losing for a given player, then we can deduce than a lot of more moves are losing for the same player. In order to exploit this fact, we introduce a notion of selection, a mathematical tool that identifies a limited number of moves for each position such that, restricting to such moves, one obtains a game equivalent to the original one (with the same winner at each position). Under specific conditions, this limited amount of moves can be minimal, in the sense that it cannot be further reduced

without impacting on the solution. Towards an efficient implementation of the algorithm, selections are essential.

The fact that restricting to a selection one obtains a game equivalent to the original one is formalized by introducing a notion of progress measure in the style of Jurdziński, for powerset games. Progress measures, that, as we mentioned, provide at each position, a witness of a winning strategy for one of the players, can be the basis for developing a global algorithm that computes the winner of each position of the game graph.

In many situations, a global algorithm is uselessly expensive. In fact, typically, a verification problem, e.g., model-checking problem, can be reduced to determine the winner of the corresponding game only at a specific position. For example, when comparing two systems with respect to a coinductive behavioral equivalence, like bisimilarity, the game graph has all pairs of states as positions, but one is interested in checking only if the pair consisting of the initial states belongs to the solution. Regarding model-checking $\mu$-calculus, one is typically interested in recognizing whether a specific state (often the initial one) satisfies a given formula, not to determine all the states satisfying the formula.

For these reasons, we focused on a local approach to solving the game, which is aimed at establishing which player wins at a specific position, exploring "on demand" the part of the game graph that is needed to take this decision. We based our approach over the local algorithm devised in [Baldan et al., 2020]. Which, in turn, took inspiration from backtracking techniques, for bisimilarity [Hirschkoff, 1998] and the $\mu$-calculus [Stevens and Stirling, 1998; Stirling, 1995]. Intuitively, the algorithm concerned checks whether a given lattice element is below the $i$-th equation solution. More in-depth, the execution alternates between two phases: the exploration phase and the backtracking phase. In the exploration phase, the algorithm performs a depth-first search of the game graph until a possibly infinite loop is discovered or one of the players remains without any move left. Both cases lead to a creation of assumptions for one of the two players that possibly win over that position, and we backtrack. During computation, assumptions can be either transformed into decisions (that intuitively are "stronger" than assumptions), or withdraw in favor of another fact that conflicts with the previous acknowledge. When, during backtracking, we come back to the root without any move left, we return a winner.

Exploiting symbolic $\exists$-moves in this algorithm is our main contribution to the algorithm presented in this thesis. This is done by associating each position for player $\exists$ with a logic formula which is used to retrieve, one at a time, all the minimal moves.

The rest of the thesis is organized as follows:

**Chapter 2** *Math background* overviews the main mathematical concepts used in the thesis. Focusing in particular on lattice theory.

**Chapter 3** *Systems of fixpoint equations and parity games* is an in-depth about the theory of fixpoint games, progress measures, and parity games. We also discuss how these notions instantiate in the case of the $\mu$-calculus, an important setting that will provide many examples throughout the thesis.

**Chapter 4** *Selections and symbolic $\exists$-moves* introduces selections and symbolic $\exists$-moves, which are later exploited for developing a local algorithm for solving the game.

**Chapter 5** *Local algorithm* devises a new algorithm for establishing the winner of a powerset game at a specific position. The chapter contains a detailed description of the pseudocode and all the components used. The algorithm is also illustrated on a specific example based on the $\mu$-calculus.

**Chaper 6** *Conclusion and future work* contains our though and conclusions and outlines some directions for future work.

# 2

# Math background

In this chapter we provide a summary of the notions that will be used throughout the thesis. In particular, this work relies on order theory. Specifically, it focus on lattice theory with special attention to distributive lattices, where the results will be simpler, allowing more efficient algorithms.

## 2.1 Lattices

Lattices constitute undoubtedly the groundwork of systems of fixpoint equations. In fact, this is the most important mathematical structure used in this thesis. They are used as the domain of systems of equations for all the verification problem discussed in the thesis.

In this section we provide an overview of the notions and results on lattices used in our development. For a thorough treatment the reader is referred to [Abramsky and Jung, 1994] and [Davey and Priestley, 2002].

**Definition 2.1** (poset and preorders)**.** A set $P$ with a binary relation $\sqsubseteq$ is called a *partially ordered set*, or *poset*, if the following holds, for all $x, y, z \in P$:

1. $x \sqsubseteq x$ *(Reflexivity)*
2. $x \sqsubseteq y \wedge y \sqsubseteq z \implies x \sqsubseteq z$ *(Transitivity)*

3. $x \sqsubseteq y \wedge y \sqsubseteq x \implies x = y$ *(Antisymmetry)*

If we drop antisymmetry from our list of requirements then we get what is known as *preorder*.

A preordered or partially ordered set $(P, \sqsubseteq)$ is often denoted simply as $P$, omitting the (pre)order relation. It is *well-ordered* if every non-empty subset $X \subseteq P$ has a minimum. The *join* and the *meet* of a subset $X \subseteq P$ (if they exist) are denoted $\bigsqcup X$ and $\bigsqcap X$, respectively.

**Definition 2.2** (lattice, complete lattice)**.** A *lattice* is a poset $(L, \sqsubseteq)$ such that each pair of elements $l', l'' \in L$ admits a *join operator* $l' \sqcup l''$ and a *meet operator* $l' \sqcap l''$.

A *complete lattice* is a poset $(L, \sqsubseteq)$ such that each subset $X \subseteq L$ admits a *join operator* $\bigsqcup X$ and a *meet operator* $\bigsqcap X$. In such structure, there always exists a *least element* (also called *bottom*) $\bot_L = \bigsqcup \emptyset$ and a *greatest element* (also called *top*) $\top_L = \bigsqcap \emptyset$.

**Definition 2.3** (upward-closure, basis)**.** Given a lattice $L$, let $l \in L$, we define its *upward-closure* as $\uparrow l = \{l' \in L \mid l \sqsubseteq l'\}$.

A *basis* is a subset $B_L \subseteq L$ such that for every $l \in L$, $l = \bigsqcup\{b \in B_L \mid b \sqsubseteq l\}$.

Let us now introduce an essential concept: least and greatest fixpoints. They will be used mainly to define the equational system based on the given verification problem. As a consequence of [Cousot and Cousot, 1977] and [Tarski, 1955], any monotone function on a complete lattice has least and greatest fixpoints.

Before giving the formal definition we introduce first the concept of ordinals, that will be used in the following definitions. We denote *ordinals* by Greek letters $\alpha, \gamma, \delta, \ldots$ and their order by $\leq$. The collection of all ordinals is well-ordered. Given any ordinal $\alpha$, the collection of ordinals dominated by $\alpha$ is a set $[\alpha] = \{\lambda \mid \lambda \leq \alpha\}$, which, seen as an ordered structure, is a lattice. Meet and join of a set $X$ of ordinals will be denoted by $inf\ X$ (which equals to $min\ X$ if $X \neq \emptyset$) and $sup\ X$ (dually, $max\ X$ if $X$ finite and non-empty).

**Definition 2.4** (monotone functions, least and greatest fixpoints)**.** Let $L$ be a complete lattice. A function $f : L \to L$ is *monotone* if for all $l, l' \in L$ it holds that if $l \sqsubseteq l'$ then $f(l) \sqsubseteq f(l')$. The least fixpoint $\mu f$ can be computed as the meet

of all pre-fixpoints, while the greatest fixpoint $\nu f$ as the join of all post-fixpoints, as follows:

- $\mu f = \bigsqcap\{l \mid f(l) \sqsubseteq l\}$
- $\nu f = \bigsqcup\{l \mid l \sqsubseteq f(l)\}$

We also have an iterative procedure to actually compute them, starting respectively from the bottom or from the top of the lattice, this is often referred to as *Kleene's theorem*, at least for the least fixpoint, which is one of the most essential theorems about abstract interpretation [Cousot and Cousot, 1977].

Given a lattice $L$, define its *height* $\lambda_L$ as the supremum of the length of every (transfinite) strictly ascending chain in $L$, observe that $\lambda_L$ is an ordinal.

**Theorem 2.5** (Kleene's iteration)**.** *Let $L$ be a lattice and let $f : L \to L$ be a monotone function. Consider the (transfinite) ascending chain $(f^\beta(\bot))_\beta$ where $\beta$ ranges over ordinal, defined by:*

- $f^0(\bot) = \bot$
- $f^{\alpha+1}(\bot) = f(f^\alpha(\bot))$*, for any successor ordinal* $\alpha$
- $f^\alpha(\bot) = \bigsqcup_{\beta<\alpha} f^\beta(\bot)$*, for any limit ordinal* $\alpha$*.*

*Then $\mu f = f^\gamma(\bot)$ for some ordinal $\gamma \leq \lambda_L$. Dually, $\nu f = f^\gamma(\top)$ for some ordinal $\gamma$ via the (transfinite) descending chain $(f^\alpha(\top))_\alpha$.*

Note that, for any ordinal $\alpha$, the element $f^\alpha(\bot)$ is always a post-fixpoint and $f^\alpha(\top)$ is always a pre-fixpoint. Notably, regarding the greatest fixpoint $\nu$, we did not give an explicit bound to the ordinal $\gamma$ since depth and height are not equivalent in (transfinite) lattices.

We now introduce the well-known concept of equivalence classes, concept hardly related to its equivalence relation.

**Definition 2.6** (equivalence relation)**.** A binary relation $\sim$ is called an *equivalence relation* if $\sim$ is a preorder and the symmetric property holds, i.e., for all $x, y$ it holds that $x \sim y \implies y \sim x$.

As a consequence of the reflexive, symmetric, and transitive properties, any equivalence relation provides a partition of the underlying set into disjoint equivalence classes. Two elements of the given set are equivalent to each other if and only if they belong to the same equivalence class. Observe that, every preorder $\sqsubseteq$ induces a natural equivalence relation $\sim_{\sqsubseteq}$, that is, $x \sim_{\sqsubseteq} y \iff x \sqsubseteq y \wedge y \sqsubseteq x$.

**Definition 2.7** (equivalence class)**.** Let $P$ be a set endowed with the equivalence relation $\sim$ and let $x \in P$. The *equivalence class* of $x$ is $[x]_\sim$ and is defined as $[x]_\sim = \{y \in P \mid x \sim y\}$. The set of all equivalence classes on $P$ under the relation $\sim$ is the set $P/_\sim$.

## 2.2 Tuples

We will often consider tuples of elements. This structure is useful in contexts where the order of the elements is relevant. For instance, when considering the solution of a system of equations, every component of the solution has a precise position and this information cannot be lost during the computation.

As an example, the tuple $t = (1, 2)$ is obviously different from $t' = (2, 1)$, while if we consider the set $s = \{1, 2\}$ there is no difference between $s$ and $s' = \{2, 1\}$.

Given a set $A$, an $n$-tuple in $A^n$ will be denoted by a boldface letter $\boldsymbol{a}$. The components of $\boldsymbol{a}$ will be denoted by using the same name of the tuple, not in boldface style and with an index, i.e., $\boldsymbol{a} = (a_1, \ldots, a_n)$. For an index $n \in \mathbb{N}$ we use the notation $\underline{n}$ to denote the integer interval $\{1, \ldots, n\}$. Given $\boldsymbol{a} \in A^n$ and two indexes $i, j \in \underline{n}$ with $i \leq j$, we write $\boldsymbol{\alpha}_{i,j}$ for the subtuple $(a_i, a_{i+1}, \ldots, a_j) \in A^{j-i+1}$.

**Definition 2.8** (order relations over tuples)**.** Let $P$ be a preorder. We will denote by $(P^n, \sqsubseteq^\wedge)$ the set of $n$-tuples endowed with the *pointwise order* (also called *and relation*) defined, for $\boldsymbol{l}, \boldsymbol{l'} \in L^n$, by $\boldsymbol{l} \sqsubseteq^\wedge \boldsymbol{l'}$ if $l_i \sqsubseteq l'_i$ for all $i \in \underline{n}$. Dually, the *or relation* is defined, for $\boldsymbol{l}, \boldsymbol{l'} \in L^n$, by $\boldsymbol{l} \sqsubseteq^\vee \boldsymbol{l'}$ if there exists $i \in \underline{n}$ such that $l_i \sqsubseteq l'_i$.

Let $L$ be a poset. For $n \in \mathbb{N}$, we will denote by $(L^n, \preceq)$ the set of $n$-tuples endowed with the *lexicographic order* (in which the last component is the most relevant), i.e., inductively, for $\boldsymbol{l}$, $\boldsymbol{l'} \in L^n$, we let $\boldsymbol{l} \preceq \boldsymbol{l'}$ if either $l_n \sqsubset l'_n$, or $l_n = l'_n$ and $\boldsymbol{l}_{1,n-1} \preceq \boldsymbol{l'}_{1,,n-1}$.

Let $n \in \mathbb{N}$, for $i \in \underline{n}$, we will denote by $(L^n, \preceq_i)$ the set of $n$-tuples endowed with the *truncated lexicographic order* defined, for $\boldsymbol{l}, \boldsymbol{l'} \in L^n$, $\boldsymbol{l} \preceq_i \boldsymbol{l'}$ if $\boldsymbol{l}_{i,n} \preceq \boldsymbol{l'}_{i,n}$.

**Remark 1.** Some observations over tuples.

- When $(L, \sqsubseteq)$ is a lattice also $(L^n, \preceq)$ is a lattice. Meet and join can be easily obtained by taking the operator of the single components, from the last to the first.
- In words, $\preceq_i$ is the lexicographic order restricted to the last components $i, i+1, \ldots, n$. For instance, if $P = \mathbb{N}$ then $(8,1,2,5) \preceq_2 (1,8,2,5)$, while $(8,1,2,5) =_3 (1,8,2,5)$ using the equivalence induced.

### 2.3 Order theory

Here we introduce some order/preorder relations that play a crucial role in Chapter 4. In particular these relations permit us to efficiently compute the solution for the verification problem concerned.

Roughly speaking, to compute the solution of a verification problem we need to play a parity game (see Section 3.3), that consists in repeatedly "playing" some element of a lattice. We will show that one can only consider moves which are minimal with respect to suitably defined preorder, like Hoare or Supremum preorder. This clearly improves efficiency with respect to trying all possible moves.

We first recall the classical notion of subset inclusion order, i.e. the powerset lattice, then we define two other preorders called respectively Hoare and Supremum preorder. Note that, the definition of "Hoare preorder" comes from [Abramsky and Jung, 1994], instead the "Supremum preorder" is an original structure, to the best of our knowledge.

**Definition 2.9** (powerset lattice)**.** Let $L$ be a lattice with a basis $B_L$ and let $X, Y \subseteq B_L$. We define the *powerset lattice* $(2^{B_L}, \subseteq)$, where $X \subseteq Y$ if for all $x \in X$ it holds that $x \in Y$.

**Definition 2.10** (Hoare and Supremum Preorder on Sets)**.** Let $L$ be a lattice with a basis $B_L$ and let $X, Y \in 2^{B_L}$. We define the *Hoare preorder* $(2^{B_L}, \sqsubseteq_H)$, where $X \sqsubseteq_H Y$ if for all $x \in X$, there is $y \in Y$ such that $x \sqsubseteq y$. We define the *Supremum preorder* $(2^{B_L}, \sqsubseteq_\sqcup)$, where $X \sqsubseteq_\sqcup Y$ if $\bigsqcup X \sqsubseteq \bigsqcup Y$.

It is immediate to see that subset inclusion implies Hoare preorder, and that Hoare preorder implies Supremum preorder. The converse does not generally

holds, unless some distributive properties explored in the next section are required.

**Lemma 2.11** (subset inclusion implies Hoare preorder)**.** *Let $(L, \sqsubseteq)$ be a lattice and let $B_L \subseteq L$ be a basis for L.Let $X, Y \in 2^{B_L}$. If $X \subseteq Y$ then $X \sqsubseteq_H Y$.*

*Proof.* From $X \subseteq Y$ we have that, whenever $x \in X$ then $x \in Y$. Hence, for all $x \in X$ we have $x \in Y$ and $x \sqsubseteq x$. □

**Lemma 2.12** (Hoare preorder implies Supremum preorder)**.** *Let $(L, \sqsubseteq)$ be a lattice and let $B_L \subseteq L$ be a basis for L. Let $X, Y \in 2^{B_L}$. If $X \sqsubseteq_H Y$ then $X \sqsubseteq_\sqcup Y$.*

*Proof.* Let $X, Y \in 2^{B_L}$ such that $X \sqsubseteq_H Y$

$$\begin{aligned}
\bigsqcup X &= \bigsqcup \{x \in B_L \mid x \in X\} && (X \text{ definition}) \\
&\sqsubseteq \bigsqcup \{y \mid x \in X \wedge y \in Y \wedge x \sqsubseteq y\} && (X \sqsubseteq_H Y) \\
&\sqsubseteq \bigsqcup Y && (\{y \mid x \in X \wedge y \in Y \wedge x \sqsubseteq y\} \subseteq Y)
\end{aligned}$$

□

We conclude this section by showing that both Hoare and Supremum are preorders.

**Lemma 2.13** ($\sqsubseteq_H$ is a pre-order)**.** *Let $(L, \sqsubseteq)$ be a lattice and let $B_L \subseteq L$ be a basis for L. Consider $(2^{B_L}, \sqsubseteq_H)$, let $X, Y, Z \in 2^{B_L}$. The followings holds:*

- *Reflexivity, $X \sqsubseteq_H X$, and*
- *Transitivity, if $X \sqsubseteq_H Y$ and $Y \sqsubseteq_H Z$ then $X \sqsubseteq_H Z$*

*Proof.* We show that $\sqsubseteq_H$ is a preorder by showing that the two properties hold.

**(Reflexivity)** To conclude that $X \sqsubseteq_H X$ just observer that, for all $x \in X$ there exists $x' = x \in X$ such that $x \sqsubseteq x'$.

**(Transitivity)** Since $X \sqsubseteq_H Y$, for all $x \in X$ there exists $y \in Y$ such that $x \sqsubseteq y$. Using $Y \sqsubseteq_H Z$, there exists $z \in Z$ such that $y \sqsubseteq z$, hence $x \sqsubseteq y \sqsubseteq z$, we conclude by transitivity of $\sqsubseteq$.

□

**Lemma 2.14** ($\sqsubseteq_\sqcup$ is a pre-order)**.** *Let $(L, \sqsubseteq)$ be a lattice and let $B_L \subseteq L$ be a basis for $L$. Consider $(2^{B_L}, \sqsubseteq_\sqcup)$, let $X, Y, Z \in 2^{B_L}$. The followings holds:*

- *Reflexivity, $X \sqsubseteq_\sqcup X$, and*
- *Transitivity, if $X \sqsubseteq_\sqcup Y$ and $Y \sqsubseteq_\sqcup Z$ then $X \sqsubseteq_\sqcup Z$*

*Proof.* We show that $\sqsubseteq_\sqcup$ is a preorder by showing that the two properties hold.

**(Reflexivity)** We have that $X \sqsubseteq_\sqcup X$ since $\bigsqcup X \sqsubseteq \bigsqcup X$ by reflexivity of $\sqsubseteq$.

**(Transitivity)** If $X \sqsubseteq_\sqcup Y$ and $Y \sqsubseteq_\sqcup Z$ then $\bigsqcup X \sqsubseteq \bigsqcup Y \sqsubseteq \bigsqcup Z$. We conclude by transitivity of $\sqsubseteq$ that $\bigsqcup X \sqsubseteq \bigsqcup Z$ and thus $X \sqsubseteq_\sqcup Z$.

□

## 2.4 Distributive lattices

In this section we will study relations between the various type of lattices under the general category of distributive lattices. More detail about this class of lattices can be found in [Garg, 2015] and [Bandelt, 1982].

Later in this thesis, thanks to these properties exposed in this section (e.g. infinite distributive law, see Definition 2.16), we will be able to compute the solution of verification problems in a faster way. In words, in context based over frames it is easier to check whether an element is under a certain given element of the lattice. Instead, in non-distributive based context, to do the same check, one needs to use slower and more complex algorithms.

**Definition 2.15** (distributive lattices)**.** A lattice $L$ is called *distributive*, if for all $a, b, c \in L$ it holds that $a \sqcap (b \sqcup c) = (a \sqcap b) \sqcup (a \sqcap c)$. Equivalently, $a \sqcup (b \sqcap c) = (a \sqcup b) \sqcap (a \sqcup c)$.

**Definition 2.16** (frames)**.** A complete lattice $L$ is called a *frame*, if finite meets may distribute over arbitrary join, i.e., if the infinite distributive law holds: $l \sqcap (\bigsqcup X) = \bigsqcup \{l \sqcap x \mid x \in X\}$, for any $l \in L$ and $X \subseteq L$.

Note that this distributive law is not equivalent to its dual statement, $l \sqcup (\bigsqcap X) = \bigsqcap\{l \sqcup x \mid x \in X\}$ which, see Example 1, which instead defines the class of coframes.

**Example 1** (infinite distributive law is not dual)**.** This example is taken from [Ranzato, 2017].

Consider the complete lattice of open subsets of $\mathbb{R}$ ordered by $\subseteq$, this lattice is a frame since in here it holds the infinite distributive law, but it is not a co-frame, which is the dual notion.

Let us first define the join and the meet operators in this lattice. The join operator $\sqcup$ is the classical union $\cup$ between open sets since this operator is closed under the lattice of open sets of $\mathbb{R}$, i.e., the union of open sets is still an open set. The $\sqcap$ operator is defined as the classical intersection $\cap$ for finite sets, instead, for infinite sets is defined as $\bigsqcap X = int(\bigcap X)$, where $int$ is the *inside* of the given set, e.g., $X = \{]-\epsilon, \epsilon[ \mid \epsilon > 0\}$ then $\bigsqcap X = \emptyset$.

So, consider the previous defined set of open sets $X$ and the open set $\mathbb{R}\backslash\{0\}$. We show that the lattice concerned is a frame by showing that the infinite distributive law holds:

$$(\mathbb{R} \setminus \{0\}) \sqcap (\bigsqcup X) = \mathbb{R} \setminus \{0\}$$
$$\bigsqcup\{(\mathbb{R} \setminus \{0\}) \sqcap x \mid x \in X\} = \bigsqcup\{x \setminus \{0\} \mid x \in X\} = \mathbb{R} \setminus \{0\}$$

while the converse does not:

$$(\mathbb{R} \setminus \{0\}) \sqcup (\bigsqcap X) = \mathbb{R} \setminus \{0\}$$
$$\bigsqcap\{(\mathbb{R} \setminus \{0\}) \sqcup x \mid x \in X\} = \mathbb{R}$$

So this is clearly a counterexample for the duality of the infinite distributive law.

Now one can go further and define orders where arbitrary joins distribute over arbitrary meets.

**Definition 2.17** (completely distributive lattices)**.** Let $L$ be a complete lattice. Consider a doubly indexed family $(x_{j,k})_{j \in J, k \in K(j)}$, where $x_{j,k} \in L$. Let $F$ be the set of choice functions $f$ providing for each index $j \in J$ some index $f(j) \in K(j)$.

The lattice $L$ is called a *completely distributive lattice*, if

$$\bigsqcap_{j\in J}\bigsqcup_{k\in K(j)} x_{j,k} = \bigsqcup_{f\in F}\bigsqcap_{j\in J} x_{j,f(j)}$$

This property is again a self-dual property, as the one in Definition 2.15. Thus dualizing the above statement yields the same class of complete lattices.

As shown in [Bonsangue and Kok, 1997] every completely distributive lattice $L$, is a frame. Similarly, every frame $L$, is a distributive lattice.

**Definition 2.18** (completely join-irreducible and completely prime)**.** Let $L$ be a complete lattice. An element $l \in L$ is called *completely join-irreducible* if, for all $X \subseteq L$ such that $l = \bigsqcup X$, then $l \in X$. An element $l \in L$ is called *completely prime* if, for all non-empty $X \subseteq L$ such that $l \sqsubseteq \bigsqcup X$, then there exists $x \in X$ such that $l \sqsubseteq x$.

Observe that $\bot$ is never completely join-irreducible since $\bot = \bigsqcup \emptyset$ but $\bot \notin \emptyset$.

Unfortunately, a basis of only completely join-irreducibles does not always exists, e.g., consider the complete lattice of real numbers in $[0, 1]$, here there is no completely join-irreducible, more formally, $\forall x \in [0, 1]$, if $X = [0, x), x = \bigsqcup X$ but $x \notin X$, which means that $x$ is not completely join-irreducible. In Theorem 2, Section 1.9 [Büchi, 1989], it is shown that the so-called discrete lattices always admit a basis of only completely join-irreducibles. Moreover,for these lattices, every basis is a superset of the set of all completely join-irreducibles.

We do not provide the details of the definition of discrete lattice. We just mention that a notion of distance is defined between elements of $L$ and $L$ is discrete if, for all $x \in S \subseteq L$ there exists $\epsilon > 0 \in \mathbb{R}$, such that for any $y \in S \setminus \{x\}$ it holds $d(x, y) > \epsilon$. Every finite lattice is discrete [Büchi, 1989].

Clearly, even in lattices that could have basis with only completely join-irreducible elements, some basis could contain elements that are not completely join-irreducibles.

**Example 2** (basis with elements that are not completely join-irreducible)**.** Let us consider a simple complete lattice of just four elements $\{\bot, a, b, \top\}$, where $a$ and $b$ are incomparable.

Then a basis can be $B_L = \{a, b, \top\}$, where $\top$ is not completely join-irreducibles since $\top = \bigsqcup\{a, b\}$ and $\top \notin \{a, b\}$.

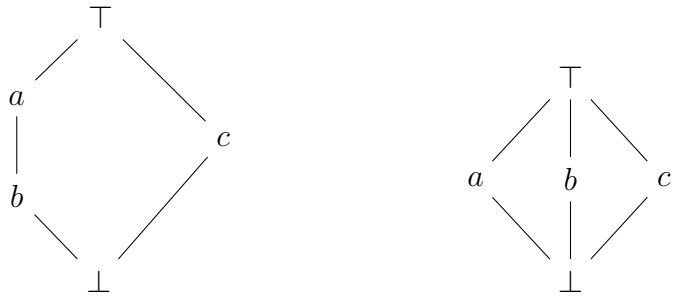


Figure 2.1: Pentagon lattice (left) and diamond lattice (right)

Additionally, since this lattice is finite, and thus discrete, there exists $B'_L \subseteq B_L$ where $B'_L$ is the set of all completely join-irreducible elements. It can be easily seen that $B'_L = \{a, b\}$.

**Lemma 2.19** (Supremum preorder implies Hoare preorder on frames)**.** *Let $L$ be a complete lattice, and $B_L$ a basis of completely join-irreducibles. Whenever $L$ is a frame, given $X, Y \subseteq B_L$, if $X \sqsubseteq_\sqcup Y$ then $X \sqsubseteq_H Y$.*

*Proof.* First we show that each $b \in B_L$ is completely prime. For all non-empty $Y \subseteq L$ such that $b \sqsubseteq \bigsqcup Y$, we have $b = b \sqcap (\bigsqcup Y) = \bigsqcup \{b \sqcap y \mid y \in Y\}$, since $L$ is a frame. Hence, $b = b \sqcap y$ for some $y \in Y$, i.e., $b \sqsubseteq y$ for some $y \in Y$, by completely join-irreducibility.

Then we show our thesis. Given $X, Y \subseteq B_L$, assume that $\bigsqcup X \sqsubseteq \bigsqcup Y$. For any $x \in X$, it holds that $x \sqsubseteq \bigsqcup X \sqsubseteq \bigsqcup Y$, hence $x \sqsubseteq y$ for some $y \in Y$ since $x$ is completely prime, by the first part of the proof. □

**Corollary 2.20** (Hoare and Supremum preorder coincide on frames)**.** *Let $L$ be a frame, with a basis $B_L$ of completely join-irreducible elements. Given $X, Y \in 2^{B_L}$, $X \sqsubseteq_H Y$ if and only if $X \sqsubseteq_\sqcup Y$.*

*Proof.* If $L$ is a frame $X \sqsubseteq_\sqcup Y$ implies $X \sqsubseteq_H Y$ by Lemma 2.19. The converse implication holds in general, even if $L$ is not a frame, by Lemma 2.12. □

**Example 3.** In Figure 2.1, we show two notable non-distributive lattices. Respectively, the pentagon lattice N5 on the left and the diamond lattice M3 on the right. It is easy to see that they are both non-distributive. For the pentagon

lattice N5 on the left, observe

$$\begin{aligned} a \sqcap (b \sqcup c) &= (a \sqcap b) \sqcup (a \sqcap c) \\ a \sqcap \top &= b \sqcup \bot \\ a &= b \end{aligned}$$

while for the diamond lattice on the right

$$\begin{aligned} a \sqcap (b \sqcup c) &= (a \sqcap b) \sqcup (a \sqcap c) \\ a \sqcap \top &= \bot \sqcup \bot \\ a &= \bot \end{aligned}$$

These are, in a sense, the prototypical non-distributive lattices, since it can be shown that every non-distributive lattice has N5 or M3 as sub-lattice.

# 3

# Systems of fixpoint equations and parity games

In the chapter before we introduced the basic math concepts that we are going to use in this thesis such as order theory, distributive property and so on. In this chapter, we will move a step forward by introducing the theoretical problem the thesis will focus on, i.e., the solution of systems of fixpoint equations in complete lattices.

## 3.1 Systems of fixpoint equations

In this thesis we deal with systems of fixpoint equations, where, for each equation, we are interested in the least or the greatest solution independently from the other. In this section, we define the system of fixpoint equations, its solution and we provide an example that will be used throughout in the thesis. We base our presentation on previous works [Baldan et al., 2020] and [Baldan et al., 2018].

**Definition 3.1** (system of fixpoint equations)**.** Let $L$ be a (complete) lattice. A *system of fixpoint equations* $E$ over the lattice $L$ is a list of $m$ equations of the

following form:

$$\begin{cases} x_1 & =_{\eta_1} f_1(x_1, \ldots, x_m) \\ & \ldots \\ x_m & =_{\eta_m} f_m(x_1, \ldots, x_m) \end{cases}$$

where $f_i : L^m \to L$ are monotone functions and $\eta_i \in \{\mu, \nu\}$. A more compact, vector like, representation of the system is $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$, where $\boldsymbol{x} = (x_1, \ldots, x_m)$, $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_m)$ and $\boldsymbol{f} = (f_1, \ldots, f_m)$. We denote the empty system as $E = \emptyset$ coherently.

Note that $\boldsymbol{f}$ can be seen as a function $\boldsymbol{f} : L^m \to L^m$. Thus, the solution of a system is a fixpoint of such function. But first, we need some preliminary notions used to properly define the solution.

**Definition 3.2** (substitution)**.** Let $E$ be a system of $m$ fixpoint equations over a complete lattice $L$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$. Let $i \in \underline{m}$ be an index and $l \in L$ an element. We write $E[x_i := l]$ for the system of $m-1$ equations obtained from $E$ by removing the $i$-th equation and replacing $x_i$ by $l$ in the other equations, i.e., if $\boldsymbol{x} = \boldsymbol{x}' x_i \boldsymbol{x}''$, $\boldsymbol{\eta} = \boldsymbol{\eta}' \eta_i \boldsymbol{\eta}''$ and $\boldsymbol{f} = \boldsymbol{f}' f_i \boldsymbol{f}''$ then $E[x_i := l]$ is $\boldsymbol{x}'\boldsymbol{x}'' =_{\boldsymbol{\eta}'\boldsymbol{\eta}''} \boldsymbol{f}'\boldsymbol{f}''(\boldsymbol{x}', l, \boldsymbol{x}'')$.

**Definition 3.3** (solution)**.** Let $E$ be a system of $m$ fixpoint equations over a complete lattice $L$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$. The *solution* of $E$, denoted $\boldsymbol{s} = sol(E) \in L^m$, is defined inductively as follows:

$$\begin{aligned} sol(\emptyset) &= () \\ sol(E) &= (sol(E[x_m := s_m], s_m)) \end{aligned}$$

where $s_m = \eta_m(\lambda x. f_m(sol(E[x_m := x]), x))$. As a tuple, solutions are endowed with indices, thus $sol_i(E)$ represent the $i$-th component of the solution.

Intuitively, the solution is obtained by considering the last variable as a fixed parameter $x$ and the system of $m-1$ equations that arises from dropping the last equations is recursively solved. The following $(m-1)$-tuple, parametric on $x$ is therefore produced, $\boldsymbol{s}_{1,m-1}(x) = sol(E[x_m := x])$. Inserting such $x$-parametrized equation into the last equation, we get an equation in a single variable

$$x =_{\eta_m} f_m(\boldsymbol{s}_{1,m-1}(x), x)$$

that can be solved by taking the least or greatest fixpoint (depending on $\eta_m$) of the function $\lambda x.f_m(\boldsymbol{s}_{1,m-1}(x), x)$ that provides $s_m = \eta_m(\lambda x.f_m(\boldsymbol{s}_{1,m-1}(x), x))$. Finally, the remaining components are computed by inserting $s_m$ into the parametric solution $\boldsymbol{s}_{1,m-1}(x)$ previously computed, $\boldsymbol{s}_{1,m-1} = \boldsymbol{s}_{1,m-1}(s_m)$.

Note that, the order of the equations matters, reordering the equations typically results in a different solution.

**Example 4.** We recall an example from [Mazzocchin, 2019], to show some peculiarities of systems of equations.

Let us consider a small equation system with only two equations, i.e., $m = 2$, over a powerset lattice $2^A$ where $A$ is a fixed non-empty set. Observe that we use both least and greatest fixpoints.

$$\begin{cases} x & =_\mu x \cap y \\ y & =_\nu x \cup y \end{cases}$$

Using a trivial substitution, it can be easily seen that the solution is $\boldsymbol{s} = (\emptyset, A)$, because $\emptyset$ is clearly the least solution for the first equation and $A$ is the biggest one for the second equation. We first substituted the second equation into $y$ on the first one and we obtained $x =_\mu x \cap (x \cup y)$ that is equivalent to $x =_\mu x$, thus $x = \emptyset$. Then we substituted $x = \emptyset$ into the second equation, so $y =_\nu \emptyset \cup y =_\nu y$. Clearly, $y = A$ since we required the greatest fixpoint.

Let us invert the equations, obtaining

$$\begin{cases} y & =_\nu x \cup y \\ x & =_\mu x \cap y \end{cases}$$

In this case the solution remains $\boldsymbol{s} = (A, \emptyset)$. However, this is not always the case. To show that also in simple examples as this one, the order of the equations matters, we swap the set operators of the first system producing

$$\begin{cases} x & =_\mu x \cup y \\ y & =_\nu x \cap y \end{cases}$$

Here the solution is $\boldsymbol{s} = (A, A)$. Now we swap the two equations of the above

system.

$$\begin{cases} y & =_\nu x \cap y \\ x & =_\mu x \cup y \end{cases}$$

this time we obtain a different solution, that is $\boldsymbol{s} = (\emptyset, \emptyset)$.

Remarkably, such strong dependency on the order of the equations can occur only when the system contains a mix of least and greatest fixpoints.

## 3.2 $\mu$-CALCULUS: A BRIEF INTRODUCTION

As a prototypical example, we gently introduce $\mu$-calculus formulae to show how they can be seen as systems of fixpoint equations. The $\mu$-calculus was created by Dana Scott and Jaco de Bakker, and was further developed and improved by Dexter Kozen in [Kozen, 1983] into its current form. Nowadays, it is mainly used to express verification problems which are then solved with the model checking technique. Its power derives from the two fixed point operators that increase his expressiveness.

For a fixed disjoint set $PVar$ of propositional variables, ranged over by $x, y, z, \ldots$ and $Prop$ of propositional symbols, ranged over by $p, q, r, \ldots$, the syntax of formulae is defined by

$$\varphi \ ::= \ \boldsymbol{t} \mid \boldsymbol{f} \mid p \mid x \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Box\varphi \mid \Diamond\varphi \mid \eta x.\varphi$$

where $p \in Prop$, $x \in PVar$ and $\eta \in \{\mu, \nu\}$. Formulae of the kind $\mu x.\varphi$ or $\nu x.\varphi$ are called fixpoint formulae. The semantic of a formula is given with respect to an unlabelled transitions system.

Given a formula $\varphi$ and an environment $\rho : Prop \bigcup PVar \to 2^{\mathbb{S}}$ mapping each proposition or propositional variable to the set of states where it holds. We denote by $[\![\varphi]\!]_\rho$ the semantic of $\varphi$, defined as usual (see [Bradfield and Walukiewicz, 2018]).

**Definition 3.4** (semantic of $\mu$-calculus formulae)**.** Given an unlabelled transitions system $(S, \to)$, an environment $\rho : Prop \bigcup PVar \to 2^{\mathbb{S}}$, the *semantic* of a

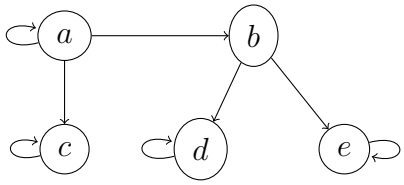


Figure 3.1: Transition system

formula $\varphi$ is defined as follows:

$$
\begin{aligned}
[\![p]\!]_\rho &= \rho(p) \\
[\![x]\!]_\rho &= \rho(x) \\
[\![\varphi_1 \wedge \varphi_2]\!]_\rho &= [\![\varphi_1]\!]_\rho \cap [\![\varphi_2]\!]_\rho \\
[\![\varphi_1 \vee \varphi_2]\!]_\rho &= [\![\varphi_1]\!]_\rho \cup [\![\varphi_2]\!]_\rho \\
[\![\Box\varphi]\!]_\rho &= \{s \in \mathbb{S} \mid \forall t \in \mathbb{S}, s \to t \implies t \in [\![\varphi]\!]_\rho\} \\
[\![\Diamond\varphi]\!]_\rho &= \{s \in \mathbb{S} \mid \exists t \in \mathbb{S}, s \to t \wedge t \in [\![\varphi]\!]_\rho\} \\
[\![\nu x.\varphi]\!]_\rho &= \bigcup\{S \subseteq \mathbb{S} \mid S \subseteq [\![\varphi]\!]_{\rho[x:=S]}\} \\
[\![\mu x.\varphi]\!]_\rho &= \bigcap\{S \subseteq \mathbb{S} \mid S \supseteq [\![\varphi]\!]_{\rho[x:=S]}\}
\end{aligned}
$$

Several authors observed that $\mu$-calculus formulae can be equivalently presented as systems of fixpoint equations, see [Cleaveland et al., 1993] and [Gawlitza and Seidl, 2011].

**Example 5** ($\mu$-calculus)**.** A $\mu$-calculus formula can be presented as a system of equations, by using an equation for each fixpoint subformula. For instance, consider $\varphi = \mu x_2.((\nu x_1.(p \wedge \Box x_1)) \vee \Diamond x_2)$ that requires that a state is eventually reached from which $p$ always holds. Its equational form is

$$
\begin{cases}
x_1 & =_\nu p \wedge \Box x_1 \\
x_2 & =_\mu x_1 \vee \Diamond x_2
\end{cases}
$$

Consider a transition system $T = (\mathbb{S}_T, \to_T)$ where $\mathbb{S}_T = \{a, b, c, d, e\}$ and $\to_T$ is depicted in Figure 3.1, with $p$ that holds in $\{b, d, e\}$.

Given a relation $R \subseteq X \times X$, we define the semantic counterpart of the modal

operators ($\Diamond$ and $\Box$) as follows:

$$\blacksquare_R, \blacklozenge_R : 2^X \to 2^X$$
$$\blacksquare_R(Y) = \{x \in X \mid \forall y \in X.(x,y) \in R \implies y \in Y\}$$
$$\blacklozenge_R(Y) = \{x \in X \mid \exists y \in Y.(x,y) \in R\}$$

for $Y \subseteq R$. Then the formula $\varphi$ interpreted over the transition system $T$ leads to the system of fixpoint equations over the powerset lattice $2^{\mathbb{S}_T}$.

$$\begin{cases} x_1 & =_\nu \{b,d,e\} \cap \blacksquare_{\to_T} x_1 \\ x_2 & =_\mu x_1 \cup \blacklozenge_{\to_T} x_2 \end{cases}$$

Finally, the solution of the system is $x_1 = \{b,d,e\}$ (states where $p$ always holds) and $x_2 = \{a,b,d,e\}$ ( states where the formula $\varphi$ holds).

## 3.3 Parity games

A parity game is played on a *parity graph*, a particular kind of directed graph equipped with a priority function, a formal definition of parity game follows (see [Jurdziński, 2000] for more details).

**Definition 3.5** (parity game)**.** A *parity game* is a tuple $\Gamma = (V, E, p, (V_\exists, V_\forall))$, where $G = (V, E, p)$ is the game graph of $\Gamma$, the set of nodes $V$ is partitioned into two sets $V_\exists, V_\forall$ and $p : V \to \mathbb{N}$ is the priority function. The two players, called $\exists$ and $\forall$ (different names can be found in the literature) form a possibly infinite path in the game graph by passing a token to each other along the edges. Firstly, an initial node is chosen and the token is initialized to this node. During the game, the vertex such token is placed on gives rise to two rules:

- let the token be on a vertex in the $V_\exists$ set, then player $\exists$ moves the token along one of the outgoing edges from the current vertex
- dually, player $\forall$ acts similarly on vertices in $V_\forall$

Parity games belong to the category of zero-sum game, a winner always exists

while the other players loses. This game is equipped with a formal winning conditions.

**Definition 3.6** (winning condition)**.** To define the winning condition for a game we need to distinguish between finite and infinite plays.

- *finite play*, the winner of a finite play is the player whose opponent is unable to move
- *infinite play*, let $\pi = (v_1, v_2, \dots)$ be an infinite play, let $Inf(\pi)$ denote the set of priorities appearing infinitely often in the nodes inside the play. We say that $\pi$ is a winning play for $\exists$ if $max(Inf(\pi))$ is even. Otherwise $\pi$ is a winning play for $\forall$.

**Definition 3.7** (winning strategy)**.** A function $\sigma : V_\exists \to V$ is a *strategy* for $\exists$ player if $(v, \sigma(v)) \in E$ for any $v \in V_\exists$.

A *play* $\pi = (v_1, v_2, \dots)$ is *consistent* with a strategy $\sigma$ (for $\exists$ player) in the event that $v_{i+1} = \sigma(v_i)$, for any $i \in \mathbb{N}$ such that $v_i \in V_\exists$.

The function $\sigma$ is called a *winning strategy* for $\exists$ player (dually $\forall$) with respect to set $W \subseteq V$, if every play (consistent with $\sigma$) starting from a vertex in $W$ is winning for $\exists$ player ($\forall$ resp.).

The problem of solving a parity game consists in deciding whether or not player $\exists$ has a winning strategy. This problem is computationally expensive, due to the large amount of vertices to be checked, it is known that the problem is polynomial time equivalent to the modal $\mu$-calculus model checking. It has been shown that this problem is in *NP* and *Co-NP*, or better yet, *UP* (unambiguous non-deterministic polynomial time) and *Co-UP* as well as in *QP* (quasipolynomial time). Anyway, it is not known whether parity games can be solved in polynomial time.

# 4

# Selections and symbolic ∃-moves

In this chapter we provide one of the core contributions of the thesis. We will introduce the notions of selections and symbolic moves as a way to restrict to a subset of all possible moves for the ∃-player. In particular we will introduce the mathematical concept of progress measure to allow us to formally prove that the game restricted to selections is still sound and complete. Furthermore, properties of selections, and hence symbolic ∃-moves, are studied in an extensive manner. We will provide example all along this chapter to help the reader in the hardest sections. Instead, we decided to keep the number of proofs to a minimum, since some of the most complex ones would be more suitable for a fully theoretical dissertation.

## 4.1 Fixpoint games

In this section, we present a game-theoretical approach to the solution of a system of fixpoint equations over a complete lattice in terms of a parity game. This is based on the fixpoint game described in [Baldan et al., 2020]. Actually, the fixpoint game for our framework was first devised in [Baldan et al., 2018] but it was limited to continuos lattices. The name *fixpoint game* has been adopted from [Hansen et al., 2017].

Given a lattice with a fixed basis and a basis element $b$, the game allow us to

check whether $b$ is smaller than the solution of a selected equation, with respect to $\sqsubseteq$. At a higher level of abstraction, this corresponds to solve the associated verification problem.

Regarding our goal, the game characterization opens the way to development of algorithms for solving the game, and thus the associated verification problem.

**Definition 4.1** (fixpoint game)**.** Let $L$ be a complete lattice and $B_L$ a basis for $L$. Given a system $E$ of $m$ equations over $L$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$, the corresponding *fixpoint game* (called also *powerset game*) is a parity game, with an existential player $\exists$ and a universal player $\forall$, defined as follows:

- The positions of $\exists$ are pairs $(b, i)$ where $b \in B_L, i \in \underline{m}$. Those of $\forall$ are tuples of subsets of the basis $\boldsymbol{X} = (X_1, \ldots, X_m) \in (2^{B_L})^m$.
- From position $(b, i)$ the possible moves of $\exists$ are $\boldsymbol{E}(b, i) = \{\boldsymbol{X} \mid \boldsymbol{X} \in (2^{B_L})^m \wedge b \sqsubseteq f_i(\bigsqcup \boldsymbol{X})\}$.
- From position $\boldsymbol{X} \in (2^{B_L})^m$ the possible moves of $\forall$ are $\boldsymbol{A}(\boldsymbol{X}) = \{(b, i) \mid i \in \underline{m} \wedge b \in X_i\}$.

For a finite play, the winner is the player who moved last. For an infinite play, let $h$ be the highest index that occurs infinitely often in a pair $(b, i)$. If $\eta_h = \nu$ then $\exists$ wins, else $\forall$ wins.

Observe that the fixpoint game is a parity game on an infinite graph and the winning condition is the natural formulation of the standard winning condition in this setting.

The correctness and completeness of the game can be proven by exploiting the connection between the lattice $L$ and the powerset lattice of its basis $2^{B_L}$. The connection is given using an abstraction, that permits us to play in the abstracted game, i.e. in the powerset lattice (that is an algebraic and thus continuos lattice), without loosing neither soundness nor correctness, see Chapter 4 [Baldan et al., 2020]. Then the correctness and completeness of the "concrete" game relies on Theorem 4.8 in [Baldan et al., 2018].

**Theorem 4.2** (correctness and completeness)**.** *Let $E$ be a system of $m$ fixpoint equations over a complete lattice $L$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$ with solution $\boldsymbol{s}$. For all*

$b \in B_L$ *and* $i \in \underline{m}$,

$$b \sqsubseteq s_i \quad \textit{iff} \quad \exists \textit{ has a winning strategy from position } (b, i).$$

*Proof.* The proof can be found in [Baldan et al., 2020]. □

## 4.2 Progress measures

In this section, we introduce the general notion of progress measure for fixpoint games over complete lattices. We will show how a complete progress measure characterizes the winning positions for the two players. In the next section, this fact will be extremely helpful to prove that the relation chosen to reduce the possible moves of the existential player does not affect the correctness and completeness of the game.

Progress measures constitute a technique that associates to each vertex of a parity graph a monotonically increasing measure, by means, following a certain path in the parity graph, the weight associated with each vertex (by this progress measure) is increasing through the path. The concept was firstly introduced in [Jurdziński, 2000] and then further improved in [Hasuo et al., 2016]. Intuitively, the measure of each vertex is lifted based on the measures of its successors, hence progress measure accounts for how favorable the game is for a given player.

Our definition of progress measures applied to fixpoint games are adapted from the Definitions in [Baldan et al., 2018]. In the previous work, progress measures were used for the concrete game.

### 4.2.1 General definition

Given an ordinal $\alpha$ we denote by $[\alpha]^m_\star = \{\beta \mid \beta \leq \alpha\}^m \bigcup \{\star\}$, the set of ordinal vectors with entries smaller than, or equal to, the ordinal $\alpha$, with an added bound $\star$, that intuitively behaves as top.

**Definition 4.3** (progress measure)**.** Let $L$ be a complete lattice and $E$ be a system of $m$ fixpoint equations over $L$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$. Given an ordinal $\lambda$, a $\lambda$-*progress measure* for $E$ is a function $R : B_L \to \underline{m} \to [\lambda]^m_\star$ such that for all $b \in B_L$, $i \in \underline{m}$, either $R(b)(i) = \star$ or there exists $\boldsymbol{X} \in \boldsymbol{E}(b, i)$ such that for all $(b', j) \in \boldsymbol{A}(\boldsymbol{X})$ it holds

- if $\eta_i = \mu$ then $R(b)(i) \succ_i R(b')(j)$;
- if $\eta_i = \nu$ then $R(b)(i) \succeq_i R(b')(j)$

A progress measure maps any element of the basis and index in $\underline{m}$ to an $m$-tuple of ordinals. In words, components relative to $\mu$-equation roughly count how many unfolding steps would be needed to reach an under-approximation $X_i$ above $b$, and thus, for $\exists$ to win the game. Components relative to $\nu$-equations are less relevant, as we will see. Note that, we refer as $R_\perp$ to the $\lambda$-progress measure that associates any basis element and index to 0, i.e., for all $b \in B_L$, $i \in \underline{m}$ it holds that $R_\perp(b)(i) = 0$.

The following lemma expresses the idea that progress measures provide a sound characterization of the solution. Only a sound characterization, not also a complete one, since whenever $R(b)(i) = \star$ we cannot derive any information on the solution, i.e., if $\boldsymbol{s}$ is the solution, we cannot conclude that $b \not\sqsubseteq s_i$.

**Lemma 4.4** (progress measures are strategies)**.** *Let $L$ be a complete lattice and $E$ be a system of $m$ fixpoint equations over $L$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$ with solution $\boldsymbol{s}$. For all $b \in B_L$ and $i \in \underline{m}$, if there exists some ordinal $\lambda$ and a $\lambda$-progress measure $R$ such that $R(b)(i) \preceq_i (\lambda, \ldots, \lambda)$, then $b \sqsubseteq s_i$.*

To be able to use progress measures as a sound and complete tool we also need the following definition, that intuitively adds to the tool the completeness part.

**Definition 4.5** (complete progress measure)**.** Let $L$ be a complete lattice and $E$ be a system of $m$ fixpoint equations over a lattice $L$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$ with solution $\boldsymbol{s}$. A $\lambda$-progress measure $R : B_L \to \underline{m} \to [\lambda]^m_\star$ is called *complete* if for all $b \in B_L$ and $i \in \underline{m}$, if $b \sqsubseteq s_i$ then $R(b)(i) \preceq_i (\lambda, \ldots, \lambda)$.

### 4.2.2 Progress measures as fixpoints

Here we show that a complete progress measure can be characterized as the least solution of a system of equations over tuples of ordinals, naturally induced by Definition 4.3.

**Definition 4.6** (progress measure equations)**.** Let $L$ be a complete lattice and $E$ be a system of $m$ fixpoint equations over $L$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$. Let $\boldsymbol{\delta}^\eta_i$, with

$i \in \underline{m}$, be, for $\eta = \nu$, the null vector and, for $\eta = \mu$, the vector such that $\delta_j = 0$ if $j \neq i$ and $\delta_i = 1$.

The *progress measure equations* for $E$ over the lattice $[\lambda_L]^m_\star$, are defined, for $b \in B_L$, $i \in \underline{m}$, as:

$$R(b)(i) = min_{\preceq_i}\{sup\{R(b')(j) + \boldsymbol{\delta}^{\eta_i}_i \mid (b', j) \in \boldsymbol{A}(\boldsymbol{X})\} \mid \boldsymbol{X} \in \boldsymbol{E}(b, i)\}$$

We will denote by $\Psi_E$ the corresponding endofunction on $L \to \underline{m} \to [\lambda_L]^m_\star$ which is defined for $R : B_L \to \underline{m} \to [\lambda]^m_\star$, by

$$\Psi_E(R)(b)(i) = min_{\preceq_i}\{sup\{R(b')(j) + \boldsymbol{\delta}^{\eta_i}_i \mid (b', j) \in \boldsymbol{A}(\boldsymbol{X})\} \mid \boldsymbol{X} \in \boldsymbol{E}(b, i)\}$$

It is immediate to see that $\Psi_E$ is monotone with respect to such order, i.e., if $R \preceq R'$ pointwise then $\Psi_E(R) \preceq \Psi_E(R')$ pointwise. This allows us to obtain a complete progress measure as a (least) fixpoint of $\Psi_E$.

We next observe that the operator $\Psi_E$ creates monotone functions and, applied to functions that respect joins, it produces functions enjoying the same property. We first introduce the formal definitions.

**Definition 4.7** (monotonicity and sup-respecting)**.** Let $L$ be a complete lattice, $\lambda$ an ordinal and $R : B_L \to \underline{m} \to [\lambda_L]^m_\star$ a $\lambda$-progress measure. The function $R$ is *monotone* if, for all $b, b' \in B_L$ and $i \in \underline{m}$, $b \sqsubseteq b'$ implies $R(b)(i) \preceq R(b')(i)$.

The function $R$ is *sup-respecting* if, for all $b \in B_L$ and $X \subseteq B_L$, $b \sqsubseteq \bigsqcup X$ implies $R(b)(i) \preceq sup\{R(b')(i) \mid b' \in X\}$.

Note that, the notion of monotonicity above is the standard one applied to the pointwise order on $\underline{m} \to [\lambda_L]^m_\star$.

Observe that $R$ is defined only on the basis element, which are possibly (and typically) not closed under joins. The requirements of being sup-respecting ensures that $R$ extends to a function on $L$ which preserves joins. Also note that a sup-respecting function $R$ is always monotone, see Lemma below.

**Lemma 4.8** (sup-respecting implies monotonicity)**.** *Let $L$ be a complete lattice, $\lambda$ an ordinal and $R : B_L \to \underline{m} \to [\lambda_L]^m_\star$ a $\lambda$-progress measure. If $R$ is sup-respecting, then $R$ is monotone.*

*Proof.* Consider the singleton $\{b'\}$ where $b' \in B_L$ such that $b \sqsubseteq b'$, since $R$ is sup-respecting, $R(b)(i) \preceq sup\{R(b')(i)\} = \{R(b')(i)\}$. □

**Lemma 4.9** ($\Psi_E(R)$ is sup-respecting)**.** *Let $E$ be a system of $m$ fixpoint equations over a complete lattice $L$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$. For every function $R : B_L \to \underline{m} \to [\lambda]^m_\star$, the function $\Psi_E(R)$ is sup-respecting.*

*Proof.* We have that $b \sqsubseteq \bigsqcup Y$ by sup-respecting Definition 4.7. We show that

$$\Psi_E(R)(b)(i) \preceq sup\{\Psi_E(R)(b')(i) \mid b' \in X\}$$

Remember that $\Psi_E(R)(b)(i) = \underset{f(\bigsqcup \boldsymbol{X}) \sqsubseteq b}{min_{\preceq_i}} \; \underset{b_j \in X_j}{sup} \{R(b_j)(j) + \boldsymbol{\delta}_i^{\eta_i}\}$. Expanding the progress measure equations definition we obtain:

$$\begin{aligned} \underset{b' \in Y}{sup}\, \Psi_E(R)(b')(i) &= \underset{b' \in Y}{sup} \; \underset{f(\bigsqcup \boldsymbol{X}) \sqsubseteq b'}{min_{\preceq_i}} \; \underset{b_j \in X_j}{sup} \{R(b_j)(j) + \boldsymbol{\delta}_i^{\eta_i}\} \\ &= \underset{\substack{\gamma : Y \to (2^{B_L})^m \\ f(\bigsqcup \gamma(b')) \sqsubseteq b}}{min_{\preceq_i}} \; \underset{b' \in Y}{sup} \; \underset{b_j \in \gamma(b')_j}{sup} \{R(b_j)(j) + \boldsymbol{\delta}_i^{\eta_i}\} \end{aligned}$$

By applying the completely distributive property.

We can conclude by observing that for all $\gamma : Y \to (2^{B_L})^m$ in the expression for $\Psi_E(R)(b)$ one can take

$$\boldsymbol{X} = \bigcup_{y \in Y} \gamma(y)$$

□

**Lemma 4.10** ($\Psi_E(R)$ is monotone)**.** *Let $E$ be a system of $m$ fixpoint equations over a complete lattice $L$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$. For every function $R : B_L \to \underline{m} \to [\lambda]^m_\star$, the function $\Psi_E(R)$ is monotone.*

*Proof.* Given $b \sqsubseteq b'$ in $B_L$, we show that $\Psi_E(R)(b)(i) \preceq \Psi_E(R)(b')(i)$. First, observe that $\boldsymbol{E}(b,i) = \{\boldsymbol{X} \mid b \sqsubseteq f_i(\bigsqcup \boldsymbol{X})\} \supseteq \{\boldsymbol{X} \mid b' \sqsubseteq f_i(\bigsqcup \boldsymbol{X})\} = \boldsymbol{E}(b',i)$, so $\boldsymbol{E}(b',i) \subseteq \boldsymbol{E}(b,i)$. Since $\Psi_E(R)(b)(i) = min_{\preceq_i}\{sup\{R(b')(j) + \boldsymbol{\delta}_i^{\eta_i} \mid (b',j) \in \boldsymbol{A}(\boldsymbol{X})\} \mid \boldsymbol{X} \in \boldsymbol{E}(b,i)\}$, taking the minimum over a larger set results in a smaller vector of ordinals, that is, $min_{\preceq_i}\{g(\boldsymbol{X}) \mid \boldsymbol{X} \in \boldsymbol{E}(b,i)\} \preceq min_{\preceq_i}\{g(\boldsymbol{X}) \mid \boldsymbol{X} \in \boldsymbol{E}(b',i)\}$, where $g(\boldsymbol{X}) = sup\{R(b')(j) + \boldsymbol{\delta}_i^{\eta_i} \mid (b',j) \in \boldsymbol{A}(\boldsymbol{X})\}$. □

## 4.3 Selections

In principle, at least in finite lattices, we are able to compute the solution and thus to prove the property of interest for the verification problem. However, from a complexity point of view, exploring the full game-graph is quite unreasonable, due to the fact that $\exists$-player could play a huge number of moves and trying all these possibilities is quite inefficient. In fact, it can be shown that most of these moves are useless, this is formalized by defining a notion of selection (as done in [Baldan et al., 2018]) which allows one to restrict the possible moves, hence making the calculation more efficient, keeping soundness and completeness of the technique.

First, note that the set of possible moves for the existential player is upward closed with respect to $\sqsubseteq_H$. The notion of selection will help us trying to descend this set as much as possible (as we will see that playing large elements is inconvenient).

Note that, in general we abuse the notion of upward closure $\uparrow_H$ instantiated to the Hoare preorder, formally $\uparrow_H \sigma(b,i) = \{\boldsymbol{X} \in (2^{B_L})^m \mid \exists \boldsymbol{Y} \in \sigma(b,i) \wedge \boldsymbol{Y} \sqsubseteq_H^{\wedge} \boldsymbol{X}\}$.

**Lemma 4.11** (moves for $\exists$ are upward closed with respect to $\sqsubseteq_H$)**.** *Let $E$ be a system of $m$ fixpoint equations over a complete lattice $L$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$. The set $\boldsymbol{E}(b,i)$ is upward closed with respect to $\sqsubseteq_H$, i.e. $\boldsymbol{E}(b,i) = \uparrow_H \boldsymbol{E}(b,i)$.*

*Proof.* We show that, for any $\boldsymbol{X} \in \boldsymbol{E}(b,i)$, whenever $\boldsymbol{X} \sqsubseteq_H^{\wedge} \boldsymbol{Y}$ for some $\boldsymbol{Y}$, then $\boldsymbol{Y} \in \boldsymbol{E}(b,i)$. So, from $\boldsymbol{X} \in \boldsymbol{E}(b,i)$ we have that $b \sqsubseteq f_i(\bigsqcup \boldsymbol{X})$. By hypothesis and Lemma 2.12, we obtain $\boldsymbol{X} \sqsubseteq_{\sqcup}^{\wedge} \boldsymbol{Y}$. So, by Definition 2.10, $\bigsqcup \boldsymbol{X} \sqsubseteq \bigsqcup \boldsymbol{Y}$. By monotonicity of $f_i$, $b \sqsubseteq f_i(\bigsqcup \boldsymbol{X}) \sqsubseteq f_i(\bigsqcup \boldsymbol{Y})$ hence $\boldsymbol{Y} \in \boldsymbol{E}(b,i)$. □

A selection is defined as a set of moves that covers all the moves of the existential player. In the sense that, when we apply the upward closure operator to the selection we obtain the set of all possible moves of $\exists$.

**Definition 4.12** (selections)**.** A *selection* is a function $\sigma : (B_L \times \underline{m}) \to 2^{(2^{B_L})^m}$ such that for all $(b,i) \in (B_L \times \underline{m})$ it holds $\uparrow_H \sigma(b,i) = \boldsymbol{E}(b,i)$.

Observe that according to the definition, a selection is a set of tuples that are moves for the $\exists$ player, such that any other move $\sqsubseteq_H$-dominates one of the moves in the selection. Intuitively, a selection provides a subset of $\boldsymbol{E}(b,i)$ that is sufficient to properly determine the winner.

**Lemma 4.13** ($\sigma$ is monotone)**.** *Let $\sigma : (B_L \times \underline{m}) \to 2^{(2^{B_L})^m}$ be a selection for $E$. For all $b, b' \in B_L$ and $i \in \underline{m}$, if $b \sqsubseteq b'$, then for all $\boldsymbol{Y} \in \sigma(b', i)$ there exists $\boldsymbol{X} \in \sigma(b, i)$ such that $\boldsymbol{X} \sqsubseteq^{\wedge}_{H} \boldsymbol{Y}$.*

*Proof.* Consider $\boldsymbol{Y} \in \sigma(b', i)$, hence $\boldsymbol{Y} \in \uparrow_H \sigma(b', i) = \boldsymbol{E}(b', i) \subseteq \boldsymbol{E}(b, i) = \uparrow_H \sigma(b, i)$. So, by $\boldsymbol{Y} \in \uparrow_H \sigma(b, i)$, there exists $\boldsymbol{X} \in \sigma(b, i)$ such that $\boldsymbol{X} \sqsubseteq^{\wedge}_{H} \boldsymbol{Y}$. $\square$

Observe that the monotonicity in the previous Lemma is not the canonical one. In fact, it does not state that, for any $b, b' \in B_L$ and $i \in \underline{m}$, then $\sigma(b, i) \subseteq \sigma(b', i)$ as one expects. Instead, it talks about the Hoare preorder.

From now on, we will discriminate between the classical notion of progress measure equations devised in Definition 4.6 and the definition with $\exists$-moves restricted to a selection $\sigma$ using a single quotation near the progress measure equations symbol $\Psi'_E(R)$, i.e.,

$$\Psi'_E(R)(b)(i) = min_{\preceq_i}\{sup\{R(b')(j) + \boldsymbol{\delta}^{\eta_i}_i \mid (b', j) \in \boldsymbol{A}(\boldsymbol{X})\} \mid \boldsymbol{X} \in \sigma(b, i)\}$$

Let us continue stating some useful lemmata about the progress measure equations restricted to selections.

**Lemma 4.14** ($\Psi'_E(R)$ is monotone)**.** *For every function $R : B_L \to \underline{m} \to [\lambda]^m_\star$, the function $\Psi'_E(R)$ is monotone.*

**Lemma 4.15** ($\Psi_E(R)$ coincides with $\Psi'_E(R)$ for monotone R's)**.** *Whenever $R : B_L \to \underline{m} \to [\lambda]^m_\star$ is monotone, then $\Psi_E(R) = \Psi'_E(R)$.*

*Proof.* We show that, for a given monotone $R$, $\Psi_E(R) = \Psi'_E(R)$ by showing that $\Psi_E(R)(b)(i) =_i \Psi'_E(R)(b)(i)$ for every $b \in B_L$ and $i \in \underline{m}$. So, let $b \in B_L$ and $i \in \underline{m}$. Let us write

$$\boldsymbol{\beta}_b = \Psi'_E(R)(b)(i) = min_{\preceq_i}\{sup\{R(b')(j) + \boldsymbol{\delta}^{\eta_i}_i \mid (b', j) \in \boldsymbol{A}(\boldsymbol{X})\} \mid \boldsymbol{X} \in \sigma(b, i)\}$$

$$\boldsymbol{\gamma}_b = \Psi_E(R)(b)(i) = min_{\preceq_i}\{sup\{R(b')(j) + \boldsymbol{\delta}^{\eta_i}_i \mid (b', j) \in \boldsymbol{A}(\boldsymbol{X})\} \mid \boldsymbol{X} \in \boldsymbol{E}(b, i)\}$$

and we show $\boldsymbol{\beta}_b =_i \boldsymbol{\gamma}_b$. So, the fact that $\boldsymbol{\gamma}_b \preceq_i \boldsymbol{\beta}_b$ follows from the observation that, by Definition 4.12, $\sigma(b, i) \subseteq \boldsymbol{E}(b, i)$, i.e, the first is a minimum over a smaller set. The converse inequality $\boldsymbol{\beta}_b \preceq_i \boldsymbol{\gamma}_b$ holds from the fact that, by Definition 4.12, for each $b \in B_L$, $i \in \underline{m}$ if $\boldsymbol{X} \in \boldsymbol{E}(b, i)$ then there exists $\boldsymbol{Y} \in \sigma(b, i)$ such that

$\boldsymbol{Y} \sqsubseteq^{\wedge}_{H} \boldsymbol{X}$ (i.e, for all $i \in \underline{m}$, $\forall b' \in Y_i$, $\exists b'' \in X_i$ such that $b' \sqsubseteq b''$). Hence, for all $(b', i) \in \boldsymbol{A}(\boldsymbol{Y})$ there exists $(b'', i) \in \boldsymbol{A}(\boldsymbol{X})$ such that $b' \sqsubseteq b''$. Thus,

$$sup\{R(b')(j) + \boldsymbol{\delta}_i^{\eta_i} \mid (b', j) \in \boldsymbol{A}(\boldsymbol{Y})\} \preceq_i sup\{R(b')(j) + \boldsymbol{\delta}_i^{\eta_i} \mid (b', j) \in \boldsymbol{A}(\boldsymbol{X})\}$$

since $R$ is monotone by hypothesis. □

Since a complete progress measures witnesses the existence of a winning strategy for ∃, the next theorem implies that whenever ∃ has a winning strategy, it has one also in the game where the moves of ∃ are restricted to be in the selection.

**Theorem 4.16** (game with selections)**.** *Let $E$ be a system of $m$ fixpoint equations over a complete lattice $L$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$. Let $\sigma : (B_L \times \underline{m}) \to 2^{(2^{B_L})^m}$ be a selection for $E$. The system of progress measures equations over $[\lambda_L]^m_\star$, defined, for $b \in B_L$, $i \in \underline{m}$, as:*

$$\Psi'_E(R)(b)(i) = min_{\preceq_i}\{sup\{R(b')(j) + \boldsymbol{\delta}_i^{\eta_i} \mid (b', j) \in \boldsymbol{A}(\boldsymbol{X})\} \mid \boldsymbol{X} \in \sigma(b, i)\}$$

*has the same least solution as the original one in Definition 4.6.*

*Proof.* To compute the least fixpoint of $\Psi_E$ we start from $R_\perp$ and we (transfinitely) iterate until we reach the fixpoint $\mu\Psi_E = {\Psi_E}^{\gamma}(R_\perp)$ for some ordinal $\gamma \leq \lambda_L$. Since $\Psi_E$ is monotone (Lemma 4.10), by Knaster-Tarsky's theorem [Tarski, 1955], the existence of such $\gamma$ is ensured. We show by transfinite induction that, for every ordinal $\alpha$, ${\Psi_E}^{\alpha}(R_\perp) = {\Psi'_E}^{\alpha}(R_\perp)$.

($\alpha = 0$) In the base case, we have ${\Psi_E}^{0}(R_\perp) = R_\perp = {\Psi'_E}^{0}(R_\perp)$ by definition.

($\alpha = 1$) In the second base case, we have ${\Psi_E}^{1}(R_\perp) = \Psi_E(R_\perp)$. Hence, $\Psi_E(R_\perp) = \Psi'_E(R_\perp)$ by Lemma 4.15, since $R_\perp$ is monotone.

($\alpha = (\beta + 1) + 1$) In the inductive step,

$${\Psi_E}^{\alpha}(R_\perp) = \Psi_E({\Psi_E}^{\beta+1}(R_\perp)) = \Psi_E(\Psi_E({\Psi_E}^{\beta}(R_\perp)))$$

From Lemma 4.10, we know that $\Psi_E({\Psi_E}^{\beta}(R_\perp)) = {\Psi_E}^{\beta+1}(R_\perp)$ is monotone. By inductive hypothesis we know that ${\Psi_E}^{\beta+1}(R_\perp) = {\Psi'_E}^{\beta+1}(R_\perp)$. Hence,

$$\Psi_E({\Psi_E}^{\beta+1}(R_\perp)) = \Psi_E({\Psi'_E}^{\beta+1}(R_\perp)) = \Psi'_E({\Psi'_E}^{\beta+1}(R_\perp))$$

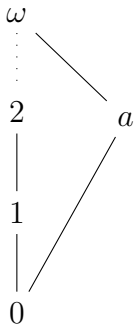


Figure 4.1: non-continuos lattice

where the last step is motivated by Lemma 4.15.

($\alpha$ **limit ordinal**) In this case, ${\Psi_E}^{\alpha}(R_\perp) = \bigsqcup_{\beta<\alpha} {\Psi_E}^{\beta}(R_\perp) = \bigsqcup_{\beta<\alpha} {\Psi'_E}^{\beta}(R_\perp) = {\Psi'_E}^{\alpha}(R_\perp)$, since from the inductive hypothesis we know that ${\Psi_E}^{\beta}(R_\perp) = {\Psi'_E}^{\beta}(R_\perp)$, for all $\beta < \alpha$.

□

**Remark 2** (game with selections, with respect to $\uparrow_\sqcup$)**.** In this remark we show, first an example where a weaker selection $\sigma^\sqcup$ could lead to a wrong result. Second, we show a counterexample for Lemma 4.15 using the same weaker selection.

Consider the (complete) lattice $W = \mathbb{N} \bigcup \{\omega, a\}$ in Figure 4.1. The basis for $W$ is defined $B_W = \mathbb{N}^+ \bigcup \{a\}$. The system $E$ considered is a single equation system, hence for this example we will avoid indexes. The system $E$ is defined as $x =_\mu f(x)$, where

$$f(x) = \begin{cases} x+1 & \text{if } x \in \mathbb{N} \\ \omega & \text{otherwise} \end{cases}$$

Suppose we are interested in checking $a \sqsubseteq \mu f$, which is true.

The weaker selection $\sigma^\sqcup$ is defined as

$$\sigma^\sqcup(b) = \begin{cases} \{(\{a\})\} & \text{if } b = a \\ \{(\{b-1\})\} & \text{if } b \in \mathbb{N}^+ \setminus \{1\} \\ \{(\emptyset)\} & \text{if } b = 1 \end{cases}$$

It can be seen that $\sigma^\sqcup$ complies with the condition $\forall b \in B_W, \uparrow_\sqcup \sigma^\sqcup(b) = \boldsymbol{E}(b)$.

1. We show that if $\exists$ could play only moves in the weak selection, $\forall$ has a winning strategy even though $\exists$ should win instead. Similarly to Example 6.2 [Baldan et al., 2020], in the powerset game from the position $(a)$, $\exists$ wins descending the chain $\mathbb{N}^+ \to (n \in \mathbb{N}^+) \to \{n-1\} \to \cdots \to (1) \to \emptyset$ and $\forall$ is stuck hence $\exists$ wins (note that every infinite play leads to a winning strategy for $\forall$ since the only equation in the system requires a least fixpoint), more details can be found in the referred Example 6.2 [Baldan et al., 2020].

   Whenever $\exists$-moves are restricted to the selection $\sigma^{\sqcup}$ above, it can be shown that $\forall$ wins. Indeed if, the game starts from $(a)$, $\exists$ now can only play $(\{a\})$. Then, $\forall$ plays again $(a)$ leading to an infinite play won by $\forall$, since $\eta_1 = \mu$, although we know that $\exists$ should have won because $a \sqsubseteq \mu f$.

2. We show also that, given a monotone $R$ , $\Psi_E(R) \neq \Psi^{\sqcup}_E(R)$, where $\Psi^{\sqcup}_E(R)$ is defined as $\Psi'_E(R)$ but restricted to $\sigma^{\sqcup}$. The function $R : B_W \to [\omega]_\star$ is defined as $R(a) = \omega$ and $R(n) = n$ is clearly monotone and it respects Definition 4.3.

   We show that, for $b = a$, $\Psi_E(R)(a) \neq \Psi^{\sqcup}_E(R)(a)$, that is

$$\begin{aligned}
\Psi_E(R)(a) &= min\{sup\{R(b') + (1) \mid (b') \in \boldsymbol{A}(\boldsymbol{X})\} \mid \boldsymbol{X} \in \boldsymbol{E}(a)\} \\
&= min\{sup\{R(b') + (1) \mid (b') \in \boldsymbol{A}(\boldsymbol{X})\} \mid \boldsymbol{X} \in \{(\mathbb{N}), (\{a, 1\}), \dots\}\} \\
&= min\{sup\{R(n) + (1) \mid n \in \boldsymbol{A}((\mathbb{N}))\}, \\
&\qquad\qquad sup\{R(1) + (1), R(a) + (1)\}, \dots\} \\
&= min\{\omega, \star = sup\{(2), \star\}, \dots\} \\
&= \omega
\end{aligned}$$

   Equivalently, we show that $\Psi^{\sqcup}_E(R)(a) \neq \omega$, thus they do not coincide.

$$\begin{aligned}
\Psi^{\sqcup}_E(R)(a) &= min\{sup\{R(b') + (1) \mid (b') \in \boldsymbol{A}(\boldsymbol{X})\} \mid \boldsymbol{X} \in \sigma^{\sqcup}(a) = \{(\{a, 1\})\}\} \\
&= sup\{R(b') + (1) \mid (b') \in \boldsymbol{A}((\{a, 1\}))\} \\
&= sup\{R(a) + (1), R(1) + (1)\} \\
&= sup\{\star, (2)\} \\
&= \star
\end{aligned}$$

Observe that, since we have only one $\mu$-equation in $E$, $\boldsymbol{\delta}_i^{\eta_i} = (1)$ and $\min_{\preceq_1} \equiv \min$.

Clearly, for computational purposes, it could be convenient to consider selections that are minimal with respect to some criterion. We define a relation (actually a pre-order) that allow us to formally characterize a concept of minimality, with respect to this order.

**Definition 4.17** (selection order $\sqsubseteq_{H_H}$)**.** Let $\sigma$ and $\sigma'$ be selections for a system $E$. Let us define a relation between selections $\sqsubseteq_{H_H}$. We write $\sigma \sqsubseteq_{H_H} \sigma'$ if for all positions $(b, i) \in (B_L \times \underline{m})$, for all $\boldsymbol{X} \in \sigma(b, i)$ there exists $\boldsymbol{Y} \in \sigma'(b, i)$ such that $\boldsymbol{X} \sqsubseteq_H^{\wedge} \boldsymbol{Y}$.

Later in this section, we will provide another relation to compare selections, the main difference between the two definitions is that the first one allows us to work with any kind of complete lattices (and thus, on those that are infinite). The other definition instead, it restricted to finite height lattices, but, on such specific subclass, it is more effective. Practically, it is the finest definition we can obtain between selections, in finite height lattices.

**Lemma 4.18** ($\sqsubseteq_{H_H}$ is a pre-order)**.** *Let $\sigma$, $\sigma'$ and $\sigma''$ be selections for $E$. The followings holds:*

- *Reflexivity, $\sigma \sqsubseteq_{H_H} \sigma$, and*
- *Transitivity, if $\sigma \sqsubseteq_{H_H} \sigma'$ and $\sigma' \sqsubseteq_{H_H} \sigma''$ then $\sigma \sqsubseteq_{H_H} \sigma''$*

*Proof.* We show that $\sqsubseteq_{H_H}$ is a preorder by showing that the two properties hold.

**(Reflexivity)** For every position $(b, i)$, we show that for all $\boldsymbol{X} \in \sigma(b, i)$ there exists $\boldsymbol{Y} \in \sigma(b, i)$ such that $\boldsymbol{X} \sqsubseteq^{\wedge} \boldsymbol{Y}$. Take $\boldsymbol{Y} = \boldsymbol{X} \in \sigma(b, i)$ and $\boldsymbol{X} \sqsubseteq^{\wedge} \boldsymbol{X}$.

**(Transitivity)** From the first hypothesis, we have that for every position $(b, i)$, for all $\boldsymbol{X} \in \sigma(b, i)$ there exists $\boldsymbol{Y} \in \sigma'(b, i)$ such that $\boldsymbol{X} \sqsubseteq^{\wedge} \boldsymbol{Y}$. From the second hypothesis, we have that for every position $(b, i)$, for all $\boldsymbol{Y'} \in \sigma'(b, i)$ there exists $\boldsymbol{Z'} \in \sigma''(b, i)$ such that $\boldsymbol{Y'} \sqsubseteq^{\wedge} \boldsymbol{Z'}$. We show that, for every position $(b, i)$, for all $\boldsymbol{X} \in \sigma(b, i)$ there exists $\boldsymbol{Z} \in \sigma''(b, i)$ such that $\boldsymbol{X} \sqsubseteq^{\wedge} \boldsymbol{Z}$.

From $\boldsymbol{X} \in \sigma(b, i)$ we have that there exists $\boldsymbol{Y} \in \sigma'(b, i)$ such that $\boldsymbol{X} \sqsubseteq^{\wedge} \boldsymbol{Y}$. By the second hypothesis, there exists $\boldsymbol{Z} \in \sigma''(b, i)$ such that $\boldsymbol{Y} \sqsubseteq^{\wedge} \boldsymbol{Z}$, hence $\boldsymbol{X} \sqsubseteq^{\wedge} \boldsymbol{Y} \sqsubseteq^{\wedge} \boldsymbol{Z}$. We conclude by transitivity of $\sqsubseteq$, Lemma 2.13.

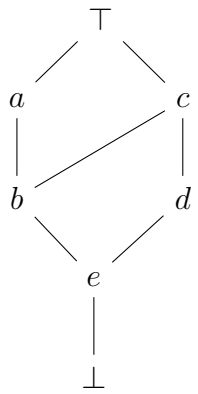


Figure 4.2: Finite distributive lattice $L$

□

As anticipated, over finite lattices this is not the finest order we could obtain, Example 6 explains why over finite lattices we would look for a stronger order definition.

**Example 6** ($\sqsubseteq_{H_H}$ over finite lattices)**.** Consider the complete lattice $L$ drawn in Figure 4.2. Let $B_L = \{a, b, c, d, e\}$ be the basis. Consider the trivial system with a single equation $x =_\mu f(x)$ where $f = id$. The set of moves is

$$\boldsymbol{E}(c) = \{\{c\}, \{b, d\}, \{a, b, d\}, \{b, c, d\}, \{a, d\}, \{b, d, e\}, \dots\}$$

We are interested in finding the minimal selection $\hat{\sigma}$ such that $\uparrow_H \hat{\sigma}(c) = \boldsymbol{E}(c)$. Ideally the minimal move would be $\{b, d\}$ since in $L$ there is no other move that has the join above $c$ and elements below $\{b, d\}$. Hence, the minimal selection that could represents the set $\boldsymbol{E}(c, 1)$ would be just $\hat{\sigma}(c) = \{\{b, d\}\}$.

Instead, with the order defined above some superset of $\{b, d\}$ could appear in the minimal selection. In other words, the set $\sigma'(c) = \{\{b, d, e\}\}$ could be a possible minimal selection, that is, for any $\sigma''$ selection then $\sigma' \sqsubseteq_{H_H} \sigma''$. In particular, $\sigma' \sqsubseteq_{H_H} \hat{\sigma}$ since $\{b, d, e\} \sqsubseteq_H \{b, d\}$ by the fact that $e \sqsubseteq b$ (or equivalently, $e \sqsubseteq d$).

On the other hand, we could not get any improvements to this order over infinite lattices. In fact, using this preorder $\sqsubseteq_{H_H}$, a minimal selection is always obtainable, even in infinite lattices (assuming some condition over equation $f$'s, see below Lemma 4.19 and Example 7). Obviously, the minimal selection retrieved

using $\sqsubseteq_{H_H}$ is not the least one, in the sense that we obtain multiple different minimal selections.

**Remark 3** ($\omega$-chains)**.** In the following we will refer to the notion of $\omega$-chain, from [Abramsky and Jung, 1994]. As a remark, a chain is a totally ordered subset of a lattice. An $\omega$-chain is a chain isomorphic to $(\mathbb{N}, \leq)$, and thus a countable structure.

**Lemma 4.19** (minimal selection, with respect to $\sqsubseteq_{H_H}$)**.** *Let $E$ be a system of $m$ fixpoint equations over a complete lattice $L$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$. A minimal selection $\hat{\sigma}$ exists if for all $i \in \underline{m}$, $f_i \circ \bigsqcup$ preserves the infimum of descending $\omega$-chains with respect to $\sqsubseteq_H$.*

*Proof.* Assume that $f_i \circ \bigsqcup$ preserves the infimum of descending $\omega$-chains, with respect to $\sqsubseteq_H$. First observe that given a descending $\omega$-chain $(\boldsymbol{X}_\alpha)_\alpha$ in $\boldsymbol{E}(b,i)$ we have that $\bigsqcap_\alpha \boldsymbol{X}_\alpha \in \boldsymbol{E}(b,i)$. In fact, for each $\alpha$ we have $b \sqsubseteq (f_i \circ \bigsqcup)(\boldsymbol{X}_\alpha)$ and thus $b \sqsubseteq \bigsqcap_\alpha((f_i \circ \bigsqcup)(\boldsymbol{X}_\alpha)) = (f_i \circ \bigsqcup)(\bigsqcap_\alpha \boldsymbol{X}_\alpha))$. Note that, the infimum on the right-hand side of the last equation is referring to the Hoare order applied pointwise.

The above implies that for each $\boldsymbol{X} \in \boldsymbol{E}(b,i)$ there exists $\boldsymbol{X}' \in \boldsymbol{E}(b,i)$, minimal, such that $\boldsymbol{X}' \sqsubseteq_H^\wedge \boldsymbol{X}$. In fact, consider the (possibly transfinite) chain of tuples $\boldsymbol{X}_\alpha$ in $\boldsymbol{E}(b,i)$ defined as follows. Start from $\boldsymbol{X}_0 = \boldsymbol{X}$. For any ordinal $\alpha$, if there is $\boldsymbol{X}' \in \boldsymbol{E}(b,i)$ such that $\boldsymbol{X}' \neq \boldsymbol{X}$ and $\boldsymbol{X}' \sqsubseteq_H^\wedge \boldsymbol{X}$, let $\boldsymbol{X}_{\alpha+1} = \boldsymbol{X}'$. If $\alpha$ is a limit ordinal, then $\boldsymbol{X}_\alpha = \bigsqcap_{\beta<\alpha} \boldsymbol{X}_\beta$, as before, the infimum is referring to the Hoare order applied pointwise.

This is a strictly descending chain, that thus necessarily stops at some ordinal $\lambda$ bounded by the length of the longest descending chain in $L$. By construction and the observation above $\boldsymbol{X}_\lambda \in \boldsymbol{E}(b,i)$ and it is minimal in $\boldsymbol{E}(b,i)$. Define $\hat{\sigma}(b,i)$ as the set of minimal elements of $\boldsymbol{E}(b,i)$ for each $(b,i) \in B_L \times \underline{m}$. It is immediate to see that this is a selection, and that it is minimal in the sense of $\sqsubseteq_{H_H}$. □

We introduce an explanatory example to clarify the role of the hypothesis in the lemma above.

**Example 7** (minimal selections might not exist)**.** To understand the existence of minimal selections with respects to $\sqsubseteq_{H_H}$, consider the lattice $L$ in Figure 4.3,

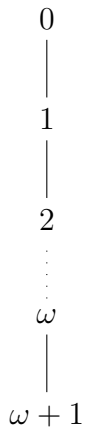


Figure 4.3: Reverse ordinal sublattice

suppose that $B_L = \mathbb{N} \bigcup \{\omega\}$. In this case we have the descending $\omega$-chain, with respect to $\sqsubseteq_H$

$$\{0\} \sqsupseteq_H \{1\} \sqsupseteq_H \{2\} \sqsupseteq_H \dots$$

We show an example in which $f : L \to L$, given below, does not preserve the infimum of the $\omega$-chain described above.

$$f(x) = \begin{cases} x & \text{if } x \in \mathbb{N} \\ \omega + 1 & \text{otherwise} \end{cases}$$

Applying the supremum at each element of the $\omega$-chain we obtain the following descending $\omega$-chain (with respect to $\sqsubseteq$):

$$\bigsqcup\{0\} \sqsupseteq \bigsqcup\{1\} \sqsupseteq \bigsqcup\{2\} \sqsupseteq \dots$$

So, regarding Lemma 4.19, $f$ has to preserve the infimum of this last $\omega$-chain presented. In our case study, $f$ does not preserve the infimum, since the infimum of the $\omega$-chain is $\omega$ and $f(\omega) = \omega + 1$. Hence, a minimal selection does not exists. Intuitively whenever we are interested in moves above $\omega$, e.g., $\boldsymbol{E}(\omega, 1)$ with the $f$ described above, we cannot find the minimal one since it is equivalent to find the maximum of $\mathbb{N}$.

### 4.3.1 Selections over finite lattices

For computational purposes, it is useful to restrict to finite (height) lattices. With this restriction, we are able to provide a finer order relation between selections. In the rest of this section all considered lattices will be of finite height.

**Definition 4.20** (selection order $\subseteq_H$)**.** For a finite height lattice $L$, let $\sigma$ and $\sigma'$ be selections for a system $E$. Let us define a relation between selections $\subseteq_H$. We write $\sigma \subseteq_H \sigma'$ if for all positions $(b, i) \in (B_L \times \underline{m})$ it holds that $\sigma(b, i) \subseteq_H^{\wedge} \sigma'(b, i)$, i.e., for all $\boldsymbol{X} \in \sigma(b, i)$ there exists $\boldsymbol{Y} \in \sigma'(b, i)$ such that $\boldsymbol{X} \subseteq^{\wedge} \boldsymbol{Y}$.

The intuition behind the above definition is that, to check if a given selection is smaller than another we just need to compare component-wise the corresponding elements using Hoare (with respect to the subset inclusion).

At a first glance this definition could look simplistic. In fact, it does not mention the upward closure or about the Hoare order $\sqsubseteq_H$. For instance, consider two selections $\sigma, \sigma'$ such that $\sigma \subseteq_H \sigma'$. Let $X, Y \in 2^{B_L}$ be moves such that $X \not\subseteq Y \not\subseteq X$, in words $X$ and $Y$ are incomparable. Whenever $X \sqsubseteq_H Y$, we would like to have $X$ inside our smaller selection $\sigma$, instead of $Y$.

More formally, the reason why we are able to keep this definition so simple is that:

- Regarding the absence of the Hoare order $\sqsubseteq_H$, if $\sigma(b, i)$ contains $\boldsymbol{X}$ such that, for some $\boldsymbol{Y} \in \boldsymbol{E}(b, i)$, $\boldsymbol{Y} \sqsubseteq_H^{\wedge} \boldsymbol{X}$ and $\boldsymbol{X} \not\sqsubseteq_H^{\vee} \boldsymbol{Y}$ (in words, $\boldsymbol{Y}$ is strictly lower than $\boldsymbol{X}$ with respect to $\sqsubseteq_H$ pointwise). Then, there exists $\boldsymbol{Y}' \in \sigma(b, i)$ such that $\boldsymbol{Y}' \sqsubseteq^{\wedge} \boldsymbol{Y}$ since $\uparrow_H \sigma(b, i) = \boldsymbol{E}(b, i)$. Hence, we do not need to discriminate by this criteria in the relation $\subseteq_H$.

- Regarding the absence of the upward closure, for all $\sigma$ and $\sigma'$ selections for $E$, $\uparrow_H \sigma(b, i) = \uparrow_H \sigma'(b, i) = \boldsymbol{E}(b, i)$, for every position $(b, i)$. Hence, we do not need to check relations between the two upward closures, with respect to $\sqsubseteq_H$, since they will always coincide.

**Lemma 4.21** ($\subseteq_H$ is a pre-order)**.** *Let $\sigma$, $\sigma'$ and $\sigma''$ be selections for $E$. Let us consider the relation between selections ($\subseteq_H$). The followings holds:*

- *Reflexivity, $\sigma \subseteq_H \sigma$, and*

- *Transitivity, if* $\sigma \subseteq_H \sigma'$ *and* $\sigma' \subseteq_H \sigma''$ *then* $\sigma \subseteq_H \sigma''$

*Proof.* We show that $\subseteq_H$ is a preorder by showing that the two properties hold.

**(Reflexivity)** For every position $(b, i)$, we show that $\sigma(b,i) \subseteq^{\wedge}_H \sigma(b,i)$ by reflexivity of $\sqsubseteq_H$ applied to $\subseteq$.

**(Transitivity)** From the first hypothesis, we have that for every position $(b, i)$, for all $\boldsymbol{X} \in \sigma(b,i)$ there exists $\boldsymbol{Y} \in \sigma'(b,i)$ such that $\boldsymbol{X} \subseteq^{\wedge} \boldsymbol{Y}$. From the second hypothesis, we have that for every position $(b, i)$, for all $\boldsymbol{Y'} \in \sigma'(b,i)$ there exists $\boldsymbol{Z'} \in \sigma''(b,i)$ such that $\boldsymbol{Y'} \subseteq^{\wedge} \boldsymbol{Z'}$. We show that, for every position $(b, i)$, for all $\boldsymbol{X} \in \sigma(b,i)$ there exists $\boldsymbol{Z} \in \sigma''(b,i)$ such that $\boldsymbol{X} \subseteq^{\wedge} \boldsymbol{Z}$.

From $\boldsymbol{X} \in \sigma(b,i)$ we have that there exists $\boldsymbol{Y} \in \sigma'(b,i)$ such that $\boldsymbol{X} \subseteq^{\wedge} \boldsymbol{Y}$. By the second hypothesis, there exists $\boldsymbol{Z} \in \sigma''(b,i)$ such that $\boldsymbol{Y} \subseteq^{\wedge} \boldsymbol{Z}$, hence $\boldsymbol{X} \subseteq^{\wedge} \boldsymbol{Y} \subseteq^{\wedge} \boldsymbol{Z}$. We conclude by transitivity of $\subseteq$.

□

From now on, selections will be always ordered with respect to $\subseteq_H$, so we will avoid to mention this every time, since there might be some concerns between $\subseteq_H$ and $\sqsubseteq_{H_H}$. Also when we are talking about minimal or least selections, the order relation considered will be always $\subseteq_H$.

**Lemma 4.22** (subset inclusion implies $\subseteq_H$)**.** *Let* $\sigma$ *and* $\sigma'$ *be selections. For all positions* $(b, i) \in (B_L \times \underline{m})$*, if* $\sigma(b,i) \subseteq \sigma'(b,i)$ *then* $\sigma \subseteq_H \sigma'$*.*

*Proof.* This is obvious since, by means of subset inclusion (Definition 2.9), whenever $\boldsymbol{X} \in \sigma(b,i)$ then $\boldsymbol{X} \in \sigma'(b,i)$ and $\boldsymbol{X} \subseteq^{\wedge} \boldsymbol{X}$. □

**Lemma 4.23** (the least selection does not contain repetitions)**.** *Let* $\hat{\sigma}$ *be the least selection for* $E$*. For all positions* $(b, i) \in (B_L \times \underline{m})$ *and* $\boldsymbol{X}, \boldsymbol{Y} \in \hat{\sigma}(b,i)$ *such that* $\boldsymbol{X} \neq \boldsymbol{Y}$*, it holds that* $\boldsymbol{X}$ *is* $\subseteq^{\wedge}$*-incomparable with* $\boldsymbol{Y}$*.*

*Proof.* Suppose we have at least two elements in $\hat{\sigma}(b,i)$. Otherwise, with just one element in $\hat{\sigma}(b,i)$, it would be easy too see that there are no repetitions. So, consider $\boldsymbol{X} \in \hat{\sigma}(b,i)$, suppose there exists another $\boldsymbol{Y} \in \hat{\sigma}(b,i)$ such that $\boldsymbol{Y} \subseteq^{\wedge} \boldsymbol{X}$ $(*)$, in words, we suppose that $\boldsymbol{Y}$ is included in $\boldsymbol{X}$ (and thus comparable with).

Let us define $\hat{\sigma}_1(b,i) = \hat{\sigma}(b,i) \backslash \{\boldsymbol{X}\}$, and $\hat{\sigma}(b',j) = \hat{\sigma}_1(b',j)$ on all other positions $(b', j)$. We show that $\hat{\sigma}_1$ is a proper selection (1) and that $\hat{\sigma}_1 \subseteq_H \hat{\sigma}$ (2).

(1) we show that $\uparrow_H \hat{\sigma}_1(b,i) = \boldsymbol{E}(b,i)$. Consider any $\boldsymbol{Y'} \in \boldsymbol{E}(b,i)$, if there exists $\boldsymbol{X'} \in \hat{\sigma}(b,i)$ such that $\boldsymbol{X} \neq \boldsymbol{X'}$ and $\boldsymbol{X'} \sqsubseteq^{\wedge}_H \boldsymbol{Y'}$, then we conclude by $\hat{\sigma}_1$ definition, that is, $\hat{\sigma}_1(b,i) = \hat{\sigma}(b,i) \setminus \{\boldsymbol{X}\}$ and so $\boldsymbol{X'} \in \hat{\sigma}_1(b,i)$. Otherwise, we are in the case where $\boldsymbol{X}$ is the element that cover $\boldsymbol{Y'}$, i.e., $\boldsymbol{X} \sqsubseteq^{\wedge}_H \boldsymbol{Y'}$. By $(*)$ we know that there exists $\boldsymbol{Y} \in \hat{\sigma}_1(b,i)$ such that $\boldsymbol{Y} \subseteq^{\wedge} \boldsymbol{X}$ and by Lemma 2.11 we have that $\boldsymbol{Y} \sqsubseteq^{\wedge}_H \boldsymbol{X}$. By transitive property we obtain the following chain $\boldsymbol{Y} \sqsubseteq^{\wedge}_H \boldsymbol{X} \sqsubseteq^{\wedge}_H \boldsymbol{Y'}$ that concludes case (1).

(2) we show that $\hat{\sigma}_1 \subseteq_H \hat{\sigma}$ by Lemma 4.22 since $\hat{\sigma}_1(b,i) \subseteq \hat{\sigma}(b,i)$.

So with the two points above we just proved that $\hat{\sigma}_1$ is a selection and it is lower than $\hat{\sigma}$ with respect to $\subseteq_H$, that is impossible since $\hat{\sigma}$ is the least selection. Thus, the only assumption made is $(*)$ which is false. □

Intuitively, the property that the least selection does not contain any repetition holds because every time we find a repetition in a selection, we can remove it to obtain a smaller selection, in the sense of $\subseteq_H$, until we reached the least one, in which there are no repetitions.

**Theorem 4.24** (least selection)**.** *Let $E$ be a system of $m$ fixpoint equations over a complete lattice $L$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$. For a finite height lattice $L$. The least selection $\hat{\sigma}$ with respect to $\subseteq_H$ always exists and it is unique.*

*Proof.* Since we restrict ourselves to finite height lattices, there not exists either infinite descending or ascending $\omega$-chain. Thus, the set of minimal moves of $\boldsymbol{E}(b,i)$, with respect to $\subseteq_H$, exists and it could always be used to define a minimal selection.

Moreover, it is the least selection. In fact, let $\hat{\sigma}_1$ be another minimal selection for $E$. So, we have that $\hat{\sigma} \subseteq_H \hat{\sigma}_1$ $(*)$ and $\hat{\sigma}_1 \subseteq_H \hat{\sigma}$ $(**)$ since they are both minimal. It holds that, for all $\boldsymbol{X} \in \hat{\sigma}(b,i)$ there exists $\boldsymbol{Y} \in \hat{\sigma}_1(b,i)$ such that $\boldsymbol{X} \subseteq^{\wedge} \boldsymbol{Y}$ from $(*)$. In addition, there exists $\boldsymbol{X'} \in \hat{\sigma}(b,i)$ such that $\boldsymbol{Y} \subseteq^{\wedge} \boldsymbol{X'}$ from $(**)$ applied to $\boldsymbol{Y}$.

We face two cases, the first is $\boldsymbol{X} = \boldsymbol{X'}$, hence $\boldsymbol{X} \subseteq^{\wedge} \boldsymbol{Y} \subseteq^{\wedge} \boldsymbol{X'}$ implies that $\boldsymbol{X} = \boldsymbol{Y}$ for every $\boldsymbol{X}$. Dually, we obtain the same result for every $\boldsymbol{Y}$, starting from $(**)$ and then using $(*)$.


Figure 4.4: Ordinal sublattice

Second case, $\boldsymbol{X} \neq \boldsymbol{X'}$, but $\boldsymbol{X} \subseteq^{\wedge} \boldsymbol{Y} \subseteq^{\wedge} \boldsymbol{X'}$ so it has to hold that $\boldsymbol{X} \subsetneq^{\vee} \boldsymbol{X'}$, i.e., there exists $i \in \underline{m}$ such that $X_i \subsetneq X'_i$. Since, the least selection does not contain repetitions by Lemma 4.23, the second case is falsified. □

We now expose why this order relation between selections is not effective with infinite ascending $\omega$-chains.

**Example 8** (ascending $\omega$-chains)**.** To understand the existence of the least selection consider the lattice with all the natural numbers plus the first limit ordinal $\omega$, shown in Figure 4.4.

Whenever we are interested in moves that could cover $\omega$ with the supremum, $\subseteq_H$ could not find a minimal move for $\exists$-player anymore.

This happens because any infinite $X \subseteq \mathbb{N}$ could cover $\omega$ since $\bigsqcup X = \omega$. But using our selections order, we only check for subset inclusion. Thus, given $X$ we could always find an infinite $Y \subseteq X$ obtained as $Y = X \setminus \{x\}$ for $x \in X$. And $Y$ would be minimal in the sense of $\subseteq_H$, this is due to the fact that for all $y \in Y$, by construction, $y \in X$ and $y \subseteq y$, that is, $Y \subseteq_H X$.

This happens when we have ascending $\omega$-chains as: $0 \sqsubseteq 1 \sqsubseteq 2 \sqsubseteq \ldots$ in the lattice, and we are interested in moves below the limit ordinal $\omega$.

The next lemma helps us in the setting of distributive lattices, where in the least selection, for each $\sim_{\sqcup}$ class (class of equivalence with respect to the supremum order, see Definition 2.7) there is exactly one element. Whereas, in a non-distributive context, for each $\sim_{\sqcup}$ class, we possibly have more than only one element. Thus, in the powerset game over frames (Definition 2.16), the possible

moves for the $\exists$ player are bounded by the number of classes of equivalence, that is a way lower than all the possible permutations of lattice elements.

Notably, we did not directly use the standard definition of set of classes of equivalence defined in Definition2.7. Since $\hat{\sigma}(b,i)$ is a set of tuples we will partition it component by component, we will do so using the relative component index as superscript. By means, for $j \in \underline{m}$, $\hat{\sigma}(b,i)/^{j}_{\sim_{\sqcup}} = (\{X_j \mid \boldsymbol{X} \in \hat{\sigma}(b,i)\})/_{\sim_{\sqcup}}$.

**Lemma 4.25** (least selection in frames)**.** *Let $E$ be a system of $m$ fixpoint equations over a frame $L$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$. Let $\hat{\sigma}$ be the least selection for $E$ over $L$. For $j \in \underline{m}$, for every position $(b,i) \in (B_L \times \underline{m})$, for each $[C]_{\sim_{\sqcup}} \in \hat{\sigma}(b,i)/^{j}_{\sim_{\sqcup}}$ it holds that $\{[C]_{\sim_{\sqcup}}\} \bigcap \{X_j \mid \boldsymbol{X} \in \hat{\sigma}(b,i)\}$ is a singleton.*

*Proof.* Let $j \in \underline{m}$ be an equation index. Since $L$ is a frame, the assumption over the equivalence classes becomes $[C]_{\sim_H} \in \hat{\sigma}(b,i)/^{j}_{\sim_H}$ by Corollary 2.20. Thus, we show that $\{[C]_{\sim_H}\} \bigcap \{X_j \mid \boldsymbol{X} \in \hat{\sigma}(b,i)\}$ is a singleton, i.e., $|\{[C]_{\sim_H}\} \bigcap \{X_j \mid \boldsymbol{X} \in \hat{\sigma}(b,i)\}| = 1$.

Consider $[C]_{\sim_H} \in \hat{\sigma}(b,i)/^{j}_{\sim_H}$, suppose that there exist two different elements $\boldsymbol{X}, \boldsymbol{Y} \in \hat{\sigma}(b,i)$ such that $X_j, Y_j \in [C]_{\sim_H}$. Hence $X_j \sqsubseteq_H Y_j \sqsubseteq_H X_j$, or equivalently $X_j \sim_H Y_j$. But then consider $\hat{\sigma}_1$, such that $\hat{\sigma}_1(b,i) = (\hat{\sigma}(b,i) \setminus \{\boldsymbol{Y}\}) \bigcup \{\boldsymbol{Y'}\}$ and $\hat{\sigma}_1(b',i') = \hat{\sigma}(b',i')$ on all other positions $(b',i')$, where $Y'_j = \bot$ and $Y'_k = Y_k$ for $k \neq j$.

Thus, we obtain another least selection different from $\hat{\sigma}$, since we removed an element covered by $\boldsymbol{X}$, which is impossible by Theorem 4.24. So, we have that $|\{[C]_{\sim_H}\} \bigcap \{X_j \mid \boldsymbol{X} \in \hat{\sigma}(b,i)\}| = 1$. □

Intuitively, consider the equivalence relation induced by the preorder $\sqsubseteq_{\sqcup}$. Given a least selection $\hat{\sigma}$, an index $j \in \underline{m}$ and a position $(b,i)$, it holds that for every $\boldsymbol{X} \in \hat{\sigma}(b,i)$, there not exists $\boldsymbol{Y} \in \hat{\sigma}(b,i)$ such that $Y_j \in [X_j]_{\sim_H}$.

H-MINIMALITY From the fact that there are no repetitions in a minimal selection, we obtain a property, called H-minimality, that holds in every element of a selection without repetitions, thus on minimal selections. In words, given $X \in 2^{B_L}$, H-minimality means that any strict subset of $X$ is not above $X$, with respect to $\sqsubseteq_H$.

**Definition 4.26** (H-minimal)**.** Let $L$ be a complete lattice, and $B_L$ a basis for $L$. Consider $X \subseteq B_L$, $X$ is called *H-minimal* if $\nexists Y \subsetneq X$ such that $X \sqsubseteq_H Y$.

This property is applied pointwise to the tuples of a selection, that is, $\boldsymbol{X}$ is H-minimal if for all $i \in \underline{m}$ it holds that $X_i$ is H-minimal.

**Lemma 4.27** (least selections contain only H-minimals)**.** *Let $E$ be a system of $m$ fixpoint equations over a complete lattice $L$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$. Whenever $\hat{\sigma}$ is the least selection for $E$, with respect to $\subseteq_H$, then, for all positions $(b, i) \in (B_L \times \underline{m})$, every $\boldsymbol{X} \in \hat{\sigma}(b, i)$ is H-minimal.*

*Proof.* Consider $\hat{\sigma}$ least selection for $E$. Thus, $\hat{\sigma} \subseteq_H \sigma_i$ for any $\sigma_i$ suitable selection for $E$. Suppose there exists $\boldsymbol{X} \in \hat{\sigma}(b, i)$ which is not H-minimal. Hence, there exists $\boldsymbol{Y}$ such that $\boldsymbol{Y} \subsetneq^{\vee} \boldsymbol{X}$ and $\boldsymbol{X} \sqsubseteq_H^{\wedge} \boldsymbol{Y}$, thus $\boldsymbol{X} \sim_H^{\wedge} \boldsymbol{Y}$ using Lemma 2.11. Let us build now a selection $\sigma'$ such that $\sigma'(b, i) = (\hat{\sigma}(b, i) \setminus \{\boldsymbol{X}\}) \bigcup \boldsymbol{Y}$ and $\sigma'(b', j) = \hat{\sigma}(b', j)$ on all other positions $(b', j)$.

It is easy to see that $\sigma' \subseteq_H \hat{\sigma}$, so for Theorem 4.24, $\sigma' = \hat{\sigma}$ but this is a contradiction because $\sigma'$ is different from $\hat{\sigma}$ by construction. □

Currently, the contrary does not hold, if a selection $\sigma$ contains only H-minimals, does not imply that $\sigma$ is the least. In fact, $\sigma(b, i)$ could contain an element $\boldsymbol{X} \in (2^{B_L})^m$ H-minimal and some other $\boldsymbol{Y} \neq \boldsymbol{X}$ such that $\boldsymbol{Y} \sqsubseteq_H^{\wedge} \boldsymbol{X}$, and thus not be minimal.

SELECTIONS FOR $\mu$-CALCULUS Since in the thesis, various examples will be based on the $\mu$-calculus, here we study what happens to selections if the given lattice is a powerset lattice. The following results are therefore specialized for this class of lattices.

Let us see first that in this context Hoare preorder and subset inclusion coincide.

**Lemma 4.28** (subset inclusion and Hoare preorder coincide)**.** *Given a set $S$, consider the powerset lattice $L = 2^S$. Consider $X, Y \subseteq S$, then $X \subseteq Y$ if and only if $X \sqsubseteq_H Y$.*

*Proof.* We already know from Lemma 2.11, that $X \subseteq Y$ implies $X \sqsubseteq_H Y$ even without the restriction on $L$. Assume that $X \sqsubseteq_H Y$. Thus we have that for all $x \in X$ there exists $y \in Y$ such that $x \sqsubseteq y$, since $S$ is a set there are no

order relation between elements. Hence, $x \sqsubseteq y$ implies that $x = y$. Therefore, $X \subseteq Y$. □

**Corollary 4.29** (subset inclusion and Supremum preorder coincide)**.** *Given a set $S$, consider the powerset lattice $L = 2^S$. Consider $X, Y \subseteq S$, then $X \subseteq Y$ if and only if $X \sqsubseteq_{\sqcup} Y$.*

*Proof.* Observe that $2^S$ is a frame, and $B_{2^S} = \{\{s\} \mid s \in S\}$ is a basis of completely join-irreducibles for $L$. Hence by Lemma 4.28 and Corollary 2.20 we conclude. □

For the lemma below, we will use the classical notion of upward closure with respect to the subset inclusion without adding any subscript, that is, given a set $X$, its upward closure is defined as

$$\uparrow X = \{y \in X \mid \exists x \in X \text{ such that } x \sqsubseteq y\}$$

**Lemma 4.30** (selections coincide in $\mu$-calculus setting)**.** *Let $S$ be a set and $E$ be a system of $m$ fixpoint equations over $L = 2^S$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$. Given a selection $\sigma$, for all positions $(b, i)$, we have that $\uparrow \sigma(b, i) = \uparrow_H \sigma(b, i) = \boldsymbol{E}(b, i)$.*

*Proof.* Since Hoare Preorder and subset inclusion coincide in $2^S$, by Lemma 4.28, also the two upward closures coincide. □

This last lemma justifies our interest for selections over the powerset lattice of a set $S$. Since the upward closure operator is antitone, in the $\mu$-calculus setting we could base the order over selections on the subset inclusion. That is, given two selections $\sigma'_s$ and $\sigma''_s$, we have that $\sigma'_s \subseteq_H \sigma''_s$ if and only if, for all positions $(b, i) \in (B_L \times \underline{m})$, $\sigma'_s(b, i) \subseteq \sigma''_s(b, i)$. Observe that, if for all positions $(b, i) \in (B_L \times \underline{m})$, $\sigma'_s(b, i) \subseteq \sigma''_s(b, i)$, then $\sigma'_s \subseteq_H \sigma''_s$ holds even in a general context, by Lemma 4.22.

## 4.4 Symbolic ∃-moves

For finite height lattices, the set of possible moves of the existential player is upward-closed, with respect to $\subseteq_H$. Such set can be conveniently represented and manipulated in logical form. Intuitively, minimal selections could be naively

described via families of logical formulae in disjunctive normal form, i.e., consisting of a disjunction of conjunctions, but more compact forms can also be obtained.

**Example 9** (exponential explosion problem)**.** For instance, the monotone function

$$f(X_1, \ldots, X_{2n}) = (X_1 \cup X_2) \cap \cdots \cap (X_{2n-1} \cup X_{2n})$$

would be of exponential size in the corresponding disjunctive normal form with $2^n$ disjuncts: $(X_1 \cap \cdots \cap X_{2n-1}) \cup \cdots \cup (X_2 \cap \cdots \cap X_{2n})$, such as the conjunction above. But we can easily give a formula of linear size.

This motivates the introduction of a propositional logic for expressing the set of moves of the existential player along with a technique to generate moves, that is our final goal.

**Definition 4.31** (logic for upward closed sets with respect to $\sqsubseteq_H$)**.** Let $L$ be a lattice and let $B_L$ be a basis for $L$. Given $m \in \mathbb{N}$, representing the number of equations, the logic $\mathcal{L}_m^H(B_L)$ has formulae defined as follows, where $b \in B_L$ and $j \in \underline{m}$:

$$\varphi ::= [b, j] \mid \bigwedge_{k \in K} \varphi_k \mid \bigvee_{k \in K} \varphi_k$$

We will write $true$ for the empty conjunction, and $false$ for the empty disjunction.

Generally, when $\varphi = [b, j]$, $true$ or $false$ we refer to it just as atom, because in functions that work inductively over logic formulae, whenever the input is $[b, j]$, $true$ or $false$ we do not need any recursive step to compute the function solution.

**Definition 4.32** (semantic for logic formulae)**.** The semantics of a formula $\varphi$ is an upward-closed set $[\![\varphi]\!] \subseteq (2^{B_L})^m$ with respect to $\sqsubseteq_H$, defined as follow:

$$\begin{aligned}
[\![[b, j]]\!] &= \{\boldsymbol{X} \in (2^{B_L})^m \mid \{b\} \sqsubseteq_H X_j\} \\
[\![\bigwedge_{k \in K} \varphi_k]\!] &= \bigcap_{k \in K} [\![\varphi_k]\!] \\
[\![\bigvee_{k \in K} \varphi_k]\!] &= \bigcup_{k \in K} [\![\varphi_k]\!]
\end{aligned}$$

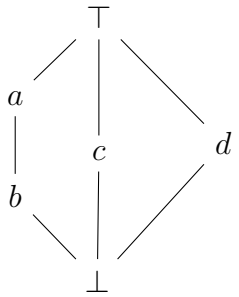


Figure 4.5: Non-distributive Lattice $L$ with four elements

Observe that, the semantic of atoms $[\![[b, j]]\!] = \{\boldsymbol{X} \in (2^{B_L})^m \mid \{b\} \sqsubseteq_H X_j\}$ has been constructed in this way to keep a connection between selections and symbolic $\exists$-moves, such that each $\boldsymbol{X}$ in a minimal selection is represented by a conjunction of atoms and all the $\boldsymbol{X}$ in a minimal selection by means of disjunctions.

**Remark 4** (semantic for $true$ and $false$ atoms)**.**

$$[\![true]\!] = [\![\bigwedge_{k\in\emptyset} \varphi_k]\!] = \bigcap_{k\in\emptyset} [\![\varphi_k]\!] = \bigcap \emptyset = \{\boldsymbol{X} \mid \boldsymbol{X} \in (2^{B_L})^m\}$$

$$[\![false]\!] = [\![\bigvee_{k\in\emptyset} \varphi_k]\!] = \bigcup_{k\in\emptyset} [\![\varphi_k]\!] = \bigcup \emptyset = \emptyset$$

**Example 10** (semantic for logic formulae)**.** This example shows why we choose this kind of semantic for logic formulae and how it supports its connection with the (minimal) selections.

Another possibility for the semantic of an atom could be

$$[\![[b, j]]\!]_s = \{\boldsymbol{X} \in (2^{B_L})^m \mid b \sqsubseteq \bigsqcup X_j\} \qquad (\star)$$

With this definition, we would have logic formulae with a lower number of atoms, compared to those regarding the actual definition (Definition 4.32). However, in general the connection with the least selection would be less immediate with this last version.

Consider the lattice $L$, in Figure 4.5, with the basis $B_L = \{a, b, c, d\}$. Suppose

that the system of equations $E$ is composed only by a single equation, such that

$$\begin{aligned}\boldsymbol{E}(b,1) ='' &\text{ all subsets of } B_L \text{ such that their } \sqcup \text{ is greater than } b''\\ = \{&(\{b\}),(\{a,b\}),(\{b,c\}),(\{b,d\}),\\ &(\{a\}),(\{a,c\}),(\{a,d\}),(\{c,d\}),\\ &(\{b,c,d\}),(\{a,c,d\}),(\{a,b,c,d\})\}\end{aligned}$$

It is easy to see that $\hat{\sigma}(b,1) = \{(\{b\}),(\{c,d\})\}$, since $\uparrow_H \{(\{b\}),(\{c,d\})\} = \boldsymbol{E}(b,1)$. Let now call $\varphi_e$ for the extended version (Definition 4.32) and $\varphi_s$ for the shortened one, defined in $(\star)$. The two formulae $\varphi_e$ and $\varphi_s$ both represent the same least selection $\hat{\sigma}$, but they differ in the way to do it.

$$\begin{aligned}\varphi_e &= [b,1] \vee ([c,1] \wedge [d,1])\\ \varphi_s &= [b,1]\end{aligned}$$

This explains the meaning of "connection" between minimal selections (and selections in general) and symbolic $\exists$-moves.

However, this is not always the case, for example consider Example 9, in that case the minimal selections would consist of $2^n$ components while the symbolic move would be a formula of linear size.

It is easy to see that indeed each upward-closed set with respect to $\sqsubseteq_H$ is denoted by a formula, showing that the logic is sufficiently expressive.

**Lemma 4.33** (formulae for upward-closed set, with respect to $\sqsubseteq_H$)**.** *Let $L$ be a lattice with basis $B_L$ and let $X \subseteq (2^{B_L})^m$ be an upward closed set, with respect to $\sqsubseteq_H$. Then $X = [\![\varphi]\!]$ where $\varphi$ is the formula in $\mathcal{L}^H_m(B_L)$ defined as follows:*

$$\varphi = \bigvee_{\boldsymbol{Y} \in X} \bigwedge \{[b,j] \mid j \in \underline{m} \wedge \{b\} \sqsubseteq_H Y_j\}$$

From the Lemma above, every least selection $\hat{\sigma}$ is represented by a formula $\varphi \in \mathcal{L}^H_m(B_L)$.

It follows the definition that remarks the connection, already predicted in Example 10, between a system of fixpoint equations $E$ and the symbolic $\exists$-moves

associated with it. For practical purposes, we should restrict to finite formulae. This can be surely done in the case of finite lattices, so from now on we restrict ourselves to work only with finite lattices.

**Definition 4.34** (symbolic $\exists$-moves)**.** Let $L$ be a finite lattice and let $f : L^m \to L$ be a monotone function. A *symbolic $\exists$-move for $f$* is a family $(\varphi_b)_{b \in B_L}$ of formulae in $\mathcal{L}^H_m(B_L)$ such that, for all $b \in B_L$ it holds $[\![\varphi_b]\!] = \boldsymbol{E}(b, f)$, i.e., $\boldsymbol{E}(b, f) = \{\boldsymbol{X} \in (2^{B_L})^m \mid b \sqsubseteq f(\bigsqcup \boldsymbol{X})\}$.

If $E$ is a system of $m$ equations of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$ over $L$, a *symbolic $\exists$-move for $E$* is a family of formulae $(\varphi^i_b)_{b \in B_L, i \in \underline{m}}$ such that for all $i \in \underline{m}$, the family $(\varphi^i_b)_{b \in B_L}$ is a symbolic $\exists$-move for $f_i$.

Interestingly, symbolic $\exists$-moves can be obtained compositionally. The formulae corresponding to a function arising as a composition of other functions can be obtained from those for the components.

**Lemma 4.35** (symbolic $\exists$-moves, compositionally)**.** *Let $L$ be a finite lattice with basis $B_L$, and let $f : L^n \to L$, $f_j : L^m \to L$ for $j \in \underline{n}$ be monotone functions and let $(\varphi_b)_{b \in B_L}$, $(\varphi^j_b)_{b \in B_L, j \in \underline{n}}$ be symbolic $\exists$-moves for $f, f_1, \ldots f_n$. Consider the function $h : L^m \to L$ obtained as the composition $h(\boldsymbol{x}) = f(f_1(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))$. Define the family $(\varphi'_b)_{b \in B_L}$ as follows. For all $b \in B_L$, the formula $\varphi'_b$ is obtained from $\varphi_b$ by replacing each occurrence of $[b', j]$ by $\varphi^j_{b'}$. Then $(\varphi'_b)_{b \in B_L}$ is a symbolic $\exists$-move for $h$.*

*Proof.* The proof can be found in the extended version of [Baldan et al., 2018]. □

The example below shows how to build symbolic $\exists$-moves compositionally.

**Example 11** ($\mu$-calculus symbolic $\exists$-moves construction)**.** Given a set of states $S$, let $L$ be the powerset lattice of $S$. For $b \in B_L$, let us define $\phi$ for each $\mu$-calculus semantic function and projection operator:

- $\cup : L \times L \to L$, we let $\phi^{\cup}_b = [b, 1] \bigvee [b, 2]$
- $\cap : L \times L \to L$, we let $\phi^{\cap}_b = [b, 1] \bigwedge [b, 2]$
- $\Diamond : L \to L$, we let $\phi^{\Diamond}_b = \bigvee \{[x, 1] \mid b \to x\}$
- $\Box : L \to L$, we let $\phi^{\Box}_b = \bigwedge \{[x, 1] \mid b \to x\}$

- $p \in Prop$ and $\rho : Prop \bigcup PVar \to 2^{B_L}$, we let $\phi_b^\rho = true$ if $b \subseteq \rho(p)$, $\phi_b^\rho = false$ otherwise
- $\pi_i : L^m \to L$, we let $\phi_b^\pi = [b, i]$

Consider the settings of Example 5. So, we use these constructions to obtain the first equation, $f^1(x_1, x_2) = \{\{b\}, \{d\}, \{e\}\} \cap \Box x_1$. Let us rewrite $f^1$ in terms of its sub-operators for the sake of clarity, $f^1(x_1, x_2) = f^\cap(\{\{b\}, \{d\}, \{e\}\}, f^\Box(\pi_1(x1, x2))$. Since we know how to build symbolic moves for the meet, predicate and square operator, $\phi^\cap_{\{b\}} = [\{b\}, 1] \bigwedge [\{b\}, 2]$, $\phi^\rho_{\{b\}} = true$ (since $b \in \{\{b\}, \{d\}, \{e\}\}$) and $\phi^\Box_{\{b\}} = \bigwedge\{[x, 1] \mid b \to x\}$ respectively. We can build the composed one, hence $\phi^1_{\{b\}} = [\{b\}, 1] \bigwedge [\{b\}, 2]$ where, by composition:

- $[\{b\}, 1]$ is replaced by $\phi^\rho_{\{b\}}$ since the first parameter of the meet operator is instantiated by the predicate $\rho$.
- $[\{b\}, 2]$ is replaced by $\phi^\Box_{\{b\}}$, that is $[\{d\}, 1] \wedge [\{e\}, 1]$. In turn, each atom is replaced by the projection function $\pi_1$ that leaves unchanged the symbolic $\exists$-move already constructed since $i = 1$ in the projection function instantiation.

Suppose that we want to obtain the symbolic $\exists$-moves for the position $(\{b\}, 1)$, by definition, we need to compute $\phi^1_{\{b\}}$, since $[\![\phi_{\{b\}}]\!] = \boldsymbol{E}(\{b\}, 1)$. Using the formula previously computed, $\phi^1_{\{b\}} = true \wedge ([\{d\}, 1] \wedge [\{e\}, 1])$.

Unfortunately, composition of symbolic $\exists$-moves of minimal size could leads to bigger moves. The composed symbolic move could have more atoms than necessary.

**Example 12** (sub-optimal composed move)**.** Recall the Example 11. The formula obtained for $\boldsymbol{E}(\{b\}, 1)$ is bigger than the necessary. $\phi^1_{\{b\}} = true \wedge ([d, 1] \wedge [e, 1])$ can be refactored to $\phi^1_{\{b\}} = [\{d\}, 1] \wedge [\{e\}, 1]$. As another example, suppose we want to compute the formula for $\boldsymbol{E}(\{a\}, 1)$, the composed result is $\phi^1_{\{a\}} = false \wedge ([\{a\}, 1] \wedge [\{b\}, 1] \wedge [\{c\}, 1])$ while this is equivalent to the much smaller formula, $\phi^1_{\{a\}} = false$.

We now are able to introduce the formal tool, called *generator*, that allow us to retrieve the moves for player $\exists$, given a symbolic $\exists$-move. Note that when player $\exists$

uses this generator, moves for player $\forall$ will be returned, i.e., $m$-tuples of powerset of the basis. This generator will not be used directly in the pseudocode in Chapter 5, but will be the fundament between the pseudocode and the theoretical tools exposed in this chapter.

Our goal is to return the smallest set possible of moves, this is due to the fact that more moves we are able to play more the game will last. The generator for $\exists$-move will be indicated by $M^{\exists}$ and it works as follows. We achieve our goal by cases, keeping a strong connection with the symbolic $\exists$-moves formulae semantic defined in Definition 4.32 to handle the input logic formula:

**(base case)** the generator $M^{\exists}$ receives as input the atom $[b, j]$. In the base case, $M^{\exists}$ has to return the singleton composed by the everywhere-emptyset $m$-tuple except for the $j$-th component, where $X_j = \{b\}$.

**(conjunctions)** the generator $M^{\exists}$ receives $\bigwedge_{k\in K} \varphi_k$ as input. Since the input is a conjunction of formulae $M$ has to return the set that merges all the possible combinations of the recursive steps. In other words, we retrieve each possible combinations with the product of all the sub-formulae $\varphi_k$ applied to $M^{\exists}$ recursively. Then, we join the components returned pointwise.

**(disjunctions)** the generator $M^{\exists}$ receives $\bigvee_{k\in K} \varphi_k$ as input. In this last case, $M^{\exists}$ has to return just the flattened set of all the sets returned from the recursive steps.

**Definition 4.36** (Generator $M^{\exists}$ for $\exists$-moves)**.** Let $E$ be a system of $m$ fixpoint equations over a finite lattice $L$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$. The *generator* $M^{\exists} : \mathcal{L}^H_m(B_L) \to 2^{(2^{B_L})^m}$ is defined as follows, for $b \in B_L$ and $j \in \underline{m}$:

$$
\begin{aligned}
M^{\exists}([b, j]) &= \boldsymbol{X} \text{ such that } X_j = \{b\} \wedge X_i = \emptyset \text{ for } i \neq j \\
M^{\exists}(\bigwedge_{k\in K} \varphi_k) &= \bigcup\{\cup^{\wedge}\{\boldsymbol{X} \mid \boldsymbol{X} \in y\} \mid y \in \prod_{k\in K} M^{\exists}(\varphi_k)\} \\
M^{\exists}(\bigvee_{k\in K} \varphi_k) &= \bigcup\{M^{\exists}(\varphi_k) \mid k \in K\}
\end{aligned}
$$

From the definition above we know that given a formula $\varphi$ if we apply it to $M^{\exists}$, the generator returns a set of real moves for $\exists$-player, i.e., positions for player $\forall$.

We introduced the definition above to be able to connect the least selection to the symbolic ∃-moves, that will be used in the pseudocode. With this connection, since we know that the game played with selections is still sound and complete, we can conclude that if the algorithm uses only moves obtained by the generator the algorithm will remain still sound and complete.

**Lemma 4.37** (symbolic ∃-moves and least selection)**.** *Let $E$ be a system of $m$ fixpoint equations over a finite lattice $L$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$. Let $(\varphi_b^i)_{b \in B_L, i \in \underline{m}}$ be a symbolic ∃-move for $E$. Given the least selection $\hat{\sigma}$ for $E$. For every position $(b, i) \in (B_L \times \underline{m})$, it holds that*

$$\hat{\sigma}(b,i) \subseteq M^{\exists}(\varphi_b^i) \subseteq \uparrow_H \hat{\sigma}(b,i) = \boldsymbol{E}(b,i)$$

The lemma above says that there is no guarantee in the goodness of the generator itself, it depends on how good is the symbolic ∃-move given as input. Hence in the best case we only play the minimal number of real moves to maintain the soundness of the game, instead in the worst case we have no pruning at all. As already said before, it depends on how symbolic ∃-moves are built.

An important fact, that may have gone unnoticed, is that our studies regarding the classes of distributive lattices, e.g., see Lemma 4.25 or Paragraph 4.3.1, would only affect the construction of symbolic ∃-moves. But, whenever the logic formulae are built, there is no more need to discriminate any criterion by the lattice concerned. All the complexity is so abstracted to the level of creation of symbolic ∃-moves. Thus, will be easier to build formulae if the lattice is a frame, or even better the powerset lattice.

Interestingly enough, we studied how symbolic ∃-moves react when the given system of equations is *normalized*. We found out that in a normalized system of equations the formulae associated with each equation are optimal, in the sense of the generator. That is, we obtained the best case of Lemma 4.37, i.e., $\hat{\sigma}(b,i) = M^{\exists}(\varphi_b^i)$. More details can be found in Appendix A.

# 5

# Local algorithm

In this last chapter, we now describe the local algorithm for characterizing the solution of a system of fixpoint equations over a complete lattice. As we mentioned before, technically, this means, given an element of the lattice basis, determining whether or not such element is dominated in the order by the solution of the system. The development relies some previous work in [Baldan et al., 2020], where they presented an algorithm for solving these kinds of local problems. For instance, in the case of the $\mu$-calculus, rather than computing the set of states satisfying a formula $\varphi$, we could be interested in checking whether a specific state enjoys or not $\varphi$. The idea consists in computing only the information needed for the local problem of interest, in the line of other local algorithms developed for bisimilarity [Hirschkoff, 1999] and $\mu$-calculus model checking [Stevens and Stirling, 1998]. In particular, the algorithm arises as a natural generalization of the one in [Stevens and Stirling, 1998] to the setting of fixpoint games, see Definition 4.1.

In this thesis, we will add some improvements mainly regarding the efficiency in computing moves for the existential player implementing the techniques described in Chapter 4. Some ideas, which were only hinted at in the mentioned paper, are formalized and fully developed. We still have not implemented a working prototype yet, but the contribution in this thesis offers all the notions and structures needed to produce a working tool.

## 5.1 Notations

In this section, we fix some notations and conventions which will be useful for describing the algorithm.

For the rest of the section, $L$ denotes a finite lattice, with a basis $B_L$, and $E$ is a system of $m$ fixpoint equations over $L$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$ with solution $\boldsymbol{s} \in L^m$.

A generic player, that can be either $\exists$ or $\forall$, is represented by the upper case letter $P$. The opponent of player $P$ is denoted by $\overline{P}$. The set of all *positions* of the game is denoted by $Pos = Pos_\exists \cup Pos_\forall$, where $Pos_\exists = B_L \times \underline{m}$, ranged over by $(b, i)$ is the set of positions controlled by $\exists$, and $Pos_\forall = (2^{B_L})^m$, ranged over by $\boldsymbol{X}$ is the set of positions controlled by $\forall$. A generic position is denoted by the upper case letter $C$ and we write $P(C)$ for the player controlling that position.

Given a position $\boldsymbol{X} \in Pos_\forall$, the possible moves for player $\forall$ are indicated by $M^\forall(C) \subseteq Pos_\exists$, see Algorithm 5.1. Instead, given a position $(b, i) \in Pos_\exists$, we retrieve the possible moves for player $\exists$ one at a time, invoking the function NextMove, which will be discussed later on.The pseudocode is presented in Algorithm 5.6.

**Algorithm 5.1** moves for the player $\forall$

```
function M^∀(X)
    P ← ∅
    for i ∈ m
        for b ∈ X_i
            P ← P ∪ {(b, i)}
    return P
```

A function $\imath : Pos \to (\underline{m} \cup \{0\})$ maps every position to a *priority*, which, for positions $(b, i)$ of player $\exists$ is the index $i$, while it is always 0 for position of $\forall$, see Algorithm 5.2.

**Algorithm 5.2** priority function $\imath$

```
function ı(C)
    if P(C) = ∀
        return 0
    else
        let b, i s.t. (b, i) = C
        return i
```

## 5.2 The algorithm

Given an element of the basis $b \in B_L$ and an index $i \in \underline{m}$, the algorithm checks whether $b$ is below the solution of the $i$-th fixpoint equation of the system, i.e, $b \sqsubseteq s_i$. According to Theorem 4.2, this corresponds to establish which of the players has a winning strategy in the fixpoint game starting from position $(b, i)$.

The procedure roughly consists of a depth-first exploration of the tree of plays arising as unfolding the game graph starting from the initial position $(b, i)$ for the player $\exists$. The algorithm optimizes the search by making assumptions on particular subtrees, which are thus pruned. Assumptions can be later confirmed or invalidated, and hence withdrawn.

The algorithm is split into two central functions and nine more auxiliary ones.

- Function Explore explores the tree of plays of the game, trying different moves from each node to determine the player who has a winning strategy from such node.
- Function Backtrack allows backtracking from a node after the algorithm has established the winner from it, transmitting the information backward.

With these two modules, the algorithm runs through the tree of plays. All the other subfunctions are used to retrieve information and to set values among the program flow properly. We conceptually partitioned these two functions from all the others because Explore and Backtrack mutually call themselves during the execution, while the auxiliary functions, when called, return the required value.

The algorithm uses the following data structures:

- The *counter* $\boldsymbol{k} \in \mathbb{N}^m$, i.e., an $m$-tuple of natural numbers, which associates each non-zero priority with the number of times the priority has been encountered in the play since a higher priority was last faced (the current position is not included). After each move, the counter is updated taking into account the priority of the current position. More precisely, the update of the counter $\boldsymbol{k}$, moving from a position with priority $i$, denoted $\textsc{Next}(\boldsymbol{k}, i)$, is defined in Algorithm 5.3. Note that, in particular, $\textsc{Next}(\boldsymbol{k}, 0) = \boldsymbol{k}$, i.e., moves from a position with priority 0, which are the moves of $\forall$, do not change $\boldsymbol{k}$.

**Algorithm 5.3** Counter $\boldsymbol{k}$ update

```
function NEXT(k, i)
    k' ← 0
    k'_i ← k_i + 1
    for j from (i + 1) to m
        k'_j ← k_j
    return k'
```

We also equip the counter data structure with two total orders $<_\exists$ and $<_\forall$, that intuitively measure how good the current advancement of the game is for the two players.

- We let $\boldsymbol{k} <_\exists \boldsymbol{k'}$ when the largest $i$ such that $k_i \neq k'_i$ is the index of a greatest fixpoint equation and $k_i < k'_i$, or it is the index of a least fixpoint equation and $k_i > k'_i$.
- The other order $<_\forall$ is the reverse of $<_\exists$, that is $\boldsymbol{k} <_\forall \boldsymbol{k'}$ if and only if $\boldsymbol{k'} <_\exists \boldsymbol{k}$.

For each player $P$, we write $\boldsymbol{k} \leq_P \boldsymbol{k'}$ if $\boldsymbol{k} <_P \boldsymbol{k'}$ or $\boldsymbol{k} = \boldsymbol{k'}$.

- The *playlist* $\rho$, i.e., a list of positions encountered from the root to the current node (empty if the current node is the root), each with the corresponding counter $\boldsymbol{k}$ and the indication of the alternative moves which have not been explored (the exploration is performed depth-first). Thus, $\rho$ is a list of triples $(C, \boldsymbol{k}, \pi)$, where $C$ is a position, $\boldsymbol{k}$ is the counter and $\pi$ indicates the unexplored moves from that position.

  We will use the object $\pi$ to restart the exploration when we backtrack. In the case of the player $\forall$, we will directly store the positions that we did not try during the current execution. Thus, we will pick a move from this set to restart the exploration. Regarding the player $\exists$, we will store the simplified symbolic $\exists$-move $\varphi_C$ associated with the current position $C \in Pos_\exists$. Thus, we will restart the exploration simplifying this symbolic move $\varphi_C$ and then calling the function NEXTMOVE returning a new $\forall$-position. Moreover, for both players, $\pi$ will also record the current counter $\boldsymbol{k}$.

- The *assumptions* for player $\exists$ and $\forall$, i.e., a pair of sets $\Gamma = (\Gamma_\exists, \Gamma_\forall)$. A position $C$ is assumed to be winning for some player when either it is en-

countered for the second time in the current playlist $\rho$, or it is inferred during the symbolic $\exists$-move simplification.

  - In the first case, this reveals the presence of a loop in the game graph that can be unfolded into an infinite play. Position $C$ is assumed to be winning for the player who would win such an infinite play. In detail, if $\boldsymbol{k}$ is the current counter and $\boldsymbol{k}'$ is the counter of the previous occurrence of $C$, then the winner $P$ is the player such that $\boldsymbol{k}' <_P \boldsymbol{k}$.
  - In the second case, during the process of simplification of a symbolic $\exists$-move (done by the function REDUCE), we can infer some assumptions, at most, one for each atom of the symbolic move. Given a formula $\varphi$, for each atom $[b, i]$, we will look into the current playlist $\rho$ for the position $(b, i) \in Pos_\exists$. This check could reveal a loop in the game graph, which can be unfolded into an infinite play. Note that we did not encounter the same position twice in $\rho$, but we would have done it, choosing that position in the next exploration step. Thus, as before, position $C$ is assumed to be winning for the player who would win such an infinite play. In detail, given an atom $[b, i]$, if the triple $((b, i), \boldsymbol{k}', \_)$ is encountered in $\rho$ (so $\boldsymbol{k}'$ would be the counter of the "previous occurrence" of $(b, i)$) and $\boldsymbol{k}$ is the current counter, then the winner $P$ is the player such that $\boldsymbol{k}' <_P \boldsymbol{k}$.

  This ensures that the highest priority in the loop is the index of a least fixpoint if $P = \forall$ and of a greatest fixpoint if $P = \exists$. The assumption is stored with the corresponding counter, i.e., $\Gamma_P$ contains pairs of the kind $(C, \boldsymbol{k})$. Since other possible paths branching from the loop or the simulation are possibly unexplored, assumptions can still be falsified afterwards.

- The *decisions* for player $\exists$ and $\forall$, i.e., a pair of sets $\Delta = (\Delta_\exists, \Delta_\forall)$. Intuitively, a decision for a player $P$ is a position $C$ of the game such that we established that $P$ has a winning strategy from $C$. The decision is stored with the corresponding counter, i.e., $\Delta_P$ contains pairs of the kind $(C, \boldsymbol{k})$. When a new decision is added, we also record its *justification*, i.e., the assumptions and decisions we relied on for deriving the new decision, if any. Sometimes, we also store a decision without any justification. In this case,

we call the decision a *truth*.

- To retrieve a symbolic ∃-move from a given ∃-position we will use a simple get statement, e.g., "get $\varphi_C$ from position $C$" retrieves the formula associated with the position $C$ if $C \in Pos_\exists$. Since all the symbolic ∃-moves are treated as an input to the algorithm concerned, they are always available through this directive.

For checking whether $b \sqsubseteq s_i$ for $b \in B_L$ and $i \in \underline{m}$, we call the function Explore$((b,i), \mathbf{0}, [], (\emptyset,\emptyset), (\emptyset,\emptyset))$, where $\mathbf{0}$ is the everywhere-zero counter. This returns the only player $P$ having a winning strategy from position $(b,i)$ and, by Theorem 4.2, $P = \exists$ if and only if $b \sqsubseteq s_i$.

We now present each of the functions used in the algorithm grouped by functionalities into five modules: Reduce, NextMove, Explore, Backtrack, and Forget.

### Reduce

Given a formula $\varphi$, the corresponding counter $\boldsymbol{k}$, the set of decisions $\Delta$ and the current playlist $\rho$, the function Reduce$(\varphi, \boldsymbol{k}, \Delta, \rho)$ returns a triple $(\varphi', \Gamma_\exists, \Gamma_\forall)$, where $\varphi'$ is the simplified symbolic ∃-move equipped with two sets of new assumptions, one for each player. The code is given in Algorithm 5.4. As you can see, the function body is split into two sub-modules:

**Algorithm 5.4** The reduce function

**function** Reduce$(\varphi, \boldsymbol{k}, \Delta, \rho)$
  $(\varphi', \Gamma_\exists, \Gamma_\forall) \leftarrow$ ApplyDecisionsAndAssumptions$(\varphi, \boldsymbol{k}, \Delta, \rho)$
  **return** (Simplify$(\varphi'), \Gamma_\exists, \Gamma_\forall)$
**Ensure:** Simplify$(\varphi')$ is $true$, $false$ or a formula without $true/false$ in its structure

- The function ApplyDecisionsAndAssumptions takes all the parameters given in input to the Reduce function. This function has to transform some atoms of the given formula $\varphi$ into $true$ or $false$ using the information in the set of decisions $\Delta$ and in the playlist $\rho$. The pseudocode of the function is given in Algorithm 5.5. Note that, $true$ and $false$ are just syntactic sugar for $\bigwedge_{k\in\emptyset} \varphi_k$ and $\bigvee_{k\in\emptyset} \varphi_k$ respectively, thus the returned object is coherent with $\mathcal{L}^H_m(B_L)$.

This function behaves in two logical steps. It first performs a recursive unfolding of the given formula until it reaches all the atoms (with the aid of the function UNFOLD). Then, when an atom $[b, i]$ is reached, it checks whether one of the following two conditions holds, otherwise it just returns the plain atom $[b, i]$. The two conditions are then translated into four if-branch due to the instantiation of the player $P$.

- If there is already a decision for the current position $(b, i)$ with a favorable $\boldsymbol{k}'$ for the player $P$, that is, $((b, i), \boldsymbol{k}') \in \Delta_P$ such that $\boldsymbol{k}' \leq_P \boldsymbol{k}$. Hence, if $P = \exists$ means that the atom $[b, i]$ would lead to a win for the player $\exists$ and so the value returned is $true$. Otherwise, the atom $[b, i]$ would lead to a win for the player $\forall$, so in this case the function returns $false$.
- If position $(b, i)$ was already encountered in the play, i.e., $((b, i), \boldsymbol{k}', \_) \in \rho$ for some counter $\boldsymbol{k}'$ such that $\boldsymbol{k}' <_P \boldsymbol{k}$, then $(b, i)$ becomes an assumption for player $P$. As before, if $P = \exists$ the atom becomes $true$, otherwise it becomes $false$.

Finally, the function APPLYDECISIONSANDASSUMPTIONS returns a triple $(\varphi', \Gamma_\exists, \Gamma_\forall)$, where $\varphi'$ is the formula with some atoms transformed into $true$ or $false$, the set $\Gamma_\exists$ includes all the new assumptions favorable for player $\exists$ and, dually, $\Gamma_\forall$ is the set of new assumptions for player $\forall$.

- The second sub-module of the function REDUCE works as follows. Given a formula $\varphi$ with some atoms to $true$ or $false$, the function SIMPLIFY returns a formula simplified in his structure. The pseudocode for this function is quite cumbersome. Thus it is not given explicitly.

  An efficient implementation of this function would exploit some programming languages low-level features as the pointers to take control over the parent of any subformula that has to be modified iteratively. Thus, we decided to explain the functionalities of the function SIMPLIFY by words.

  Intuitively, it just performs a $false$-$\vee$ and $true$-$\wedge$ elimination, $true$-$\vee$ collapsing to $true$ and $false$-$\wedge$ collapsing to $false$. Note that, if the given formula has no $true$ or $false$ into its leafs, then SIMPLIFY returns the input without any changes.

**Algorithm 5.5** The functions ApplyDecisionsAndAssumptions and Unfold

**function** ApplyDecisionsAndAssumptions($\varphi, \boldsymbol{k}, \Delta, \rho$)
  **switch** $\varphi$
    **case** $[b, i]$
      **if** exists $((b, i), \boldsymbol{k}') \in \Delta_\exists$ s.t. $\boldsymbol{k}' \leq_\exists \boldsymbol{k}$
        **return** $(true, \emptyset, \emptyset)$
      **if** exists $((b, i), \boldsymbol{k}') \in \Delta_\forall$ s.t. $\boldsymbol{k}' \leq_\forall \boldsymbol{k}$
        **return** $(false, \emptyset, \emptyset)$
      **if** exists $((b, i), \boldsymbol{k}', \_) \in \rho$ s.t. $\boldsymbol{k}' <_\exists \boldsymbol{k}$
        **return** $(true, \{((b, i), \boldsymbol{k}')\}, \emptyset)$
      **if** exists $((b, i), \boldsymbol{k}', \_) \in \rho$ s.t. $\boldsymbol{k}' <_\forall \boldsymbol{k}$
        **return** $(false, \emptyset, \{((b, i), \boldsymbol{k}')\})$
      **return** $([b, i], \emptyset, \emptyset)$
    **case** $\bigwedge_{j \in J} \varphi_j$
      **return** Unfold$(\varphi, J, \wedge, true)$
    **case** $\bigvee_{j \in J} \varphi_j$
      **return** Unfold$(\varphi, J, \vee, false)$

**function** Unfold$(\varphi, J, op, init)$
  $\Gamma \leftarrow (\emptyset, \emptyset)$
  $\varphi' \leftarrow init$
  **for** $j \in J$
    $(\varphi'_j, \Gamma_\exists, \Gamma_\forall) \leftarrow$ ApplyDecisionsAndAssumptions$(\varphi_j, \boldsymbol{k}, \Delta, \rho)$
    $\Gamma \leftarrow (\Gamma_1 \cup \Gamma_\exists, \Gamma_2 \cup \Gamma_\forall)$
    $\varphi' \leftarrow \varphi'\ op\ \varphi'_j$
  **return** $(\varphi', \Gamma_1, \Gamma_2)$

**Example 13** (SIMPLIFY pruning)**.** The following table will give a complete view of the rules used to simplify symbolic ∃-moves, where each formula on the lefthand side is associated with its simplification on the righthand side.

$$x \wedge true \rightsquigarrow x$$
$$x \wedge false \rightsquigarrow false$$
$$x \vee true \rightsquigarrow true$$
$$x \vee false \rightsquigarrow x$$

Some examples follow as well:

$$\textsc{Simplify}(((x \wedge y \wedge true) \vee (false \wedge y)) \wedge x) = (x \wedge y) \wedge x$$
$$\textsc{Simplify}(x \vee x \vee (y \wedge true \wedge false)) = x \vee x$$
$$\textsc{Simplify}(false \vee x \vee y \vee z) = x \vee y \vee z$$

As you can see from the examples above the function SIMPLIFY only performs a simplification, looking for $true$ or $false$. However, it does not run any other simplification/reduction step like a $SAT$ solver.

### NEXTMOVE

Given a logic formula $\varphi_C$ associated with the position $C$, the invocation of the function NEXTMOVE($\varphi_C$) returns a ∀-position, or the token *null* to indicate the absence of moves left. To work properly, i.e., to return a new minimal move regarding the position $C$ every time the function is called, the function requires $\varphi$ to be already simplified and have not been called before with the same argument. Otherwise, the function may return duplicates or inefficient moves. The pseudocode is shown in Algorithm 5.6.

The function NEXTMOVE($\varphi_C$) just checks whether the formula $\varphi_C$ is exactly $false$ or $true$, otherwise it calls the auxiliary function BUILDNEXTMOVE (see Algorithm 5.7) that performs the unfolding of the given formula. In case $\varphi_C = false$, the formula cannot generate any other moves. In this way the function returns the token *null*, while if $\varphi_C = true$, the formula can possibly generate all the moves in $(2^{B_L})^m$, thus the function returns the lowest one, i.e., the everywhere-

emptyset $m$-tuple (written $\boldsymbol{\emptyset}$), which is a proper position for player $\forall$.

**Algorithm 5.6** The function NextMove

```
Require: φ already simplified using Reduce, NextMove(φ) has never been called before
function NextMove(φ)
    if φ = false
        return null
    if φ = true
        return ∅
    return BuildNextMove(φ)
```

Regarding the function BuildNextMove($\varphi$), it requires that the input formula does not contain any $true$ or $false$ in its leafs. This check is ensured by the fact that BuildNextMove is called only by NextMove, so $\varphi \neq false$ and $\varphi \neq true$ and that NextMove, in turn, requires that the formula is already simplified using the function Reduce. Thus the formula $\varphi$ does not contain $true$ or $false$ in its structure.

BuildNextMove unfolds the formula structure until it reaches the atoms. For each atom $[b, i]$ it returns the move where the $i$-th components is the only non-emptyset component set to the singleton $\{b\}$. For each $\wedge$-node, it returns the union of the recursive calls applied to the subformulae. Instead, in case of an $\vee$-node, it returns the recursive call applied to just one picked subformula. The "pick" statement can be implemented random or fixed, it does not affect the correctness of the function.

**Algorithm 5.7** The function BuildNextMove

```
Require: φ does not have true or false leafs
function BuildNextMove(φ)
    C ← ∅
    switch φ
        case [b, i]
            C_i ← {b}
        case ⋀_{j∈J} φ_j
            for j ∈ J
                C ← C ∪^ BuildNextMove(φ_j)
        case ⋁_{j∈J} φ_j
            pick j ∈ J
            C ← BuildNextMove(φ_j)
    return C
```

EXPLORE

Given the current position $C$, the corresponding counter $\boldsymbol{k}$, the playlist $\rho$ describing the path that led to $C$, and the set of assumptions $\Gamma$ and decisions $\Delta$, function EXPLORE$(C, \boldsymbol{k}, \rho, \Gamma, \Delta)$ checks if one of the following five conditions holds, each one corresponding to a different if-branch.

- If $M^{\forall}(C) = \emptyset$ or $C = false$, then the controller $P(C)$ of position $C$ cannot move and its opponent $\overline{P(C)}$ wins. This check is operated by the function ISEMPTY, see Algorithm 5.8. Therefore, a new truth, i.e., decision without justification, for the current position $C$ is added for the opponent, and then we backtrack.

**Algorithm 5.8** Function ISEMPTY

```
function IsEmpty(C)
    if P(C) = ∀
        return M^∀(C) = ∅
    else
        get φ_C from position C
        return φ_C = false
```

- If there is already a decision for a player $P$ for the current position $C$, that is, $(C, \boldsymbol{k}') \in \Delta_P$ and $\boldsymbol{k}' \leq_P \boldsymbol{k}$, the we can reuse that information to assert that $P$ would win from the current position as well. The requirement $\boldsymbol{k}' \leq_P \boldsymbol{k}$ intuitively ensures that we arrived to the current position $C$ with a play that is at least as good for $P$ as the play which led to the previous decision $(C, \boldsymbol{k}')$.

- If the current position $C$ was already encountered in the play, i.e., $(C, \boldsymbol{k}, \_) \in \rho$ for some $\boldsymbol{k}'$, then $C$ becomes an assumption for the player $P$ for which the counter got strictly better, that is, $\boldsymbol{k}' <_P \boldsymbol{k}$. Then we backtrack.

- If none of the conditions above holds and the player $P = \forall$, the exploration continues from $C$. A move $C' \in M^{\forall}(C)$ is chosen to be explored. The playlist is thus extended by adding $(C, \boldsymbol{k}, \pi)$ where $\pi$ records the remaining moves to be explored. The counter $\boldsymbol{k}$ is updated according to the priority of the past position $C$.

- If none of the conditions above holds and the player $P = \exists$, we first retrieve the associated formula $\varphi_C$ from the current position $C \in Pos_\exists$. Then we simplify that logic formula using the function REDUCE. If any assumptions are created, we merge them to the set of assumptions $\Gamma$ pointwise, i.e., we combine assumptions for each player $P$ with the newly created assumptions for $P$. Now we check whether the simplified formula $\varphi'$ became $false$, if so, $C$ becomes a new truth for player $\forall$ and we backtrack. Otherwise, a new move $C'$, extracted by NEXTMOVE, is chosen to be explored. As done previously, the playlist is thus extended by adding $(C, \boldsymbol{k}, \pi)$ where $\pi$ stores the simplified symbolic $\exists$-move $\varphi'$. The counter $\boldsymbol{k}$ is updated according to the priority of the past position $C$.

See Algorithm 5.9 for the pseudocode of this procedure.

**Algorithm 5.9** Function EXPLORE

```
function Explore(C, k, ρ, Γ, Δ)
    if IsEmpty(C)
        Δ_{¬P(C)} ← Δ_{¬P(C)} ∪ {(C, k)}
        return Backtrack(¬P(C), C, ρ, Γ, Δ)
    if there is (C, k') ∈ Δ_P s.t. k' ≤_P k
        return Backtrack(P, C, ρ, Γ, Δ)
    if there is (C, k', _) ∈ ρ
        let P s.t. k' <_P k
        Γ_P ← Γ_P ∪ {(C, k')}
        return Backtrack(P, C, ρ, Γ, Δ)
    if P(C) = ∀
        pick C' ∈ M_∀(C)
        k' ← next(k, I(C))
        π ← (M(C) \ {C'}) × {k'}
        return Explore(C', k', ((C, k, π) :: ρ), Γ, Δ)
    get φ_C from position C
    (φ'_C, Γ'_∃, Γ'_∀) ← Reduce(φ_C, k, ρ, Δ)
    Γ ← (Γ_∃ ∪ Γ'_∃, Γ_∀ ∪ Γ'_∀)
    if φ'_C = false
        Δ_∀ ← Δ_∀ ∪ {(C, k)}
        return Backtrack(∀, C, ρ, Γ, Δ)
    C' ← NextMove(φ'_C)
    k' ← next(k, I(C))
    π ← (φ'_C, k')
    return Explore(C', k', ((C, k, π) :: ρ), Γ, Δ)
```

Regarding player $\forall$, the choice of moves to explore (performed by the action

"pick" in the pseudocode) is random, since there are no differences between choosing a move or another. However, this is induced by the fact that we only explore the minimal moves on the ∃-player's shift. This is one of the significant improvement from the previously developed pseudocode in [Baldan et al., 2020].

Notably, the moves for player ∃ returned by the function NextMove during the computation (which are those explored by the algorithm) are less than those that the game would prescribe to explore, although considering only minimal moves. More considerations about this fact are given in the correctness Section 5.3.

### Backtrack

Function Backtrack$(P, C, \rho, \Gamma, \Delta)$ is used to backtrack from a position $C$, reached via the playlist $\rho$, after assuming or deciding that player $P$ will win from such position $C$. The pseudocode is shown in Algorithm 5.10.

The algorithm of Backtrack works as follows. If $\rho = []$, we are back at the root, the position from which the computation started, and the exploration is concluded. The algorithm decides that player $P$ is the winner from such a position.

Otherwise we pop the head $(C', \boldsymbol{k}, \pi)$ of the playlist $\rho$ and the status of position $C'$ is investigated.

- If $C'$ is controlled by the opponent of $P$, i.e., $P(C') \neq P$ we face two more cases:
  - If player ∀ controls $C'$, i.e., $P(C') = \forall$ and there are still unexplored moves, i.e., $\pi \neq \emptyset$, we must explore such moves before deciding the winner from $C'$. Then a new move is extracted from $\pi$ and thus explored.
  - If instead, player ∃ controls $C'$, i.e., $P(C') = \exists$, we retrieve $(\varphi, \boldsymbol{k}'')$ from $\pi$. Then we simplify $\varphi$ into $\varphi'$(remembering to add the newly created assumptions, if any). If $\varphi' \neq false$, we proceed retrieving the next move $C''$ to be explored from $\varphi'$. Before exploring the move $C''$, we update into $\rho$ the simplified formula $\varphi'$ instead of the old one $\varphi$, with the same counter $\boldsymbol{k}''$.

- If instead $C'$ is controlled by the player $P$, i.e., $P(C') = P$ then $P$ wins also from $C'$. Hence $C'$ is inserted in $\Delta_P$, justified by the move $C$ from where we backtracked. Similarly, if the controller of $C'$ is the opponent of $P$, i.e., $P(C') \neq P$, we already explored all possible moves from $C'$, i.e., $\pi = \emptyset$ or $\varphi' = false$, and all turn out to be winning for player $P$, again we decide that $P$ wins from $C'$, which is inserted in $\Delta_P$, justified by all possible moves from $C'$. Since we decided that $P$ would win from $C'$ we can now continue to backtrack. However, before backtracking we must discard all assumptions for the opponent of $P$ in conflict with the newly taken decision, and this must be propagated to the decisions depending on such assumptions. This is done by the invocation $\textsc{Forget}(\Delta_{\overline{P}}, \Gamma_{\overline{P}}, (C', \boldsymbol{k}'))$.

### Forget

The function Forget is not given explicitly. The precise definition of the property that the function Forget must satisfy to ensure the algorithm's correctness is quite technical. Intuitively, when an assumption in $\Gamma_P$ fails and is withdrawn, we must remove from $\Delta_P$ at least all the decisions depending on such an assumption.

With the new mechanism introduced by NextMove and Reduce, the situation is even more complicated than the Forget developed in [Baldan et al., 2020], since we can no longer use the dependencies graph of justifications to implement a sound Forget. This issue is due to the newly created assumptions during the simplification phase. Furthermore, whenever we find a winning $\exists$-move during the simplification process of a symbolic move, that winning move will not be a real move for the related position (that is the original symbolic $\exists$-move). The simplified move is not a big deal unless this is not a "correct" justification for the new decision for the $\exists$-position. That is, "correct" means that, if we used the playlist $\rho$ to transform atoms of the logic formula to *true*, and we found a winning move afterwards, this move would not be connected to such assumptions. Therefore, whenever one of those assumptions fail, Forget would be invoked to forget the decisions relying on the failed assumptions, but the decision concerned would not be deleted. All this while assuming that Forget is implemented using the dependencies graph over justifications. The same also applies in the dual case,

**Algorithm 5.10** Function BACKTRACK

**function** BACKTRACK$(P, C, \rho, \Gamma, \Delta)$
  **if** $\rho = []$
    **return** $P$
  let $C', \boldsymbol{k}', \pi, t$ s.t. $\rho = ((C', \boldsymbol{k}', \pi) :: t)$
  **if** $P(C') \neq P$
    **if** $P(C') = \forall$ and $\pi \neq \emptyset$
      pick $(C'', \boldsymbol{k}'') \in \pi$
      $\pi' = \pi \setminus \{(C'', \boldsymbol{k}'')\}$
      **return** EXPLORE$(C'', \boldsymbol{k}'', ((C', \boldsymbol{k}', \pi') :: t), \Gamma, \Delta)$
    **if** $P(C') = \exists$
      let $\varphi, \boldsymbol{k}''$ s.t. $\pi = (\varphi, \boldsymbol{k}'')$
      $(\varphi'_C, \Gamma'_\exists, \Gamma'_\forall) \leftarrow$ REDUCE$(\varphi, \boldsymbol{k}', \rho, \Delta)$
      $\Gamma \leftarrow (\Gamma_\exists \cup \Gamma'_\exists, \Gamma_\forall \cup \Gamma'_\forall)$
      **if** $\varphi' \neq false$
        $C'' \leftarrow$ NEXTMOVE$(\varphi')$
        $\pi' \leftarrow (\varphi', \boldsymbol{k}'')$
        **return** EXPLORE$(C'', \boldsymbol{k}'', ((C', \boldsymbol{k}', \pi') :: t), \Gamma, \Delta)$
  **if** $P(C') = P$
    $\Delta_P \leftarrow \Delta_P \cup \{(C', \boldsymbol{k}')\}$ justified by $C$
  **else**
    **if** $P = \forall$
      $\Delta_\forall \leftarrow \Delta_\forall \cup \{(C', \boldsymbol{k}')\}$ justified by $M^\forall(C')$
    **else**
      get $\varphi_{C'}$ from position $C'$
      $\Delta_\exists \leftarrow \Delta_\exists \cup \{(C', \boldsymbol{k}')\}$ justified by $\varphi_{C'}$
  $\Gamma_P \leftarrow \Gamma_P \setminus \{(C', \boldsymbol{k}')\}$
  **if** there is $(C', \boldsymbol{k}') \in \Gamma_{\overline{P}}$
    $\Delta_{\overline{P}} \leftarrow$ FORGET$(\Delta_{\overline{P}}, \Gamma_{\overline{P}}, (C', \boldsymbol{k}'))$
    $\Gamma_{\overline{P}} \leftarrow \Gamma_{\overline{P}} \setminus \{(C', \boldsymbol{k}')\}$
  **return** BACKTRACK$(P, C', t, \Gamma, \Delta)$

regarding losing positions for player $\exists$, that is when to fail was an assumption for player $\forall$ that was used to substitute an atom to $false$.

Anyway, different sound realizations of FORGET are then possible (see [Stevens and Stirling, 1998]). Experimentally, it can be seen that those removing only the least possible set of decisions can be practically inefficient. For instance, a simple sound FORGET could be based over a temporal criterion: when an assumption fails, all decisions taken after that assumption are deleted. The temporal criterion can be implemented by associating timestamps with decisions and assumptions, avoiding the complex management of justifications.

## 5.3 CORRECTNESS

In the line of Section 7.2 in [Baldan et al., 2020], we show that when the lattice is finite, which is an assumption that holds since Section 4.4, the algorithm terminates. Moreover, when it terminates, it provides the correct answer.

Termination on finite lattices can be proved by observing that the set of positions (which are either element of the basis or tuples of sets of elements of the basis) is finite. The length of playlists is bounded by the number of positions, since, whenever a position repeats in a playlist, it necessarily becomes an assumption and backtracking starts. Finally, one can observe that it is impossible to cycle indefinitely between two positions, so that termination immediately follows.

Since the pseudocode developed in this thesis differs only in the number of moves played by player $\exists$, depending on the fact that, in this new algorithm, the set of moves played by $\exists$ is smaller than that played in the original algorithm, we can conclude using the Lemma 7.2 in [Baldan et al., 2020] regarding the termination. Nevertheless, for a better understanding, we also add that lemma here.

**Lemma 5.1** (termination)**.** *Let $L$ be a finite lattice. Given a fixpoint game over $L$, any call* EXPLORE$(C_0, \mathbf{0}, [], (\emptyset, \emptyset), (\emptyset, \emptyset))$ *terminates, hence at some point the function* BACKTRACK$(P, C_0, [], (\emptyset, \emptyset), \Delta)$ *is invoked, for some player $P$ and a set $\Delta$.*

Correctness of the local algorithm is not obvious, since we play on a sort of restricted game (without the possibility of trying every possible move playable by

player $\exists$). However, the correctness can be proved relying on the correctness of the original algorithm, with some additional observations concerning the functions Reduce and Explore.

More precisely, we distinguish two possibilities concerning the moves of the player $\exists$ not explored in the algorithm.

1. The move discarded from player $\exists$ does not belong to a selection. This case is completely covered by Theorem 4.16, which states that the game played with moves restricted to selections is equivalent to the original game played without any restriction.

2. The move discarded from player $\exists$ belongs to a selection. Let us call this move $C \in Pos_{\forall}$. Since $C$ was discarded during the simplification step, a decision or assumption $x$ must have lead to the transformation of the corresponding atom (to *true* or $false$). Hypothetically, in the next exploration step, if we had played $C$, we would have benefited from the same conclusions since $x$ would have been applied when exploring the aforementioned atom. Indeed, this arises from the fact that the same necessary conditions, specified in the function Explore, are also checked in ApplyDecisionsAndAssumptions, as can be seen by comparing Algorithm 5.9 and 5.5.

Summarizing, the proof of correctness relies on Lemma 7.3 [Baldan et al., 2020] in conjunction with the notes above.

**Theorem 5.2** (correctness)**.** *Let $L$ be a finite lattice. Given a fixpoint game over $L$, if the invocation to* Explore$(C, \mathbf{0}, [], (\emptyset, \emptyset), (\emptyset, \emptyset))$ *returns a player $P$, then $P$ wins the game from $C$.*

Notice that it is unnecessary to prove also the converse implication since the game can never result in a draw. Furthermore, proving that if $P$ wins the game from $C$, the the call Explore$(C, \mathbf{0}, [], (\emptyset, \emptyset), (\emptyset, \emptyset))$ returns $P$ is equivalent to prove that if the call Explore$(C, \mathbf{0}, [], (\emptyset, \emptyset), (\emptyset, \emptyset))$ returns $\overline{P}$, then $\overline{P}$ wins the game from position $C$. Currently, this is already stated in Theorem 5.2.

## 5.4 Comparison with previous algorithms

We have presented a new algorithm based over the original one devised in [Baldan et al., 2020], which provides a way to compute local problems arising from systems

of fixpoint equations. In this section we will provide a running execution as a comparison with the previous algorithm. We recall the same settings of Example 7.1 in [Baldan et al., 2020].

**Example 14** (algorithm execution instance)**.** Consider the transition system $T$ in Figure 3.1 and the $\mu$-calculus formula

$$\phi = \mu x_2.((\nu x_1.(p \wedge \Box x_1)) \vee \Diamond x_2)$$

presented in Example 5. As already discussed, the formula $\phi$ interpreted over $T$ leads to the following systems, over the powerset lattice $2^{\mathbb{S}_T}$:

$$\begin{cases} x_1 & =_\nu \{b, d, e\} \cap \blacksquare_{\to_T} x_1 \\ x_2 & =_\mu x_1 \cup \blacklozenge_{\to_T} x_2 \end{cases}$$

Suppose we want to verify whether the state $a \in \mathbb{S}$ satisfies the formula $\phi$. This is equivalent to determining the winner of the powerset game from position $(a, 2)$ which is done by invoking $\textsc{Explore}((a, 2), \mathbf{0}, [], (\emptyset, \emptyset), (\emptyset, \emptyset))$. A computation performed by the algorithm is schematized in Figure 5.1 where we show all the possibly played moves by each player in this instance. We indicate with a bold line (either straight or dashed) the path chosen when a non-deterministic pick is required. Note that the chosen instance is an unlucky event, since all the other routes lead to faster executions, e.g., see Figure 5.2. In the diagram, positions of player $\exists$ are drawn as diamonds, while those of player $\forall$ are represented as boxes. The counter $\boldsymbol{k}$ associated with the positions is on their righthand side.

Since symbolic $\exists$-moves are already available as input, we show for each $\exists$-position the associated logic formula.

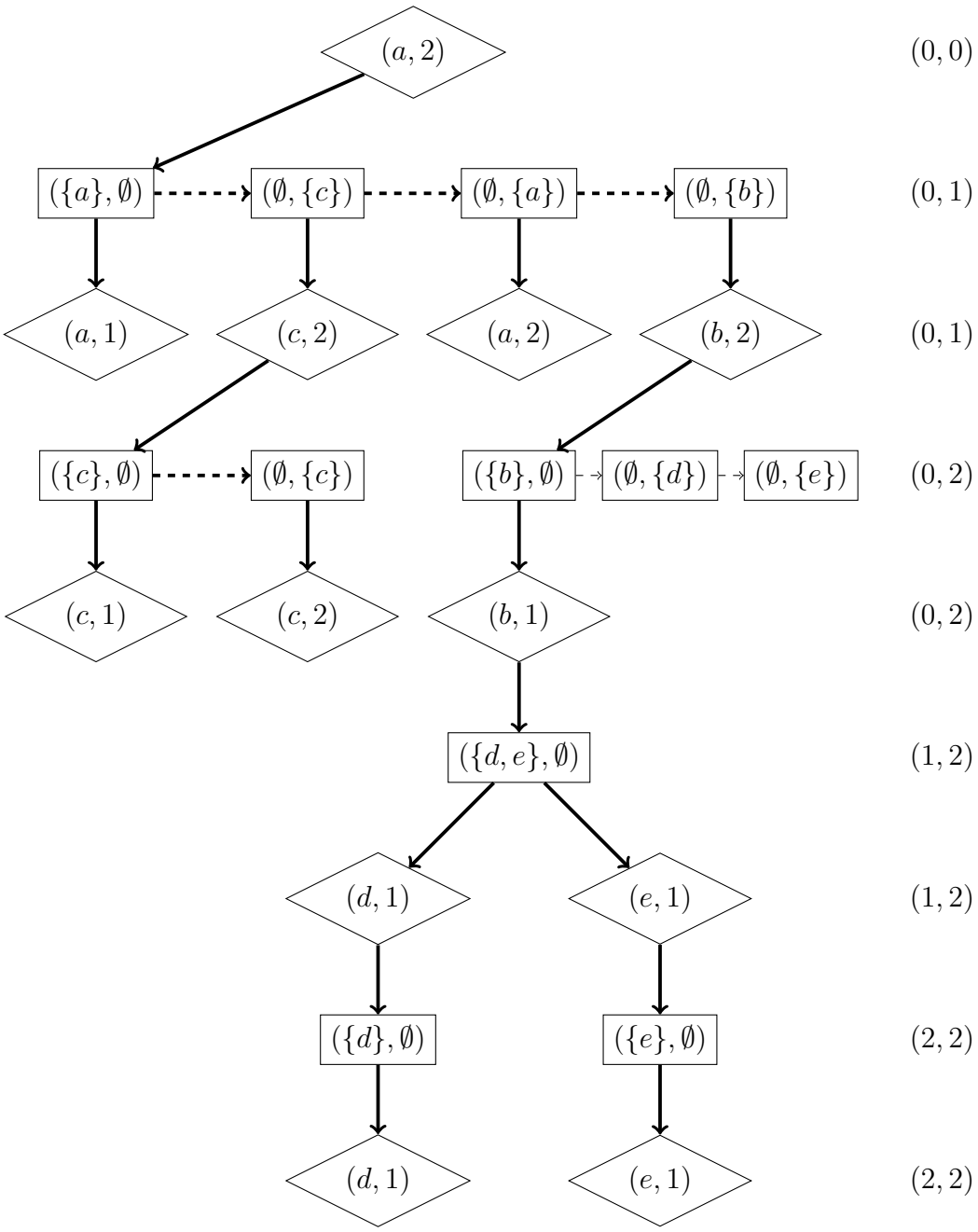


Figure 5.1: An execution of the local algorithm

$$
\begin{aligned}
(a,1) &\implies false \\
(b,1) &\implies [\{d\},1] \wedge [\{e\},1] \\
(c,1) &\implies false \\
(d,1) &\implies [\{d\},1] \\
(e,1) &\implies [\{e\},1] \\
(a,2) &\implies [\{a\},1] \vee [\{a\},2] \vee [\{b\},2] \vee [\{c\},2] \\
(b,2) &\implies [\{b\},1] \vee [\{d\},2] \vee [\{e\},2] \\
(c,2) &\implies [\{c\},1] \vee [\{c\},2] \\
(d,2) &\implies [\{d\},1] \vee [\{d\},2] \\
(e,2) &\implies [\{e\},1] \vee [\{e\},2]
\end{aligned}
$$

To build all the formulae, we used the composition of symbolic $\exists$-moves, more details are given in Example 11. We will indicate each reduction phase performed over a given formula $\varphi$ with a prime, e.g., if the symbolic $\exists$-move has simplified twice, we will indicate the new formula as $\varphi''$.

Starting from the initial position $(a,2)$, with counter $(0,0)$, we retrieve the corresponding formula $\varphi_{(a,2)} = [\{a\},1] \vee [\{a\},2] \vee [\{b\},2] \vee [\{c\},2]$. The reduction phase does not modify the formula since no positions are in $\Delta$ or in the playlist $\rho$. Thus, $\varphi'_{(a,2)} = \varphi_{(a,2)}$. Assuming "pick $j \in J$" returns the subformula $[\{a\},1]$, the next move from $\varphi'_{(a,2)}$ is the tuple $(\{a\},\emptyset)$, thus the formula, the new counter $(0,1) = \textsc{Next}((0,0),2)$ and the updated playlist are stored and the exploration can continue.

The search proceeds in this way and player $\forall$ plays the only move possible for position $(\{a\},\emptyset)$, i.e., $(a,1)$. Over this position, player $\exists$ retrieves the formula $false$, that remains as it is also after the simplification phase, that generates a new truth for player $\forall$, since no new move is feasible from this position, and we backtrack. In the playlist the last position played was $(\{a\},\emptyset)$ and the winning player is $\forall$, that coincides with $P((\{a\},\emptyset))$ so a new decision for player $\forall$ is generated justified by this position and the backtracking takes one more step.

In position $(a, 2)$, controlled by the opponent, player $\exists$ has moves left, so the simplified formula $\varphi'_{(a,2)}$ is simplified again. Atom $[\{a\}, 1]$ is reduced to $false$ and eliminated, since a decision for position $(a, 1)$ is found in $\Delta_\forall$ and $(0, 1) \leq_\forall (0, 1)$ where the first counter is the one stored with $\varphi'_{(a,2)}$ and the second is the actual position's counter. Thus, $\varphi''_{(a,2)} = [\{a\}, 2] \vee [\{b\}, 2] \vee [\{c\}, 2]$. Suppose that the algorithm chooses to explore the (next) move $(\emptyset, \{c\})$, as highlighted by the dashed bold arrow.

Player $\forall$ now plays $(c, 2)$ as its only option. Then, the associated formula for the chosen position $(c, 2)$ is $\varphi_{(c,2)} = [\{c\}, 1] \vee [\{c\}, 2]$. Note that this formula coincides with its simplification $\varphi'_{(c,2)}$. We now face two possible moves for player $\exists$, i.e., $(\{c\}, \emptyset)$ or $(\emptyset, \{c\})$, discriminated by the picking action. Assuming the first position is picked by the invocation of NextMove, position $(\{c\}, \emptyset)$ brings us to position $(c, 1)$ that is associated with a $false$-formula. Hence, we backtrack until the second option is chosen, i.e., $(\emptyset, \{c\})$, since the simplified formula became $\varphi''_{(c,2)} = [\{c\}, 2]$. Thus, position $(c, 2)$ is played again for player $\exists$, with counter $(0, 2)$ this time. Since the counter stored with the first occurrence of $(c, 2)$ was $(0, 1)$ and $(0, 1) <_\forall (0, 2)$, then the pair position and counter $((c, 2), (0, 1))$ is added as an assumption for player $\forall$ and the algorithm starts backtracking until the root again. That is due to the fact that another simplification phase led to $\varphi'''_{(c,2)} = false$. Observe that the atom $[\{c\}, 2]$ became $false$ for the same reason we started backtracking in the current execution branch (atom already found in the current playlist $\rho$). Notably, decisions for player $\forall$ are generated throughout the backtracking route.

Here in the root, we simplify again the formula, where atom $[c, 2]$ disappeared ($false$ atoms in logic conjunctions disappear). Thus, $\varphi'''_{(a,2)} = [\{a\}, 2] \vee [\{b\}, 2]$ and the next move chosen is $(\emptyset, \{a\})$. Thus, player $\forall$ has to play $(a, 2)$ which is already encountered in the playlist (as it is the root) and it becomes a favorable assumption for player $\forall$. We backtrack again until the root, and decisions for player $\forall$ are generated throughout the backtracking route.

Position $(a, 2)$, controlled by player $\exists$ has another last move left. The simplified logic formula became $\varphi''''_{(a,2)} = [\{b\}, 2]$, hence the next move chosen is $(\emptyset, \{b\})$. From this position, player $\forall$ plays $(b, 2)$, hence we retrieve the formula $\varphi_{(b,2)} = [\{b\}, 1] \vee [\{d\}, 2] \vee [\{e\}, 2]$, which is already simplified in the current context, i.e., in the current playlist $\rho$ and decisions set $\Delta$. The search proceeds in this way

along the moves, assuming picking choice during the computation

$$(b,2) \overset{\exists}{\rightsquigarrow} (\{b\},\emptyset) \overset{\forall}{\rightsquigarrow} (b,1) \overset{\exists}{\rightsquigarrow} (\{d,e\},1)$$
$$\overset{\forall}{\rightsquigarrow} (d,1) \overset{\exists}{\rightsquigarrow} (\{d\},\emptyset) \overset{\forall}{\rightsquigarrow} (d,1)$$

until position $(d,1)$ occurs again, with counter $(2,2)$. Since the counter stored with the first occurrence of $(d,1)$ was $(1,2)$ and $(1,2) <_\exists (2,2)$, then the pair position and counter $((d,1),(1,2))$ is added as an assumption for player $\exists$ and we backtrack. While backtracking, the algorithm generates a decision for player $\exists$, which is $((\{d\},\emptyset),(2,2))$, justified by the only possible move $(d,1)$ for player $\forall$. When it comes back to the first occurrence of $(d,1)$, since $P((d,1)) = \exists$, the procedure transforms the assumption $((d,1),(1,2))$ into a decision for player $\exists$ justified by the move $(\{d\},\emptyset)$. Then it backtracks to position $(\{d,e\},\emptyset)$ which is controlled by the opponent and there is still an unexplored move $(e,1)$. Therefore, the algorithm starts exploring again from $(e,1)$, and does it similarly to the previous branch of $(d,1)$. After making decisions for those positions as well, the algorithm resumes backtracking from $(\{d,e\},\emptyset)$, since all possible moves have been explored, making decisions for player $\exists$ along the way back. This goes on up until the root is reached again. The last invocation $\textsc{Backtrack}(\exists,(a,2),[],\Gamma,\Delta)$ terminates since $\rho = []$, and returns player $\exists$. Indeed, player $\exists$ wins starting from position $(a,2)$ since state $a$ satisfies the formula $\phi$, as the solution is $x_1 = \{b,d,e\}$ and $x_2 = \{a,b,d,e\}$ from Example 5.

As explained before this instance of the algorithm is the worst case scenario, Figure 5.2 shows the shortest one instead, where the bold arrows indicate the chosen path.

Comparing this algorithm, presented in Chapter 5, to the one devised in [Baldan et al., 2020], it can be easily seen that our new improvements have abated the iterations needed to compute the winner in the worst case scenario, e.g., consider the original algorithm running on this situation, from the initial position $(a,2)$ player $\exists$ could play position $(\{a,b,d\},\{b,c,e\})$ since it is a superset of $(\{a\},\emptyset)$. That is, a huge number of moves is always accessible.

Indicatively, from the first position $(a,2)$ in the example concerned we would have about a hundred different possible positions from the starting one. All of them are supersets of the ones available from the initial position $(a,2)$ running

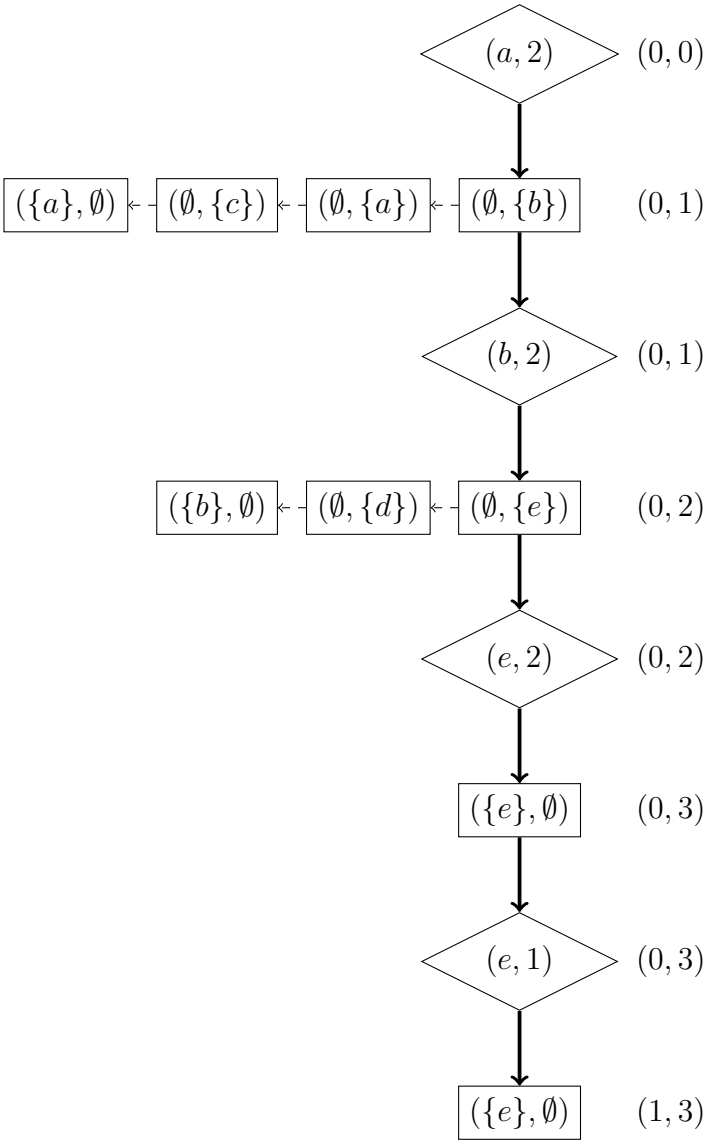


Figure 5.2: Another execution of the local algorithm

the new algorithm, i.e., $(\{a\}, \emptyset)$, $(\emptyset, \{a\})$, $(\emptyset, \{b\})$ or $(\emptyset, \{c\})$.

So with this improvement, we properly captured the fact that playing only minimal moves leads to a faster game. So a feasible working tool can be now implemented. We see this as the natural prosecution of the work done in this thesis.

# 6

# Conclusion and future work

Systems of (mixed) least and greatest fixpoint equations over complete lattices are at the core of several verification and analysis tasks ranging from the model-checking of various specification logics, notably the $\mu$-calculus, to the verification of coinductive behavioral equivalences, like bisimilarity, to the static analysis of programs.

In this thesis, we built over some recent characterization of the solution of such systems of equations in terms of suitable parity games, referred to as powerset games. More precisely,

- We explored a notion of progress measures, in the sense of Jurdziński [Jurdziński, 2000], that, as in their original formulation, compactly characterize the existence of a winning strategy for a player at any possible position of the game.

- We studied a notion of selection to restrict the moves that need to be explored, showing that playing on the restricted game is equivalent to playing on the original one. We also identify sufficient conditions that ensure the existence of a minimal selection, ensuring that the number of moves to be explored will be as small as possible.

- We introduced a notion of symbolic $\exists$-moves that allow for a compact rep-

resentation of the selections, to be used in algorithms to solve the powerset game.

- We developed a local algorithm for solving the game locally, which means establishing whether an element of the basis is dominated by some solution component. This is normally sufficient to solve the verification problem of interest and possibly avoids the full exploration of the game graph, required by global approaches. Our work is based on the local algorithm devised in [Baldan et al., 2020], which has been guided from backtracking techniques, see [Stevens and Stirling, 1998; Stirling, 1995]. The algorithm flow alternates two distinct phases: the exploration phase explores a branch depthwise until some conditions are fulfilled, or the backtracking phase that proceeds backward. The use of symbolic $\exists$-moves guarantees that only "clever" moves are tried, which belong to selections and for which the winner is still unknown.

FUTURE WORK The algorithm devised in this thesis is still in the form of pseudocode. The first natural advance is thus the development of a working prototype running on general systems of fixpoint equations over finite height lattices. The tool would take as input the basis of the lattice, the basic operators used to define the functions involved in the equations, and the symbolic $\exists$-moves associated with such the basic operators. The tool would be a sort of generic model-checker capable of checking various kinds of specifications logics, with special attention for those in the $\mu$-calculus family. Some inspiration could be provided by an analogous work recently developed in a similar setting but for a global algorithm [Mazzocchin, 2019].

A direction of future research is related to the fact that we restricted ourselves to finite height lattices, something that ensures the termination of the local algorithm. It could be interesting to investigate what sort of abstraction is needed to adapt the developed algorithm to possibly infinite-height lattices. We could take inspiration from the work done in [Baldan et al., 2020], wherewith the aid of abstract interpretation, they abstracted the fixpoint game to relax the continuity condition. Our thoughts rely on the fact that whenever one abstracts the infinite concrete lattice, the game played on this abstraction would compute the

solution only in a finite (restricted) number of steps. In contrast, if one tries to compute the fixpoint using the Knaster-Tarski theorem [Tarski, 1955] over infinite lattices, the computation could lead indefinitely. As it happens typically for other approximation approaches, notably abstract interpretation [Cousot and Cousot, 1977], we would expect to maintain the soundness of the game while losing the completeness.

When approximating in a partial order, there are no guarantees on the quality of these over approximations. It would be interesting to study which circumstances can obtain guarantees to get the exact solution or, working in a non-discrete setting, to get a solution that is sufficiently close to the exact one.

Connected to the above, another exciting area is integrating up-to techniques in our local algorithm [Sangiorgi, 2005; Pous, 2007], which have been recently shown to be interpretable as complete abstractions for greatest fixpoints [Baldan et al., 2020].

# A

# Normalized systems

In the normalized system we can guarantee a sort of goodness property for the generator $M^{\exists}$, see Definition 4.36. Note that, the normalized system gives to the fixpoint game a wider play graph, instead, the classical one rises to a leaner and long game graph. Since it involves a deep understand of how the parity game react to this normalized system, we are still not sure if it is always convenient to use the normalized system instead of the conventional one.

In a normalized system every symbolic $\exists$-move is "normalized". Let us first define what is a normalized $\exists$-moves.

**Definition A.1** (normalized $\exists$-moves)**.** Let $(\varphi_b^i)_{b\in B_L, i\in \underline{m}}$ be a symbolic $\exists$-move for $E$. We say that $\varphi_b^i$ *is normalized*, if and only if, either

$$\varphi_b^i = [b', j] \text{ such that } b' \in B_L \text{ and } j \in \underline{m}, \text{ or}$$
$$\varphi_b^i = (\bigwedge_{k\in K} \varphi_k) \wedge (\forall k \in K. \exists b' \in B_L, j \in \underline{m} \text{ such that } \varphi_k = [b', j]), \text{ or}$$
$$\varphi_b^i = (\bigvee_{k\in K} \varphi_k) \wedge (\forall k \in K. \exists b' \in B_L, j \in \underline{m} \text{ such that } \varphi_k = [b', j])$$

We say that the system $E$ corresponding to the family $(\varphi_b^i)_{b\in B_L, i\in \underline{m}}$ is *normalized* if for any $b \in B_L$ and $i \in \underline{m}$, $\varphi_b^i$ is normalized.

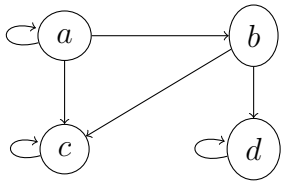


Figure A.1: $\mu$-calculus transition system

Using the definition above we are now able to give some bounds to the generator $M^{\exists}$ to limit its cardinality and to reduce its computational complexity.

**Definition A.2** (generator $M^{\exists}$ in the normalized system)**.** Given a normalized system $E$ of $m$ fixpoint equations over a finite lattice $L$ of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$. Consider the *generator for the normalized system* $M_n^{\exists} : \mathcal{L}_m^H(B_L) \to 2^{(2^{B_L})^m}$ defined as follows, for $b \in B_L$ and $j \in \underline{m}$:

$$
\begin{aligned}
M_n^{\exists}([b,j]) &= \boldsymbol{X} \text{ such that } X_j = \{b\} \wedge X_i = \emptyset \text{ for } i \neq j \\
M_n^{\exists}(\bigwedge_{k \in K} \varphi_k) &= \cup^{\wedge}\{M_n^{\exists}(\varphi_k) \mid k \in K\} \\
M_n^{\exists}(\bigvee_{k \in K} \varphi_k) &= \bigcup_{k \in K} M_n^{\exists}(\varphi_k)
\end{aligned}
$$

It can be seen that in a normalized system of fixpoint equations, for any formula $\varphi$, the two generators return the same set, i.e., $M^{\exists}(\varphi) = M_n^{\exists}(\varphi)$. Observe that in the definition above, the generator $M_n^{\exists}$ is not defined recursively, in the two last cases it just recall the base case since each sub-formula is an atom (in the normalized system), thanks to Definition A.1.

Additionally, we found out that in the $\mu$-calculus settings (and not only) a normalized system is always obtainable given the starting system of equations. The next example, based on $\mu$-calculus, shows two important peculiarities: how to obtain a normalized system given a un-normalized one and how symbolic $\exists$-moves behave in such systems.

**Example 15** (normalized system in the $\mu$-calculus settings)**.** Suppose we are in $\mu$-calculus settings where $\mathbb{S} = \{a, b, c, d\}$ and the transition system is depicted in Figure A.1. Consider the system of fixpoint equations $E$ (conventional form) over

the powerset lattice $2^{\mathbb{S}}$:

$$\begin{cases} x_1 & =_\nu x_2 \cup \blacksquare_{\to_T} x_1 \\ x_2 & =_\mu x_1 \cap \blacklozenge_{\to_T} x_2 \end{cases}$$

First, we translate this system into the normalized one (called $E_n$), making sure that its solution will be untouched, i.e., $sol(E) = sol(E_n)$.

Without giving a formal procedure to transform the system $E$ into the normalized one $E_n$, we just write down some notes:

- Every equation has to apply just one operator at a time.
- If $f_i$ does not respect the rule above, we add a coherent $y_j$ variable to the system (above the $i$-th equation) for each parameter that is passed to the associated operator that is not a variable or a predicate.
- We iterate on the previous rule until each equation respects the first rule introduced.

Thus, we obtain the following system (still not a normalized one):

$$\begin{cases} y_1 & =_\nu \blacksquare_{\to_T} x_1 \\ x_1 & =_\nu x_2 \cup y_1 \\ y_2 & =_\mu \blacklozenge_{\to_T} x_2 \\ x_2 & =_\mu x_1 \cap y_2 \end{cases}$$

We merge all the $x$s and $y$s into a unique naming system, then we obtain the normalized system $E_n$

$$\begin{cases} z_1 & =_\nu \blacksquare_{\to_T} z_2 \\ z_2 & =_\nu z_4 \cup z_1 \\ z_3 & =_\mu \blacklozenge_{\to_T} z_4 \\ z_4 & =_\mu z_2 \cap z_3 \end{cases}$$

Recall the symbolic $\exists$-move construction procedure for $\mu$-calculus exposed in Ex-

ample 11, we obtain the following symbolic ∃-moves, one for each ∃-position:

$$
\begin{aligned}
(a,1) &\implies [\{a\},2] \wedge [\{b\},2] \wedge [\{c\},2] \\
(b,1) &\implies [\{b\},2] \wedge [\{d\},2] \\
(c,1) &\implies [\{c\},2] \\
(d,1) &\implies [\{d\},2] \\
(a,2) &\implies [\{a\},1] \vee [\{a\},4] \\
(b,2) &\implies [\{b\},1] \vee [\{b\},4] \\
(c,2) &\implies [\{c\},1] \vee [\{c\},4] \\
(d,2) &\implies [\{d\},1] \vee [\{d\},4] \\
(a,3) &\implies [\{a\},4] \vee [\{b\},4] \vee [\{c\},4] \\
(b,3) &\implies [\{b\},4] \vee [\{d\},4] \\
(c,3) &\implies [\{c\},4] \\
(d,3) &\implies [\{d\},4] \\
(a,4) &\implies [\{a\},2] \wedge [\{a\},3] \\
(b,4) &\implies [\{b\},2] \wedge [\{b\},3] \\
(c,4) &\implies [\{c\},2] \wedge [\{c\},3] \\
(d,4) &\implies [\{d\},2] \wedge [\{d\},3]
\end{aligned}
$$

While, regarding the un-normalized system, the symbolic ∃-moves generated are the following (again one for each ∃-position):

$$
\begin{aligned}
(a,1) &\implies [\{a\},2] \vee ([\{a\},1] \wedge [\{b\},1] \wedge [\{c\},1]) \\
(b,1) &\implies [\{b\},2] \vee ([\{b\},1] \wedge [\{d\},1]) \\
(c,1) &\implies [\{c\},2] \vee [\{c\},1] \\
(d,1) &\implies [\{d\},2] \vee [\{d\},1] \\
(a,2) &\implies [\{a\},1] \wedge ([\{a\},2] \vee [\{b\},2] \vee [\{c\},2]) \\
(b,2) &\implies [\{b\},1] \wedge ([\{b\},2] \vee [\{d\},2]) \\
(d,2) &\implies [\{c\},1] \wedge [\{c\},2] \\
(c,2) &\implies [\{d\},1] \wedge [\{d\},2]
\end{aligned}
$$

As we expected, we obtain more positions playable for player $\exists$, that is why we increase the number of equations in the pair $(b, i) \in Pos_{\exists}$, the index $i$ ranges over $m$ and, in the normalized system $E_n$, the number of equations $m$ increased to four. In the other hand, no mixing between conjunctions and disjunctions is possible in the normalized system, while in the classical one this often happens. This fact bounds the growth in width of the play graph, whereas it expands its height. Intuitively, in the normalized system we have more moves to be played (positions for player $\exists$), thus the height increases. By the way, for each $\exists$-position $C$, it decreases the set of possible moves playable from $C$, i.e., it decreases the cardinality of the set $M^{\exists}(\varphi_C)$.

# References


Abramsky, S. and Jung, A. (1994). Domain Theory. 3:167.

Baldan, P., König, B., Mika-Michalski, C., Padoan, T., and Mika-Michalski, C. (2018). Fixpoint games on continuous lattices. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–29.

Baldan, P., König, B., and Padoan, T. (2020). Abstraction, Up-to Techniques and Games for Systems of Fixpoint Equations. 1(January):1–28.

Bandelt, H. J. (1982). $\mathfrak{R}$-Distributive Lattices. *Archiv der Mathematik*, 39(5):436–442.

Bonsangue, M. M. and Kok, J. N. (1997). Infinitary domain logic for finitary transition systems. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 1281, pages 213–232. Springer Verlag.

Bradfield, J. and Walukiewicz, I. (2018). The mu-calculus and model checking. In *Handbook of Model Checking*, pages 871–919. Springer International Publishing.

Büchi, J. R. (1989). *Finite Automata, Their Algebras and Grammars*. Springer New York.

Calude, C. S., Jain, S., Khoussainov, B., Li, W., and Stephan, F. (2017). Deciding parity games in quasipolynomial time. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, volume Part F1284, pages 252–263. Association for Computing Machinery.

Cleaveland, R., Klein, M., and Steffen, B. (1993). Faster model checking for the modal mu-calculus. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 663 LNCS, pages 410–422. Springer Verlag.

Cousot, P. and Cousot, R. (1977). Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Annual ACM Symposium on Principles of Programming Languages*, volume Part F1307, pages 238–252. Association for Computing Machinery.

Davey, B. A. and Priestley, H. A. (2002). *Introduction to Lattices and Order*. Cambridge University Press.

Emerson, E. A. and Jutla, C. S. (1991). Tree automata, mu-calculus and determinacy. In *Annual Symposium on Foundations of Computer Science (Proceedings)*, pages 368–377. Publ by IEEE.

Garg, V. K. (2015). Distributive Lattices. In *Introduction to Lattice Theory with Computer Science Applications*, pages 99–106. John Wiley & Sons, Inc.

Gawlitza, T. M. and Seidl, H. (2011). Solving systems of rational equations through strategy iteration. *ACM Transactions on Programming Languages and Systems*, 33(3).

Hansen, H. H., Kupke, C., Marti, J., and Venema, Y. (2017). Parity Games and Automata for Game Logic (Extended Version).

Hasuo, I., Shimizu, S., and Cîrstea, C. (2016). Lattice-theoretic progress measures and coalgebraic model checking. *ACM SIGPLAN Notices*, 51(1):718–732.

Hirschkoff, D. (1998). Automatically proving up-to bisimulation. In *Electronic Notes in Theoretical Computer Science*, volume 18, pages 75–89. Elsevier.

Hirschkoff, D. (1999). Mise en oeuvre de preuves de bisimulation. *Ph.D. Dissertation, Ecole Nationale des Ponts et Chaussées (ENPC)*.

Jurdziński, M. (2000). Small progress measures for solving parity games. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1770(February):290–301.

Kozen, D. (1983). Results on the propositional $\mu$-calculus. *Theoretical Computer Science*, 27(3):333–354.

Mazzocchin, G. (2019). Solving fixpoint equations using Progress Measures. *Master dissertation, Department of Mathematics, University of Padua*, (April).

Pous, D. (2007). Complete lattices and up-to techniques. In *APLAS*, volume 4807 of *Lecture Notes in Computer Science*, pages 351–366. Springer.

Ranzato, F. (2017). A new characterization of Complete Heyting and Co-Heyting Algebras. *Logical Methods in Computer Science*, 13:1–11.

Sangiorgi, D. (2005). Beyond bisimulation: The "up-to" techniques. In *FMCO*, volume 4111 of *Lecture Notes in Computer Science*, pages 161–171. Springer.

Sangiorgi, D. (2011). *Introduction to bisimulation and coinduction*, volume 9781107003. Cambridge University Press.

Stevens, P. and Stirling, C. (1998). Practical model-checking using games. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 1384, pages 85–101. Springer Verlag.

Stirling, C. (1995). Local model checking games. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 962, pages 1–11. Springer Verlag.

Tarski, A. (1955). A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309.

# Acknowledgments

Throughout the writing of this dissertation, I have received a great deal of support and assistance. First, I would like to thank my supervisor, Professor Paolo Baldan, whose expertise was invaluable, and my co-supervisor, Postdoc Tommaso Padoan, both from the Department of Mathematics, University of Padua. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level. I also want to thank you for your patient support about my writing skills.

I would like to thank a remote helper, researcher Emil Jeřábek from the Institute of Mathematics of the Czech Academy of Sciences, for the assistance he gave me about some proofs in chapter 2.

Finally, I would like to thank my friends Joel Parman and Elisa Tobaldo for helping me in the compilation of this dissertation.