Abstraction, Up-to Techniques and Games for Systems of Fixpoint Equations

Paolo Baldan
Dipartimento di Matematica "Tullio Levi-Civita"
Università di Padova
Padova, Italy
baldan@math.unipd.it

Barbara König
Fakultät für Ingenieurwissenschaften, Abteilung Informatik und Angewandte Kognitionswissenschaft
Universität Duisburg-Essen
Duisburg, Germany
barbara_koenig@uni-due.de

Tommaso Padoan
Dipartimento di Matematica "Tullio Levi-Civita"
Università di Padova
Padova, Italy
padoan@math.unipd.it

Abstract

Systems of fixpoint equations over complete lattices, consisting of (mixed) least and greatest fixpoint equations, allow one to express a number of verification tasks such as model-checking of various kinds of specification logics or the check of coinductive behavioural equivalences. In this paper we develop a theory of approximation for systems of fixpoint equations in the style of abstract interpretation: a system over some concrete domain is abstracted to a system in a suitable abstract domain, with conditions ensuring that the abstract solution represents a sound/complete over-approximation of the concrete solution. Interestingly, up-to techniques, a classical approach used in coinductive settings to obtain easier or feasible proofs, can be interpreted as abstractions in a way that they naturally fit in our framework and extend to systems of equations. Additionally, relying on the approximation theory, we can provide a characterisation of the solution of systems of fixpoint equations over complete lattices in terms of a suitable parity game, generalising some recent work that was restricted to continuous lattices. The game view opens the way to the development of on-the-fly algorithms for characterising the solution of such equation systems.

Keywords fixpoint equation systems, complete lattices, parity games, abstract interpretation, up-to techniques, local algorithms, μ-calculus, bisimilarity

1 Introduction

Systems of fixpoint equations over complete lattices, consisting of (mixed) least and greatest fixpoint equations, allow one to uniformly express a number of verification tasks. Notable examples come from the area of model-checking. Invariant/safety properties can be characterised as greatest fixpoints, while liveness/reachability properties as least fixpoints. Using both least and greatest fixpoints leads to very expressive specification logics. The μ-calculus [Kozen 1983] is a prototypical example, encompassing various other logics such as LTL and CTL. Another area of special interest for the present paper is that of behavioural equivalences, which typically arise as solutions of greatest fixpoint equations. The most famous example is bisimilarity that can be seen as the greatest fixpoint of a suitable operator over the lattice of binary relations on states (see, e.g., [Sangiorgi 2011]).

In the first part of the paper, after a general introduction to systems of equations over complete lattices, clarifying how the modal μ-calculus, its probabilistic variants and bisimilarity can be viewed as examples, we develop a theory of approximation for systems of equations in the style of abstract interpretation. The general idea of abstract interpretation [Cousot and Cousot 1977, 1979a] is simple and effective. It consists of extracting properties of programs by defining an approximated program semantics over a so-called abstract domain, typically a complete lattice, whose elements can be seen as properties of the concrete semantics. Moving from the concrete to the abstract domain, on the one hand, allows one to focus on the program properties of interest, on the other hand, it is often essential to make the check effective.

Concrete and abstract program semantics are typically expressed in terms of (systems of) least fixpoint equations, and suitable conditions can be imposed ensuring that the approximation obtained is sound, namely that, roughly, properties derived from the abstract semantics are also valid at concrete level. In an ideal situation also the converse holds, a situation referred to as completeness of the abstract interpretation (see [Giacobazzi et al. 2000] and references therein), which ensures the absence of false alarms.

We generalise this idea to systems of fixpoint equations, where least and greatest fixpoints can coexist. A system over some concrete domain C is abstracted by a system over some abstract domain A, which is intended to provide an over-approximation of the concrete one. Suitable conditions are identified that ensure the soundness and completeness of

arXiv, 2020
2020.

the approximation. This enables the use of the approximation theory on a number of verification tasks. As an example, we show that some results on property preserving abstractions for the μ-calculus [Loiseaux et al. 1995] arise as instances of our theory. We also discuss the use of the approximation theory for systems of fixpoint equations over the real interval $[0, 1]$ arising from a fixpoint extension of Łukasiewicz logic, considered in [Mio and Simpson 2017] as a precursor to model-checking PCTL or probabilistic μ-calculi.

When dealing with greatest fixpoints, a key proof technique relies on the coinduction principle. It naturally spills out of Tarski's Theorem, that states that a monotone function f over a complete lattice has a greatest fixpoint νf, which is the join of all post-fixpoints, i.e., the elements l such that $l \sqsubseteq f(l)$. As a consequence proving $l \sqsubseteq f(l)$ suffices to conclude that $l \sqsubseteq \nu f$. Note that the focus here is on underapproximations of the fixpoint: for instance, if f is the operator on relations having bisimilarity as the greatest fixpoint, one is normally interested in checking the bisimilarity of two states, say s_1 and s_2, i.e., in checking whether $\{(s_1, s_2)\} \subseteq \nu f$.

In this setting, up-to techniques have been proposed for "simplifying" proofs [Milner 1989; Pous 2007; Pous and Sangiorgi 2011; Sangiorgi and Milner 1992]. They turn out to be helpful not only for shortening hand-written proofs, but also for reducing the search space in fully automatic verification algorithms (see e.g. [Bonchi and Pous 2013] where up-to techniques provide an exponential speed-up for language equivalence of non-deterministic automata). Roughly speaking, a sound up-to function is a function u on the lattice such that $\nu(f \circ u) \sqsubseteq \nu f$ so that $l \sqsubseteq f(u(l))$ implies $l \sqsubseteq \nu(f \circ u) \sqsubseteq \nu f$. The characteristics of u (typically, extensiveness) should make it easier to show that an element is a post-fixpoint of $f \circ u$ rather than a post-fixpoint of f.

We show that up-to techniques admit a natural interpretation as abstractions in our approximation framework. This fact, besides being of interest in itself, allows us to generalise smoothly the theory of up-to techniques to systems of fixpoint equations. It also contributes to the understanding of the relation between abstract interpretation and up-to techniques, a theme that received some recent attention [Bonchi et al. 2018a].

Some recent work [Baldan et al. 2019] has shown that the solution of systems of fixpoint equations can be characterised in terms of a parity game when working in a suitable subclass of complete lattices, the so-called continuous lattices [Scott 1972]. Here, relying on our approximation theory, we get rid of the continuity hypothesis and design a game that works for systems of equations over general complete lattices. The simple but crucial observation is that a system of equations over any complete lattice L can be "transferred" to a system of equations over the powerset of a basis $\mathbf{2}^{B_L}$ (which is always continuous) by means of a Galois insertion.

The above results opens the way to the development of algorithms, possibly integrating abstraction and up-to techniques, for solving the game, i.e., for determining winning and losing positions for the players, which in turn corresponds to solving the associated verification problem. Global algorithms establishing the winner at each position can be based, e.g., on progress measures (originally proposed in [Jurdziński 2000] and adapted to systems of equations in [Baldan et al. 2019; Hasuo et al. 2016]). Local algorithms, confining the attention to specific positions, can be devised taking inspiration from backtracking methods for bisimilarity [Hirschkoff 1998] and for the μ-calculus [Stevens and Stirling 1998; Stirling 1995]. We will outline an on-the-fly algorithm for the case of a single equation in §6.2. This will allow us to establish a link with some recent work relating abstract interpretation and up-to techniques [Bonchi et al. 2018a] and exploiting up-to techniques for computing language equivalence on NFAs [Bonchi and Pous 2013].

Subsequently we will consider a version of the algorithm for the general case.

Our contributions can be summarized as follows:

▹ We develop a theory of (sound and complete) approximations for systems of fixpoint equations. This is challenging due to the interplay of least and greatest fixpoints. (§4)

▹ We derive a theory of up-to functions for systems of fixpoint equations. This is done by suitably instantiating the theory developed earlier, since an up-to function can be canonically transformed into a closure which in turn can be seen as a Galois insertion. (§5)

▹ Generalising [Baldan et al. 2019], we present a game that allows to characterize the solution of systems of fixpoint equations. Differently from [Baldan et al. 2019] it works for all complete lattices, not just for continuous ones. (§6.1)

▹ We use the theory of games and up-to techniques to provide on-the-fly algorithms for characterising the solution of a single fixpoint equation. In particular, we focus on the special case in [Bonchi et al. 2018a] where the function – whose fixpoint we want to determine – is a right adjoint. (§6.2)

▹ We give a local on-the-fly algorithm for solving the game in the general case, i.e. for checking whether a given lattice element is below the solution. This algorithm generalises the one proposed in [Stevens and Stirling 1998] for μ-calculus model-checking and applies to the solution of arbitrary equations systems. We also show how this algorithm can be enhanced with up-to techniques. (§7)

Proofs can be found in the appendix.

2 Preliminaries and notation

We provide the basic order theoretic notions used in the paper and fix the notation for tuples of elements that will be useful when dealing with systems of equations.

A preordered or partially ordered set $\langle P, \sqsubseteq \rangle$ is often denoted simply as P, omitting the (pre)order relation. Given $X \subseteq P$, we denote by $\downarrow X = \{p \in P \mid \exists x \in X.\, p \sqsubseteq x\}$ the *downward-closure* of X. The *join* and the *meet* of a subset $X \subseteq P$ (if they exist) are denoted $\bigsqcup X$ and $\bigsqcap X$, respectively.

Definition 2.1 (complete lattice, basis)**.** A *complete lattice* is a partially ordered set $(L, \sqsubseteq)$ such that each subset $X \subseteq L$ admits a join $\bigsqcup X$ and a meet $\bigsqcap X$. A complete lattice $(L, \sqsubseteq)$ always has a least element $\bot = \bigsqcup \emptyset$ and a greatest element $\top = \bigsqcap \emptyset$. A *basis* for a complete lattice is a subset $B_L \subseteq L$ such that for each $l \in L$ it holds that $l = \bigsqcup \{b \in B_L \mid b \sqsubseteq l\}$.

For instance, the powerset of any set X, ordered by subset inclusion $(2^X, \subseteq)$ is a complete lattice. Join is union, meet is intersection, top is X and bottom is $\emptyset$. A basis is the set of singletons $B_{2^X} = \{\{x\} \mid x \in X\}$. Another complete lattice used in the paper is the real interval $[0, 1]$ with the usual order $\leq$. Join and meet are the sup and inf over real numbers, 0 is bottom and 1 is top. Any dense subset, e.g., the set of rationals $\mathbb{Q} \cap (0, 1]$, is a basis.

A function $f \colon L \to L$ is *monotone* if for all $l, l' \in L$, if $l \sqsubseteq l'$ then $f(l) \sqsubseteq f(l')$. By Knaster-Tarski's theorem [Tarski 1955, Theorem 1], any monotone function f on a complete lattice has a least and a greatest fixpoint, denoted respectively μf and νf, characterised as the meet of all pre-fixpoints respectively the join of all post-fixpoints: $\mu f = \bigsqcap \{l \mid f(l) \sqsubseteq l\}$ and $\nu f = \bigsqcup \{l \mid l \sqsubseteq f(l)\}$.

Given a complete lattice L, a subset $X \subseteq L$ is *directed* if $X \neq \emptyset$ and every pair of elements in X has an upper bound in X. If L, L' are complete lattices, a function $f : L \to L'$ is *(directed-)continuous* if for any directed set $X \subseteq L$ it holds $f(\bigsqcup X) = \bigsqcup f(X)$. The function f is called *strict* if $f(\bot) = \bot$. Co-continuity and co-strictness are defined dually.

Definition 2.2 (Galois connection)**.** Let $(C, \sqsubseteq)$, $(A, \leq)$ be complete lattices. A *Galois connection* (or *adjunction*) is a pair of monotone functions $\langle \alpha, \gamma \rangle$ such that $\alpha : C \to A$, $\gamma : A \to C$ and for all $a \in A$ and $c \in C$

$$\alpha(c) \sqsubseteq a \quad \text{iff} \quad c \sqsubseteq \gamma(a).$$

Equivalently, for all $a \in A$ and $c \in C$, (i) $c \sqsubseteq \gamma(\alpha(c))$ and (ii) $\alpha(\gamma(a)) \leq a$. In this case we will write $\langle \alpha, \gamma \rangle : C \to A$. The Galois connection is called an *insertion* when $\alpha \circ \gamma = id_A$.

For a Galois connection $\langle \alpha, \gamma \rangle : C \to A$, the function α is called the left (or lower) adjoint and γ the right (or upper) adjoint. The left adjoint α preserves all joins and the right adjoint γ preserves all meets. Hence, in particular, the left adjoint is strict and continuous, while the right adjoint is co-strict and co-continuous.

A function $f : L \to L$ is *idempotent* if $f \circ f = f$ and *extensive* if $l \sqsubseteq f(l)$ for all $l \in L$. When f is monotone, extensive and idempotent it is called an *(upper) closure*. In this case, $\langle f, i \rangle : L \to f(L)$, where i is the inclusion, is a Galois insertion. Moreover, $f(L) = \{f(l) \mid l \in L\}$ is a complete lattice (by Knaster-Tarski's theorem, since $f(L)$ is the set of fixpoints of f).

We will often consider tuples of elements. Given a set A, an n-tuple in A^n is denoted by a boldface letter $\boldsymbol{a}$. The components of an n-tuple $\boldsymbol{a}$ are denoted as $\boldsymbol{a} = (a_1, \ldots, a_n)$. For an index $n \in \mathbb{N}$ we write $\underline{n}$ for the integer interval $\{1, \ldots, n\}$. Given $\boldsymbol{a} \in A^n$ and $i, j \in \underline{n}$ we write $\boldsymbol{a}_{i,j}$ for the subtuple $(a_i, a_{i+1}, \ldots, a_j)$. The empty tuple is denoted by $()$. Given two tuples $\boldsymbol{a} \in A^m$ and $\boldsymbol{a'} \in A^n$ we denote by $(\boldsymbol{a}, \boldsymbol{a'})$ or simply by $\boldsymbol{a}\boldsymbol{a'}$ their concatenation in A^{m+n}.

Definition 2.3 (pointwise order)**.** Given a complete lattice $(L, \sqsubseteq)$ we will denote by $(L^n, \sqsubseteq)$ the set of n-tuples endowed with the *pointwise order* defined, for $\boldsymbol{l}, \boldsymbol{l'} \in L^n$, by $\boldsymbol{l} \sqsubseteq \boldsymbol{l'}$ if $l_i \sqsubseteq l'_i$ for all $i \in \underline{n}$.

The structure $(L^n, \sqsubseteq)$ is a complete lattice. More generally, for any set X, the set of functions $L^X = \{f \mid f \colon X \to L\}$, endowed with pointwise order, is a complete lattice.

A tuple of functions $\boldsymbol{f} = (f_1, \ldots, f_m)$ with $f_i : X \to Y$, will be seen itself as a function $\boldsymbol{f} : X \to Y^m$, defined by $\boldsymbol{f}(x) = (f_1(x), \ldots, f_m(x))$. We will also need to consider the *product function* $\boldsymbol{f}^{\times} : X^m \to Y^m$, defined by $\boldsymbol{f}^{\times}(x_1, \ldots, x_m) = (f_1(x_1), \ldots, f_m(x_m))$.

3 Systems of Fixpoint Equations over Complete Lattices

In the paper we deal with systems of (fixpoint) equations over some complete lattice, where, for each equation one can be interested either in the least or in the greatest solution. We define systems, their solutions and we provide some examples that will be used as running examples in the paper.

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

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

where $f_i \colon L^m \to L$ are monotone functions (with respect to the pointwise order on L^m) and $\eta_i \in \{\mu, \nu\}$. The system will often be denoted as $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$, where $\boldsymbol{x}$, $\boldsymbol{\eta}$ and $\boldsymbol{f}$ are the obvious tuples. We denote by $\emptyset$ the system with no equations.

Systems of equations of this kind have been considered in the literature in connection to verification problems (see e.g., [Baldan et al. 2019; Cleaveland et al. 1992; Hasuo et al. 2016; Seidl 1996]. In particular, [Baldan et al.

2019; Hasuo et al. 2016] work on general classes of complete lattices.

Note that $\boldsymbol{f}$ can be seen as a function $\boldsymbol{f}: L^m \to L^m$. The solution of the system is a selected fixpoint of such function. We first need some auxiliary notation.

Definition 3.2 (substitution)**.** Given a system E of m equations over a complete lattice L of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$, an index $i \in \underline{m}$ and $l \in L$ 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 $\boldsymbol{x}'\boldsymbol{x}'' =_{\boldsymbol{\eta}'\boldsymbol{\eta}''} \boldsymbol{f}'\boldsymbol{f}''(\boldsymbol{x}', l, \boldsymbol{x}'')$.

We can now define the solution of a system of equations.

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

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

where $s_m = \eta_m(\lambda x.\, f_m(sol(E[x_m := x]), x))$.

In words, for solving a system of m equations, the last variable is considered as a fixed parameter x and the system of $m-1$ equations that arises from dropping the last equation is recursively solved. This produces an $(m-1)$-tuple parametric on x, i.e., we get $\boldsymbol{s}_{1,m-1}(x) = sol(E[x_m := x])$. Inserting this parametric solution 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 for the function $\lambda x.\, f_m(\boldsymbol{s}_{1,m-1}(x), x)$, the least or greatest fixpoint, depending on whether the last equation is a μ- or ν-equation. This provides the m-th component of the solution $s_m = \eta_m(\lambda x.\, f_m(\boldsymbol{s}_{1,m-1}(x), x))$. The remaining components of the solution are obtained inserting s_m in the parametric solution $\boldsymbol{s}_{1,m-1}(x)$ previously computed, i.e., $\boldsymbol{s}_{1,m-1} = \boldsymbol{s}_{1,m-1}(s_m)$.

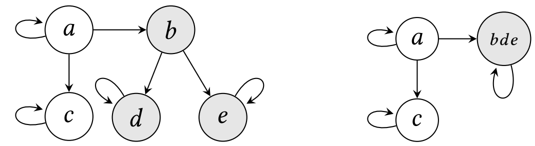
Note that the order of equation matters, reordering the equations typically results in a different solution.

Example 3.4 (μ-calculus)**.** Several authors observed that μ-calculus formulae can be equivalently presented as systems of fixpoint equations (see, e.g., [Cleaveland et al. 1992; Seidl 1996]). We adopt a standard μ-calculus syntax. For fixed disjoint sets *PVar* of propositional variables, ranged over by $x, y, z, \ldots$ and *Prop* of propositional symbols, ranged over by $p, q, r, \ldots$, formulae are defined by

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

where $p \in Prop$, $x \in PVar$ and $\eta \in \{\mu, \nu\}$.

The semantics of a formula is given with respect to an unlabelled transitions system (or Kripke structure) $T = (\mathbb{S}_T, \to_T)$ where $\mathbb{S}_T$ is the set of states and $\to_T \subseteq \mathbb{S}_T \times \mathbb{S}_T$



(a) (b)

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

(c)

$$\begin{array}{rcl} x_1 & =_\nu & \{b, d, e\} \cap \blacksquare_T x_1 \\ x_2 & =_\mu & x_1 \cup \blacklozenge_T x_2 \end{array}$$

(d)

Figure 1

is the transition relation. Given a formula φ and an environment $\rho: Prop \cup PVar \to 2^{\mathbb{S}_T}$ mapping each proposition or propositional variable to the set of states where it holds, we denote by $\|\varphi\|_\rho^T$ the semantics of φ defined as usual (see, e.g., [Bradfield and Walukiewicz 2018]).

A μ-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. The equational form is reported in Fig. 1c. Consider a transition system $T = (\mathbb{S}_T, \to_T)$ where $\mathbb{S}_T = \{a, b, c, d, e\}$ and $\to_T$ is as depicted in Fig. 1a, with p that holds in the grey states b, d and e. Define the semantic counterpart of the modal operators as follows: given a relation $R \subseteq X \times X$ let us write $\blacklozenge_R, \blacksquare_R : 2^X \to 2^X$ for the functions defined by $\blacklozenge_R(Y) = \{x \in X \mid \exists y \in Y.\ (x, y) \in R\}$, $\blacksquare_R(Y) = \{x \in X \mid \forall y \in X.(x, y) \in R \Rightarrow y \in Y\}$ for $Y \subseteq X$.

Then the formula φ interpreted over the transition system T leads to the system of equations over the lattice $2^{\mathbb{S}_T}$ in Fig. 1d, where we write $\blacklozenge_T$ and $\blacksquare_T$ for $\blacklozenge_{\to_T}$ and $\blacksquare_{\to_T}$.

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

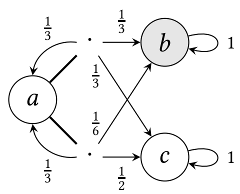
Example 3.5 (Łukasiewicz μ-terms)**.** Systems of equations over the real interval $[0, 1]$ have been considered in [Mio and Simpson 2017] as a precursor to model-checking PCTL or probabilistic μ-calculi. More precisely, the authors study a fixpoint extension of Łukasiewicz logic, referred to as Łukasiewicz μ-terms, whose syntax is as follows:

$$t ::= \mathbf{1} \mid \mathbf{0} \mid x \mid r \cdot t \mid t \sqcup t \mid t \sqcap t \mid t \oplus t \mid t \odot t \mid \eta x.t$$

where $x \in PVar$ is a variable (ranging over $[0, 1]$), $r \in [0, 1]$ and $\eta \in \{\mu, \nu\}$. The various syntactic operators have a semantic counterpart. For all $x, y \in [0, 1]$

$$\begin{array}{ll} \mathbf{0}(x) = 0,\ \mathbf{1}(x) = 1 & \text{(constant)} \\ r \cdot x = rx & \text{(scalar multiplication)} \\ x \sqcup y = \max(x, y) & \text{(weak disjunction)} \\ x \sqcap y = \min(x, y) & \text{(weak conjunction)} \\ x \oplus y = \min(x + y, 1) & \text{(strong disjunction)} \\ x \odot y = \max(x + y - 1, 0) & \text{(strong conjunction)} \end{array}$$

Then, each Łukasiewicz μ-term, in an environment $\rho : PVar \to [0, 1]$, can be assigned a semantics which is a



(a) A PNDT

$$\varphi \quad \begin{cases} x_1 =_\nu p \odot \Box x_1 \\ x_2 =_\mu x_1 \oplus \Diamond x_2 \end{cases}$$

$$\varphi' \quad \begin{cases} x_1 =_\nu p \odot \Box x_1 \\ x_2 =_\mu x_1 \oplus \Box x_2 \end{cases}$$

(b) Formulas as systems

Figure 2

real number in $[0,1]$, denoted as $\|t\|_\rho$. Exactly as for the μ-calculus, a Łukasiewicz μ-term can be naturally seen as a system of fixpoint equations over the lattice $[0,1]$. For instance, the term $\nu x_2.\,(\mu x_1.\,(\frac{5}{8} \oplus \frac{3}{8}x_2) \odot (\frac{1}{2} \sqcup (\frac{3}{8} \oplus \frac{1}{2}x_1)))$ from an example in [Mio and Simpson 2017], can be written as the system:

$$\begin{aligned} x_1 &=_\mu (\frac{5}{8} \oplus \frac{3}{8}x_2) \odot (\frac{1}{2} \sqcup (\frac{3}{8} \oplus \frac{1}{2}x_1)) \\ x_2 &=_\nu x_1 \end{aligned}$$

Example 3.6 (Łukasiewicz μ-calculus)**.** The Łukasiewicz μ-calculus, as defined in [Mio and Simpson 2017], extends the Łukasiewicz μ-terms with propositions and modal operators. The syntax is as follows:

$$\varphi \ ::= \ p \mid \bar{p} \mid r \cdot \varphi \mid \varphi \sqcup \varphi \mid \varphi \sqcap \varphi \mid \varphi \oplus \varphi \mid \varphi \odot \varphi \mid \Diamond\varphi \mid \Box\varphi \mid \eta x.t$$

where x ranges in a set *PVar* of propositional variables, p ranges in a set *Prop* of propositional symbols, each paired with an associated complement $\bar{p}$, and $\eta \in \{\mu, \nu\}$.

The Łukasiewicz μ-calculus can be seen as a logic for probabilistic transition systems. It extends the quantitative modal μ-calculus of [Huth and Kwiatkowska 1997; McIver and Morgan 2007] and it allows to encode PCTL [Bianco and de Alfaro 1995]. For a finite set $\mathbb{S}$, the set of (discrete) probability distributions over $\mathbb{S}$ is defined as $\mathcal{D}(\mathbb{S}) = \{d : \mathbb{S} \to [0,1] \mid \sum_{s \in \mathbb{S}} d(s) = 1\}$. A formula is interpreted over a *probabilistic non-deterministic transition system (PNDT)* $N = (\mathbb{S}, \to)$ where $\to \subseteq \mathbb{S} \times \mathcal{D}(\mathbb{S})$ is the transition relation. An example of PNDT can be found in Fig. 2a. Imagine that the aim is to reach state b. State a has two transitions. A "lucky" one where the probability to get to b is $\frac{1}{3}$ and an "unlucky" one where b is reached with probability $\frac{1}{6}$. For both transitions, with probability $\frac{1}{3}$ one gets back to a and then, with the residual probability, one moves to c. Once in states b or c, the system remains in the same state with probability 1.

Given a formula φ and an environment $\rho : Prop \cup PVar \to (\mathbb{S} \to [0,1])$ mapping each proposition or propositional variable to a real-valued function over the states, the semantics of φ is a function $\|\varphi\|^N_\rho : \mathbb{S} \to [0,1]$ defined as expected using the semantic operators. In addition to those already discussed, we have the semantic operators for the complement and the modalities: for $v : \mathbb{S} \to [0,1]$

$$\bar{v}(x) = 1 - v(x) \qquad \blacklozenge_N(v)(x) = \max_{x \to d} \sum_{y \in \mathbb{S}} d(y)\, v(y)$$

$$\blacksquare_N(v)(x) = \min_{x \to d} \sum_{y \in \mathbb{S}} d(y)\, v(y)$$

As it happens for the propositional μ-calculus, also formulas of the Łukasiewicz μ-calculus can be seen as systems of equations, but on a different complete lattice, i.e., $[0,1]^{\mathbb{S}}$. For instance, consider the formulas $\varphi = \mu x_2.(\nu x_1.(p \odot \Box x_1) \oplus \Diamond x_2)$ and $\varphi' = \mu x_2.(\nu x_1.(p \odot \Box x_1) \oplus \Box x_2)$, rendered as (syntactic) equations in Fig. 2b. Roughly speaking, they capture the probability of eventually satisfying forever p, with an angelic scheduler and a daemonic one, choosing at each step the best or worst transition, respectively. Assuming that p holds with probability 1 on b and 0 on a and c, we have $\|\varphi\|_\rho(a) = \frac{1}{2}$ and $\|\varphi'\|_\rho(a) = \frac{1}{4}$.

Example 3.7 ((bi)similarity over transition systems)**.** For defining (bi)similarity uniformly with the example on μ-calculus, we work on unlabelled transition systems with atoms $T = (\mathbb{S}, \to, A)$ where $A \subseteq \mathbf{2}^{\mathbb{S}}$ is a fixed set of atomic properties over the states. Everything can be easily adapted to labelled transition systems.

Given a transition system with atoms $T = (\mathbb{S}, \to, A)$, consider the lattice of relations on $\mathbb{S}$, namely $\mathsf{Rel}(\mathbb{S}) = (\mathbf{2}^{\mathbb{S} \times \mathbb{S}}, \subseteq)$. We consider as basis the set of singletons, i.e., $B_L = \{\{(x,y)\} \mid x, y \in \mathbb{S}\}$. The *similarity relation* on T, denoted $\precsim_T$, is defined as the greatest fixpoint of the function $sim_T : \mathsf{Rel}(\mathbb{S}) \to \mathsf{Rel}(\mathbb{S})$, defined by

$$sim_T(R) = \left\{ \begin{array}{l|l} (x,y) \in R & \forall a \in A.\, x \in a \Rightarrow y \in a \\ & \forall x \to x'.\, \exists y \to y'.\, (x',y') \in R \end{array} \right\}$$

In other words it can be seen as the solution of a system consisting of a single greatest fixpoint equation

$$x =_\nu sim_T(x).$$

For instance, consider the transition system T in Fig. 1a and take $p = \{b, d, e\}$ as the only atom. Then similarity $\precsim_T$ is the transitive and reflexive closure of $\{(c,a), (a,b), (b,d), (d,e), (e,b)\}$.

Bisimilarity $\sim_T$ can be obtained analogously as the greatest fixpoint of

$$bis_T(R) = sim_T(R) \cap sim_T(R^{-1}).$$

In the transition system T above, bisimilarity $\sim_T$ is the equivalence such that $b \sim_T d \sim_T e$.

4 Approximation for Systems of Fixpoint Equations

In this section we design a theory of approximation for systems of fixpoint equations over complete lattices.

The general setup is borrowed from abstract interpretation [Cousot and Cousot 1977, 1979a], where a concrete domain C and an abstract domain A are fixed. Semantic operators on the concrete domain C have a counterpart in the abstract domain A, and suitable conditions can be imposed on such operators to ensure that the least fixpoints of the abstract operators (or of functions built out of such operators) are sound and/or complete approximations of the fixpoints of their concrete counterparts.

Similarly, here we will have a system of equations $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^C(\boldsymbol{x})$ defined over a concrete domain C and its abstract counterpart $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^A(\boldsymbol{x})$ defined over an abstract domain A, and we want to ensure that the solution of the latter provides an approximation of the solution of the former. The presence of least and greatest fixpoints requires special care in order to single out conditions working for both kinds of fixpoints at the same time.

Let us first focus on the case of a single equation. Let $(C, \sqsubseteq)$ and $(A, \leq)$ be complete lattices and let $f^C : C \to C$ and $f^A : A \to A$ be monotone functions. The fact that f^A is a sound (over)approximation of f^C can be formulated in terms of a concretisation function $\gamma : A \to C$, that maps each abstract element $a \in A$ to a concrete element $\gamma(a) \in C$, for which, intuitively, a is an overapproximation. In the setting of abstract interpretation, where the interest is for program semantics, typically expressed in terms of least fixpoints, the desired *soundness* property is

$$\mu f^C \sqsubseteq \gamma(\mu f^A)$$

A standard sufficient condition for soundness (see [Cousot and Cousot 1977, 1979a; Miné 2017]) is

$$f^C \circ \gamma \sqsubseteq \gamma \circ f^A \tag{1}$$

The same condition ensures soundness also for greatest fixpoints, i.e., $\nu f^C \sqsubseteq \gamma(\nu f^A)$, provided that γ is co-continuous and co-strict (see Lemma A.1(a) in the appendix).

For dealing with systems of equations, we rely on the above results, carefully combining the conditions for least and greatest fixpoints. We will allow a different concretisation function for each equation.

Theorem 4.1 (sound concretisation for systems)**.** *Let $(C, \sqsubseteq)$ and $(A, \leq)$ be complete lattices, let E_C of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^C(\boldsymbol{x})$ and E_A of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^A(\boldsymbol{x})$ be systems of m equations over C and A, with solutions $\boldsymbol{s}^C \in C^m$ and $\boldsymbol{s}^A \in A^m$, respectively. Let $\boldsymbol{\gamma}$ be an m-tuple of monotone functions, with $\gamma_i : A \to C$ for $i \in \underline{m}$. If $\boldsymbol{\gamma}$ satisfies*

$$\boldsymbol{f}^C \circ \boldsymbol{\gamma}^\times \sqsubseteq \boldsymbol{\gamma}^\times \circ \boldsymbol{f}^A \tag{2}$$

with γ_i co-continuous and co-strict for each $i \in \underline{m}$ such that $\eta_i = \nu$, then $\boldsymbol{s}^C \sqsubseteq \boldsymbol{\gamma}^\times(\boldsymbol{s}^A)$.

The standard abstract interpretation framework of [Cousot and Cousot 1979c] relies on Galois connections: concretisation functions γ are right adjoints, whose left adjoint, the abstraction function α, intuitively maps each concrete element in C to its "best" overapproximation in A. When $\langle \alpha, \gamma \rangle$ is a Galois connection, each component determines the other. Moreover, α is automatically continuous and strict, while γ is co-continuous and co-strict. This leads to the following result, where, besides the soundness conditions, we make also explicit the dual conditions that we refer to as completeness conditions.

Theorem 4.2 (abstraction via Galois connections)**.** *Let $(C, \sqsubseteq)$ and $(A, \leq)$ be complete lattices, let E_C of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^C(\boldsymbol{x})$ and E_A of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^A(\boldsymbol{x})$ be systems of m equations over C and A, with solutions $\boldsymbol{s}^C \in C^m$ and $\boldsymbol{s}^A \in A^m$, respectively. Let $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$ be m-tuples of monotone functions, with $\langle \alpha_i, \gamma_i \rangle : C \to A$ forming a Galois connection for each $i \in \underline{m}$.*

a) Soundness*: If $\boldsymbol{\gamma}$ satisfies $\boldsymbol{f}^C \circ \boldsymbol{\gamma}^\times \sqsubseteq \boldsymbol{\gamma}^\times \circ \boldsymbol{f}^A$ (2) or equivalently $\boldsymbol{\alpha}$ satisfies*

$$\boldsymbol{\alpha}^\times \circ \boldsymbol{f}^C \leq \boldsymbol{f}^A \circ \boldsymbol{\alpha}^\times, \tag{3}$$

then $\boldsymbol{\alpha}^\times(\boldsymbol{s}^C) \leq \boldsymbol{s}^A$ (equivalent to $\boldsymbol{s}^C \leq \boldsymbol{\gamma}^\times(\boldsymbol{s}^A)$).

b) Completeness (for abstraction)*: If $\boldsymbol{\alpha}$ satisfies*

$$\boldsymbol{f}^A \circ \boldsymbol{\alpha}^\times \leq \boldsymbol{\alpha}^\times \circ \boldsymbol{f}^C \tag{4}$$

with α_i co-continuous and co-strict for each $i \in \underline{m}$ such that $\eta_i = \nu$, then $\boldsymbol{s}^A \leq \boldsymbol{\alpha}^\times(\boldsymbol{s}^C)$.

c) Completeness (for concretisation)*: If $\boldsymbol{\gamma}$ satisfies*

$$\boldsymbol{\gamma}^\times \circ \boldsymbol{f}^A \sqsubseteq \boldsymbol{f}^C \circ \boldsymbol{\gamma}^\times \tag{5}$$

with γ_i continuous and strict for each $i \in \underline{m}$ such that $\eta_i = \mu$, then $\boldsymbol{\gamma}^\times(\boldsymbol{s}^A) \sqsubseteq \boldsymbol{s}^C$.

Completeness for the abstraction, i.e., $\boldsymbol{s}^A \leq \boldsymbol{\alpha}^\times(\boldsymbol{s}^C)$, together with soundness, leads to $\boldsymbol{\alpha}^\times(\boldsymbol{s}^C) = \boldsymbol{s}^A$. This is a rare but very pleasant situation in which the abstraction does not lose any information as far as the abstract properties are concerned. We remark that here the notion of "completeness" slightly deviates from the standard abstract interpretation terminology where soundness is normally indispensable, and thus complete abstractions in abstract interpretation (see, e.g., [Giacobazzi et al. 2000]) are, by default, also sound.

Moreover, completeness for the concretisation is normally of no or limited interest in abstract interpretation. In fact, alone, it states that the abstract solution is an underapproximation of the concrete one, while typically the interest is for overapproximations. Together with soundness, it leads to $\boldsymbol{s}^C = \boldsymbol{\gamma}^\times(\boldsymbol{s}^A)$, a very strong property which is not meaningful in program analysis. In our case, keeping the concepts of soundness and completeness separated and considering also completeness for the concretisation is helpful in some cases, especially when dealing with up-to functions, which are designed to provide underapproximations of fixpoints.

As in the standard abstract interpretation framework, dealing with Galois connections, we can consider the best (smallest) sound abstraction of the concrete system in the abstract domain.

Definition 4.3 (best abstraction)**.** Let $(C, \sqsubseteq)$ and $(A, \leq)$ be complete lattices, let E_C be a system of m equations over C of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$. Let $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$ be m-tuples of monotone functions, with $\langle \alpha_i, \gamma_i \rangle : C \to A$ a Galois connection for each $i \in \underline{m}$. The *best abstraction* of E_C is the system over A defined by $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^{\#}(\boldsymbol{x})$, where $\boldsymbol{f}^{\#} = \boldsymbol{\alpha}^{\times} \circ \boldsymbol{f} \circ \boldsymbol{\gamma}^{\times}$.

Standard arguments shows that $\boldsymbol{f}^{\#}$ is a sound abstraction of $\boldsymbol{f}$ over A, and it is the smallest one.

Moreover, sound abstract operators can be obtained compositionally out of basic ones.Similarly, standard arguments show that concretisations can be composed preserving soundness (in particular, observe that the composition of (co-)continuous and (co-)strict functions remain so).

Example 4.4 (abstraction for the μ-calculus)**.** The paper [Loiseaux et al. 1995] observes that (bi)simulations over transition systems can be seen as Galois connections and interpreted as abstractions. Then it characterises fragments of the μ-calculus which are preserved and strongly preserved by the abstraction. We next discuss how this can be derived as an instance of our framework.

Let $T_C = (\mathbb{S}_C, \to_C)$ and $T_A = (\mathbb{S}_A, \to_A)$ be transition systems and let $\langle \alpha, \gamma \rangle : \mathbf{2}^{\mathbb{S}_C} \to \mathbf{2}^{\mathbb{S}_A}$ be a Galois connection. It is a *simulation*, according to [Loiseaux et al. 1995], if it satisfies the following condition:

$$\alpha \circ \blacklozenge_{T_C} \circ \gamma \subseteq \blacklozenge_{T_A} \tag{6}$$

In this case T_A is called a $\langle \alpha, \gamma \rangle$*-abstraction* of T_C, written $T_C \sqsubseteq_{\langle \alpha, \gamma \rangle} T_A$. This can be shown to be equivalent to the ordinary notion of simulation between transition systems [Loiseaux et al. 1995, Propositions 9 and 10]. In particular, if $R \subseteq \mathbb{S}_C \times \mathbb{S}_A$ is a simulation in the ordinary sense then one can consider $\langle \blacklozenge_{R^{-1}}, \blacksquare_R \rangle : \mathbf{2}^{\mathbb{S}_C} \to \mathbf{2}^{\mathbb{S}_A}$, where $\blacklozenge_{R^{-1}}$ is the function $\blacklozenge_{R^{-1}}(X) = \{y \in \mathbb{S}_A \mid \exists x \in X.\, (x, y) \in R\}$. This is a Galois connection inducing a simulation in the above sense, i.e., $\blacklozenge_{R^{-1}} \circ \blacklozenge_{T_C} \circ \blacksquare_R \subseteq \blacklozenge_{T_A}$.

When $T_C \sqsubseteq_{\langle \alpha, \gamma \rangle} T_A$, by [Loiseaux et al. 1995, Theorem 2], one has that α "preserves" the $\mu\Diamond$-calculus, i.e., the fragment of the μ-calculus without $\Box$ operators. More precisely, for any formula φ of the $\mu\Diamond$-calculus, we have $\alpha(\|\varphi\|_\rho^{T_C}) \subseteq \|\varphi\|_{\alpha \circ \rho}^{T_A}$. This means that for each state $s_C \in \mathbb{S}_C$, if s_C satisfies φ in the concrete system, then all the states in $\alpha(\{s_C\})$ satisfy φ in the abstract system, provided that each proposition p is interpreted in A with the abstraction of its interpretation in C, i.e., by $\alpha(\rho(p))$.

This can be obtained as an easy consequence of our Theorem 4.2, where we use the same function α as an abstraction for all equations. Condition (6) above can be rewritten as

$$\alpha \circ \blacklozenge_{T_C} \subseteq \blacklozenge_{T_A} \circ \alpha$$

which is the soundness condition (3) in Theorem A.4 for the semantics of the diamond operator. For the other operators the soundness condition is trivially shown to hold. In fact,

- for $\mathbf{t}$ and $\mathbf{f}$ we have $\alpha(\emptyset) = \emptyset$ and $\alpha(\mathbb{S}_C) \subseteq \mathbb{S}_A$;
- for $\wedge$ and $\vee$ we have $\alpha(X \cup Y) = \alpha(X) \cup \alpha(Y)$ and $\alpha(X \cap Y) \subseteq \alpha(X) \cap \alpha(Y)$;
- a proposition p represents the constant function $\rho(p)$ in T_C and $\alpha(\rho(p))$ in T_A.

In order to extend the logic by including negation on propositions, in [Loiseaux et al. 1995], an additional condition is required, called *consistency* of the abstraction with respect to the interpretation: for all p, it has to be $\alpha(\rho(p)) \cap \alpha(\overline{\rho(p)}) = \emptyset$. This is easily seen to be equivalent to $\alpha(\overline{\rho(p)}) \subseteq \overline{\alpha(\rho(p))}$ which is exactly the soundness condition (3) for negated propositions.

Note that our framework naturally suggests generalisations of the results in [Loiseaux et al. 1995]. For instance, we could work with an abstraction function not being part of a connection, thus going beyond ordinary simulations. In fact, exploiting the dual of Theorem 4.1, one can see that continuity and strictness of α are sufficient to retain the results.

Example 4.5 (abstraction for Łukasiewicz μ-terms)**.** For Łukasiewicz μ-terms, as introduced in Example 3.5, leading to systems of fixpoint equations over the reals, we can consider as an abstraction a form of discretisation: for some fixed n define the abstract domain $[0,1]_{/n} = \{0\} \cup \{k/n \mid k \in \underline{n}\}$ and the insertion $\langle \alpha_n, \gamma_n \rangle : [0,1] \to [0,1]_{/n}$ with α_n defined by $\alpha_n(x) = \lceil n * x \rceil / n$ and γ_n the inclusion. We can consider for all operators op, their best abstraction $op^{\#} = \alpha_n \circ op \circ \boldsymbol{\gamma_n}^{\times}$, thus getting a sound abstraction (see Definition 4.3).

Note that for all semantic operators, $op^{\#}$ is the restriction of op to the abstract domain, with the exception of $r \cdot^{\#} x = \alpha_n(r \cdot x)$ for $x \in [0,1]_{/n}$. Moreover, for $x, y \in [0,1]$ we have

- $\alpha_n(\mathbf{0}(x)) = \mathbf{0}^{\#}(\alpha_n(x))$, $\alpha_n(\mathbf{1}(x)) = \mathbf{1}^{\#}(\alpha_n(x))$;
- $\alpha_n(r \cdot x) \leq r \cdot^{\#} \alpha_n(x)$;
- $\alpha_n(x \sqcup y) = \alpha_n(x) \sqcup^{\#} \alpha_n(y)$, $\alpha_n(x \sqcap y) = \alpha_n(x) \sqcap^{\#} \alpha_n(y)$;
- $\alpha_n(x \oplus y) \leq \alpha_n(x) \oplus^{\#} \alpha_n(y)$, $\alpha_n(x \odot y) \leq \alpha_n(x) \odot^{\#} \alpha_n(y)$ since $\alpha_n(x + y) \leq \alpha_n(x) + \alpha_n(y)$

i.e., the abstraction is complete for $\mathbf{0}$, $\mathbf{1}$, $\sqcup$, $\sqcap$, while it is just sound for the remaining operators.

For instance, the system in Example 3.5 can be shown to have solution $x_1 = x_2 = 0.2$. With abstraction α_{10} we get $x_1 = x_2 = 0.8$, with a more precise abstraction α_{100} we get $x_1 = x_2 = 0.22$ and with α_{1000} we get $x_1 = x_2 = 0.201$.

Example 4.6 (abstraction for Łukasiewicz μ-calculus)**.** Although space limitations prevent a detailed discussion, observe that when dealing with Łukasiewicz μ-calculus over some probabilistic transition system $N = (\mathbb{S}, \to)$, we can lift the Galois insertion above to $[0,1]^{\mathbb{S}}$. Define $\alpha_n^{\to} : [0,1]^{\mathbb{S}} \to [0,1]_{/n}^{\mathbb{S}}$ by letting, $\alpha_n^{\to}(v) = \alpha_n \circ v$ for $v \in [0,1]^{\mathbb{S}}$. Then

$\langle \alpha_n^{\rightarrow}, \gamma_n^{\rightarrow} \rangle : [0,1]^{\mathbb{S}} \to [0,1]^{\mathbb{S}}_{/n}$, where $\gamma_n^{\rightarrow}$ is the inclusion, is a Galois insertion and, as in the previous case, we can consider the best abstraction for the operators of the Łukasiewicz μ-calculus (see Definition 4.3).

For instance, consider the system for φ' in Example 3.6. Recall that the exact solution is $x_2(a) = 0.25$. With abstraction α_{10} we get $x_2(a) = 0.3$, with α_{15} we get $x_2(a) = 0.2\bar{6}$.

5 Up-to Techniques

Up-to techniques have been shown effective in easing the proof of properties of greatest fixpoints. Originally proposed for coinductive behavioural equivalences [Milner 1989; Sangiorgi and Milner 1992], they have been later studied in the setting of complete lattices [Pous 2007, 2016]. Some recent work [Bonchi et al. 2018a] started the exploration of the relation between up-to techniques and abstract interpretation. Roughly, they work in a setting where the semantic functions of interest $f^* : L \to L$ admits a left adjoint $f_* : L \to L$, the intuition being that f^* and f_* are predicate transformers mapping a condition into, respectively, its strongest postcondition and weakest precondition. Then complete abstractions for f^* and sound up-to functions for f_* are shown to coincide. This result has a natural interpretation in our game theoretic framework, that will be discussed in §6.2.

Here we take another view. We work with general semantic functions and, in §5.1, we first argue that up-to techniques can be naturally interpreted as abstractions where the concretisation is complete (and sound, if the up-to function is a closure). Then, in §5.2 we can smoothly extend up-to techniques from a single fixpoint to systems of fixpoint equations.

5.1 Up-To Techniques as Abstractions

The general idea of up-to techniques is as follows. Given a monotone function $f : L \to L$ one is interested in the greatest fixpoint νf. In general, the aim is to establish whether some given element of the lattice $l \in L$ is under the fixpoint, i.e., if $l \sqsubseteq \nu f$. In turn, since by Tarski's Theorem, $\nu f = \bigsqcup\{x \mid x \sqsubseteq f(x)\}$, this amounts to proving that l is under some post-fixpoint l', i.e., $l \sqsubseteq l' \sqsubseteq f(l')$. For instance, consider the function $bis_T : \mathrm{Rel}(\mathbb{S}) \to \mathrm{Rel}(\mathbb{S})$ for bisimilarity on a transition system T in Example 3.7. Given two states $s_1, s_2 \in \mathbb{S}$, proving $\{(s_1, s_2)\} \subseteq \nu bis_T$, i.e., showing the two states bisimilar, amounts to finding a post-fixpoint, i.e., a relation R such that $R \subseteq bis_T(R)$ (namely, a bisimulation) such that $\{(s_1, s_2)\} \subseteq R$.

Definition 5.1 (up-to function)**.** Let L be a complete lattice and let $f : L \to L$ be a monotone function. A *sound up-to function* for f is any monotone function $u : L \to L$ such that $\nu(f \circ u) \sqsubseteq \nu f$. It is called *complete* if also the converse inequality $\nu f \sqsubseteq \nu(f \circ u)$ holds.

When u is sound, if l is a post-fixpoint of $f \circ u$, i.e., $l \sqsubseteq f(u(l))$ we have $l \sqsubseteq \nu(f \circ u) \sqsubseteq \nu f$. The idea is that the characteristics of u should make it easier to prove that l is a postfix-point of $f \circ u$ than proving that it is one for f. This is clearly the case, for instance, when u is extensive. In fact by extensiveness of u and monotonicity of f we get $f(l) \sqsubseteq f(u(l))$, in a way that obtaining $l \sqsubseteq f(u(l))$ is "easier" than obtaining $l \sqsubseteq f(l)$. Observe that extensiveness also implies "completeness" of the up-to function: since $f \sqsubseteq f \circ u$ clearly $\nu f \sqsubseteq \nu(f \circ u)$. We remark that for up-to functions, since the interest is for underapproximating fixpoints, the terms soundness and completeness are somehow reversed with respect to their meaning in abstract interpretation.

A sufficient condition, commonly used for ensuring soundness of up-to functions is compatibility [Pous 2007].

Definition 5.2 (compatibility)**.** Let L be a complete lattice and let $f : L \to L$ be a monotone function. A monotone function $u : L \to L$ is *f-compatible* if

$$u \circ f \sqsubseteq f \circ u \tag{7}$$

When u is f-compatible, and, as it commonly happens in most applications, it is a closure (i.e., extensive and idempotent) there is a natural interpretation of the up-to technique in terms of abstractions. In fact, since u is a closure, $u(L)$ is a complete lattice that can be seen as an abstract domain in a way that $\langle u, i \rangle : L \to u(L)$, with i being the inclusion, is a Galois insertion. Moreover $f_{|u(L)}$ can be easily shown to provide an abstraction of both f and $f \circ u$ over L, sound and complete with respect to the inclusion i, seen as the concretisation. To the best of our knowledge, this view of up-to techniques as special abstractions is an original observation.

The formal details are given in the result below. Since we later aim to apply up-to techniques in the setting of systems of equations, we deal with not only greatest but also least fixpoints.

Lemma 5.3 (compatible up-to functions as sound and complete abstractions)**.** *Let $f : L \to L$ be a monotone function and let $u : L \to L$ be an f-compatible closure. Consider the Galois insertion $\langle u, i \rangle : L \to u(L)$ where $i : u(L) \to L$ is the inclusion. Then*

a) f restricts to $u(L)$, i.e., $f_{|u(L)} : u(L) \to u(L)$;

b) $f_{|u(L)}$ is a sound and complete abstraction of both f and $f \circ u$. Therefore $\nu f = i(\nu f_{|u(L)}) = \nu(f \circ u)$. Moreover, if u is continuous and strict then $\mu f = i(\mu f_{|u(L)}) = \mu(f \circ u)$.

$$\begin{tikzcd} L \arrow[loop above, "f"] \arrow[loop left, "f \circ u"] \arrow[r, shift right, "u"'] & u(L) \arrow[l, shift right, "i"'] \arrow[loop right, "f_{|u(L)}"] \end{tikzcd}$$

Whenever the up-to function is just f-compatible (hence sound), but possibly not a closure, we can canonically turn it into an f-compatible closure (hence sound and complete), by taking the least closure above u. This has been considered already in [Cousot and Cousot 1979b], with a slightly different construction.

Definition 5.4 (least upper closure)**.** Let L be a complete lattice and let $u : L \to L$ be a monotone function. We let $\bar{u} : L \to L$ be the function defined by $\bar{u}(x) = \mu(\hat{u}_x)$ where $\hat{u}_x(y) = u(y) \sqcup x$.

Lemma 5.5 (properties of $\bar{u}$)**.** *Let $u : L \to L$ be a monotone function. Then*

a) $\bar{u}$ is the least closure larger than u;
b) if u is f-compatible then $\bar{u}$ is;
c) if u is continuous and strict then $\bar{u}$ is.

Using Lemmas 5.3 and 5.5, whenever u is a compatible up-to function for f, we have that $\bar{u}$ is a sound and complete up-to function for f. The soundness of u immediately follows.

Corollary 5.6 (soundness of compatible up-to functions)**.** *Let $f : L \to L$ be a monotone function and let $u : L \to L$ be an f-compatible up-to function. Then $\nu(f \circ u) \sqsubseteq \nu(f \circ \bar{u}) = \nu f$. If u is continuous and strict, then $\mu(f \circ u) \sqsubseteq \mu(f \circ \bar{u}) = \mu f$.*

In [Pous 2007] the proof of soundness of a compatible up-to technique u relies on the definition of a function u^ω defined as $u^\omega(x) = \bigsqcup\{u^n(x) \mid n \in \mathbb{N}\}$, where $u^n(x)$ is defined inductively as $u^0(x) = x$ and $u^{n+1}(x) = u(u^n(x))$. The function u^ω is extensive but not idempotent in general, and it can be easily seen that $u^\omega \sqsubseteq \bar{u}$. The paper [Pous 2016] shows that for any monotone function one can consider the largest compatible up-to function, the so-called companion, which is extensive and idempotent. The companion could be used in place of $\bar{u}$ for part of the theory. However, we find it convenient to work with $\bar{u}$ since, despite not discussed in the present paper, it plays a key role for the integration of up-to techniques into the verification algorithms. Furthermore the companion is usually hard to determine.

5.2 Up-to techniques for systems of equations

Exploiting the fact that up-to functions can be viewed as abstractions, moving to systems of equations is almost immediate. As in the case of abstractions, we allow to have a different up-to function for each equation of the system.

Definition 5.7 (compatible up-to for systems of equations)**.** Let $(L, \sqsubseteq)$ be a complete lattice and let E be $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$, a system of m equations over L. A *compatible tuple of up-to functions* for E is an m-tuple of monotone functions $\boldsymbol{u}$, with $u_i : L \to L$, satisfying compatibility:

$$\boldsymbol{u}^\times \circ \boldsymbol{f} \sqsubseteq \boldsymbol{f} \circ \boldsymbol{u}^\times \tag{8}$$

with u_i continuous and strict for each $i \in \underline{m}$ such that $\eta_i = \mu$.

We can then generalise Corollary 5.6 to systems of equations. Below, given an m-tuple $\boldsymbol{u}$ of functions $u_i : L \to L$ for $i \in \underline{m}$, we write $\bar{\boldsymbol{u}}$ for the m-tuple $(\bar{u}_1, \ldots, \bar{u}_m)$.

Theorem 5.8 (up-to for systems)**.** *Let $(L, \sqsubseteq)$ be a complete lattice and let E be $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$, a system of m equations over L, with solution $\boldsymbol{s} \in L^m$. Let $\boldsymbol{u}$ be a compatible tuple of up-to functions for E. Let $\boldsymbol{s}'$ and $\bar{\boldsymbol{s}}$ be the solutions of the systems $E\boldsymbol{u}$ with equations $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{u}^\times(\boldsymbol{x}))$ and $E\bar{\boldsymbol{u}}$ with equations $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\bar{\boldsymbol{u}}^\times(\boldsymbol{x}))$, respectively. Then $\boldsymbol{s}' \sqsubseteq \bar{\boldsymbol{s}} = \boldsymbol{s}$. If additionally $\boldsymbol{u}$ is extensive then $\boldsymbol{s}' = \boldsymbol{s}$.*

We will now state a corollary of this theorem that paves the way to using up-to techniques in algorithms (cf. §6.2).

Corollary 5.9. *Let E be $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$, a system of m equations over a complete lattice L and let $\boldsymbol{u}$ be a compatible tuple of extensive up-to functions for E.*

For all $b \in B_L$, if $b \sqsubseteq u_i(\bigsqcup B')$ for $B' \subseteq B_L$ where all $b' \in B'$ satisfy $b' \sqsubseteq sol_i(E)$ then $b \sqsubseteq sol_i(E)$.

Example 5.10 (μ-calculus up-to (bi)similarity)**.** Consider the problem of model-checking the μ-calculus over some transition system with atoms $T = (\mathbb{S}, \to, A)$.

Assuming that we have some a priori knowledge about the similarity relation $\precsim$ over (some of) the states in T, then, restricting to a suitable fragment of the μ-calculus we can avoid checking the same formula on similar states. This intuition can be captured in the form of an up-to technique, that we refer to as up-to similarity, based on an up-to function $\boldsymbol{u}_\precsim : \mathbf{2}^{\mathbb{S}} \to \mathbf{2}^{\mathbb{S}}$ defined, for $X \in \mathbf{2}^{\mathbb{S}}$, by

$$u_\precsim(X) = \{s \in \mathbb{S} \mid \exists s' \in X.\ s' \precsim s\}.$$

It can be easily seen that $u_\precsim$ is monotone, extensive, and idempotent. It is also clearly continuous and strict.

We observe that $u_\precsim$ is a compatible (and thus sound) up-to function for the $\mu\Diamond$-calculus where propositional variables are interpreted as atoms. In fact, $\precsim$ is a simulation (the largest one) and the function $u_\precsim$ is the associated abstraction as defined in Example 4.4, namely $u_\precsim = \blacklozenge_\succsim$. Therefore, compatibility $u_\precsim \circ f \sqsubseteq f \circ u_\precsim$ corresponds to condition (6) in Example 4.4 which has been already observed to coincide with soundness in the sense of Theorem 4.2 for the operators of the $\mu\Diamond$-calculus. Concerning propositional variables, in Example 4.4, they were interpreted, in the target transition system, by the abstraction of the interpretation in the source transition system. Since here we have a single transition system and a single interpretation $\rho : \mathit{Prop} \to \mathbf{2}^{\mathbb{S}}$, we must have $\rho(p) = u_\precsim(\rho(p))$, i.e., $\rho(p)$ upward-closed with respect to $\precsim$. This automatically holds by the fact that $\precsim$ is a simulation.

Similarly, we can define up-to bisimilarity. In this case the up-to function is:

$$u_\sim(X) = \{s \in \mathbb{S} \mid \exists s' \in X.\ s \sim s'\}.$$

Reasoning as above, one can see that compatibility $u_\sim \circ f \sqsubseteq f \circ u_\sim$ holds for the full μ-calculus with propositional variables interpreted as atoms. For instance, consider the formula φ in Example 3.4 and the transition system in Fig. 1a. Using the up-to function $u_\sim$ corresponds to working in the bisimilarity quotient Fig. 1b. Note, however, that when using a local algorithm (see §6.2) the quotient does not need to be actually computed. Rather, only the bisimilarity over the states explored by the searching procedure is possibly exploited.

Example 5.11 (bisimilarity up-to transitivity)**.** Consider the problem of checking bisimilarity on a transition system $T = \langle \mathbb{S}, \rightarrow \rangle$. A number of well-known sound up-to techniqueshave been introduced in the literature [Pous and Sangiorgi 2011]. As a simple example, we consider the up-to function $u_{tr} : \mathsf{Rel}(\mathbb{S}) \rightarrow \mathsf{Rel}(\mathbb{S})$ performing a single step of transitive closure. It is defined as:

$$u_{tr}(R) = R \circ R = \{(x, y) \mid \exists z \in \mathbb{S}.\ (x, z) \in R \ \wedge\ (z, y) \in R\}.$$

It is easy to see that u_{tr} is monotone and compatible with respect to the function $bis_T : \mathsf{Rel}(\mathbb{S}) \rightarrow \mathsf{Rel}(\mathbb{S})$ of which bisimilarity is the greatest fixpoint (see Example 3.7).

Note that u_{tr} is neither idempotent nor extensive. The corresponding closure $\bar{u}_{tr}$ is the function mapping a relation to its (full) transitive closure (which is known to be itself a sound up-to technique, a fact that we can also derive from the compatibility of u_{tr} and Corollary 5.6).

6 Solving systems of equations via games

In this section, we first provide a characterisation of the solution of a system of fixpoint equations over a complete lattice in terms of a parity game. This is a variation of a result in [Baldan et al. 2019]. While the original result was limited to continuous lattices, here, exploiting the results on abstraction in §4, we devise a game working for any complete lattice.

The game characterisation opens the way to the development of algorithms for solving the game and thus the associated verification problem. A proper treatment of these aspects is beyond the scope of the present paper. Here, in §6.2, we hint at the algorithmic potentials focusing on the case of a single equation.

6.1 Game characterization

We show that the solution of a system of equations over a complete lattice can be characterised using a parity game.

Definition 6.1 (fixpoint game)**.** Let L be a complete lattice and let B_L be 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 *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 $\mathbf{E}(b, i) = \{\boldsymbol{X} \mid \boldsymbol{X} \in (2^{B_L})^m \ \wedge\ b \sqsubseteq f_i(\bigsqcup \boldsymbol{X})\}$.
- From position $\boldsymbol{X} \in (2^{B_L})^m$ the possible moves of $\forall$ are $\mathbf{A}(\boldsymbol{X}) = \{(b, i) \mid i \in \underline{m} \ \wedge\ b \in X_i\}$.

The game is schematised in Table 1. 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.

Position	Player	Moves
(b, i)	$\exists$	$\boldsymbol{X}$ s.t. $b \sqsubseteq f_i(\bigsqcup \boldsymbol{X})$
$\boldsymbol{X}$	$\forall$	(b', j) s.t. $b' \in X_j$

Table 1. The fixpoint game on the powerset of the basis

If we instantiate the game to the setting of standard μ-calculus model-checking, we obtain an alternative encoding of μ-calculus into parity games, typically resulting in more compact games.

Example 6.2. We provide a simple example illustrating the game. Consider the infinite lattice $L = \mathbb{N} \cup \{\omega, \omega+1\}$ (where $n \leq \omega \leq \omega + 1$ for every $n \in \mathbb{N}$) with basis $B_L = L$. Furthermore let $f : L \rightarrow L$ be a monotone function with $f(n) = n+1$ for $n \in \mathbb{N}$ and $f(\omega) = \omega$, $f(\omega + 1) = \omega + 1$. Hence $\mu f = \omega$.

We set $b = \omega$ and attempt to show via the game that $b \leq \mu f$, by exhibiting a winning strategy for $\exists$. Note that since we are dealing with a μ-equation, in order to win $\exists$ must ensure that $\forall$ eventually has no moves left. Since there is only one fixpoint equation, we omit the indices. Starting with $b = \omega$, $\exists$ plays $X = \mathbb{N}$, which is a valid move since $\omega \leq f(\bigsqcup X) = f(\omega)$. Now $\forall$ has to pick some $n \in X$. In the next move, $\exists$ can play $X = \{n - 1\}$, which means that $\forall$ picks $n - 1$. Hence we obtain a descending chain, leading to 1, which can be covered by $\exists$ by choosing $X = \emptyset$, since $1 \leq f(\bigsqcup \emptyset) = f(0)$. Now $\forall$ has no moves left and $\exists$ wins.

Instead for $b = \omega + 1 \not\leq \mu f$, $\exists$ has no winning strategy since she has to play a set X that contains $\omega + 1$. Then player $\forall$ can reply by choosing $\omega + 1$ and the game will continue forever. This is won by $\forall$ since we are dealing with a μ-equation.

Interestingly, the correctness and completeness of the game can be proved by exploiting the results in §4. The crucial observation is that there is a Galois insertion between L and the powerset lattice of its basis (which is algebraic hence continuous) $\langle \alpha, \gamma \rangle : 2^{B_L} \rightarrow L$ where abstraction α is the join $\alpha(X) = \bigsqcup X$ and concretisation γ takes the lower cone $\gamma(l) = \downarrow l \cap B_L$. Then a system of equations over a complete lattice L can be "transferred" to a system of equations over the powerset of the basis 2^{B_L} along such insertion, in a way that the system in L can be seen as a sound and complete abstraction of the one in 2^{B_L}.

Theorem 6.3 (correctness and completeness)**.** *Let E be a system of m 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 $i \in \underline{m}$,*

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

6.2 An Algorithmic View

The game theoretical characterisation can be the basis for the development of algorithms, possibly integrating abstraction and up-to techniques, for solving the game, i.e., for determining winning and losing positions for the players, which in turn corresponds to solving the associated verification problem. Here we consider on-the-fly algorithms for the case of a single equation, establishing a link with some recent work relating abstract interpretation and up-to techniques [Bonchi et al. 2018a] and exploiting up-to techniques for computing language equivalence on NFAs [Bonchi and Pous 2013].

An algorithm for the unrestricted case based on [Stevens and Stirling 1998] can also be given, but is considerably more difficult. Hence we postpone it to § 7 and we will first focus on the special case of a single (greatest) fixpoint equation $x =_\nu f(x)$.

6.2.1 Selections

For a practical use of the game it can be useful to observe that the set of moves of the existential player can be suitably restricted without affecting the completeness of the game, by introducing a notion of selection, similarly to what is done in [Baldan et al. 2019].

Given a lattice L define a preorder $\sqsubseteq_H$ on 2^{B_L} by letting, for $X, Y \in 2^{B_L}$, $X \sqsubseteq_H Y$ if $\bigsqcup X \sqsubseteq \bigsqcup Y$. (The subscript H comes from the fact that for completely distributive lattices, if B_L is the set of irreducible elements, $\sqsubseteq_H$ can be seen to coincide with the "Hoare preorder" [Abramsky and Jung 1994], requiring that for all $x \in X$ there exists $y \in Y$ such that $x \sqsubseteq y$). Observe that $\sqsubseteq_H$ is not antisymmetric. We write $\equiv_H$ for the corresponding equivalence, i.e., $X \equiv_H Y$ when $X \sqsubseteq_H Y \sqsubseteq_H X$. When $X \subseteq L$ is finite, the subset $X' \subseteq X$ of its maximal elements is clearly equivalent to X, i.e., $X' \equiv_H X$.

The moves of player $\exists$ can be ordered by the pointwise extension of $\sqsubseteq_H$, thus leading to the following definition. Since we deal with a single equation, we will omit the indices from the positions of player $\exists$ and write b instead of $(b, 1)$.

Definition 6.4 (selection)**.** Given an equation $x =_\nu f(x)$ over a complete lattice L, a *selection* is a function $\sigma : B_L \to 2^{B_L}$ such that for all positions $b \in B_L$ it holds that $\uparrow_H \sigma(b) = \mathbf{E}(b)$, where $\uparrow_H$ is the upward-closure with respect to $\sqsubseteq_H$.

For the case of a single fixpoint equation it is easy to see that Theorem 6.3 continues to hold if we restrict the moves of player $\exists$ to those prescribed by a selection, i.e., the restriction of the moves of $\exists$ does not reduce her power.

Theorem 6.5 (game with selections)**.** *Let $x =_\nu f(x)$ be an equation over a complete lattice L with solution s. For all $b \in B_L$, it holds that $b \sqsubseteq s$ iff $\exists$ has a winning strategy from position b in the game restricted to selections.*

In practice, it can be convenient to consider selections that are minimal with respect to some criterion.

6.2.2 On-the-fly algorithm for a special case

We will now assume that $f : L \to L$ preserves non-empty meets, i.e., for $X \neq \emptyset$, $f(\bigsqcap X) = \bigsqcap f(X)$. Observe that this is equivalent to asking $f(x) = f^*(x) \sqcap c$ for some $c \in L$ (just take $c = f(\top)$), with f^* being a right adjoint of a map f_*, a setting that has been studied also in [Bonchi et al. 2018a]. Note that the adjunction $\langle f_*, f^* \rangle$ is completely orthogonal to the adjunctions studied so far.

In this case, there is a selection function defined, for all $b \in B_L$, by

$$\sigma(b) = \begin{cases} \{X\} \text{ with } X \equiv_H \downarrow f_*(b) \cap B_L & \text{when } b \sqsubseteq c \\ \emptyset & \text{otherwise} \end{cases}$$

In order to see that this is a selection, note that, if $b \sqsubseteq c$ then given $X \subseteq B_L$ it holds that $X \in \mathbf{E}(b)$, i.e., $b \sqsubseteq f(\bigsqcup X) = f^*(\bigsqcup X) \sqcap c$, iff $b \sqsubseteq f^*(\bigsqcup X)$ iff $f_*(b) \sqsubseteq \bigsqcup X$, where the last step is motivated by adjointness. This means that either the existential player is stuck or she has a best move. As a consequence, the game introduced in §6.1 can be greatly simplified. We distinguish two cases, depending on whether the basis B_L contains all (non bottom) elements or is generic.

Case 1: The basis contains all elements. Assume that $B_L = L \setminus \{\bot\}$. Then we can restrict to a completely deterministic game for both the existential and universal player.

Let $b \in B_L$. The game for checking $b \sqsubseteq \nu f$, can be described as follows. Start from $b_0 = b$ and at each position b_i of the existential player:

a) if $b_i \not\sqsubseteq c$ then $\sigma(b_i) = \emptyset$ and $\exists$ loses;
b) otherwise, $\exists$ has to play $\sigma(b_i) = \{X_i\}$ with $X_i \equiv_H \downarrow f_*(b_i) \cap B_L$,
   i) if $f_*(b_i) = \bot$ then take $X_i = \emptyset$; hence $\exists$ wins since $\forall$ has no moves;
   ii) if instead, $f_*(b_i) \neq \bot$ we can take $X_i = \{f_*(b_i)\} \equiv_H \downarrow f_*(b_i) \cap B_L$ and thus player $\forall$ can only play $b_{i+1} = f_*(b_i)$ and the game continues.

If the game continues indefinitely, player $\exists$ wins.

The game can be further simplified by observing that the following winning condition for player $\exists$ is sound:

whenever, for some i, it holds $b_i \sqsubseteq \bigsqcup_{j<i} b_j$, then $\exists$ wins.

In fact, if $b_i = \bot$ then player $\exists$ indeed wins. Otherwise, if $b_i \neq \bot$, first note that $b_i \sqsubseteq c$. In fact, since the game did not finish yet, we have $b_j \sqsubseteq c$ for all $j < i$ and thus $b_i \sqsubseteq \bigsqcup_{j<i} b_j \sqsubseteq c$. Moreover, at the next step, we will have

$$b_{i+1} = f_*(b_i) \sqsubseteq f_*(\bigsqcup_{j<i} b_j) = \bigsqcup_{j<i} f_*(b_j) = \bigsqcup_{j<i} b_{j+1} \sqsubseteq c.$$

An inductive argument thus shows that by iterating f_* we will never go beyond c, hence player $\exists$ cannot lose.

Hence, the game in this special case can be formulated as in Fig. 3. Its correctness follows directly from the arguments

Checking $b \sqsubseteq \nu f = \nu(f^* \sqcap c)$ [basis $B_L = L \setminus \{\bot\}$]

Start with $b_0 = b$ and

- if $b_i \not\sqsubseteq c$ then $\exists$ loses
- else, if $b_i \sqsubseteq \bigsqcup_{j<i} b_j$ then $\exists$ wins
- else continue with $b_{i+1} = f_*(b_i)$

If the game continues forever, $\exists$ wins.

Figure 3. The simplified game for a single equation with adjoint semantic function, when $B_L = L \setminus \{\bot\}$.

above and the fact that a play in the simplified game corresponds exactly to the one in the fixpoint game where moves of player $\exists$ are restricted using the selection obtained via f_*.

Theorem 6.6 (Case 1: correctness and completeness)**.** *Let f be a monotone function over a complete lattice L, with basis $B_L = L \setminus \{\bot\}$, such that f preserves non-empty meets. Given the Galois connection $\langle f_*, f^* \rangle : L \to L$ such that $f(x) = f^*(x) \sqcap f(\top)$, then, for all $b \in B_L$, $b \sqsubseteq \nu f$ iff $\exists$ wins the game of Case 1 from position b.*

Observe that, since f_* is a left adjoint and thus continuous, we have that $\bigsqcup_{i \in \mathbb{N}} b_i = \bigsqcup_i f_*^i(b)$ is the least fixpoint of f_* above b, which in turn coincides with the least fixpoint of $f^* \sqcup b$. This establishes a direct link with [Bonchi et al. 2018a] which shows that for $b \in L$ it holds that $\mu(f^* \sqcup b) \sqsubseteq c$ iff $b \sqsubseteq \nu(f^* \sqcap c) = \nu f$.

Furthermore, we can bring up-to techniques into the picture: given an up-to function $u$we can modify the algorithm in Fig. 3 by replacing the winning condition for $\exists$, that is, $b_i \sqsubseteq \bigsqcup_{j<i} b_j$, by $b_i \sqsubseteq u(\bigsqcup_{j<i} b_j)$. The algorithm remains clearly complete and it is also correct due to Corollary 5.9.

Case 2: General basis. Assume now that we take any basis B_L such that $\bot \notin B_L$. Considerations similar to Case 1 apply, but now the move of player $\exists$ prescribed by the selection $\sigma(b) = \{X\}$ with $X \equiv_H \downarrow f_*(b) \cap B_L$ is typically not a singleton, which means that moves of player $\forall$ are no longer deterministic and the game has a tree structure. The corresponding algorithmic view is outlined in Fig. 4, where W represents the set of positions assumed winning for $\exists$. As in the previous case, a canonical choice for X, including only maximal elements will be possible (at least in finite lattices). Player $\exists$ loses at a position b' if $b' \not\sqsubseteq c$, on the other hand, a position b' is considered winning for $\exists$ if it is dominated by the join of other visited positions in the tree.

Theorem 6.7 (Case 2: correctness and completeness)**.** *Let f be a monotone function over a complete lattice L, with basis $B_L \subseteq L \setminus \{\bot\}$, such that f preserves non-empty meets. Given the Galois connection $\langle f_*, f^* \rangle : L \to L$ such that $f(x) = f^*(x) \sqcap f(\top)$, then, for all $b \in B_L$, $b \sqsubseteq \nu f$ iff $\exists$ wins the game of Case 2 from position b.*

Checking $b \sqsubseteq \nu f = \nu(f^* \sqcap c)$ [generic $B_L \subseteq L \setminus \{\bot\}$]

Set $W = \emptyset$ and explore b.

Explore b':

- if $b' \not\sqsubseteq c$ then $\exists$ loses
- else, if $b' \sqsubseteq \bigsqcup W$ then stop exploring this branch
- else set $W \leftarrow W \cup \{b'\}$, choose $X \subseteq B_L$ such that $X \equiv_H \downarrow f_*(b') \cap B_L$ and explore each $b'' \in X$.

If no losing position for $\exists$ is encountered during the exploration, then $\exists$ wins.

Figure 4. The simplified game for a single equation with adjoint semantic function, with a general basis.

Up-to techniques can be integrated as in Case 1, exploiting Corollary 5.9, by replacing the stop condition $b' \sqsubseteq \bigsqcup W$ by $b' \sqsubseteq u(\bigsqcup W)$ (for an up-to function u). This allows us to cover the algorithm in [Bonchi and Pous 2013] which checks language equivalence for non-deterministic automata. It performs on-the-fly determinization and constructs a bisimulation up-to congruence on the determinized automaton. A more detailed comparison can be found in Appendix E.

7 On-the-fly algorithm for solving the game in the general case

We now describe the on-the-fly algorithm for the general case, which gives us a local algorithm for determining whether a lattice element is below the solution, extending the specialized algorithm in the previous section. For instance, in the case of the μ-calculus, rather than computing the set of states enjoying some formula φ, one could be interested in checking whether a specific state enjoys or not φ. For probabilistic logics, rather than determining the full evaluation of φ, we could be interested in determining the value for a specific state or only in establishing a bound for such a value. Similarly, in the case of behavioural equivalences, rather than computing the full behavioural relation, one could be interested in determining whether two specific states are equivalent.

In this section we devise an on-the-fly algorithm for solving these kind of local problems. 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 for μ-calculus model checking [Stevens and Stirling 1998]. In particular, our algorithm arises as a natural generalisation of the one in [Stevens and Stirling 1998] to the setting of fixpoint games (see Definition 6.1).

We fix some notation and conventions which will be useful for describing the algorithm.

Notation For the rest of the section, L denotes a complete 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 usually 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 usually denoted by the upper case letter C and we write $\mathsf{P}(C)$ for the player controlling the position C.

Given a position $C \in Pos$, the possible moves for player $\mathsf{P}(C)$ are indicated by $\mathsf{M}(C) \subseteq Pos$. In particular, if $C \in Pos_\exists$ then $\mathsf{M}(C) \subseteq Pos_\forall$, otherwise $\mathsf{M}(C) \subseteq Pos_\exists$. A function $\mathsf{i} : Pos \to (\underline{m} \cup \{0\})$ maps every position to a *priority*, which, for positions (b, i) of player $\exists$ is the index i, while it is 0 for positions of $\forall$. With this notation, the winning condition can be expressed as follows:

- Every finite play is won by the player who moved last.
- Every infinite play, seen as a sequence of positions $(C_1, C_2, \ldots)$, is won by player $\exists$ (resp. $\forall$) if there exists a priority $h \in \underline{m}$ s.t. $\eta_h = \nu$ (resp. μ), the set $\{j \mid \mathsf{i}(C_j) = h\}$ is infinite and the set $\{j \mid \mathsf{i}(C_j) > h\}$ is finite.

7.1 The algorithm

Given an element of the basis $b \in B_L$ and some 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 6.3, this corresponds to establish which of the players has a winning strategy in the fixpoint game starting from the position (b, i). The procedure roughly consists in a depth-first exploration of the tree of plays arising as unfolding of the game graph starting from the initial position (b, i). The algorithm optimises the search by making assumptions on particular subtrees, which are thus pruned. Assumptions can be later confirmed or invalidated, and thus withdrawn. The algorithm is split into three different functions (see Fig. 5).

- Function Explore explores the tree of plays of the game, trying different moves from each node in order to determine the player who has a winning strategy from such node.
- Function Backtrack allows to backtrack from a node after the algorithm has established who was the winner from it, transmitting the information backwards.
- Sometimes the algorithm makes erroneous assumptions when pruning the search in some position, this leads it to incorrectly designate a player as the winner from that position. However, the algorithm is able to detect this fact and correct its decisions. The correction is performed by the function Forget.

The algorithm uses the following data structures:

- The *counter* $\boldsymbol{k}$, 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 an higher priority was last encountered (the current positions is not included). After any move, the counter is updated taking into account the priority of the current position. More precisely, the update of a counter $\boldsymbol{k}$ when moving from a position with priority i, denoted $next(\boldsymbol{k}, i)$, is defined as follows: $next(\boldsymbol{k}, i)_j = 0$ for all $j < i$, $next(\boldsymbol{k}, i)_i = k_i + 1$, and $next(\boldsymbol{k}, i)_j = k_j$ for all $j > i$. Note that, in particular, $next(\boldsymbol{k}, 0) = \boldsymbol{k}$, i.e., moves from a position with priority 0, which are the moves of $\forall$, do not change $\boldsymbol{k}$. We also define two total orders $<_\exists$ and $<_\forall$ on counters, 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 s.t. $k_i \neq k'_i$ is the index of a greatest fixpoint equation and $k_i < k'_i$, or it is the index of a least fixpoint and $k_i > k'_i$. The other order $<_\forall$ is the reverse of $<_\exists$, that is $\boldsymbol{k} <_\forall \boldsymbol{k}'$ iff $\boldsymbol{k}' <_\exists \boldsymbol{k}$. For each player P, we write $\boldsymbol{k} \leq_P \boldsymbol{k}'$ for $\boldsymbol{k} <_P \boldsymbol{k}'$ or $\boldsymbol{k} = \boldsymbol{k}'$.
- The *playlist* ρ, i.e., a list of the 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 (exploration is performed depth-first). Thus, ρ is a list of triples $(C, \boldsymbol{k}, \pi)$, where C is a position, $\boldsymbol{k}$ is a counter and $\pi \subseteq Pos$ is the set of the unexplored moves from that position.
- The *assumptions* for players $\exists$ 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 it is encountered for the second time in the current playlist ρ. This reveals the presence of a loop in the game graph which 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 fact, 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., Γ_P contains pairs of the kind $(C, \boldsymbol{k})$. Since other possible paths branching from the loop 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., Δ_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.

For checking whether $b \sqsubseteq s_i$ for $b \in B_L$ and $i \in \underline{m}$, we call the function $\textsc{Explore}((b,i), \mathbf{0}, [], (\emptyset,\emptyset), (\emptyset,\emptyset))$, 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 6.3, $P = \exists$ if and only if $b \sqsubseteq s_i$.

Given the current position C, the corresponding counter $\boldsymbol{k}$, the playlist ρ describing the path that led to C, and the sets of assumptions Γ and decisions Δ, function $\textsc{Explore}(C, \boldsymbol{k}, \rho, \Gamma, \Delta)$ checks if one of the following three conditions holds, each one corresponding to a different **if** branch.

- If $\mathsf{M}(C) = \emptyset$, then the controller $\mathsf{P}(C)$ of position C cannot move and its opponent $\overline{\mathsf{P}(C)}$ wins. Therefore, a new decision for the current position is added for the opponent, and we backtrack. A decision of this kind, with empty justification is called a *truth*.
- If there is already a decision for a player P for the current position C, that is, $(C, \boldsymbol{k}') \in \Delta_P$ and $\boldsymbol{k}' \leq_P \boldsymbol{k}$, then 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 lead to the previous decision $(C, \boldsymbol{k}')$.
- If the current position C was already encountered in the play, i.e., $(C, \boldsymbol{k}', \pi) \in \rho$ for some $\boldsymbol{k}'$ and π, then C becomes an assumption for the 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, the exploration continues from C. A move $C' \in \mathsf{M}(C)$ is chosen to be explored. The playlist is thus extended by adding $(C, \boldsymbol{k}, \pi)$ where π records the remaining moves to be explored. The counter $\boldsymbol{k}$ is updated according to the priority of the now past position C.

Function $\textsc{Backtrack}(P, C, \rho, \Gamma, \Delta)$ is used to backtrack from a position C, reached via the playlist ρ, after assuming or deciding that player P would win from such position.

- If $\rho = []$ we are back at the root, the position from where the computation started, and the exploration is concluded. The algorithm decides that player P is the winner from such a position.
- Otherwise, the head $(C', \boldsymbol{k}, \pi)$ of the playlist ρ is popped and the status of position C' is investigated.
  - If C' is controlled by the opponent of P ($\mathsf{P}(C') \neq P$) and there are still unexplored moves ($\pi \neq \emptyset$), we must explore such moves before deciding the winner from C'. Then, a new move is extracted from π and explored.
  - If instead the controller of C' is P ($\mathsf{P}(C') = P$) then P wins also from C'. Hence C' is inserted in Δ_P, justified by the move C from where we backtracked. Similarly, if the controller of C' is the opponent of P ($\mathsf{P}(C') \neq P$), we already explored all possible moves from C' ($\pi = \emptyset$) and all turn out to be winning for P, again we decide that P wins from C', which is inserted in Δ_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 $\textsc{Forget}(\Delta_{\overline{P}}, \Gamma_{\overline{P}}, (C', \boldsymbol{k}'))$.

In general the choice of moves to explore, performed by the action "pick" in the pseudocode, is random. However, we observed in §6.1, that for player $\exists$ it can be shown that it is sufficient to explore the minimal moves. Furthermore, it is usually convenient to give priority to moves which are immediately reducible to valid decisions or assumptions for the player who is moving. A practical way to do this is to check if there is a decision for a position C', with a valid counter wrt. the current one, such that either the current position $C = (b, i), C' = (b', i)$ and $b \sqsubseteq b'$, or $C = \boldsymbol{X}, C' = \boldsymbol{X}'$ and $\boldsymbol{X}' \subseteq \boldsymbol{X}$. Then, the move to pick is the one justifying such decision, which by those features is guaranteed to be a move also from the current position C.

The function $\textsc{Forget}$ is not given explicitly. The precise definition of the property that function $\textsc{Forget}$ must satisfy in order to ensure the correctness of the algorithm is quite technical (it can be found in the appendix provided as extra material). Intuitively, when an assumption in Γ_P fails and is withdrawn, then it must remove from Δ_P at least all the decisions depending on such assumption. It is possible that decisions taken on the base of the deleted assumption remain valid because they can be justified by other decisions or assumptions, possibly introduced later. Different sound realisations of $\textsc{Forget}$ are then possible (see [Stevens and Stirling 1998]) and, experimentally, it can been seen that those removing only the least possible set of decisions can be practically inefficient. A simple sound implementation, which, at least in the setting of the μ-calculus, resulted to be the most efficient is based on a temporal criterion: when an assumption fails, all decisions which have been taken after that assumption are deleted. This can be implemented by associating timestamps with decisions and assumptions, and avoiding the complex management of justifications.

Example 7.1 (model-checking μ-calculus)**.** Consider the transition system $T = (\mathbb{S}, \to)$ in Fig. 1a and the μ-calculus

function Explore(C, $\boldsymbol{k}$, ρ, Γ, Δ)
  if $\mathsf{M}(C) = \emptyset$ **then**
    $\Delta_{\overline{\mathsf{P}(C)}} := \Delta_{\overline{\mathsf{P}(C)}} \cup \{(C, \boldsymbol{k})\}$;
    Backtrack($\overline{\mathsf{P}(C)}$, C, ρ, Γ, Δ);
  else if there is $(C, \boldsymbol{k}') \in \Delta_P$ s.t. $\boldsymbol{k}' \leq_P \boldsymbol{k}$ **then**
    Backtrack(P, C, ρ, Γ, Δ);
  else if there is $(C, \boldsymbol{k}', \pi) \in \rho$ **then**
    let P s.t. $\boldsymbol{k}' <_P \boldsymbol{k}$;
    $\Gamma_P := \Gamma_P \cup \{(C, \boldsymbol{k}')\}$;
    Backtrack(P, C, ρ, Γ, Δ);
  else
    pick $C' \in \mathsf{M}(C)$;
    $\boldsymbol{k}' := \mathit{next}(\boldsymbol{k}, \mathsf{i}(C))$;
    $\pi := (\mathsf{M}(C) \smallsetminus \{C'\}) \times \{\boldsymbol{k}'\}$;
    Explore(C', $\boldsymbol{k}'$, $((C, \boldsymbol{k}, \pi) :: \rho)$, Γ, Δ);
  end if
end function

function Backtrack(P, C, ρ, Γ, Δ)
  if $\rho = []$ **then**
    P;
  else if $\rho = ((C', \boldsymbol{k}', \pi) :: t)$ **then**
    if $\mathsf{P}(C') \neq P$ and $\pi \neq \emptyset$ **then**
      pick $(C'', \boldsymbol{k}'') \in \pi$;
      $\pi' := \pi \smallsetminus \{(C'', \boldsymbol{k}'')\}$;
      Explore(C'', $\boldsymbol{k}''$, $((C', \boldsymbol{k}', \pi') :: t)$, Γ, Δ);
    else
      if $\mathsf{P}(C') = P$ **then**
        $\Delta_P := \Delta_P \cup \{(C', \boldsymbol{k}')\}$ justified by C;
      else
        $\Delta_P := \Delta_P \cup \{(C', \boldsymbol{k}')\}$ justified by $\mathsf{M}(C')$;
      end if
      $\Gamma_P := \Gamma_P \smallsetminus \{(C', \boldsymbol{k}')\}$;
      if there is $(C', \boldsymbol{k}') \in \Gamma_{\overline{P}}$ **then**
        $\Delta_{\overline{P}} :=$ Forget($\Delta_{\overline{P}}$, $\Gamma_{\overline{P}}$, $(C', \boldsymbol{k}')$);
        $\Gamma_{\overline{P}} := \Gamma_{\overline{P}} \smallsetminus \{(C', \boldsymbol{k}')\}$;
      end if
      Backtrack(P, C', t, Γ, Δ);
    end if
  end if
end function

Figure 5. The general local algorithm.

formula $\varphi = \mu x_2.((\nu x_1.(p \wedge \Box x_1)) \vee \Diamond x_2)$ discussed in Example 3.4. As already discussed, the formula φ interpreted over T leads to the system E in Fig. 1d over the lattice $\mathbf{2}^{\mathbb{S}}$.

Suppose that we want to verify whether the state $a \in \mathbb{S}$ satisfies the formula φ. This requires to determine the winner of the fixpoint game from position $(a, 2)$, which can be done by invoking Explore($(a, 2)$, $\mathbf{0}$, $[]$, $(\emptyset, \emptyset)$, $(\emptyset, \emptyset)$). A computation performed by the algorithm is schematised in Fig. 6, where we only consider minimal moves. Since the choice of moves is non-deterministic, other search sequences are possible. In the diagram, positions of player $\exists$ are represented as diamonds, while those of $\forall$ are represented as boxes, the counters associated with the positions is on their lefthand side

Recall that the second equation is $x_2 =_\mu x_1 \cup \blacklozenge_T x_2$. Then, from the initial position $(a, 2)$, with counter $(0, 0)$, there are four available minimal moves, i.e., $(\{a\}, \emptyset)$, $(\emptyset, \{a\})$, $(\emptyset, \{b\})$ and $(\emptyset, \{c\})$, represented by the four outgoing edges from position $(a, 2)$ in the diagram, all four will have counter $(0, 1) = \mathit{next}((0, 0), 2)$. Indeed, it is easy to see that $a \in \{a\} \cup \blacklozenge_T \emptyset = \emptyset \cup \blacklozenge_T \{a\} = \emptyset \cup \blacklozenge_T \{b\} = \{a\} \subseteq \emptyset \cup \blacklozenge_T \{c\} = \{a, c\}$. Suppose that the algorithm chooses to explore the move $(\emptyset, \{b\})$, as highlighted by the bold arrow. Even though not shown in the diagram, the other moves are stored in the set of unexplored moves π associated with the position $(a, 2)$ in the playlist ρ. The search proceeds in this way along the

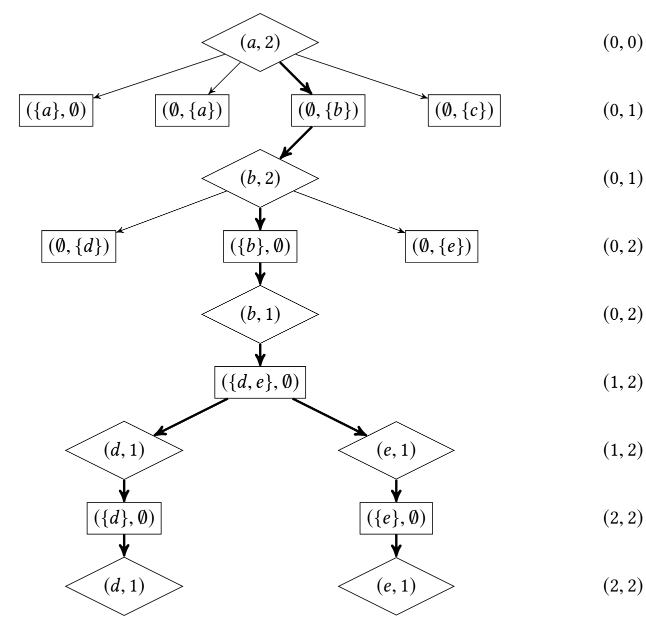


Figure 6. An execution of the local algorithm.

moves

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

until position $(d, 1)$ occurs again, with counter $(2, 2)$. Since the counter associated 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 the algorithm starts backtracking. While backtracking it generates a decision for $\exists$, which is $((\{d\}, \emptyset), (2, 2))$ justified by the only possible move $(d, 1)$ of player $\forall$. When it comes back to the first occurrence of $(d, 1)$, since it is a position controlled by $\exists$, the procedure transforms the assumption $((d, 1), (1, 2))$ into a decision for $\exists$ justified by the move $(\{d\}, \emptyset)$. Then, it backtracks to position $(\{d, e\}, \emptyset)$, which is controlled by player $\forall$ and there is still an unexplored move $(e, 1)$. Therefore, the algorithm starts exploring again from $(e, 1)$, and does so 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 Backtrack($\exists$, (a,2), [], Γ, Δ) terminates since $\rho = []$, and returns player $\exists$. Indeed, $\exists$ wins starting from position $(a, 2)$ since the state a satisfies the formula φ.

7.2 Correctness

We show that, when the lattice is finite, the algorithm terminates. Moreover, when it terminates (which could happen also on infinite lattices), it provides a correct answer.

Termination on finite lattices can be proved by observing that the set of positions (which are either elements 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 not possible to cycle indefinitely between two positions, so that termination immediately follows.

Lemma 7.2 (termination). *Given a fixpoint game on a finite lattice, any call* Explore*(C_0, $\mathbf{0}$, [], $(\emptyset, \emptyset)$, $(\emptyset, \emptyset)$) terminates, hence at some point* Backtrack*(P, C_0, [], $(\emptyset, \emptyset)$, Δ) is invoked, for some player P and pairs of sets Γ and Δ.*

The proof of correctness is long and technical. The underlying idea is to prove that, at any invocation of Explore($\cdot$, $\cdot$, ρ, Γ, Δ) and Backtrack($\cdot$, $\cdot$, ρ, Γ, Δ), the justifications for the decisions Δ_P, can be interpreted as a winning strategy for player P from the positions $C \in \Delta_P$, in a modified game where P immediately wins on the assumptions Γ_P. Since at termination, the set of assumptions is empty, the modified game coincides with the original one and thus we conclude.

Theorem 7.3 (correctness). *Given a fixpoint game, if a call* 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 the converse implication, that is, if P wins the game from C, then the call Explore(C, $\mathbf{0}$, [], $(\emptyset, \emptyset)$, $(\emptyset, \emptyset)$) returns P. Indeed, since the game can never result in a draw, this is equivalent to show that if the call Explore(C, $\mathbf{0}$, [], $(\emptyset, \emptyset)$, $(\emptyset, \emptyset)$) returns $\overline{P}$, then $\overline{P}$ wins the game from C. And this already holds by Theorem 7.3.

7.3 Using up-to techniques in the algorithm

In the literature about bisimilarity checking, up-to techniques have been fruitfully integrated with local checking algorithm for speeding up the computation (see, e.g., [Hirschkoff 1999]). Here we show that a similar idea can be developed for our local algorithm for general systems of fixpoint equations.

Let E be a system of m equations of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$ over a complete lattice L and let $\boldsymbol{u}$ be a compatible tuple of up-to functions for E. By Theorem 5.8 we have that the system $E\bar{\boldsymbol{u}}$ with equations $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\bar{\boldsymbol{u}} \cdot \boldsymbol{x})$ has the same solution as E. Now, since $\bar{\boldsymbol{u}}$ is a tuple of functions obtained as least fixpoints (see Definition 5.4), the system $E\bar{\boldsymbol{u}}$ can be "equivalently" written as the system of $2m$ equations that we denote by $d(E, \boldsymbol{u})$, defined as follows:

$$\boldsymbol{y} =_{\mu} (\boldsymbol{u} \cdot \boldsymbol{y}) \sqcup \boldsymbol{x}$$
$$\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{y})$$

More precisely, we can show the following result.

Theorem 7.4 (preserving solutions with up-to). *Let E be a system of m equations of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$ over a complete lattice L. Let $\boldsymbol{u}$ be a m-tuple of up-to functions compatible for E (Definition 5.7). The solution of the system $d(E, \boldsymbol{u})$ is $sol(d(E, \boldsymbol{u})) = (sol(E), sol(E))$.*

By relying on Theorem 7.4 we can derive an algorithm that exploits the up-to function $\boldsymbol{u}$. It is obtained by instantiating the general algorithm discussed before to the system $d(E, \boldsymbol{u})$ and suitably restricting the moves considered in the exploration. Roughly, the idea is to allow the use of the up-to function only when it leads immediately to an assumption or a decision. This is in some sense similar to what is done for bisimilarity checking in [Hirschkoff 1999], where the up-to function is used only to enlarge the set of states which are considered bisimilar. More precisely, when the exploration is in a position (b, i) corresponding to one of the added equations $y_i =_{\mu} u_i(y_i) \sqcup x_i$, according to the definition of the game, a possible move would be any $2m$-tuple of sets $(\boldsymbol{Y}, \boldsymbol{X})$ such that $b \sqsubseteq u_i(\bigsqcup Y_i) \sqcup \bigsqcup X_i$. First of all, since only the i-th and $(m + i)$-th components Y_i and X_i play a role and we can restrict to minimal moves (see §6.1), we can assume $X_j = Y_j = \emptyset$ for $j \neq i$. Moreover, for X_i and Y_i, we only allow two types of moves:

a) $X_i = \{b\}$ and $Y_i = \emptyset$, which means that we keep the focus on element b and just jump to the "original" equation $x_i =_{\eta_i} f(y_i)$, or
b) $X_i = \emptyset$ and all positions in Y_i will immediately become assumptions or decisions when explored.

At the level of the pseudocode, this only means that the action "pick" needs to be refined. Instead of simply choosing randomly a move in $\mathsf{M}(C)$, in some cases it has to perform a constrained choice. This is made precise below.

Definition 7.5 (up-to algorithm)**.** Let E be a system of m fixpoint equations over the complete lattice L and let $\boldsymbol{u}$ be a compatible tuple of up-to function for E. The up-to algorithm for E based on $\boldsymbol{u}$ is just the algorithm in Fig. 5 applied to the system $d(E, \boldsymbol{u})$, where, in function $\textsc{Explore}(C, \boldsymbol{k}, \rho, \Gamma, \Delta)$, when $C = (b, i)$ with $i \in \underline{m}$, the action "pick" can select only moves $C' = (\boldsymbol{Y}, \boldsymbol{X})$ such that $Y_j = X_j = \emptyset$ for $j \neq i$ and X_i, Y_i complying with either of the following conditions

a) $Y_i = \emptyset$ and $X_i = \{b\}$ or
b) $X_i = \emptyset$ and for all $b' \in Y_i$ it holds
  i) $((b', i), \boldsymbol{k}') \in \Delta_\exists$ with $\boldsymbol{k}' \leq_\exists \mathit{next}(\boldsymbol{k}, i)$ or
  ii) $((b', i), \boldsymbol{k}', \pi) \in \rho$ with $\boldsymbol{k}' <_\exists \mathit{next}(\boldsymbol{k}, i)$.

Condition (a) has been already clarified above. Condition (b) is a formal translation of the fact that Y_i can contain only positions for which there are usable decisions (case (b.i)) or that will immediately become assumptions (case b.ii)).

Clearly the modification does not affect termination on finite lattices (in fact, we just restrict the possible moves of a procedure which is known to be terminating). We next show that the up-to algorithm is also correct.

Theorem 7.6 (correctness with up-to)**.** *Let E be a system of m equations of the kind $\boldsymbol{x} =_\eta \boldsymbol{f}(\boldsymbol{x})$ over a complete lattice L. Let $\boldsymbol{u}$ a compatible m-tuple of up-to functions for E. Then the up-to algorithm associated with the system $d(E, \boldsymbol{u})$ as given in Definition 7.5 is correct, i.e., if a call* $\textsc{Explore}(C, \mathbf{0}, [], (\emptyset, \emptyset), (\emptyset, \emptyset))$ *returns a player P, then P wins the game from C.*

The proof is based on the observation that any winning strategy for player $\exists$ in the game associated with the original system E can be replicated in the game associated with the modified system $d(E, \boldsymbol{u})$, even when the moves are restricted as in Definition 7.5. This is done by choosing always moves corresponding to case (a) in Definition 7.5. Then strategies in the constrained game for $d(E, \boldsymbol{u})$ are also valid in the unconstrained game. We conclude since, by Theorem 7.4, we know that winning positions for player $\exists$ are the same in the game for E and in the game for $d(E, \boldsymbol{u})$.

Further optimizations of the up-to algorithm are possible by exploiting the fact that a variable y_i has the same solution of the corresponding x_i in the system $d(E, \boldsymbol{u})$. Intuitively, decisions and assumptions for positions associated with a variables y_i could be used as decisions and assumptions for the corresponding positions of variable x_i, and the other way around.

Example 7.7 (model-checking μ-calculus up-to bisimilarity)**.** In Example 7.1 we showed how the algorithm would solve a model-checking problem by exploring the corresponding fixpoint game. Suppose that this time we also want to use up-to bisimilarity as an up-to technique to answer the same question, that is, whether the state $a \in \mathbb{S}$ satisfies the formula $\varphi = \mu x_2.((\nu x_1.(p \wedge \Box x_1)) \vee \Diamond x_2)$. In Example 5.10 we presented the up-to function $u_\sim : \mathbf{2}^\mathbb{S} \to \mathbf{2}^\mathbb{S}$ corresponding to up-to bisimilarity defined as $u_\sim(X) = \{s \in \mathbb{S} \mid s \sim_T s' \wedge s' \in X\}$. In order to apply the procedure described above, first we need to build the system $d(E, (u_\sim, u_\sim))$, which is

$$y_1 =_\mu u_\sim(y_1) \cup x_1 \qquad x_1 =_\nu \{b, d, e\} \cap \blacksquare_T y_1$$
$$y_2 =_\mu u_\sim(y_2) \cup x_2 \qquad x_2 =_\mu y_1 \cup \blacklozenge_T y_2$$

Then, to check whether the state a satisfies the formula φ we invoke the function $\textsc{Explore}((a, 4), \mathbf{0}, [], (\emptyset, \emptyset), (\emptyset, \emptyset))$, where the index 4 is that of the variable x_2 in the system $d(E, (u_\sim, u_\sim))$. Then, the algorithm behaves in similar fashion to what described in Example 7.1. However, this time the exploration of position $(d, 1)$ with counter $(0, 0, 1, 2)$ is pruned by using the up-to function. Recalling that position $(b, 1)$ occurred in the past, hence it is included in the playlist, with counter $(0, 0, 0, 2)$, we have that condition (b) above holds here for the move $(\{b\}, \emptyset, \emptyset, \emptyset)$ since $d \sim b$, hence $d \in u_\sim(\{b\}) \cup \emptyset$, and $(0, 0, 0, 2) <_\exists \mathit{next}((0, 0, 1, 2), 1) = (1, 0, 1, 2)$. This leads to making an assumption for $(b, 1)$ and then backtracking up to the root. The same happens when exploring the other branch, that is position $(e, 1)$, since also $e \sim b$. Similarly to the previous example, the last invocation $\textsc{Backtrack}(\exists, (a,4), [], \Gamma, \Delta)$ returns player $\exists$. Indeed, $\exists$ wins starting from position $(a, 4)$ since the state a satisfies the formula φ.

8 Conclusion

We have presented a theory of (sound and complete) abstractions for solving fixpoint equation systems, of which up-to techniques are a special case, including on-the-fly procedures for solving such equation systems.

Related work: Our work draws inspiration from various sources. Clearly our contribution is based on the notion of approximation as formalised in the theory of abstract interpretation, and, in particular, on the idea of capturing abstractions in the form of Galois connections, heavily used in program analysis and advocated in [Cousot and Cousot 1977, 1979a]. Due to the intimate connection of Galois connections and closure functions, there is a close correspondence with up-to techniques for enhancing coinduction proofs [Pous 2007; Pous and Sangiorgi 2011], originally developed for CCS [Milner 1989]. However, as far as we know, recent research has only started to explore this connection: [Bonchi et al. 2018a] explains the relation between sound up-to techniques and complete abstract domains in the setting where the semantic function has an adjoint. This adjunction or Galois connection plays a different role than the abstractions: it gives the existential player a unique best move, a concept explored in §6.2.2.

Fixpoint equation systems largely derive their interest from μ-calculus model-checking (see [Bradfield and Walukiewicz 2018] for an overview). Evaluating μ-calculus formulae on a transition system can be reduced to solving a parity game and the exact complexity of this task is still open. Progress measures, a tool introduced by [Jurdziński 2000], allow one to solve parity games with a complexity which is polynomial in the number of states and exponential in (half of) the alternation depth of the formula. Recent approaches have devised quasi-polynomial algorithms for parity games [Calude et al. 2017; Jurdzinski and Lazic 2017; Lehtinen 2018].

Instead of improving the complexity bounds, our main aim here is to introduce heuristics, based on an on-the-fly algorithm and up-to functions that are known to achieve good efficiency improvements in practice.

We also showed – for a special case – how on-the-fly algorithms inspired by [Bonchi et al. 2018a; Bonchi and Pous 2013; Hirschkoff 1998, 1999] for a single (greatest) fixpoint equation can be adapted to the case of general lattices. For the general case of arbitrary fixpoint equation systems a generalisation along the lines of [Stevens and Stirling 1998] is possible, but we omitted it due to lack of space.

The use of assumptions as stopping conditions in the algorithm is reminiscent of parameterized coinduction, where the basic idea is to parameterize the fixpoint over the accumulated knowledge of the "proof so far" [Hur et al. 2013; Sprunger and Moss 2017]. This allows to introduce extra assumptions into coinductive proofs. As spelled out in [Pous 2016] parameterized coinduction is closely related to the companion and hence to up-to techniques.

As applications beyond bisimulation checking, the area where up-to techniques have originated [Milner 1989], we have considered fixpoint equation systems over the reals [Mio and Simpson 2017] and abstraction in μ-calculus model-checking, based on simulations [Loiseaux et al. 1995]. The latter technique has been extended to modal respectively mixed transition systems that feature both may and must transitions [Dams et al. 1997; Larsen and Thomsen 1988; Schmidt 2000], which allow to both preserve and reflect the validity of a formula.

Future work: There are some interesting questions that can be derived from our work. First, the notion of progress measures that has been studied in [Baldan et al. 2019] can be adapted to the game for arbitrary complete (rather than just continuous) lattices, introduced in this paper. A first natural question to ask is whether the on-the-fly algorithm arises as an instance of the single equation algorithm instantiated with the progress measure fixpoint equation.

With respect to the applications, we will investigate whether our case study on abstractions respectively simulations for μ-calculus model-checking can also be generalised to modal transition systems [Grumberg et al. 2007; Larsen and Thomsen 1988].

Furthermore, we studied approximations in connection with solving fixpoint equations over the reals, in turn closely connected to the model checking of probabilistic logics. While the technique is sound, there are no guarantees on the quality of these overapproximations, in particular for non-continuous functions the upper bound might be too coarse. We plan to investigate under which circumstances one can obtain guarantees to be close to the exact solution or to compute the exact solution directly. Another interesting area is the use of up-to techniques for behavioural metrics [Bonchi et al. 2018b].

References

Samson Abramsky and Achim Jung. 1994. Domain Theory. In *Handbook of Logic in Computer Science*, Samson Abramsky, Dov Gabbay, and Thomas Stephen Edward Maibaum (Eds.). Oxford University Press, 1–168.

Paolo Baldan, Barbara König, Christina Mika-Michalski, and Tommaso Padoan. 2019. Fixpoint Games in Continuous Lattices. In *Proceedings of POPL'19*, Stephanie Weirich (Ed.), Vol. 3. ACM, 26:1–26:29.

Andrea Bianco and Luca de Alfaro. 1995. Model Checking of Probabalistic and Nondeterministic Systems. In *Proc. of FSTTCS '95 (Lecture Notes in Computer Science)*, Vol. 1026. Springer, 499–513.

Filippo Bonchi, Pierre Ganty, Roberto Giacobazzi, and Dusko Pavlovic. 2018a. Sound up-to techniques and Complete abstract domains. In *Proc. of LICS '18*. ACM, 175–184.

Filippo Bonchi, Barbara König, and Daniela Petrişan. 2018b. Up-To Techniques for Behavioural Metrics via Fibrations. In *Proc. of CONCUR '18 (LIPIcs)*, Vol. 118. Schloss Dagstuhl – Leibniz Center for Informatics, 17:1–17:17.

Filippo Bonchi and Damien Pous. 2013. Checking NFA equivalence with bisimulations up to congruence. In *POPL*. ACM, 457–468.

Julian Bradfield and Igor Walukiewicz. 2018. The mu-Calculus and Model Checking. In *Handbook of Model Checking*, Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem (Eds.). Springer, 871–919.

Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. 2017. Deciding parity games in quasipolynomial time. In *Proc. of STOC '17*. ACM, 252–263.

Rance Cleaveland, Marion Klein, and Bernhard Steffen. 1992. Faster model checking for the modal Mu-Calculus. In *Proc. of CAV 1992 (Lecture Notes in Computer Science)*, Vol. 663. Springer, 410–422.

Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proc. of POPL '77*. ACM, 238–252.

Patrick Cousot and Radhia Cousot. 1979a. Systematic design of program analysis frameworks. In *Proc. of POPL '79*. ACM, 269–282.

Radhia Cousot and Patrick Cousot. 1979b. A constructive characterization of the lattices of all retractions, preclosure, quasi-closure and closure operators on a complete lattice. *Portugaliae Mathematica* 38, 1-2 (1979), 185–198.

Radhia Cousot and Patrick Cousot. 1979c. Constructive versions of Tarski's fixed point theorems. *Pacific J. Math.* 82, 1 (1979), 43–57.

Dennis Dams, Rob Gerth, and Orna Grumberg. 1997. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.* 19, 2 (1997), 253–291.

Roberto Giacobazzi, Francesco Ranzato, and Francesca Scozzari. 2000. Making abstract interpretations complete. *J. ACM* 47, 2 (2000), 361–416.

Orna Grumberg, Martin Lange, Martin Leucker, and Sharon Shoham. 2007. When not losing is better than winning: Abstraction and refinement for the full mu-calculus. *Information and Computation* 205, 8 (2007), 1130–1148.

Ichiro Hasuo, Shunsuke Shimizu, and Corina Cîrstea. 2016. Lattice-theoretic progress measures and coalgebraic model checking. In *Proc. of POPL '16*. ACM, 718–732.

Daniel Hirschkoff. 1998. Automatically Proving Up To Bisimulation. In *Proc. of MFCS '98 Workshop on Concurrency (Electronic Notes in Theoretical Computer Science)*. Elsevier, 75–89.

Daniel Hirschkoff. 1999. *Mise en oeuvre de preuves de bisimulation.* Ph.D. Dissertation. Ecole Nationale des Ponts et ChaussÃl'es.

Chung-Kil Hur, Georg Neis, Derek Dreyer, and Viktor Vafeiadis. 2013. The power of parameterization in coinductive proof. In *POPL*. ACM, 193–206.

Michael Huth and Marta Kwiatkowska. 1997. Quantitative analysis and model checking. In *Proc. of LICS '97*. IEEE, 111–122.

Marcin Jurdziński. 2000. Small Progress Measures for Solving Parity Games. In *Proc. of STACS '00 (Lecture Notes in Computer Science)*, Vol. 1770. Springer, 290–301.

Marcin Jurdzinski and Ranko Lazic. 2017. Succinct progress measures for solving parity games. In *Proc. of LICS '17*. ACM/IEEE, 1–9.

Dexter Kozen. 1983. Results on the Propositional μ-Calculus. *Theoretical Computer Science* 27, 3 (1983), 333–354.

Kim Guldstrand Larsen and Bent Thomsen. 1988. A Modal Process Logic. In *Proc. of LICS '88*. IEEE, 203–210.

Karoliina Lehtinen. 2018. A modal μ perspective on solving parity games in quasi-polynomial time. In *Proc. of LICS '18*. ACM/IEEE, 639–648.

Claire Loiseaux, Susanne Graf, Joseph Sifakis, Ahmed Bouajjani, and Saddek Bensalem. 1995. Property Preserving Abstractions for the Verification of Concurrent Systems. *Formal Methods in System Design* 6 (1995), 1–35.

Annabelle McIver and Carroll Morgan. 2007. Results on the quantitative μ-calculus qMμ. *ACM Trans. Comp. Log.* 8, 1:3 (2007), 43.

Robin Milner. 1989. *Communication and Concurrency.* Prentice Hall.

Antoine Miné. 2017. Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation. *Foundations and Trends in Programming Languages* 4, 3-4 (2017), 120–372.

Matteo Mio and Alex Simpson. 2017. Łukasiewicz μ-calculus. *Fundamenta Informaticae* 150, 3-4 (2017), 317–346.

Damien Pous. 2007. Complete Lattices and Up-To Techniques. In *Proc. of APLAS '07*. Springer, 351–366. LNCS 4807.

Damien Pous. 2016. Coinduction All the Way Up. In *Proc. of LICS'16*. ACM, 307–316.

Damien Pous and Davide Sangiorgi. 2011. Enhancements of the bisimulation proof method. In *Advanced Topics in Bisimulation and Coinduction*, Davide Sangiorgi and Jan Rutten (Eds.). Cambridge University Press.

Davide Sangiorgi. 2011. *Introduction to Bisimulation and Coinduction.* Cambridge University Press.

Davide Sangiorgi and Robin Milner. 1992. The problem of "weak bisimulation up to". In *Proc. of CONCUR'92*, W.R. Cleaveland (Ed.). Springer, 32–46.

David A. Schmidt. 2000. Binary Relations for Abstraction and Refinement. In *Workshop on Refinement and Abstraction*. Elsevier Electronic.

Dana Scott. 1972. Continuous lattices. In *Toposes, Algebraic Geometry and Logic (Lecture Notes in Mathematics)*, F. W. Lawvere (Ed.). Springer, 97–136.

Helmut Seidl. 1996. Fast and simple nested fixpoints. *Inform. Process. Lett.* 59, 6 (1996), 303–308.

David Sprunger and Lawrence S. Moss. 2017. Precongruences and Parametrized Coinduction for Logics for Behavioral Equivalence. In *Proc. of CALCO '17 (LIPIcs)*, Vol. 72. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 23:1–23:15.

Perdita Stevens and Colin Stirling. 1998. Practical Model-Checking Using Games. In *Proc. of TACAS '98 (Lecture Notes in Computer Science)*, Vol. 1384. Springer, 85–101.

Colin Stirling. 1995. Local Model Checking Games. In *Proc. of CONCUR '95 (Lecture Notes in Computer Science)*, Vol. 962. Springer, 1–11.

Alfred Tarski. 1955. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.* 5 (1955), 285–309.

A Proofs for Section 4 (Approximation for Systems of Fixpoint Equations)

Lemma A.1 (concretisation for single fixpoints). *Let $\gamma : A \to C$ be a monotone function.*

a) If

$$f_C \circ \gamma \sqsubseteq \gamma \circ f_A \tag{9}$$

then $\mu f_C \sqsubseteq \gamma(\mu f_A)$; if, in addition, γ is co-continuous and co-strict $\nu f_C \sqsubseteq \gamma(\nu f_A)$.

b) If

$$\gamma \circ f_A \sqsubseteq f_C \circ \gamma \tag{10}$$

then $\gamma(\nu f_A) \sqsubseteq \nu f_C$; if, in addition, γ is continuous and strict then $\gamma(\mu f_A) \sqsubseteq \mu f_C$.

Proof. We focus on the soundness results since the completeness results follow by duality.

For least fixpoint, we prove that for all ordinals β we have $f_C^\beta(\bot_C) \leq \gamma(f_A^\beta(\bot_A))$, whence the thesis, since $\mu f_C = f_C^\beta(\bot_C)$ and $\mu f_A = f_A^\beta(\bot_A)$ for some ordinal β (just take the largest of the ordinals needed to reach the two fixpoints).

We proceed by transfinite induction:

- ($\beta = 0$) We have $f_C^0(\bot_C) = \bot_C \sqsubseteq \gamma(f_A^0(\bot_A))$, as desired.
- ($\beta \to \beta + 1$) Observe that

$$\begin{aligned}
f_C^{\beta+1}(\bot_C) &= f_C(f_C^\beta(\bot_C)) \\
&\sqsubseteq f_C(\gamma(f_C^\beta(\bot_C))) && \text{[by ind. hyp. and monotonicity of } f_C\text{]} \\
&\leq \gamma(f_A(f_A^\beta(\bot_A))) && \text{[by (9)]} \\
&= \gamma(f_A^{\beta+1}(\bot_A))
\end{aligned}$$

- (β limit ordinal) In this case

$$\begin{aligned}
f_C^\beta(\bot_C) &= \bigsqcup_{\beta' < \beta} f_C^{\beta'}(\bot_C) \\
&\sqsubseteq \bigsqcup_{\beta' < \beta} \gamma(f_A^{\beta'}(\bot_A)) && \text{[by ind. hyp.]} \\
&\sqsubseteq \gamma(\bigsqcup_{\beta' < \beta} f_A^{\beta'}(\bot_A)) && \text{[by properties of joins]} \\
&= \gamma(f_A^\beta(\bot_A))
\end{aligned}$$

For greatest fixpoints, we prove that for all ordinals β we have $f_C^\beta(\top_C) \leq \gamma(f_A^\beta(\top_A))$, again by transfinite induction.

- ($\beta = 0$) We have $f_C^0(\top_C) = \top_C = \gamma(\top_A) = \gamma(f_A^0(\top_A))$, since γ is assumed to be co-strict, hence we have the desired inequality.
- ($\beta \to \beta + 1$) Observe that

$$\begin{aligned}
f_C^{\beta+1}(\top_C) &= f_C(f_C^\beta(\top_C)) \\
&\sqsubseteq f_C(\gamma(f_A^\beta(\top_A))) && \text{[by ind. hyp. and monotonicity of } f_C\text{]} \\
&\sqsubseteq \gamma(f_A(f_A^\beta(\top_A))) && \text{[by (9)]} \\
&= \gamma(f_A^{\beta+1}(\top_A))
\end{aligned}$$

- (β limit ordinal) In this case

$$
\begin{aligned}
f_C^{\beta}(\top_C) &= \bigsqcap_{\beta'<\beta} f_C^{\beta'}(\top_C)) \\
&\sqsubseteq \bigsqcap_{\beta'<\beta} \gamma(f_A^{\beta'}(\top_A)) && \text{[by ind. hyp.]} \\
&= \gamma(\bigsqcap_{\beta'<\beta} f_A^{\beta'}(\top_A)) && \text{[since } \gamma \text{ is co-continuous]} \\
&= \gamma(f_A^{\beta}(\top_A))
\end{aligned}
$$

□

We can get analogous results for abstractions, by duality.

Lemma A.2 (abstraction for single fixpoints). *Let $\alpha : C \to A$ be an abstraction function.*

a) If

$$\alpha \circ f_C \leq f_A \circ \alpha \tag{11}$$

then $\alpha(\nu f_C) \leq \nu f_A$; if, in addition, α is continuous and strict $\alpha(\mu f_C) \leq \mu f_A$.

b) If

$$f_A \circ \alpha \leq \alpha \circ f_C \tag{12}$$

then $\mu f_A \leq \alpha(\mu f_C)$; if, in addition, α is co-continuous and co-strict then $\nu f_A \leq \alpha(\nu f_C)$.

Lemma A.3 (Galois insertions). *Let $f_C : C \to C$ and $f_A : A \to A$ be monotone functions and let $\langle \alpha, \gamma \rangle : C \to A$ be a Galois insertion.*

a) Assume soundness for α i.e., (11) (equivalent to soundness for γ, i.e., (9)), and completeness for both α and β, i.e., (12), (10). Then

$$\alpha(\eta f_C) = \eta f_A \text{ for } \eta \in \{\mu, \nu\} \qquad \nu f_C = \gamma(\nu f_A) \qquad \mu f_C \sqsubseteq \gamma(\mu f_A)$$

b) Assume

$$f_C = \gamma \circ f_A \circ \alpha \tag{13}$$

then $\alpha(\eta f_C) = \eta f_A$ and $\eta f_C = \gamma(\eta f_A)$ for $\eta \in \{\mu, \nu\}$.

Proof. a) Just using Lemma A.1 and Lemma A.2, we obtain

$$\text{(a) } \alpha(\mu f_C) = \mu f_A \quad \text{(b) } \nu f_C = \gamma(\nu f_A) \quad \text{(c) } \alpha(\mu f_C) \leq \mu f_A \quad \text{(d) } \mu f_C \sqsubseteq \gamma(\mu f_A)$$

From (b), applying α, we obtain $\alpha(\nu f_C) = \alpha(\gamma(\nu f_A) = \nu f_A$, and we are done.

b) In this case, from the assumption $f_C = \gamma \circ f_A \circ \alpha$ one can easily deduce the soundness and completeness conditions for α and γ, i.e., (11), (12), (9), (10). Therefore, by the previous point we get all desired inequalities but $\gamma(\mu f_A) \sqsubseteq \mu f_C$. For this observe that

$$
\begin{aligned}
\gamma(\mu f_A) &= \gamma(\alpha(\mu f_C)) && \text{[since } \mu f_A = \alpha(\mu f_C)\text{]} \\
&= \gamma(\alpha(f_C(\mu f_C))) && \text{[since } \mu f_C \text{ is a fixpoint of } f_C\text{]} \\
&= \gamma(\alpha(\gamma(f_A(\alpha(\mu f_C))))) && \text{[since } f_C = \gamma \circ f_A \circ \alpha\text{]} \\
&= \gamma(f_A(\alpha(\mu f_C))) && \text{[since } \alpha \circ \gamma = id_A\text{]} \\
&= f_C(\mu f_C) && \text{[since } f_C = \gamma \circ f_A \circ \alpha\text{]} \\
&= \mu f_C && \text{[since } \mu f_C \text{ is a fixpoint of } f_C\text{]}
\end{aligned}
$$

□

Theorem 4.1 (sound concretisation for systems). *Let $(C, \sqsubseteq)$ and $(A, \leq)$ be complete lattices, let E_C of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^C(\boldsymbol{x})$ and E_A of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^A(\boldsymbol{x})$ be systems of m equations over C and A, with solutions $\boldsymbol{s}^C \in C^m$ and $\boldsymbol{s}^A \in A^m$, respectively. Let $\boldsymbol{\gamma}$ be an m-tuple of monotone functions, with $\gamma_i : A \to C$ for $i \in \underline{m}$. If $\boldsymbol{\gamma}$ satisfies*

$$\boldsymbol{f}^C \circ \boldsymbol{\gamma}^\times \sqsubseteq \boldsymbol{\gamma}^\times \circ \boldsymbol{f}^A \tag{2}$$

with γ_i co-continuous and co-strict for each $i \in \underline{m}$ such that $\eta_i = \nu$, then $\boldsymbol{s}^C \sqsubseteq \boldsymbol{\gamma}^\times(\boldsymbol{s}^A)$.

Proof. We proceed by induction on m. The case $m = 0$ is trivial.

For the inductive case, consider systems with $m+1$ equations. Recall that, in order to solve the system, the last variable x_{m+1} is considered as a fixed parameter x and the system of m equations that arises from dropping the last equation is recursively solved. This produces an m-tuple $\boldsymbol{t^z}_{1,m}(x) = sol(E_z[x_{m+1} := x])$ parametric on x, for $z \in \{A, C\}$. For all $a \in A$, by inductive hypothesis applied to the systems $E_A[x_{m+1} := a]$ and $E_C[x_{m+1} := \gamma_{m+1}(a)]$ we obtain

$$\boldsymbol{t^C}_{1,m}(\gamma_{m+1}(a)) \sqsubseteq \boldsymbol{\gamma}_{1,m}{}^{\times}(\boldsymbol{t^A}_{1,m}(a)) \tag{14}$$

Inserting the parametric solution into the last equation, we get an equation in a single variable

$$a =_{\eta_m} f^A_{m+1}(\boldsymbol{t^A}_{1,m}(a), a).$$

This equation can be solved by taking the corresponding fixpoint, i.e., if we define $f_A(a) = f^A_{m+1}(\boldsymbol{t^A}_{1,m}(a), a)$, then $s^A_{m+1} = \eta_{m+1} f_A$. In the same way, $s^C_{m+1} = \eta_{m+1} f_C$ where $f_C(c) = f^C_{m+1}(\boldsymbol{t^C}_{1,m}(c), c)$.

Observe that $f_C \circ \gamma_{m+1} \sqsubseteq \gamma_{m+1} \circ f_A$. In fact

$$\begin{aligned}
& f_C(\gamma_{m+1}(a)) = && \\
& = f^C_{m+1}(\boldsymbol{t^C}_{1,m}(\gamma_{m+1}(a)), \gamma_{m+1}(a))) && \text{[definition of } f_C\text{]} \\
& \sqsubseteq f^C_{m+1}(\boldsymbol{\gamma}_{1,m}{}^{\times}(\boldsymbol{t^A}_{1,m}(a)), \gamma_{m+1}(a))) && \text{[by (14)]} \\
& \sqsubseteq f^C_{m+1}(\boldsymbol{\gamma}^{\times}(\boldsymbol{t^A}_{1,m}(a), a)) && \text{[application of } \boldsymbol{\gamma}\text{]} \\
& \sqsubseteq \gamma_{m+1}(f^A_{m+1}(\boldsymbol{t^A}_{1,m}(a), a)) && \text{[hypothesis (2)]} \\
& = \gamma_{m+1}(f_A(a)) && \text{[definition of } f_A\text{]}
\end{aligned}$$

Therefore, recalling that when $\eta_{m+1} = \mu$ we are assuming co-continuity and co-strictness for γ_{m+1}, we can apply Lemma A.1(a) and deduce that

$$s^C_{m+1} = \eta_{m+1} f_C \sqsubseteq \gamma_{m+1}(\eta_{m+1} f_A) = \gamma_{m+1}(s^A_{m+1}) \tag{15}$$

Finally, recall that the first m components of the solutions are $\boldsymbol{s^z}_{1,m} = \boldsymbol{t^z}_{1,m}(s^z_{m+1})$ for $z \in \{C, A\}$. Therefore, exploiting (14), we have

$$\begin{aligned}
& \boldsymbol{s^C}_{1,m} = && \\
& = \boldsymbol{t^C}_{1,m}(s^C_{m+1}) && \\
& \sqsubseteq \boldsymbol{t^C}_{1,m}(\gamma_{m+1}(s^A_{m+1})) && \text{[by (15)]} \\
& \sqsubseteq \boldsymbol{\gamma}_{1,m}{}^{\times}(\boldsymbol{t^A}_{1,m}(s^A_{m+1})) && \text{[by (14)]} \\
& = \boldsymbol{\gamma}_{1,m}{}^{\times}(\boldsymbol{s^A}_{1,m}) &&
\end{aligned}$$

This concludes the inductive step. □

Everything can be dually formulated in terms of abstraction functions.

Theorem A.4 (sound abstraction for systems). *Let $(C, \sqsubseteq)$ and $(A, \leq)$ be complete lattices and let E_C of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^C(\boldsymbol{x})$ and E_A of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^A(\boldsymbol{x})$ be systems of m equations over C and A, with solutions $\boldsymbol{s}^C \in C^m$ and $\boldsymbol{s}^A \in A^m$, respectively. Let $\boldsymbol{\alpha}$ be an m-tuple of monotone functions, with $\alpha_i : C \to A$ for $i \in \underline{m}$. If $\boldsymbol{\alpha}$ satisfies*

$$\boldsymbol{\alpha}^{\times} \circ \boldsymbol{f}^C \leq \boldsymbol{f}^A \circ \boldsymbol{\alpha}^{\times}$$

with α_i continuous and strict for each $i \in \underline{m}$ such that $\eta_i = \mu$, then $\boldsymbol{\alpha}^{\times}(\boldsymbol{s}^C) \leq \boldsymbol{s}^A$.

Proof. This follows from Lemma 4.1 by duality. □

Theorem 4.2 (abstraction via Galois connections). *Let $(C, \sqsubseteq)$ and $(A, \leq)$ be complete lattices, let E_C of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^C(\boldsymbol{x})$ and E_A of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^A(\boldsymbol{x})$ be systems of m equations over C and A, with solutions $\boldsymbol{s}^C \in C^m$ and $\boldsymbol{s}^A \in A^m$, respectively. Let $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$ be m-tuples of monotone functions, with $\langle \alpha_i, \gamma_i \rangle : C \to A$ forming a Galois connection for each $i \in \underline{m}$.*

a) Soundness*: If $\boldsymbol{\gamma}$ satisfies $\boldsymbol{f}^C \circ \boldsymbol{\gamma}^{\times} \sqsubseteq \boldsymbol{\gamma}^{\times} \circ \boldsymbol{f}^A$ (2) or equivalently $\boldsymbol{\alpha}$ satisfies*

$$\boldsymbol{\alpha}^{\times} \circ \boldsymbol{f}^C \leq \boldsymbol{f}^A \circ \boldsymbol{\alpha}^{\times}, \tag{3}$$

then $\boldsymbol{\alpha}^{\times}(\boldsymbol{s}^C) \leq \boldsymbol{s}^A$ (equivalent to $\boldsymbol{s}^C \sqsubseteq \boldsymbol{\gamma}^{\times}(\boldsymbol{s}^A)$).

b) Completeness (for abstraction)*: If $\boldsymbol{\alpha}$ satisfies*

$$\boldsymbol{f}^A \circ \boldsymbol{\alpha}^\times \leq \boldsymbol{\alpha}^\times \circ \boldsymbol{f}^C \tag{4}$$

with α_i co-continuous and co-strict for each $i \in \underline{m}$ such that $\eta_i = \nu$, then $\boldsymbol{s}^A \leq \boldsymbol{\alpha}^\times(\boldsymbol{s}^C)$.

c) Completeness (for concretisation)*: If $\boldsymbol{\gamma}$ satisfies*

$$\boldsymbol{\gamma}^\times \circ \boldsymbol{f}^A \sqsubseteq \boldsymbol{f}^C \circ \boldsymbol{\gamma}^\times \tag{5}$$

with γ_i continuous and strict for each $i \in \underline{m}$ such that $\eta_i = \mu$, then $\boldsymbol{\gamma}^\times(\boldsymbol{s}^A) \sqsubseteq \boldsymbol{s}^C$.

Proof. Due to Theorems 4.1 and A.4 (and the fact that we can apply the theorems to lattices with reversed order), the only thing to prove is that the conditions (3) $\boldsymbol{\alpha}^\times \circ \boldsymbol{f}^C \leq \boldsymbol{f}^A \circ \boldsymbol{\alpha}^\times$ and (2) $\boldsymbol{f}^C \circ \boldsymbol{\gamma}^\times \sqsubseteq \boldsymbol{\gamma}^\times \circ \boldsymbol{f}^A$ are equivalent. If we assume (3), by definition of Galois connection, we get $\boldsymbol{f}^C \sqsubseteq \boldsymbol{\gamma}^\times \circ \boldsymbol{f}^A \circ \boldsymbol{\alpha}^\times$. Now, post-composing with $\boldsymbol{\gamma}^\times$ and exploiting the fact that $\boldsymbol{\alpha}^\times \circ \boldsymbol{\gamma}^\times \sqsubseteq \boldsymbol{id}^\times$ we obtain

$$\boldsymbol{f}^C \circ \boldsymbol{\gamma}^\times \sqsubseteq \boldsymbol{\gamma}^\times \circ \boldsymbol{f}^A \circ \boldsymbol{\alpha}^\times \circ \boldsymbol{\gamma}^\times \sqsubseteq \boldsymbol{\gamma} \circ \boldsymbol{f}^A$$

as desired.

The converse implication is analogous. □

For Galois insertions, we make explicit a very special case where we get rid of all the (co-)continuity and (co-)strictness requirements, and get soundness and completeness both for the abstraction and the concretisation.

Lemma A.5 (Galois insertions for systems)**.** *Let $(C, \sqsubseteq)$ and $(A, \leq)$ be complete lattices, let E_C of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^C(\boldsymbol{x})$ and E_A of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^A(\boldsymbol{x})$ be systems of m equations over C and A, with solutions $\boldsymbol{s}^C \in C^m$ and $\boldsymbol{s}^A \in A^m$, respectively. Let $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$ be m-tuples of abstraction and concretisation functions, with $\langle \alpha_i, \gamma_i \rangle : C \to A$ forming a Galois insertion for each $i \in \underline{m}$. If*

$$\boldsymbol{f}_C = \boldsymbol{\gamma}^\times \circ \boldsymbol{f}^A \circ \boldsymbol{\alpha} \tag{16}$$

then $\boldsymbol{\alpha}^\times(\boldsymbol{s}^C) = \boldsymbol{s}^A$ and $\boldsymbol{s}^C = \boldsymbol{\gamma}^\times(\boldsymbol{s}^A)$.

Lemma A.6 (sound abstract operators, compositionally)**.** *Let $(C, \sqsubseteq)$ and $(A, \leq)$ be complete lattices and, for $Z \in \{C, A\}$, let $\boldsymbol{f}^Z$ and $\boldsymbol{g}^Z$ be m-tuples of functions $f_i^Z, g_i^Z : Z^m \to Z$ for $i \in \underline{m}$. Let $\boldsymbol{\gamma}$ be an m-tuple of monotone functions, with $\gamma_i : A \to C$ for $i \in \underline{m}$.*

If $\boldsymbol{f}^C \circ \boldsymbol{\gamma}^\times \sqsubseteq \boldsymbol{\gamma}^\times \circ \boldsymbol{f}^A$ and $\boldsymbol{g}^C \circ \boldsymbol{\gamma}^\times \sqsubseteq \boldsymbol{\gamma}^\times \circ \boldsymbol{g}^A$ then $(\boldsymbol{f}^C \circ \boldsymbol{g}^C) \circ \boldsymbol{\gamma}^\times \sqsubseteq \boldsymbol{\gamma}^\times \circ (\boldsymbol{f}^A \circ \boldsymbol{g}^A)$.

Proof. This derives from a routine calculation using the hypotheses and monotonicity of the involved functions.

$$\boldsymbol{f}^C \circ \boldsymbol{g}^C \circ \boldsymbol{\gamma}^\times \sqsubseteq \boldsymbol{f}^C \circ \boldsymbol{\gamma}^\times \circ \boldsymbol{g}^A \sqsubseteq \boldsymbol{\gamma}^\times \circ \boldsymbol{f}^A \circ \boldsymbol{g}^A$$

□

B Proofs for Section 5 (Up-to Techniques)

Lemma 5.3 (compatible up-to functions as sound and complete abstractions)**.** *Let $f : L \to L$ be a monotone function and let $u : L \to L$ be an f-compatible closure. Consider the Galois insertion $\langle u, i \rangle : L \to u(L)$ where $i : u(L) \to L$ is the inclusion. Then*

a) f restricts to $u(L)$, i.e., $f_{|u(L)} : u(L) \to u(L)$;

b) $f_{|u(L)}$ is a sound and complete abstraction of both f and $f \circ u$. Therefore $\nu f = i(\nu f_{|u(L)}) = \nu(f \circ u)$. Moreover, if u is continuous and strict then $\mu f = i(\mu f_{|u(L)}) = \mu(f \circ u)$.

$$\begin{tikzcd} L \arrow[loop above, "f"] \arrow[loop left, "f\circ u"] \arrow[r, shift right, "u"'] & u(L) \arrow[l, shift right, "i"'] \arrow[loop right, "f_{|u(L)}"] \end{tikzcd}$$

Proof. a) We have that for all $l \in u(L)$, the f-image $f(l) \in u(L)$. Let $l \in u(L)$, i.e., $l = u(l')$ for some $l' \in L$. Observe that

$$\begin{aligned} f(l) &\sqsubseteq u(f(l)) && \text{[by extensiveness]} \\ &\sqsubseteq f(u(l)) && \text{[by compatibility]} \\ &= f(u(u(l'))) \\ &= f(u(l')) && \text{[by idempotency]} \\ &= f(l) \end{aligned}$$

Hence $f(l) = u(f(l))$, which means that $f(l) \in u(L)$.

b) We first prove that $\nu f = \nu f_{|u(L)}$. Consider

$$\begin{tikzcd} L \arrow[loop below, "f"] \arrow[r, shift left, "\alpha=u"] & u(L) \arrow[l, shift left, "\gamma=i"] \arrow[loop below, "f|_{u(L)}"] \end{tikzcd}$$

Note that for all $l \in u(L)$, we have $f(\gamma(l)) = f(l) = \gamma(f_{|u(L)}(l))$, i.e., γ satisfies soundness (9) and completeness (10) in Lemma A.1. Therefore, $\nu f = \gamma(\nu f_{|u(L)}) = \eta f_{|u(L)}$, as desired.

Next we prove that $\nu(f \circ u) = \nu f_{|u(L)}$ Consider

$$\begin{tikzcd} L \arrow[loop below, "f\circ u"] \arrow[r, shift left, "\alpha=u"] & u(L) \arrow[l, shift left, "\gamma=i"] \arrow[loop below, "f_{|u(L)}"] \end{tikzcd}$$

Again, for all $l \in u(L)$, we have $f \circ u(\gamma(l)) = f(u(l)) = f(l) = \gamma(f_{|u(L)}(l))$, i.e., γ satisfies soundness (9) and completeness (10) in Lemma A.1. Therefore, $\nu(f \circ u) = \gamma(\nu f_{|u(L)}) = \nu f_{|u(L)}$, as desired.

Finally, if u is continuous and strict then also $\gamma = i$ is so: First, since $\bot = u(\bot) \in u(L)$ and hence the inclusion i maps $\bot$ to $\bot$. Second, since u is continuous, directed suprema in both lattices coincide: let $D \subseteq u(L)$, then $\bigsqcup D = \bigsqcup\{u(d) \mid d \in D\} = u(\bigsqcup D) \in u(L)$. Hence i preserves directed suprema.
Hence we get the previous results also for least fixpoints.

□

Lemma 5.5 (properties of $\bar{u}$). *Let $u : L \to L$ be a monotone function. Then*

a) $\bar{u}$ is the least closure larger than u;
b) if u is f-compatible then $\bar{u}$ is;
c) if u is continuous and strict then $\bar{u}$ is.

Proof. a) We first observe that $\bar{u}$ is a closure. For extensiveness, just observe that $\hat{u}_x(y) = u(y) \sqcup x \sqsupseteq x$ for all $y \in L$ and thus obviously $\bar{u}(x) = \mu(\hat{u}_x) \sqsupseteq x$.
In order to show that $\bar{u}$ is idempotent, note that, by extensiveness, $\bar{u} \sqsubseteq \bar{u} \circ \bar{u}$. Hence to conclude, we just need to prove the converse inequality $\bar{u} \circ \bar{u} \sqsubseteq \bar{u}$. For all $x \in L$, we have $\bar{u}(\bar{u}(x)) = \mu(\hat{u}_{\bar{u}(x)}) = \hat{u}^{\gamma}_{\bar{u}(x)}$ for some ordinal γ. We prove, by transfinite induction that for all α, that $\hat{u}^{\alpha}_{\bar{u}(x)} \sqsubseteq \bar{u}(x)$.

($\alpha = 0$) We have that $\hat{u}^{0}_{\bar{u}(x)} = \bot \sqsubseteq \bar{u}(x)$.

($\alpha \to \alpha + 1$) We have that

$$\begin{aligned}
\hat{u}^{\alpha+1}_{\bar{u}(x)} &= \hat{u}_{\bar{u}(x)}(\hat{u}^{\alpha}_{\bar{u}(x)}) \\
&= u(\hat{u}^{\alpha}_{\bar{u}(x)}) \sqcup \bar{u}(x) && [\text{by def. } \hat{u}_{\bar{u}(x)}\ ] \\
&\sqsubseteq u(\bar{u}(x)) \sqcup \bar{u}(x) && [\text{by ind. hyp.}] \\
&\sqsubseteq \hat{u}_x(\bar{u}(x)) \sqcup \bar{u}(x) && [\text{since } u \sqsubseteq \hat{u}_x] \\
&= \bar{u}(x) \sqcup \bar{u}(x) && [\text{since } \hat{u}_x(\bar{u}(x)) = \bar{u}(x)] \\
&= \bar{u}(x)
\end{aligned}$$

(α limit) We have that

$$\begin{aligned}
\hat{u}^{\alpha}_{\bar{u}(x)} &= \bigsqcup_{\beta<\alpha} \hat{u}^{\beta}_{\bar{u}(x)} \\
&\sqsubseteq \bigsqcup_{\beta<\alpha} \bar{u}(x) && [\text{by ind. hyp.}] \\
&= \bar{u}(x)
\end{aligned}$$

Moreover, $\bar{u}$ is larger than u, i.e., $u \sqsubseteq \bar{u}$. In fact,

$$\begin{aligned}
\bar{u}(x) &= \hat{u}_x(\bar{u}(x)) && [\text{since } \bar{u}(x) \text{ is a fixpoint of } \hat{u}_x]\\
&= u(\bar{u}(x)) \sqcup x && [\text{by def. of } \hat{u}_x]\\
&\sqsupseteq u(x) \sqcup x && [\text{since } \bar{u} \text{ is extensive}]\\
&\sqsupseteq u(x)
\end{aligned}$$

Finally, let v any closure such that $u \sqsubseteq v$. We show that for all $x \in L$, $\hat{u}_x^\alpha \sqsubseteq v(x)$, whence $\bar{u}(x) \sqsubseteq v(x)$, as desired.
($\alpha = 0$) We have that $\hat{u}^0_{\bar{u}(x)} = \bot \sqsubseteq v(x)$.

($\alpha \to \alpha + 1$) We have that

$$\begin{aligned}
\hat{u}^{\alpha+1}_{\bar{u}(x)} &= \hat{u}_x(\hat{u}_x^\alpha)\\
&= u(\hat{u}_x^\alpha) \sqcup x && [\text{by def. } \hat{u}_x\ ]\\
&\sqsubseteq u(v(x)) \sqcup x && [\text{by ind. hyp.}]\\
&\sqsubseteq v(v(x)) \sqcup x && [\text{since } u \sqsubseteq v]\\
&= v(x) \sqcup x && [\text{by idempotency of } v]\\
&= v(x) && [\text{by extensiveness of } v]
\end{aligned}$$

(α limit) We have that

$$\begin{aligned}
\hat{u}_x^\alpha &= \bigsqcup_{\beta<\alpha} \hat{u}_x^\beta\\
&\sqsubseteq \bigsqcup_{\beta<\alpha} v(x) && [\text{by ind. hyp.}]\\
&= v(x)
\end{aligned}$$

b) Observe that for all $x \in L$, we have $\bar{u}(f(x)) = \hat{u}^\gamma_{f(x)}$ for some ordinal γ. Hence also here we proceed by transfinite induction, showing that for all α

$$\hat{u}^\alpha_{f(x)} \sqsubseteq f(\bar{u}(x))$$

($\alpha = 0$) We have that $\hat{u}^0_{f(x)} = \bot \sqsubseteq f(\bar{u}(x))$.

($\alpha \to \alpha + 1$) We have that

$$\begin{aligned}
\hat{u}^{\alpha+1}_{f(x)} &= \hat{u}_{f(x)}(\hat{u}^\alpha_{f(x)})\\
&\sqsubseteq \hat{u}_{f(x)}(f(\bar{u}(x)) && [\text{by ind. hyp.}]\\
&= u(f(\bar{u}(x))) \sqcup f(x) && [\text{by def. of } \hat{u}_{f(x)}]\\
&\sqsubseteq f(u(\bar{u}(x))) \sqcup f(x) && [\text{by compatibility of } f]\\
&\sqsubseteq f(u(\bar{u}(x)) \sqcup x) && [\text{by general properties of } \sqcup]\\
&= f(\hat{u}_x(\bar{u}(x)))) && [\text{by def. of } \hat{u}_x]\\
&= f(\bar{u}(x)) && [\text{since } \hat{u}(x) \text{ is a fixpoint}]
\end{aligned}$$

(α limit) We have that

$$\begin{aligned}
\hat{u}^\alpha_{f(x)} &= \bigsqcup_{\beta<\alpha} \hat{u}^\beta_{f(x)}\\
&\sqsubseteq \bigsqcup_{\beta<\alpha} f(\bar{u}(x)) && [\text{by ind. hyp.}]\\
&= f(\bar{u}(x))
\end{aligned}$$

c) Assume that u is continuous and strict. Then $\hat{u}_x$ is continuous for all $x \in L$. In fact, for each directed set $D \subseteq L$ we have

$$\begin{aligned}\hat{u}_x(\bigsqcup D) &= u(\bigsqcup D) \sqcup x\\ &= \bigsqcup \{u(d) \mid d \in D\}) \sqcup x\\ &= \bigsqcup \{u(d) \sqcup x \mid d \in D\})\\ &= \bigsqcup \{\hat{u}_x(d) \mid d \in D\})\end{aligned}$$

Now, we can show that $\bar{u}$ is continuous. Let $D \subseteq L$ be a directed set. We have to prove that $\bar{u}(\bigsqcup D) = \bigsqcup_{d\in D} \bar{u}(d)$. It is sufficient to prove that $\bar{u}(\bigsqcup D) \sqsubseteq \bigsqcup_{d\in D} \bar{u}(d)$, as the other inequality follows by monotonicity and general properties of $\bigsqcup$. As usual, we recall that $\bar{u}(\bigsqcup D) = \hat{u}^{\gamma}_{\bigsqcup D}$ for some γ and thus show, by transfinite induction on α that

$$\hat{u}^{\alpha}_{\bigsqcup D} \sqsubseteq \bigsqcup_{d\in D} \bar{u}(d).$$

($\alpha = 0$) We have that $\hat{u}^0_{\bigsqcup D} = \bot \sqsubseteq \bigsqcup_{d\in D} \bar{u}(d)$.

($\alpha \to \alpha + 1$) We have that

$$\begin{aligned}\hat{u}^{\alpha+1}_{\bigsqcup D} &= \hat{u}_{\bigsqcup D}(\hat{u}^{\alpha}_{\bigsqcup D})\\ &\sqsubseteq \hat{u}_{\bigsqcup D}(\bigsqcup_{d\in D} \bar{u}(d)) && \text{[by ind. hyp.]}\\ &= \bigsqcup_{d\in D} \hat{u}_{\bigsqcup D}(\bar{u}(d)) && \text{[by continuity of } \hat{u}_{\bigsqcup D}]\\ &= \bigsqcup_{d\in D} (u(\bar{u}(d)) \sqcup \bigsqcup D) && \text{[by def. of } \hat{u}_{\bigsqcup D}]\\ &\sqsubseteq \bigsqcup_{d\in D} (\hat{u}_d(\bar{u}(d)) \sqcup \bigsqcup D) && \text{[since } u \sqsubseteq \hat{u}_d]\\ &= \bigsqcup_{d\in D} (\bar{u}(d) \sqcup \bigsqcup D) && \text{[since } \hat{u}(d) \text{ is a fixpoint]}\\ &= \bigsqcup_{d\in D} (\bar{u}(d) \sqcup d)\\ &= \bigsqcup_{d\in D} \bar{u}(d) && \text{[by extensiveness of } \bar{u}]\end{aligned}$$

(α limit) We have that

$$\begin{aligned}\hat{u}^{\alpha}_{\bigsqcup D} &= \bigsqcup_{\beta<\alpha} \hat{u}^{\beta}_{\bigsqcup D}\\ &\sqsubseteq \bigsqcup_{\beta<\alpha} \bigsqcup_{d\in D} \bar{u}(d) && \text{[by ind. hyp.]}\\ &= \bigsqcup_{d\in D} \bar{u}(d)\end{aligned}$$

Furthermore, $\bar{u}$ is strict since $\hat{u}_{\bot}(\bot) = u(\bot) \sqcup \bot = \bot \sqcup \bot = \bot$, and thus $\bar{u}(\bot) = \mu(\hat{u}_{\bot}) = \bot$. ◻

Theorem 5.8 (up-to for systems)**.** *Let $(L, \sqsubseteq)$ be a complete lattice and let E be $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$, a system of m equations over L, with solution $\boldsymbol{s} \in L^m$. Let $\boldsymbol{u}$ be a compatible tuple of up-to functions for E. Let $\boldsymbol{s}'$ and $\bar{\boldsymbol{s}}$ be the solutions of the systems $E\boldsymbol{u}$ with equations $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{u}^{\times}(\boldsymbol{x}))$ and $E\bar{\boldsymbol{u}}$ with equations $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\bar{\boldsymbol{u}}^{\times}(\boldsymbol{x}))$, respectively. Then $\boldsymbol{s}' \sqsubseteq \bar{\boldsymbol{s}} = \boldsymbol{s}$. If additionally $\boldsymbol{u}$ is extensive then $\boldsymbol{s}' = \boldsymbol{s}$.*

Proof. Immediate extension to systems of the proofs of the Lemma 5.3 and Corollary 5.6, exploiting Theorem 4.1. ◻

Corollary 5.9. *Let E be $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$, a system of m equations over a complete lattice L and let $\boldsymbol{u}$ be a compatible tuple of extensive up-to functions for E.*

For all $b \in B_L$, if $b \sqsubseteq u_i(\bigsqcup B')$ for $B' \subseteq B_L$ where all $b' \in B'$ satisfy $b' \sqsubseteq sol_i(E)$ then $b \sqsubseteq sol_i(E)$.

Proof. This is a straightforward corollary of Theorem 5.8: According to the theorem, the solutions of the systems E and $E\boldsymbol{u}$ coincide and we denote such solution by $\boldsymbol{s}$.

We can now prove that $u_i(s_i) \sqsubseteq s_i$ for all $i \in \underline{m}$:

$$u_i(s_i) = u_i(f_i(s_1, \dots, s_m)) \sqsubseteq f_1(u_1(s_1), \dots, u_m(s_m)) = s_i.$$

Here we use the fact that the solution is a fixpoint of $\boldsymbol{f}$ respectively $\boldsymbol{f} \circ \boldsymbol{u}$ and furthermore we exploit that $\boldsymbol{u}$ is a compatible tuple of up-to functions for E (in order to derive the inequality).

Since $b' \sqsubseteq sol_i(E) = s_i$ for all $b' \in B'$, we obtain $\bigsqcup B' \sqsubseteq s_i$. Furthermore $b \sqsubseteq u_i(\bigsqcup B') \sqsubseteq u_i(s_i) \sqsubseteq s_i$, which concludes the proof. ◻

C Proofs for Section 6 (Solving systems of equations via games)

Theorem 6.3 (correctness and completeness)**.** *Let E be a system of m 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. Define $\langle \alpha, \gamma \rangle : \mathbf{2}^{B_L} \to L$, by letting $\alpha(X) = \bigsqcup X$ for $X \in \mathbf{2}^{B_L}$ and $\gamma(l) = \downarrow l \cap B_L$ for $l \in L$. It is immediate to see that this is a Galois insertion: for all $X \in \mathbf{2}^{B_L}$ we have $X \subseteq \gamma(\alpha(X)) = (\downarrow \bigsqcup X) \cap B_L$ and, for $l \in L$ we have $l = \alpha(\gamma(l)) = \bigsqcup(\downarrow l \cap B_L)$.

Below we abuse the notation and write $\downarrow$ and $\bigsqcup$ for the m-tuples where each function is $\downarrow$ and $\bigsqcup$ applied componentwise, respectively.

$$\begin{tikzcd}
((\mathbf{2}^{B_L})^m, \subseteq) \arrow[loop below, "f^C = \downarrow f \bigsqcup"] \arrow[r, shift right, "\boldsymbol{\alpha} = \bigsqcup _"'] & L^m \arrow[l, shift right, "\boldsymbol{\gamma} = \downarrow _ \cap B_L"'] \arrow[loop below, "f"]
\end{tikzcd}$$

Define a "concrete" system $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^C(\boldsymbol{x})$ where $\boldsymbol{f}^C = \boldsymbol{\gamma}^\times \circ \boldsymbol{f} \circ \boldsymbol{\alpha}^\times : (\mathbf{2}^{B_L})^m \to (\mathbf{2}^{B_L})^m$. Then we can use Lemma A.5 to deduce that, if we denote by $\boldsymbol{S}^C$ the solution of the concrete system and by $\boldsymbol{s}$ the solution of the original system, we have $\boldsymbol{S}^C = \downarrow \boldsymbol{s} \cap B_L^m$.

Now, $(\mathbf{2}^{B_L}, \subseteq)$ is an algebraic, hence continuous lattice. Therefore, by [Baldan et al. 2019, Theorem 4.8], the lattice game for the "concrete" system on $(\mathbf{2}^{B_L})^m$ is sound and complete.

It is immediate to realise that, if we fix as basis for $\mathbf{2}^{B_L}$ the set of singletons, this corresponds exactly to what we called here the powerset game. In fact, the game aims to show that $\{b\} \subseteq S_i^C = \downarrow s_i$, for some $b \in B_L$ and $i \in \underline{m}$, and this amounts to $b \sqsubseteq s_i$. Positions of $\exists$ are pairs $(\{b\}, i)$ where $b \in B_L$ and $i \in \underline{m}$, and she has to play some tuples $\boldsymbol{X} \in (\mathbf{2}^{B_L})^m$ such that $\{b\} \subseteq f_i^C(\boldsymbol{X}) = \downarrow f_C(\bigsqcup \boldsymbol{X})$ which amounts to $b \sqsubseteq f_C(\bigsqcup \boldsymbol{X})$. Positions of $\forall$ are tuples $\boldsymbol{X} \in (\mathbf{2}^{B_L})^m$ and he chooses some $j \in \underline{m}$ and $b' \in X_j$. This is exactly the powerset game, hence we conclude. ◻

Theorem 6.5 (game with selections)**.** *Let $x =_\nu f(x)$ be an equation over a complete lattice L with solution s. For all $b \in B_L$, it holds that $b \sqsubseteq s$ iff $\exists$ has a winning strategy from position b in the game restricted to selections.*

Proof. Assume that $\exists$ has a winning strategy in the original game: given b she would play X, where all $b' \in \mathbf{A}(X)$ are winning positions.

Instead, in the game restricted by selections, she might only be able to play Y where $\bigsqcup Y \sqsubseteq \bigsqcup X$. Now $\forall$ picks $b' \in Y$. By construction $b' \sqsubseteq \bigsqcup X$. Since all elements of X are winning positions in the original game (and hence below the solution), b' is also a winning position and we can continue. Now either $\exists$ wins directly or the game continues forever, giving us a winning strategy in the restricted game. ◻

Theorem 6.6 (Case 1: correctness and completeness)**.** *Let f be a monotone function over a complete lattice L, with basis $B_L = L \setminus \{\bot\}$, such that f preserves non-empty meets. Given the Galois connection $\langle f_*, f^* \rangle : L \to L$ such that $f(x) = f^*(x) \sqcap f(\top)$, then, for all $b \in B_L$, $b \sqsubseteq \nu f$ iff $\exists$ wins the game of Case 1 from position b.*

Proof. First we prove that if $b \sqsubseteq \nu f$, then $\exists$ wins the simplified game of Case 1. Observe that by monotonicity of f, we have that $b \sqsubseteq \nu f = f(\nu f) \sqsubseteq f(\top)$, so $\exists$ does not immediately lose. Moreover, consider $b' = f_*(b)$. Since $b \sqsubseteq \nu f$, by monotonicity of f_* we have $b' \sqsubseteq f_*(\nu f) = f_*(f(\nu f)) = f_*(f^*(\nu f) \sqcap f(\top)) \sqsubseteq f_*(f^*(\nu f)) \sqsubseteq \nu f$ because of the properties of the Galois connection. Since $b' \sqsubseteq \nu f$, the same argument as before holds for b' as well, and thus the game would either continue forever or terminate because at some point $b_i \sqsubseteq \bigsqcup_{j<i} b_j$. In both cases $\exists$ wins.

Now we prove that if $\exists$ wins starting from $b_0 = b$, then $b \sqsubseteq \nu f$. Actually, we show that if $\exists$ wins the simplified game in Case 1, then she wins also the general fixpoint game for the single fixpoint equation $x =_\nu f(x)$, and so by Theorem 6.3 we know that $b \sqsubseteq \nu f$. Since $\exists$ wins, we have two possible cases:

- the game continues forever, thus for all i, $b_i \sqsubseteq f(\top)$ and $b_{i+1} = f_*(b_i)$. Then, for all i, observe that by the Galois connection we have $b_i = b_i \sqcap f(\top) \sqsubseteq f^*(f_*(b_i)) \sqcap f(\top) = f^*(b_{i+1}) \sqcap f(\top) = f(b_{i+1}) = f(\bigsqcup\{b_{i+1}\})$. This means that, for all i, $\{b_{i+1}\} \in \mathbf{E}(b_i)$ is a valid move for player $\exists$ from position b_i in the fixpoint game. Therefore, the infinite sequence $(b_0, \{b_1\}, b_1, \{b_2\}, \ldots)$ is an infinite play in the fixpoint game, which is won by $\exists$ since there is a single greatest fixpoint equation. Since the moves of $\forall$ are all deterministic $\exists$ wins the fixpoint game.
- the game terminates because at some point it holds $b_i \sqsubseteq \bigsqcup_{j<i} b_j$ and $b_i \sqsubseteq f(\top)$. Note that since the game reached the position b_i, for all $j < i$ we must have $b_j \sqsubseteq f(\top)$. We then have two more cases. If $b_i = f_*(b_{i-1}) = \bot$, then for the same reasoning in the previous case it holds $b_{i-1} \sqsubseteq f(\bot) = f(\bigsqcup \emptyset)$, thus $\emptyset \in \mathbf{E}(b_{i-1})$, and the sequence $(b_0, \{b_1\}, b_1, \{b_2\}, \ldots, b_{i-1}, \emptyset)$ is a finite play in the fixpoint game leading to the position $\emptyset$ where player $\forall$ cannot move, hence $\exists$ wins. Otherwise $b_i \neq \bot$, and so we have that $b_{i+1} = f_*(b_i) \sqsubseteq f_*(\bigsqcup_{j<i} b_j) = \bigsqcup_{j<i} f_*(b_j) = \bigsqcup_{j<i} b_{j+1} \sqsubseteq f(\top)$ since f_*, as a left adjoint, preserves non-empty joins. Moreover, for the same reasoning of before we have that $b_i \sqsubseteq f(b_{i+1}) = f(\bigsqcup\{b_{i+1}\})$, hence $\{b_{i+1}\} \in \mathbf{E}(b_i)$. An inductive argument thus proves that iterating f_* we will never go beyond $f(\top)$, and so there exists an infinite sequence $(b_0, \{b_1\}, b_1, \{b_2\}, \ldots)$ such that for all j, $\{b_{j+1}\} \in \mathbf{E}(b_j)$. Then this is an infinite play of the fixpoint game won by $\exists$. Since in both cases the moves of player $\forall$ are all deterministic $\exists$ wins the fixpoint game.

□

Theorem 6.7 (Case 2: correctness and completeness)**.** *Let f be a monotone function over a complete lattice L, with basis $B_L \subseteq L \smallsetminus \{\bot\}$, such that f preserves non-empty meets. Given the Galois connection $\langle f_*, f^* \rangle : L \to L$ such that $f(x) = f^*(x) \sqcap f(\top)$, then, for all $b \in B_L$, $b \sqsubseteq \nu f$ iff $\exists$ wins the game of Case 2 from position b.*

Proof. First we prove that if $b \sqsubseteq \nu f$, then $\exists$ wins the simplified game of Case 2. Observe that by monotonicity of f, we have that $b \sqsubseteq \nu f = f(\nu f) \sqsubseteq f(\top)$, so $\exists$ does not immediately lose. Moreover, let $X \subseteq B_L$ such that $X \equiv_H \downarrow f_*(b) \cap B_L$, hence $\bigsqcup X = f_*(b)$. Since $b \sqsubseteq \nu f$, by monotonicity of f_* we have $\bigsqcup X = f_*(b) \sqsubseteq f_*(\nu f) = f_*(f(\nu f)) = f_*(f^*(\nu f) \sqcap f(\top)) \sqsubseteq f_*(f^*(\nu f)) \sqsubseteq \nu f$ because of the properties of the Galois connection. Since $\bigsqcup X \sqsubseteq \nu f$ we must have that $b' \sqsubseteq \nu f$ for all $b' \in X$, therefore the same argument as before holds on all $b' \in X$ as well, and so no position losing for $\exists$ can ever be reached, hence $\exists$ wins.

Now we prove that if $\exists$ wins starting from $b_0 = b$, then $b \sqsubseteq \nu f$. Actually, we show that if $\exists$ wins the simplified game in Case 2 then she wins also the general fixpoint game for the single fixpoint equation $x =_\nu f(x)$, and so by Theorem 6.3 we know that $b \sqsubseteq \nu f$. Since $\exists$ wins, for every path $(b_0, b_1, \ldots)$ in the tree of positions explored we have three possible cases:

- the path is infinite, thus for all i, $b_i \sqsubseteq f(\top)$ and $b_{i+1} \in X_i$ for some $X_i \subseteq B_L$ such that $X_i \equiv_H \downarrow f_*(b_i) \cap B_L$, hence $\bigsqcup X_i = f_*(b_i)$. Then, for all i, observe that by the Galois connection we have $b_i = b_i \sqcap f(\top) \sqsubseteq f^*(f_*(b_i)) \sqcap f(\top) = f^*(\bigsqcup X_i) \sqcap f(\top) = f(\bigsqcup X_i)$. This means that, for all i, $X_i \in \mathbf{E}(b_i)$ is a valid move for player $\exists$ from position b_i in the fixpoint game. Furthermore, $b_{i+1} \in \mathbf{A}(X_i)$ is a valid move for player $\forall$ in the fixpoint game. Therefore, the infinite sequence $(b_0, X_0, b_1, X_1, \ldots)$ is an infinite play in the fixpoint game, which is won by $\exists$ since there is a single greatest fixpoint equation.
- the path is finite and the exploration has been stopped because at some point $b_i \sqsubseteq f(\top)$ and $f_*(b_i) = \bot$ thus the only possible $X_i \subseteq B_L$ such that $X_i \equiv_H \downarrow \bot \cap B_L$ is $X_i = \emptyset$. Similarly to before, for all $j < i$, we have $b_{j+1} \in X_j$ for some $X_j \subseteq B_L$ such that $X_j \equiv_H \downarrow f_*(b_j) \cap B_L$. Note that since the game reached the position b_i, for all $j < i$ we must have $b_j \sqsubseteq f(\top)$. For the same reasons in the previous case, $b_i \sqsubseteq f(\bigsqcup X_i) = f(\bigsqcup \emptyset)$, thus $\emptyset \in \mathbf{E}(b_i)$, and the sequence $(b_0, X_0, b_1, X_1, \ldots, b_i, \emptyset)$ is a finite play in the fixpoint game leading to the position $\emptyset$ where player $\forall$ cannot move, hence $\exists$ wins.
- the path is finite and the exploration has been stopped because at some point it holds $b_i \sqsubseteq \bigsqcup W$ and $b_i \sqsubseteq f(\top)$. Again, for all $j < i$, we have $b_{j+1} \in X_j$ for some $X_j \subseteq B_L$ such that $X_j \equiv_H \downarrow f_*(b_j) \cap B_L$. Observe that since W is the set of positions previously encountered, it contains every position previously explored, thus not losing for $\exists$, including all b_j for $j < i$. Then, for all $b' \in W$ we must have $b' \sqsubseteq f(\top)$. Furthermore, note that positions are put in W only when all their successors are going to be explored. Therefore, for all $b' \in W$, we have $f_*(b') \sqsubseteq f(\top)$, otherwise there would exists a successor $b'' \in X' \equiv_H \downarrow f_*(b') \cap B_L$ such that $b'' \not\sqsubseteq f(\top)$ contradicting the fact that $\exists$ wins the simplified game. Let $X_i \subseteq B_L$ such that $X_i \equiv_H \downarrow f_*(b_i) \cap B_L$. Then, we have that $\bigsqcup X_i = f_*(b_i) \sqsubseteq f_*(\bigsqcup W) = \bigsqcup_{b' \in W} f_*(b') \sqsubseteq f(\top)$ since $b_i \sqsubseteq \bigsqcup W$, f_* as a left adjoint preserves non-empty joins and $f_*(b') \sqsubseteq f(\top)$ for all $b' \in W$. Then, for all $b' \in X_i$, this implies that $b' \sqsubseteq f(\top)$. Moreover, for the same reasoning used in the first case we have that $b_i \sqsubseteq f(\bigsqcup X_i)$, hence $X_i \in \mathbf{E}(b_i)$. An inductive argument thus proves that every path continuing the exploration from a $b' \in X_i$ we will never

go beyond $f(\top)$, and so for each of those paths there exists an infinite sequence $(b_0, X_0, b_1, X_1, \ldots)$ such that for all j, $X_j \in \mathbf{E}(b_j)$. Then this is an infinite play of the fixpoint game won by $\exists$.

Since all the possible moves of player $\forall$ in every set X are explored, and all the paths obtained in this way (divided in the three cases above) correspond to plays in the fixpoint game won by $\exists$, we can conclude that, indeed, $\exists$ wins the fixpoint game. □

D Proofs for Section 7 (On-the-fly algorithm for solving the game in the general case)

Definition D.1 (sound forget). Whenever function Forget(Δ_P, Γ_P, $(C, \boldsymbol{k})$) is invoked, returning Δ'_P, for every decision $(C', \boldsymbol{k}') \in \Delta'_P$, for every position C'' justifying that decision, there exists $(C'', \boldsymbol{k}'') \in \Delta'_P$ such that $\boldsymbol{k}'' \leq_P next(\boldsymbol{k}', \mathrm{i}(C'))$ or there exists $(C'', \boldsymbol{k}'') \in \Gamma_P \smallsetminus \{(C, \boldsymbol{k})\}$ such that $\boldsymbol{k}'' <_P next(\boldsymbol{k}', \mathrm{i}(C'))$.

Lemma D.2 (assumptions and plays). *Given a fixpoint game, whenever functions* Explore*$(\cdot, \cdot, \rho, \Gamma, \Delta)$ and* Backtrack*$(\cdot, \cdot, \rho, \Gamma, \Delta)$ are invoked, for every player P, for all $(C, \boldsymbol{k}) \in \Gamma_P$ it holds $(C, \boldsymbol{k}, \pi) \in \rho$ for some π.*

Proof. Easily proved by an inspection of the code. Initially, on the call Explore(C_0, $\mathbf{0}$, [], $(\emptyset, \emptyset)$, $(\emptyset, \emptyset)$), the property vacuously holds since both $\Gamma_\exists$ and $\Gamma_\forall$ are empty. Now, the only way that could make the property fail are by adding new assumptions or backtracking, hence shortening the playlist ρ. The only position in the code where new assumptions are added is in the function Explore. A new assumption $(C, \boldsymbol{k}')$ is added only if $(C, \boldsymbol{k}', \pi) \in \rho$, for some π, thus the property still holds. On the other hand, the only place where the backtracking really happens, that is, ρ is effectively shorten, is at the end of the backtracking function, when Backtrack(P, C', t, Γ, Δ) is invoked. More precisely, the head $(C', \boldsymbol{k}', \pi)$ is removed from the playlist ρ. However, before the aforementioned invocation, $(C', \boldsymbol{k}')$ was already removed from Γ_P and from $\Gamma_{\overline{P}}$, if it were in $\Gamma_{\overline{P}}$. And so again the property still holds. □

Lemma 7.2 (termination). *Given a fixpoint game on a finite lattice, any call* Explore*(C_0, $\mathbf{0}$, [], $(\emptyset, \emptyset)$, $(\emptyset, \emptyset)$) terminates, hence at some point* Backtrack*(P, C_0, [], $(\emptyset, \emptyset)$, Δ) is invoked, for some player P and pairs of sets Γ and Δ.*

Proof. Consider the sequence σ of invocations to functions Explore and Backtrack in the order they happen, originating from a call Explore(C_0, $\mathbf{0}$, [], $(\emptyset, \emptyset)$, $(\emptyset, \emptyset)$). Let τ be the subsequence of σ obtained removing all calls to Backtrack. We show that such sequence is finite. First, since the lattice is finite, hence *Pos* is finite, the set of playlists ρ in the invocations in τ is also finite. Actually, this is not true in general for any set of playlists, but it holds for the set of lists we obtain during any computation. Indeed, this can be seen inductively, showing that every playlist ρ has length bounded by $|Pos|$. At the beginning we have the empty list [] which is clearly bounded by $|Pos|$. Then, by inspecting the code it can be seen that the only function which increases the size of ρ is Explore, and it happens only if the current position C, with counter $\boldsymbol{k}$, is not already contained in ρ with a counter $\boldsymbol{k}'$ s.t. $\boldsymbol{k}' <_P \boldsymbol{k}$ for some player P. But whenever a position C already in ρ is encountered again it must be with a counter strictly larger for one of the players. The only case where this could possibly fail is when the subsequence of ρ between the two occurrences of C contains only positions with priority 0. But, as already mentioned, this cannot happen because players alternate during the game and only $\forall$ has positions with priority 0. Thus, every time a position recurs, the playlist is not extended any more. So, the size of the playlist is necessarily bounded by the size of *Pos*. Furthermore, the set of playlists of length bounded by $|Pos|$ is finite because every π in them is bounded as well, since $\pi \subseteq Pos$, and the same happens for the counters $\boldsymbol{k}$ since they are computed starting from $\mathbf{0}$ and increased at most by 1 in some component only when the list is extended. Therefore, τ must contain only a finite number of different playlists ρ, possibly with repetitions. Now, in order to show that τ is finite, we define a partial order $\leq$ over the playlists in τ as follows, $\forall \rho, \rho', \rho'', C, \boldsymbol{k}, \pi, \pi'$:

- $\rho'\rho \leq \rho$
- if $\pi \subsetneq \pi'$, then $\rho''((C, \boldsymbol{k}, \pi) :: \rho) \leq \rho'((C, \boldsymbol{k}, \pi') :: \rho)$.

It is easy to see that such order is reflexive, antisymmetric, and transitive. Since the set of playlists in τ is finite, so is the corresponding poset with the given partial order. By an inspection of the code it can be seen that for every two playlists ρ, ρ' in consecutive invocations of Explore in τ, we have that $\rho' < \rho$, since:

- function Explore extends the playlist ρ until function Backtrack is invoked
- function Backtrack shortens the playlist ρ until it is empty or function Explore is invoked, after shortening the set of unexplored moves π in ρ.

So the playlists in τ form a strictly descending chain in a finite poset, thus τ must be finite. And this immediately proves that σ is finite as well, because otherwise from a certain point on we would have infinitely many calls to Backtrack only, which would shorten the playlist infinitely many times. And so we can conclude that any computation originating from a call Explore(C_0, $\mathbf{0}$, [], $(\emptyset, \emptyset)$, $(\emptyset, \emptyset)$) must terminate. Finally, since the only instruction returning a value (hence terminating the execution) is in the function Backtrack and it is reached only when $\rho = []$, then Backtrack(P, C, [], Γ, Δ) must have been invoked on some P, C, Γ, Δ. Furthermore, $C = C_0$ because $\rho = []$ is the list of positions from the root C_0 to the current node C.

We immediately conclude that $\Gamma = (\emptyset, \emptyset)$ by exploting Lemma D.2. ◻

Lemma D.3 (backtracking position). *Given a fixpoint game, whenever function* Backtrack*(P, C, ρ, Γ, Δ) is invoked, it holds $(C, \boldsymbol{k}) \in \Delta_P \cup \Gamma_P$ for some $\boldsymbol{k}$.*

Proof. Immediate by inspecting the invocations of Backtrack in the code. ◻

Lemma D.4 (uncontrolled decisions). *Given a fixpoint game, whenever functions* Explore*($\cdot$, $\cdot$, $\cdot$, Γ, Δ) and* Backtrack*($\cdot$, $\cdot$, $\cdot$, Γ, Δ) are invoked, for every player P, for all $(C, \boldsymbol{k}) \in \Delta_P$, if $\mathsf{P}(C) \neq P$, then for all $C' \in \mathsf{M}(C)$ it holds $(C', \boldsymbol{k}') \in \Delta_P \cup \Gamma_P$ for some $\boldsymbol{k}'$.*

Proof. By inspecting the code it is easy to see that every time we add a new decision $(C, \boldsymbol{k})$ for a player P that is not the owner of C, either:

- $\mathsf{M}(C) = \emptyset$, thus the property vacuously holds, or
- the procedure already explored all possible moves $\mathsf{M}(C)$ and they all became decisions or assumptions for P, since we are in the case where $\mathsf{P}(C) \neq P$ and $\pi = \emptyset$.

Furthermore, such a decision $(C, \boldsymbol{k})$ is justified by $\mathsf{M}(C)$. Therefore, if one of those moves were to be deleted from the assumptions or decisions of P at some point, the function Forget would delete $(C, \boldsymbol{k})$ as well. ◻

For the next results we make use of fixpoint games suitably modified for a set of assumptions for a player. For a set S of decisions or assumptions we denote by $\mathsf{C}(S)$ its first projection, that is, the set of positions appearing as first component in the elements of S.

Definition D.5 (game with assumptions). Given a fixpoint game G and a player P, the corresponding *game with assumptions* Γ_P is a parity game $G(\Gamma_P)$ obtained from G where for all $C \in Pos$, if $C \in \mathsf{C}(\Gamma_P)$, then $\mathsf{P}(C) = \overline{P}$ and $\mathsf{M}(C) = \emptyset$, otherwise they are the same as in G.

Notice that when the set of assumptions is empty $\Gamma_P = \emptyset$, the modified game is the same of the original one.

Then, we define a kind of strategies based on decisions and assumptions for a player, which fit the modified games above. Such strategies are history-free partial strategies. Indeed they only prescribe moves from decisions.

Definition D.6 (strategy with assumptions). Let G be a fixpoint game. Given a player P, a *strategy with assumptions* Γ_P *from decisions* Δ_P for P is a function $s_P : \mathsf{C}(\Delta_P \cup \Gamma_P) \to 2^{\mathsf{C}(\Delta_P \cup \Gamma_P)}$ where for all $C \in \mathsf{C}(\Gamma_P)$, $s_P(C) = \emptyset$, and for all $C \in \mathsf{C}(\Delta_P) \setminus \mathsf{C}(\Gamma_P)$, $s_P(C)$ is the set of positions, possibly empty, justifying the decision $(C, \min_{\leq_P}\{\boldsymbol{k} \mid (C, \boldsymbol{k}) \in \Delta_P\})$. Given a position $C \in \mathsf{C}(\Delta_P)$, we denote by $d_P(C) = \min_{\leq_P}\{\boldsymbol{k} \mid (C, \boldsymbol{k}) \in \Delta_P\}$ the counter that was associated with C.

We say that the strategy s_P is *winning* when it is winning in the modified game $G(\Gamma_P)$, that is, every play in $G(\Gamma_P)$ following s_P starting from a position in $\mathsf{C}(\Delta_P)$ is won by player P.

The definition above is well given since by Lemmata D.3 and D.4 we know that when we add a new decision justified by some other, those are already included in the decisions or assumptions for the same player. Moreover, notice that the minimum of $\{\boldsymbol{k} \mid (C, \boldsymbol{k}) \in \Delta_P\}$ is guaranteed to be in the set itself because $\leq_P$ is a total order and the set is never empty since $C \in \mathsf{C}(\Delta_P)$.

In the modified game $G(\Gamma_P)$, given the strategy s_P with assumptions Γ_P from decisions Δ_P, for each position $C \in \mathsf{C}(\Delta_P)$ we can build a tree including all the plays starting from C where player P follows the strategy s_P.

Definition D.7 (tree of plays). Let G be a fixpoint game. Given a player P and the strategy s_P with assumptions Γ_P from decisions Δ_P, for each position $C \in \mathsf{C}(\Delta_P)$, the *tree of the plays following s_P starting from C* is the tree $\tau^C_{s_P}$ rooted in C, where every node C' in it has successors $s_P(C')$.

Such trees can contain both finite and infinite paths. Finite complete paths terminate in assumptions or truths, infinite ones contain only decisions. By construction and definition of strategy with assumptions every node is either a decision or an assumption for P. More precisely, every inner node is a position in $\mathsf{C}(\Delta_P)$, and every leaf corresponds to either a truth in Δ_P or an assumption in Γ_P. It is easy to see that a tree $\tau^C_{s_P}$ includes all the possible plays from C following s_P since the successors of inner nodes owned by the opponent are all the possible moves from those positions (decisions controlled by the opponent are justified by all the possible opponent's moves, Lemma D.4).

The trees defined above are all we need to show that a strategy with assumptions is winning. Indeed, it is enough to show that every complete path in each of those trees corresponds to a play won by the player. To this end, first we observe some key properties of the paths in the trees.

Lemma D.8 (priorities in strategy paths). *Given a fixpoint game, whenever functions* EXPLORE$(\cdot, \cdot, \cdot, \Gamma, \Delta)$ *and* BACKTRACK$(\cdot, \cdot, \cdot, \Gamma, \Delta)$ *are invoked, for every player P, given the strategy s_P with assumptions Γ_P from decisions Δ_P, for all $\hat{C} \in \mathsf{C}(\Delta_P)$, the tree of plays $\tau^{\hat{C}}_{s_P}$ satisfies the following properties*

a) for every pair of inner nodes C, C' in $\tau^{\hat{C}}_{s_P}$ s.t. C' is a successor of C, it holds $d_P(C') \leq_P \mathit{next}(d_P(C), \mathsf{i}(C))$

b) for every non-empty inner path $C_1, \ldots, C_n$ in $\tau^{\hat{C}}_{s_P}$, if $d_P(C_1) <_P \mathit{next}(d_P(C_n), \mathsf{i}(C_n))$, then $P = \exists$ iff $\eta_h = \nu$, where h is the highest priority occurring along the path.

Proof. We prove the two properties separately.

a) Observe that we must have $C' \in s_P(C)$ by definition of $\tau^{\hat{C}}_{s_P}$. This means that there exists a decision $(C, d_P(C)) \in \Delta_P$ justified by the position C'. Then $(C, d_P(C))$ must have been added by a call to BACKTRACK. By inspecting the code it is easy to see that we were backtracking either after adding a new decision $(C', \mathit{next}(d_P(C), \mathsf{i}(C)))$ or because there was already a decision $(C', \boldsymbol{k}')$ s.t. $\boldsymbol{k}' \leq_P \mathit{next}(d_P(C), \mathsf{i}(C))$. Since $d_P(C') = \min_{\leq_P}\{\boldsymbol{k} \mid (C', \boldsymbol{k}) \in \Delta_P\}$, in both cases we can immediately conclude that $d_P(C') \leq_P \mathit{next}(d_P(C), \mathsf{i}(C))$.

b) We assume that $d_P(C_1) <_P \mathit{next}(d_P(C_n), \mathsf{i}(C_n))$ and $P = \exists$, and we prove that $\eta_h = \nu$, where h is the highest priority occurring along the path. A dual reasoning holds for $P = \forall$. Let next^j be a function that computes the counter after a subsequence of positions $C_1, \ldots, C_j$ in the path $C_1, \ldots, C_n$, for $j \in \underline{n}$. The function is inductively defined by $\mathit{next}^j(\boldsymbol{k}) = \mathit{next}(\mathit{next}^{j-1}(\boldsymbol{k}), \mathsf{i}(C_j))$ for all $j \in \underline{n}$, and $\mathit{next}^0(\boldsymbol{k}) = \boldsymbol{k}$. The inductive computation just repeatedly applies the function next for each position encountered along the sequence starting from a given counter $\boldsymbol{k}$. We observe that the function satisfies the property $d_\exists(C_j) \leq_\exists \mathit{next}^{j-1}(d_\exists(C_1))$ for all $j \in \underline{n}$. We show this by induction on j. Clearly it holds for $j = 1$, since by definition $\mathit{next}^0(d_\exists(C_1)) = d_\exists(C_1)$. Then, assuming it holds for j, we prove it for $j + 1$. Since we know that next is monotone wrt. the input counter, by inductive hypothesis we obtain that $\mathit{next}(d_\exists(C_j), \mathsf{i}(C_j)) \leq_P \mathit{next}(\mathit{next}^{j-1}(d_\exists(C_1)), \mathsf{i}(C_j)) = \mathit{next}^j(d_\exists(C_1))$, where the last equality holds by definition of next^j. Furthermore, we know that $d_\exists(C_{j+1}) \leq_\exists \mathit{next}(d_\exists(C_j), \mathsf{i}(C_j))$ by (a) above, since C_{j+1} is a successor of C_j. And so we can immediately deduce that indeed $d_\exists(C_{j+1}) \leq_\exists \mathit{next}^j(d_\exists(C_1))$. From this and the initial assumptions we have that $d_\exists(C_1) <_\exists \mathit{next}(d_\exists(C_n), \mathsf{i}(C_n)) \leq_\exists \mathit{next}^n(d_\exists(C_1))$, where the last inequality holds by definition of next^n and monotonicity of next. Observe that since next^n just recursively applies the function next on the positions $C_1, \ldots, C_n$, the final result and the initial counter $d_\exists(C_1)$ can only differ on priorities among those of the positions $C_1, \ldots, C_n$ and lower ones (which could have been zeroed). Therefore, the highest priority on which $d_\exists(C_1)$ and $\mathit{next}^n(d_\exists(C_1))$ do not coincide must be the highest priority h appearing along the path. Furthermore, we must have $d_\exists(C_1)_h < \mathit{next}^n(d_\exists(C_1))_h$, because values can only increase or become zero, when a higher priority is encountered (and its value increased), but this would contradict the fact that h is the highest. Now we can easily conclude since by hypothesis $d_\exists(C_1) <_\exists \mathit{next}^n(d_\exists(C_1))$, and so by definition of the order $<_\exists$ we must have that $\eta_h = \nu$.

□

We observe that winning strategies with assumptions are preserved by a sound function FORGET after removing an assumption and the related decisions.

Lemma D.9 (strategies and forget). *Given a fixpoint game, whenever* FORGET$(\Delta_P, \Gamma_P, (C, \boldsymbol{k}))$ *is invoked, returning Δ'_P, if the strategy with assumptions Γ_P from decisions Δ_P is winning in the modified game with assumptions Γ_P, then the strategy with assumptions $\Gamma_P \setminus \{(C, \boldsymbol{k})\}$ from decisions Δ'_P is winning in the modified game with assumptions $\Gamma_P \setminus \{(C, \boldsymbol{k})\}$.*

Proof. It follows immediately from Definitions D.1 and D.6. □

Lemma D.10 (winning strategy from decisions). *Given a fixpoint game, whenever functions* EXPLORE$(\cdot, \cdot, \cdot, \Gamma, \Delta)$ *and* BACKTRACK$(\cdot, \cdot, \cdot, \Gamma, \Delta)$ *are invoked, for every player P, the strategy with assumptions Γ_P from decisions Δ_P is winning in the modified game with assumptions Γ_P.*

Proof. We prove this by induction on the sequence of functions calls. Initially, on the first call EXPLORE$(C, \mathbf{0}, [], (\emptyset, \emptyset), (\emptyset, \emptyset))$, the property vacuously holds since $\Delta_\exists = \Delta_\forall = \emptyset$. Now, assuming that the property holds when a function is called, we show that it holds also on every invocation performed by such function.

Assume that the property holds when EXPLORE$(C, \boldsymbol{k}, \rho, \Gamma, \Delta)$ is called. The only invocation where the property could possibly fail is BACKTRACK$(\overline{\mathsf{P}(C)}, C, \rho, \Gamma, \Delta)$ after $(C, \boldsymbol{k})$ has been added to the decisions for $\overline{\mathsf{P}(C)}$, when $\mathsf{M}(C) = \emptyset$. However we can immediately see that $\overline{\mathsf{P}(C)}$ wins from C since the opponent $\mathsf{P}(C)$ cannot move (the strategy is always winning from C). On all the other calls the property is preserved since all decisions are unchanged and no assumption has been removed.

Assume that the property holds when Backtrack(P, C, ρ, Γ, Δ) is called. There are only two invocations to check. Clearly the property is preserved on the first one, i.e., Explore(C'', $\boldsymbol{k}''$, ρ, Γ, Δ), since all decisions and assumptions are unchanged. The second case is instead more complex. This is when the function Backtrack(P, C', t, Γ, Δ) is invoked. Let us analyse the strategy for one player at a time. First, consider the opponent $\overline{P}$. Even though the assumption $(C', \boldsymbol{k}')$ might have been removed from $\Gamma_{\overline{P}}$, all decisions in $\Delta_{\overline{P}}$ depending on such assumption have been removed as well via the function Forget($\Delta_{\overline{P}}$, $\Gamma_{\overline{P}}$, $(C', \boldsymbol{k}')$). Let $\Delta'_{\overline{P}}$ be the remaining decisions. By Lemma D.9 we know that the strategy with assumptions $\Gamma_{\overline{P}} \smallsetminus \{(C', \boldsymbol{k}')\}$ from decisions $\Delta'_{\overline{P}}$ is winning as long as the strategy with assumptions $\Gamma_{\overline{P}}$ from decisions $\Delta_{\overline{P}}$ was winning. Then by inductive hypothesis the property still holds for $\overline{P}$. Now we need to prove the property for player P as well. That is, the strategy s_P with assumptions $\Gamma_P \smallsetminus \{(C', \boldsymbol{k}')\}$ from decisions $\Delta_P \cup \{(C', \boldsymbol{k}')\}$ is winning in the modified game with assumptions $\Gamma_P \smallsetminus \{(C', \boldsymbol{k}')\}$. To do this we just need to show that for every position $\hat{C} \in \mathrm{C}(\Delta_P \cup \{(C', \boldsymbol{k}')\})$, every complete path in the tree of plays $\tau^{\hat{C}}_{s_P}$ is a play won by P. First, recall that every finite complete path in $\tau^{\hat{C}}_{s_P}$ terminates in a position of an assumption or a truth. In both cases such a finite play is always won by P since in the modified game assumptions and truths correspond to positions owned by the opponent with no available moves. By inductive hypothesis we know that the strategy s'_P with assumptions Γ_P from decisions Δ_P was winning in the modified game with assumptions Γ_P. Notice that the two strategies can only differ on the position C' of the new decision $(C', \boldsymbol{k}')$. It may be that s'_P was not defined on C', if there was no decision or assumption for such position before now. Anyway, this means that if C' never occurs along the path, then the play must be won by P since s_P and s'_P coincide on all the positions in the path and s'_P was winning by inductive hypothesis. Therefore we just need to check those paths containing C'. If C' appears just finitely many times along the path, consider the subpath starting from the successor C'' of the last occurrence of C'. Such subpath does not contain C' and it is still infinite. Recalling that all positions in infinite paths must come from decisions and $C'' \neq C'$, then the subpath must be one of the complete paths in the tree of plays $\tau^{C''}_{s'_P}$. Thus, by inductive hypothesis the subpath, as well as the initial one, must be a play won by P. Otherwise, C' appears infinitely many times along the path. Consider every subpath between two consecutive occurrences of C', including only the first one. In such subpath let $C'' \neq C'$ be the last position, which is the predecessor of the second occurrence of C'. Observe that no decision $(C', \boldsymbol{k})$ could have been added after exploring $(C', \boldsymbol{k}')$ and before now, because we would necessarily have either $\boldsymbol{k} <_P \boldsymbol{k}'$ or $\boldsymbol{k} <_{\overline{P}} \boldsymbol{k}'$, thus satisfying the condition of the third **if** branch of function Explore, in which case the exploration would have stopped and $(C', \boldsymbol{k})$ would have never been added as a decision. Furthermore, any decision $(C', \boldsymbol{k})$ added before exploring $(C', \boldsymbol{k}')$ must be such that $\boldsymbol{k}' < \boldsymbol{k}$, because otherwise the exploration would have stopped satisfying the second **if** branch of function Explore and $(C', \boldsymbol{k}')$ would have never been added as a decision. Therefore we must have $d_P(C') = \boldsymbol{k}'$ and, if $C' \in \mathrm{C}(\Delta_P) \smallsetminus \mathrm{C}(\Gamma_P)$ hence s'_P is defined on C', $d_P(C') <_P d'_P(C')$ since $d'_P(C')$ is the minimum $\boldsymbol{k}$ among the decisions for C' added before $(C', \boldsymbol{k}')$. Moreover, in the latter case, by Lemma D.8(a) we obtain that $d_P(C') <_P d'_P(C') \leq_P \mathit{next}(d_P(C''), \mathrm{i}(C''))$ since C' succeeds C''. If instead $C' \notin \mathrm{C}(\Delta_P) \smallsetminus \mathrm{C}(\Gamma_P)$, then we must have that $(C', \boldsymbol{k}') \in \Gamma_P$, since $C' \in s_P(C'') = s'_P(C'') \subseteq \mathrm{C}(\Delta_P \cup \Gamma_P)$ and $C' \in \mathrm{C}(\Delta_P \cup \{(C', \boldsymbol{k}')\}) \smallsetminus \mathrm{C}(\Gamma_P \smallsetminus \{(C', \boldsymbol{k}')\})$ because $s_P(C') \neq \emptyset$. In fact, by inspecting the code it can be seen that C' must have been added as an assumption after exploring C'', which then became a decision $(C'', d_P(C''))$, and it must have held $\boldsymbol{k}' <_P \mathit{next}(d_P(C''), \mathrm{i}(C''))$ as required by the third **if** branch in the function Explore. Thus, in both cases we have $\boldsymbol{k}' = d_P(C') <_P \mathit{next}(d_P(C''), \mathrm{i}(C''))$. And so by Lemma D.8(b) we know that $P = \exists$ iff $\eta_h = \nu$, where h is the highest priority appearing along the subpath. For now assume $P = \exists$. Since this holds for all subpaths between two consecutive occurrences of C', and there are infinitely many of them, which sequenced form the initial infinite path, then there must exist a priority h s.t. $\eta_h = \nu$ and it is the highest priority appearing infinitely many times along the complete path. A dual reasoning holds for $P = \forall$. Recalling that an infinite play is won by player $\exists$ (resp. $\forall$) if the highest priority $h \in \underline{m}$ appearing infinitely often is s.t. $\eta_h = \nu$ (resp. μ), we deduce that the path is won by P, whoever P is. And so we conclude that s_P is indeed winning in the modified game with assumptions $\Gamma_P \smallsetminus \{(C', \boldsymbol{k}')\}$. ◻

Now we can finally present the correctness result.

Theorem 7.3 (correctness). *Given a fixpoint game, if a call* Explore*(C, $\mathbf{0}$, [], $(\emptyset, \emptyset)$, $(\emptyset, \emptyset)$) returns a player P, then P wins the game from C.*

Proof. Assume that the call Explore(C, $\mathbf{0}$, [], $(\emptyset, \emptyset)$, $(\emptyset, \emptyset)$) returns some player P. Since the only instruction returning a value is in the function Backtrack and it is reached only when $\rho = []$, then Backtrack(P, C', [], Γ, Δ) must have been invoked for some Γ and Δ. Furthermore, $C' = C$ because $\rho = []$ is the list of positions from the root C to the current node C'. Also, by Lemma D.2 we have that $\Gamma_P = \emptyset$. Thus, by Lemma D.3 we have that $(C, \boldsymbol{k}) \in \Delta_P$ for some counter $\boldsymbol{k}$. And so by Lemma D.10 we can immediately conclude that P wins the game from C, since the modified game with no assumptions coincides with the original one. ◻

Theorem 7.4 (preserving solutions with up-to)**.** *Let E be a system of m equations of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$ over a complete lattice L. Let $\boldsymbol{u}$ be a m-tuple of up-to functions compatible for E (Definition 5.7). The solution of the system $d(E, \boldsymbol{u})$ is $sol(d(E, \boldsymbol{u})) = (sol(E), sol(E))$.*

Proof. We proceed by induction on the length m of the original system. The base case is vacuously true since, for $m = 0$, both systems have empty solution. Then, for $m > 0$, assume that the property holds for systems of size $m - 1$. By definition of solution we have that the solution of x_m is

$$sol_{2m}(d(E, \boldsymbol{u})) \quad = \quad \eta_m(\lambda x.\, f_m(sol_{1,m}(d(E, \boldsymbol{u})[x_m := x])))$$

and the parametric solution of y_m is the function $s' : L^m \to L$

$$s'(\boldsymbol{x}') \quad = \quad sol_m(d(E, \boldsymbol{u})[\boldsymbol{x} := \boldsymbol{x}']) = \mu(\lambda y.\, u_m(y) \sqcup x'_m).$$

Observe that since $s'(\boldsymbol{x}')$ depends only on x'_m, we can define the parametric solution of y_m using just a function $s : L \to L$ instead of s'

$$s(x) \quad = \quad \mu(\lambda y.\, u_m(y) \sqcup x).$$

Substituting the parametric solution of y_m in the solution of x_m we obtain

$$sol_{2m}(d(E, \boldsymbol{u})) \quad = \quad \eta_m(\lambda x.\, f_m(sol_{1,m-1}(d(E, \boldsymbol{u})[x_m := x][y_m := s(x)]), s(x))).$$

Let $h(x) = f_m(sol_{1,m-1}(d(E, \boldsymbol{u})[x_m := x][y_m := s(x)]), s(x))$ and $g_x(y) = u_m(y) \sqcup x$, so that $sol_{2m}(d(E, \boldsymbol{u})) = \eta_m(h)$ and $s(x) = \mu(g_x)$. Clearly h and g are both monotone (hence s as well). The former because the solutions of a system (see [Baldan et al. 2019]) and $\boldsymbol{f}$ are monotone, the latter because both u_m and the supremum are. Also notice that s is an extensive function, i.e., $x \sqsubseteq s(x)$ for all x. In fact, since s computes a (least) fixpoint we have that $s(x) = u_m(s(x)) \sqcup x$, and clearly $x \sqsubseteq u_m(s(x)) \sqcup x$ by definition of supremum. Furthermore, we can prove that s is compatible (wrt. h, i.e., $s(h(x)) \sqsubseteq h(s(x))$ for all x), continuous, and strict, whenever u_m satisfies those conditions, respectively. First, if u_m is continuous, then so is g in both variables, since $\sqcup$ is continuous. Then, since $s(x)$ is the least fixpoint of g_x, it is immediate that s is continuous as well. Recalling that $s(x) = g_x^\alpha(\bot)$ for some ordinal α, both remaining properties can be proved by transfinite induction on $g_x^\alpha(\bot)$ for every α. First we show that for all x, $g_{h(x)}^\alpha(\bot) \sqsubseteq h(s(x))$ for every ordinal α (hence $s(h(x)) \sqsubseteq h(s(x))$). For $\alpha = 0$, we have $g_{h(x)}^0(\bot) = \bot \sqsubseteq h(s(x))$. For a successor ordinal $\alpha = \beta + 1$, we have $g_{h(x)}^{\beta+1}(\bot) = g_{h(x)}(g_{h(x)}^\beta(\bot))$, and by inductive hypothesis we know that $g_{h(x)}^\beta(\bot) \sqsubseteq h(s(x))$. Then

$$\begin{aligned}
& g_{h(x)}(g_{h(x)}^\beta(\bot)) \\
& \sqsubseteq g_{h(x)}(h(s(x))) && \text{[since g is monotone]} \\
& = u_m(h(s(x))) \sqcup h(x) && \text{[by definition of g]} \\
& = u_m(f_m(sol_{1,m-1}(d(E, \boldsymbol{u})[x_m := s(x)][y_m := s^2(x)]), s^2(x))) \sqcup h(x) && \text{[by definition of h]} \\
& \sqsubseteq f_m(\boldsymbol{u} \cdot (sol_{1,m-1}(d(E, \boldsymbol{u})[x_m := s(x)][y_m := s^2(x)]), s^2(x))) \sqcup h(x) && \text{[by compatibility of $\boldsymbol{u}$]}
\end{aligned}$$

Observe that $u_m(s(z)) \sqsubseteq s(z) = g_z(s(z)) = u_m(s(z)) \sqcup z$ for all z. A similar reasoning applies to the other solutions as well, obtaining that $u_i(sol_i(d(E, \boldsymbol{u})[x_m := s(x)][y_m := s^2(x)])) \sqsubseteq sol_i(d(E, \boldsymbol{u})[x_m := s(x)][y_m := s^2(x)])$ for all $i \in \underline{m-1}$. Therefore we have

$$\begin{aligned}
& f_m(\boldsymbol{u} \cdot (sol_{1,m-1}(d(E, \boldsymbol{u})[x_m := s(x)][y_m := s^2(x)]), s^2(x))) \sqcup h(x) \\
& \sqsubseteq f_m(sol_{1,m-1}(d(E, \boldsymbol{u})[x_m := s(x)][y_m := s^2(x)]), s^2(x)) \sqcup h(x) && \text{[since f_m is monotone]} \\
& = h(s(x)) \sqcup h(x) && \text{[by definition of h]} \\
& \sqsubseteq h(s(x) \sqcup x) && \text{[since h is monotone]} \\
& = h(s(x)) && \text{[since s is extensive]}
\end{aligned}$$

And so we established that $g_{h(x)}^{\beta+1}(\bot) \sqsubseteq h(s(x))$. For α limit ordinal, by inductive hypothesis we immediately have that $g_{h(x)}^\alpha(\bot) = \bigsqcup_{\beta<\alpha} g_{h(x)}^\beta(\bot) \sqsubseteq \bigsqcup h(s(x)) = h(s(x))$. Now we show that $g_\bot^\alpha(\bot) = \bot$ for every ordinal α. For $\alpha = 0$, we have $g_\bot^0(\bot) = \bot$. For $\alpha = \beta + 1$, by inductive hypothesis we have that $g_\bot^{\beta+1}(\bot) = g_\bot(g_\bot^\beta(\bot)) = g_\bot(\bot)$. And in turn, $g_\bot(\bot) = u_m(\bot) \sqcup \bot = \bot$, since u_m is strict. For α limit ordinal, by inductive hypothesis we obtain that $g_\bot^\alpha(\bot) = \bigsqcup_{\beta<\alpha} g_\bot^\beta(\bot) = \bigsqcup \bot = \bot$. Now we have two different cases depending on η_m.

- $\eta_m = \nu$
  In this case $sol_{2m}(d(E, \boldsymbol{u})) = h^\alpha(\top)$ for some ordinal α. Here we show that actually $s(h^\alpha(\top)) = h^\alpha(\top)$ for every ordinal α. Since as we mentioned above s is extensive, we just need to prove that $s(h^\alpha(\top)) \sqsubseteq h^\alpha(\top)$ for every ordinal α. We proceed by transfinite induction on α. For $\alpha = 0$, we have $s(h^0(\top)) \sqsubseteq \top = h^0(\top)$. If α is a successor ordinal $\beta + 1$, assuming the property holds for β, we show that $s(h^{\beta+1}(\top)) \sqsubseteq h^{\beta+1}(\top)$. Since h is monotone, by inductive hypothesis we have that $h(s(h^\beta(\top))) \sqsubseteq h(h^\beta(\top)) = h^{\beta+1}(\top)$. Recalling that $s(h(x)) \sqsubseteq h(s(x))$ for all x, we also have that $s(h^{\beta+1}(\top)) = s(h(h^\beta(\top))) \sqsubseteq h(s(h^\beta(\top)))$. When α is a limit ordinal we have that $h^\alpha(\top) = \bigsqcap_{\beta<\alpha} h^\beta(\top)$. Since s is monotone, we have that $s(h^\alpha(\top)) = s(\bigsqcap_{\beta<\alpha} h^\beta(\top)) \sqsubseteq \bigsqcap_{\beta<\alpha} s(h^\beta(\top))$. And since by inductive hypothesis $s(h^\beta(\top)) \sqsubseteq h^\beta(\top)$ for all $\beta < \alpha$, we conclude also that $\bigsqcap_{\beta<\alpha} s(h^\beta(\top)) \sqsubseteq \bigsqcap_{\beta<\alpha} h^\beta(\top)$.
- $\eta_m = \mu$
  In this case $sol_{2m}(d(E, \boldsymbol{u})) = h^\alpha(\bot)$ for some ordinal α. Recall also that since $\eta_m = \mu$, by hypothesis we know that u_m is continuous and strict. In such case, as shown above, s is continuous and strict as well. Again, we already know that s is extensive, so we just prove by transfinite induction that $s(h^\alpha(\bot)) \sqsubseteq h^\alpha(\bot)$ for every ordinal α. For $\alpha = 0$, we have $s(h^0(\bot)) = s(\bot) = \bot$, since s is strict. If α is a successor ordinal $\beta + 1$, assuming the property holds for β, we show that $s(h^{\beta+1}(\bot)) \sqsubseteq h^{\beta+1}(\bot)$. Since h is monotone, by inductive hypothesis we have that $h(s(h^\beta(\bot))) \sqsubseteq h(h^\beta(\bot)) = h^{\beta+1}(\bot)$. Recalling that $s(h(x)) \sqsubseteq h(s(x))$ for all x, we also have that $s(h^{\beta+1}(\bot)) = s(h(h^\beta(\bot))) \sqsubseteq h(s(h^\beta(\bot)))$. When α is a limit ordinal we have that $h^\alpha(\bot) = \bigsqcup_{\beta<\alpha} h^\beta(\bot)$. Since s is continuous, we have that $s(h^\alpha(\bot)) = s(\bigsqcup_{\beta<\alpha} h^\beta(\bot)) = \bigsqcup_{\beta<\alpha} s(h^\beta(\bot))$. And since by inductive hypothesis $s(h^\beta(\bot)) \sqsubseteq h^\beta(\bot)$ for all $\beta < \alpha$, we conclude also that $\bigsqcup_{\beta<\alpha} s(h^\beta(\bot)) \sqsubseteq \bigsqcup_{\beta<\alpha} h^\beta(\bot)$.

So in both cases we have $s(h^\alpha(\top)) = h^\alpha(\top)$ or $s(h^\alpha(\bot)) = h^\alpha(\bot))$, respectively, for every ordinal α. Consider the function $h'(x) = f_m(sol_{1,m-1}(d(E, \boldsymbol{u})[x_m := x][y_m := x]), x)$. The previous fact implies that actually $\eta_m(h') = \eta_m(h) = sol_{2m}(d(E, \boldsymbol{u}))$. Furthermore, for the same reason we have that $s(sol_{2m}(d(E, \boldsymbol{u}))) = sol_{2m}(d(E, \boldsymbol{u}))$. Since $sol_{2m}(d(E, \boldsymbol{u}))$ is the solution of x_m and by definition of solution $s(sol_{2m}(d(E, \boldsymbol{u}))) = sol_m(d(E, \boldsymbol{u}))$ is that of y_m, this means that x_m and y_m have the same solution in $d(E, \boldsymbol{u})$. So we can rewrite the solutions of x_m and y_m as $\eta_m(h')$, that is

$$sol_{2m}(d(E, \boldsymbol{u})) = sol_m(d(E, \boldsymbol{u})) \quad = \quad \eta_m(\lambda x.\, f_m(sol_{1,m-1}(d(E, \boldsymbol{u})[x_m := x][y_m := x]), x)).$$

Now, observe that the system $d(E, \boldsymbol{u})[x_m := x][y_m := x]$ is actually $d(E[x_m := x], \boldsymbol{u}_{1,m-1})$. Therefore, since $E[x_m := x]$ has size $m - 1$, by inductive hypothesis we know that $sol_{1,m-1}(d(E, \boldsymbol{u})[x_m := x][y_m := x]) = sol_{m,2m-2}(d(E, \boldsymbol{u})[x_m := x][y_m := x]) = sol(E[x_m := x])$. Thus, substituting these solutions in those of x_m and y_m above, we obtain

$$sol_{2m}(d(E, \boldsymbol{u})) = sol_m(d(E, \boldsymbol{u})) \quad = \quad \eta_m(\lambda x.\, f_m(sol(E[x_m := x]), x))$$

which is also the definition of the solution of x_m in E. Which means that $sol_{2m}(d(E, \boldsymbol{u})) = sol_m(d(E, \boldsymbol{u})) = sol_m(E)$. Then, the remaining solutions are

$$\begin{aligned}
&(sol_{1,m-1}(d(E, \boldsymbol{u})), sol_{m+1,2m-1}(d(E, \boldsymbol{u})))\\
&= sol(d(E, \boldsymbol{u})[x_m := sol_{2m}(d(E, \boldsymbol{u}))][y_m := sol_m(d(E, \boldsymbol{u}))]) && \text{[by definition of solution]}\\
&= sol(d(E, \boldsymbol{u})[x_m := sol_m(E)][y_m := sol_m(E)])\\
&= (sol(E[x_m := sol_m(E)]), sol(E[x_m := sol_m(E)])) && \text{[by inductive hypothesis]}\\
&= (sol_{1,m-1}(E), sol_{1,m-1}(E)) && \text{[by definition of solution]}
\end{aligned}$$

This and the previous fact allow us to conclude that $sol(d(E, \boldsymbol{u})) = (sol_{1,m-1}(E), sol_m(E), sol_{1,m-1}(E), sol_m(E))$, that is indeed $sol(d(E, \boldsymbol{u})) = (sol(E), sol(E))$. □

Theorem 7.6 (correctness with up-to). *Let E be a system of m equations of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$ over a complete lattice L. Let $\boldsymbol{u}$ a compatible m-tuple of up-to functions for E. Then the up-to algorithm associated with the system $d(E, \boldsymbol{u})$ as given in Definition 7.5 is correct, i.e., if a call* Explore*(C, $\mathbf{0}$, [], $(\emptyset, \emptyset)$, $(\emptyset, \emptyset)$) returns a player P, then P wins the game from C.*

Proof. Let G be the fixpoint game associated with the initial system E, G_u be the one associated with the modified system $d(E, \boldsymbol{u})$, and G'_u be the game obtained from G_u by restricting the moves of player $\exists$ from positions associated with variables y_i to only those satisfying either condition (a) or (b). Observe that the moves from every position controlled by player $\exists$ of G are included in the moves from the corresponding position in G'_u since they satisfy condition (a), since in E there are no up-to functions. Therefore, every winning strategy for $\exists$ in G can be easily converted into a winning strategy for the same player in G'_u. So the winning positions of player $\exists$ in G are necessarily included in those of G'_u. Furthermore, the same clearly happens between G'_u and G_u since the moves of $\exists$ in G'_u are defined as a restriction of those in G_u. Then, calling $W_\exists(G)$ the set of winning positions of player $\exists$ in the corresponding G, we have that $W_\exists(G) \subseteq W_\exists(G'_u) \subseteq W_\exists(G_u) = W_\exists(G)$, where the last equality holds by Theorem 7.4. Since in our case every position not winning for $\exists$ is necessarily winning for $\forall$, this means that even if we restrict certain moves of player $\exists$, thus playing in the game G'_u, we still have the same exact winning positions for both players. $\square$

E Comparison to the Bonchi/Pous algorithm

In a seminal paper [Bonchi and Pous 2013] Bonchi and Pous revisited the question of checking language equivalence for non-deterministic automata and presented an algorithm based on an up-to congruence technique that behaves very well in practice.

We will here give a short description of this algorithm and then explain how it arises as a special case of the algorithm developed in §6.2.2 (Case 2).

We are given a non-deterministic finite automaton (Q, Σ, δ, F), where Q is the finite set of states, Σ is the finite alphabet, $\delta \colon Q \times \Sigma \to \mathbf{2}^Q$ is the transition function and $F \subseteq Q$ is the set of final states. Note that we omit initial states. Given $a \in \Sigma$, $X \subseteq Q$ we define $\delta_a(X) = \bigcup_{q \in X} \delta(q, a)$.

Given $q_1, q_2 \in Q$, the aim is to show whether q_1, q_2 accept the same language (in the standard sense).

In order to do this, the algorithm performs an on-the-fly determinization and constructs a bisimulation relation $R \subseteq \mathbf{2}^Q \times \mathbf{2}^Q$ on the determinized automaton. This relation has to satisfy the following properties:

- $\{q_1\}\, R\, \{q_2\}$
- Whenever $X_1\, R\, X_2$, then
  - $\delta_a(X_1)\, R\, \delta_a(X_2)$ for all $a \in \Sigma$ (*transfer property*)
  - and $X_1 \cap F \neq \emptyset \iff X_2 \cap F \neq \emptyset$ (*one set is accepting iff the other is accepting*)

Due to the up-to technique there is no need to fully enumerate R. Instead in the second item above, it suffices to show that $\delta_a(X_1)\, c(R)\, \delta_a(X_2)$ where $c(R)$ is the congruence closure of R, i.e., the least relation R' containing R that is an equivalence and satisfies that $X_1\, R\, X_2$ implies $X_1 \cup X\, R\, X_2 \cup X$ (for $X_1, X_2, X \subseteq Q$). A major contribution of [Bonchi and Pous 2013] is an algorithm for efficiently checking whether two given sets are in the congruence closure of a given relation. Here we will simply assume that this procedure is given and use it as a black box.

We will now translate this into our setting: the lattice is $L = \mathbf{2}^{\mathbf{2}^Q \times \mathbf{2}^Q}$ (the lattice of all relations over the powerset of states) with inclusion as partial order. The basis B consists of all singletons $\{(X_1, X_2)\}$ where $X_1, X_2 \subseteq Q$. That is, we consider Case 2 of §6.2.2.

The behaviour map f is given as follows: $f(R) = f^*(R) \cap C$ where

$$\begin{aligned} f^*(R) &= \{(X_1, X_2) \mid (\delta_a(X_1), \delta_a(X_2)) \in R \text{ for all } a \in \Sigma\} \\ C &= \{(X_1, X_2) \mid X_1 \cap F = \emptyset \iff X_2 \cap F = \emptyset\} \end{aligned}$$

We want to solve a single fixpoint equation $R =_\nu f(R)$ where we are interested in the greatest fixpoint. In particular, we want to check whether $(Q_1, Q_2) \in R$ (where $Q_1 = \{q_1\}$, $Q_2 = \{q_2\}$) or alternatively $I = \{(Q_1, Q_2)\} \subseteq R$.

Since we have determinized the automaton, f^* has a left adjoint f_*, given as

$$f_*(R) = \{(\delta_a(X_1), \delta_a(X_2)) \mid (X_1, X_2) \in R, a \in \Sigma\}.$$

Now we can start exploring the game positions. Starting with $I = \{(Q_1, Q_2)\} \subseteq F$, the only move of $\exists$ is to play $\{\{(X_1, X_2)\} \mid (X_1, X_2) \in f_*(I)\}$, then it is the turn of $\forall$ who can choose any singleton set $\{(X_1, X_2)\}$ and one has to explore all those singletons. This continues until one encounters a singleton $\{(X_1, X_2)\} \not\subseteq C$ (which implies that $\exists$ has no move and loses) or one finds a set $\{(X_1, X_2)\}$ where one can cut off a branch due to the up-to technique – more concretely $(X_1, X_2) \in c(R)$ where R is the collection of all pairs visited so far on all paths and $c(R)$ is its congruence closure. One can conclude that $\exists$ wins if all encountered pairs are in C. This is a straightforward instance of the more general algorithm (Case 2), enriched with an up-to technique, as explained in §6.2.2.