THÈSE

présentée à

l'ÉCOLE POLYTECHNIQUE

pour l'obtention du titre de

DOCTEUR DE L'ÉCOLE POLYTECHNIQUE EN INFORMATIQUE

Damien MASSÉ

14 décembre 2002

Vérification par Interprétation Abstraite Guidée par une Propriété Temporelle

Temporal Property-driven Verification by Abstract Interpretation

---

Président :	Patrick Cousot *Professeur, École Normale Supérieure (Paris)*
Rapporteurs :	E. Allen Emerson *Professor, The University of Texas à Austin (États-Unis)*
	Alan Mycroft *Reader, Cambridge University Computer Laboratory (Royaume-Uni)*
	David A. Schmidt *Professor, Kansas State University (États-Unis)*
Examinateur :	Roberto Giacobazzi *Professeur, Université de Vérone (Italie)*
Directeur de thèse :	Radhia Cousot *Directeur de Recherche au CNRS*

Résumé de la thèse

L'analyse statique par interprétation abstraite permet d'inférer des propriétés sur un programme par abstraction de sa sémantique. Ces propriétés permettent entre autres de vérifier si le programme satisfait ou non une spécification. Toutefois, si la spécification est connue par avance, on peut en tenir compte lors de l'analyse pour orienter celle-ci vers la vérification de cette spécification. Dans cette thèse, nous considérons le cas de spécifications complexes, en prenant comme exemple des propriétés temporelles comme CTL.

Une façon de prendre en compte la spécification est la combinaison des analyses avant et arrière, de sorte que les résultats de l'une servent à améliorer les résultats de l'autre. Nous montrons comment cette combinaison est possible pour des propriétés temporelles complexes, incluant CTL et des logiques de jeu, en gardant une analyse d'accessibilité comme analyse avant.

Pour obtenir de meilleurs résultats, il est nécessaire d'adapter l'analyse avant. Nous définissons alors les *systèmes de transition étendus*, qui combinent le programme et sa spécification. Nous définissons aussi les sémantiques avant et arrière de ces systèmes de transition étendus et montrons les liens entre elles. Nous présentons aussi un moyen de les générer.

Une autre approche consiste à inclure la spécification dans le calcul du point fixe de l'analyse. A partir d'une spécification, nous dérivons par une première analyse abstraite un « guide », sous la forme d'un opérateur de fermeture, qui sert à spécialiser l'analyse principale vers la vérification de la spécification. Nous montrons enfin qu'il est possible, là aussi, d'itérer le processus. Cette approche permet de dériver automatiquement une combinaison correcte d'analyses à partir d'une seule analyse et d'une spécification.

Remerciements

Pour le doctorant, une thèse est une épreuve difficile. L'étape des remerciements est une occasion de reconnaître l'importance de l'entourage professionnel et personnel du doctorant au cours de cette épreuve. C'est avec plaisir que j'utilise cette occasion pour exprimer ma reconnaissance envers ceux qui ont contribué à ce que cette thèse se déroule dans les meilleures conditions possibles.

Je remercie Allen Emerson, Alan Mycroft et David Schmidt pour avoir accepté d'être rapporteurs de ma thèse, ainsi que pour leurs remarques et conseils. Je remercie vivement Patrick Cousot, qui m'a fait découvrir l'analyse sémantique de programmes au cours de ma formation, m'a donné d'inestimables conseils tout au long de ces années, et m'a fait l'honneur de présider le jury de ma thèse. Je remercie aussi Roberto Giacobazzi pour avoir accepté d'être examinateur de mon travail.

Radhia Cousot a accepté d'être mon directeur de thèse. Je ne pourrais pas la remercier assez pour son encadrement, la grande liberté qu'elle m'a laissée et l'ensemble de son aide tout au long de cette thèse.

Je remercie aussi le Laboratoire d'Informatique de l'École Polytechnique, qui fut le cadre de mes recherches, et sa secrétaire Evelyne Rayssac.

Je remercie également mes collègues, thésards ou chercheurs, qui m'ont aidé, soutenu et conseillé au cours de cette thèse, entre autres et sans ordre précis, David Monniaux, Francesco Logozzo, Charles Hymans, Jérôme Feret, Antoine Miné, Laurent Mauborgne et Bruno Blanchet. Je prie à l'avance ceux que j'aurais pu oublier de m'en excuser.

Finalement, je voudrais remercier Gaëlle, qui m'a apporté un encouragement essentiel à la fin de ma thèse, ainsi que ma famille dont le soutien moral a été constant.

Contents

Chapter 1

Overview

Verifying complex temporal specifications automatically is a hard challenge. Simulation and testing are not safe. Deductive methods ultimately require human interactions to assist the theorem prover. Model-checking faces resource explosion. Abstract interpretation-based static analysis relies on the idea of approximation. Rough abstractions are fast but may be not precise enough, while precise abstractions appear too costly. A main issue is to take into account the temporal specification in the abstract analysis: by directing the analysis towards the checking of the specification as much as possible, we can expect more precise results with fast abstractions. We show several new approaches to achieve this result.

A common point of all these approaches is the *combination* of analyses: two abstract analyses are designed, each one contributing to the verification, and these analyses are combined, such that the results of one refine the results of the other, in an iterative manner. In general, one analysis is derived directly from the semantics of the temporal formulas used for the specification, whereas the other starts from the condition these formulas must satisfy and computes constraints on the semantics of the former analysis. Hence, this second analysis is used to drive the first one towards the checking of the property.

In chapter 3, we study this approach in the case where the second analysis is merely a reachability analysis. In this case our approach appears as an extension for complex temporal properties of the already known backward-forward combination used in abstract testing. The first analysis is based on an algorithm designed to check forward temporal properties (which use only backward operators). We show how the combination of this analysis with reachability analyses can be used, under some conditions, to check a given property. This combination gives better results than the simple intersection of both abstract analyses made separately. We show that this combination is not possible for all temporal properties, but can be used with a superset of CTL and some game semantics.

The main advantage of using a reachability analysis is that it is easy to implement using existing analyzers. However, general-purpose analyses are not very precise, es-

pecially when non-relational abstractions are used. Since the forward analysis is not "specialised" for the checking of the specification, it cannot keep specific constraints.

In chapter 4, we incorporate the temporal specification into the description of the program to create a new semantics. To do this, we define an extension of transition systems: an *extended transition system* associates with each state several sets of potential successors. This extension engenders a kind of game semantics, where the goal of the game is to prove the specification. We define forward and backward semantics for this extension, both described as sets of free trees, and we show the relationship between them. Furthermore, we describe how to combine these semantics, and how to generate an extended transition system from a program and its specification.

Because they can keep the history of the calculus, like trace semantics for classical transition systems, these tree-based semantics are very powerful, but they are quite complex and very hard to abstract precisely and efficiently while keeping the soundness of the combination.

Another approach, developed in chapter 5, is to express the result of the second analysis by a lower closure operator which is applied on the first one to guide it. Then the condition of soundness of the combination is easy to formalize and to prove. Following this idea, we present a generic construction of the second analysis from the first, which works for many specifications, including specifications which use extended transition systems. In the concrete case, this construction is similar to the construction of fixpoint-complete closure operators. Our contribution allows to abstract it while keeping the soundness of the combination, even when it is iterated.

Less powerful than tree-based semantics (but we can use both), this approach may be faster and easier to implement. To design analyzers using this approach, we need abstractions on domains of lower closure operators. Hence we give some examples of abstractions.

To complete our work, we develop a small toy analyzer based on the last construction. A brief description of the analyzer, and some results, are given in chapter 6.

Most of the results already appeared in proceedings of conferences (chapter 3 derives from [Mas01], chapter 4 from [Mas02], and chapter 5 from [Mas03]).

Chapter 2

Introduction

Let us begin with a few definitions, notations and results which will be used throughout the thesis.

2.1 Mathematical notations

- $\mathbb{Z}$ is the set of integers, and $\mathbb{Z}^\infty = \mathbb{Z} \cup \{-\infty, +\infty\}$.
- $A \to B$ is the set of mappings from A to B. If A and B are partially ordered, $A \xrightarrow{m} B$ denotes the set of monotone operators from A to B.
- $\wp(X)$ is the set of subsets of X;
- if $(D, \leq)$ is a partially ordered set and $a \in D$, the principal upper ideal generated by a is denoted $(\uparrow_\leq a) = \{b \in D \mid a \leq b\}$[1]. Furthermore if $X \subseteq D$, we define min X as min $X = \{a \in X \mid \forall b \in X, b \leq a \Rightarrow b = a\}$.
- The *least fixpoint* of an operator f in $A \to A$ is denoted, when it exists, lfp f. The *greatest fixpoint* is denoted gfp f. The notation lgfp will be used to denote either lfp or gfp.
- The least fixpoint greater than a of an operator f in $A \to A$, when it exists, is denoted $\text{lfp}_a\, f$, or, when this fixpoint is the limit of an upper iteration sequence [CC79a], luis (f, a). Dually, the greatest fixpoint lower than a is denoted $\text{gfp}_a\, f$, or llis (f, a).

[1] When there is no ambiguity, we will note $(\uparrow a)$ instead of $(\uparrow_\leq a)$.

2.2 Semantics of a program

2.2.1 Transition system

The goal of static analysis is to analyse automatically programs and give informations about their behaviours. Thus, the first step of static analysis is the derivation of a mathematical meaning for each program, its *semantics*.

Even if our goal is to analyse programs written in real languages (and we intend to use the structure of the language in the conception of the analyzer), for general results we want to deal with programs independently of the language used. Therefore, we will consider programs as *transition systems*.

Definition 2.2.1 (Transition system) *A transition system is a tuple* $\langle \Sigma, \tau \rangle$ *where* Σ *is a set of* states*, and* $\tau \subseteq \Sigma \times \Sigma$ *is a* transition relation *on* Σ*.*

In general, transition systems are generated by a small-step operational semantics [Plo81]. τ being a transition relation, we will note τ^* the transitive closure of τ, and τ^{-1} the inverse of τ.

Given a transition relation τ, we define the four predicate transformers:

Definition 2.2.2 (Predicate transformers) τ *being a transition relation on a set* Σ*, and* X *being a subset of* Σ*, we define:*

$$\begin{array}{l} post[\tau](X) = \{\sigma' \in \Sigma \mid \exists \sigma \in X, (\sigma, \sigma') \in \tau\} \\ \widetilde{post}[\tau](X) = \{\sigma' \in \Sigma \mid \forall \sigma \in \Sigma, (\sigma, \sigma') \in \tau \Rightarrow \sigma \in X\} \\ pre[\tau](Y) = \{\sigma \in \Sigma \mid \exists \sigma' \in Y, (\sigma, \sigma') \in \tau\} \\ \widetilde{pre}[\tau](Y) = \{\sigma \in \Sigma \mid \forall \sigma' \in \Sigma, (\sigma, \sigma') \in \tau \Rightarrow \sigma' \in Y\} \end{array} \tag{2.1}$$

When there is no ambiguity, we will write $post(X)$, $\widetilde{post}(X)$,..., instead of $post[\tau](X)$, $\widetilde{post}[\tau](X)$,.... $post(X)$ are the successors of X, whereas $\widetilde{post}(X)$ are the states which have all their predecessors in X. These relations are straightforward:

$$\begin{array}{rcl} \widetilde{post}(X) & = & \Sigma \backslash post(\Sigma \backslash X) \\ \widetilde{pre}(X) & = & \Sigma \backslash pre(\Sigma \backslash X) \\ Y \subseteq \widetilde{post}(X) & \Leftrightarrow & pre(Y) \subseteq X \\ Y \subseteq \widetilde{pre}(X) & \Leftrightarrow & post(Y) \subseteq X \end{array}$$

post and $\widetilde{post}$ are *forward* predicate transformers, whereas *pre* and $\widetilde{pre}$ are *backward* predicate transformers.

2.2.2 Semantics

Many semantics can be defined for transition systems, most of them related [Cou02]. We present here several semantics expressed as sets of states:

Proposition 2.2.3 ([Cou81]) *Let $\langle \Sigma, \tau \rangle$ be a transition system.*

- *The set of states* reachable *from a set of initial states I is equal to $\mathcal{F}_I = post[\tau^*](I) = \mathrm{lfp}\, \lambda X.(I \cup post[\tau](X))$. It is called the forward collecting semantics of the transition system.*
- *The set of states which can lead to one of the final states F is equal to $\mathcal{B}_F = pre[\tau^*](F) = \mathrm{lfp}\, \lambda X.(F \cup pre[\tau](X))$. It is called the backward collecting semantics of the transition system.*

$\mathcal{F}_I$ is a *forward* semantics of the transition system $\langle \Sigma, \tau \rangle$, whereas $\mathcal{B}_F$ is a *backward* semantics.

More generally, forward semantics propagate information from the past to the future of the computation (or reproduce the execution of the program), whereas backward semantics propagate information from the future to the past of the computation (or reproduce the reverse execution of the program).

2.3 Temporal properties

Following examples from the model-checking community [CGP99], we will use *temporal logics* for specifying programs. Temporal logics have proved to be useful because they can describe the relation between temporal events, without expressing time explicitely.

Many temporal logics are defined in the literature. For example, the $\overset{\curvearrowleft}{\mu}^*$-calculus, described in [CC00], is a complex temporal logic with backward and forward temporal operators, and which formulas are interpreted as sets of traces. However, we will restrict ourselves to quite simple temporal logics:

1. As we want to specify programs, at any time only the current state of the computer is important, and not the trace. Thus, we will restrict the formulas to be *state formulas*, which are true (or false) in a specific state.
2. In general, checking the specification consists of verifying that the initial states satisfy the temporal formulas. Thus, these formulas describe the future of the computation. Following this approach, we will restrict ourselves to *forward* temporal logics (which use only *predecessor* operators).

On the other hand, we will also use *game* logics, to give the possibility of distinguishing between the various kinds of non-determinism in the program.

Following these principles, we present the temporal logics we will use: a form of μ-calculus and its subset CTL, and the game logics $A\mu$-calculus.

2.3.1 μ-calculus and CTL

μ-calculus

Kozen's μ-calculus [Koz83] is a powerful language for expressing properties of transition systems, which includes many temporal and program logics. Since formulas are interpreted relatively to a transition system, this is highly convenient to specify properties on the program. For the sake of simplicity, we will use the μ-calculus with only one action.

The μ-calculus is defined by the following grammar:

$$\varphi ::= p \mid \neg p \mid X \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Diamond\varphi \mid \Box\varphi \mid \mu X.\varphi \mid \nu X.\varphi.$$

Here $p \in \mathrm{AP}$ is an *atomic proposition*, $X \in \mathcal{V}$ is a variable, $\neg$ is negation, $\wedge$ conjunction, $\vee$ disjunction, $\Diamond$ and $\Box$ are the backward operators, and μ and ν the least and greatest fixpoint operators. We consider only *closed formulas* with no free variables.

We use the notation $\frac{\mu}{\nu}$ to denote either μ or ν.

We restrict negation to atomic propositions. This restriction ensures the monotonicity of formulas inside fixpoint operators, and is possible because the negation can be pushed down using the dualities:

$$\begin{array}{rcl}
\neg(\varphi_1 \vee \varphi_2) & \rightarrow & (\neg\varphi_1) \wedge (\neg\varphi_2) \\
\neg(\varphi_1 \wedge \varphi_2) & \rightarrow & (\neg\varphi_1) \vee (\neg\varphi_2) \\
\neg(\Box\varphi) & \rightarrow & \Diamond(\neg\varphi) \\
\neg(\Diamond\varphi) & \rightarrow & \Box(\neg\varphi) \\
\neg(\nu X.\varphi(X)) & \rightarrow & \mu X.(\neg\varphi(\neg X)) \\
\neg(\mu X.\varphi(X)) & \rightarrow & \nu X.(\neg\varphi(\neg X))
\end{array}$$

To interpret a formula in a transition system $\langle \Sigma, \tau \rangle$, we need a mapping $L : \Sigma \rightarrow \wp(\mathrm{AP})$ which gives the set of atomic proposition true in a state. As we need to interpret non-closed formulas, we use $e : \mathcal{V} \rightarrow \wp(\Sigma)$ to denote an *environment*.

Then, given L, the set $[\![\varphi]\!]_e$ of states in which φ is true for the environment e is defined as follows:

$$\begin{array}{l}
[\![p]\!]_e = \{\sigma \mid p \in L(\sigma)\} \\
[\![\neg p]\!]_e = \{\sigma \mid p \notin L(\sigma)\} \\
[\![X]\!]_e = e(X) \\
[\![\varphi_1 \wedge \varphi_2]\!]_e = [\![\varphi_1]\!]_e \cap [\![\varphi_2]\!]_e \\
[\![\varphi_1 \vee \varphi_2]\!]_e = [\![\varphi_1]\!]_e \cup [\![\varphi_2]\!]_e \\
[\![\Diamond\varphi]\!]_e = \{\sigma \mid \exists \sigma' \in [\![\varphi]\!]_e, (\sigma, \sigma') \in \tau\} \\
[\![\Box\varphi]\!]_e = \{\sigma \mid \forall \sigma' \in \Sigma, (\sigma, \sigma') \in \tau \Rightarrow \sigma' \in [\![\varphi]\!]_e\} \\
[\![\mu X.\varphi]\!]_e = \mathrm{lfp}\, \lambda S.[\![\varphi]\!]_{e[X \leftarrow S]} \\
[\![\nu X.\varphi]\!]_e = \mathrm{gfp}\, \lambda S.[\![\varphi]\!]_{e[X \leftarrow S]}
\end{array}$$

We can see that:

$$[\![\Diamond\varphi]\!]_e = pre[\tau]([\![\varphi]\!]_e),$$
$$[\![\Box\varphi]\!]_e = \widetilde{pre}[\tau]([\![\varphi]\!]_e).$$

Thus we can interpret a μ-calculus formula with the two predicate transformers *pre* and $\widetilde{pre}$, and the fixpoint operators. When the set of states is finite, we can compute $[\![\varphi]\!]$ using *pre* and $\widetilde{pre}$. This is the principle of the *model-checking* algorithms for the verification of finite-state models.

When φ is closed, we will omit the environment and write its interpretation $[\![\varphi]\!]$.

CTL

CTL (Computation Tree Logic) [CE81] can be defined as a part of the μ-calculus. CTL formulas are defined by the following grammar:

$$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{AX}\varphi \mid \mathbf{EX}\varphi \mid \mathbf{AF}\varphi \mid \mathbf{EF}\varphi \mid \mathbf{AG}\varphi \mid \mathbf{EG}\varphi$$
$$\mid \mathbf{A}\varphi\mathbf{U}\varphi \mid \mathbf{E}\varphi\mathbf{U}\varphi \mid \mathbf{A}\varphi\mathbf{R}\varphi \mid \mathbf{E}\varphi\mathbf{R}\varphi$$

A and **E** are *path quantifiers*, and distinguish between conditions for all paths (**A**) and for some path (**E**) starting from a particular state.

X, **F**, **G**, **U** and **R** are the *temporal operators*, and describe requirements on the future states. **X** ("next time") refers to the next state, **F** ("eventually") to some state in the future, **G** ("globally") to all states in the future. **U** ("until") requires that the second property holds at some state in the future, and that the first one holds until this state. **R** ("release") is the logical dual of **U**. It requires that the second properties holds up to and including a state where the first property holds, or infinitely if the first property never holds.

The correspondences between CTL and μ-calculus are as follows:

$$\begin{aligned}
\mathbf{EX}\varphi &\longleftrightarrow \Diamond\varphi, \\
\mathbf{AX}\varphi &\longleftrightarrow \Box\varphi, \\
\mathbf{EF}\varphi &\longleftrightarrow \mu X.(\varphi \vee \Diamond X), \\
\mathbf{AF}\varphi &\longleftrightarrow \mu X.(\varphi \vee \Box X), \\
\mathbf{EG}\varphi &\longleftrightarrow \nu X.(\varphi \wedge \Diamond X), \\
\mathbf{AG}\varphi &\longleftrightarrow \nu X.(\varphi \wedge \Box X), \\
\mathbf{E}\varphi_1\mathbf{U}\varphi_2 &\longleftrightarrow \mu X.(\varphi_2 \vee (\varphi_1 \wedge \Diamond X)), \\
\mathbf{A}\varphi_1\mathbf{U}\varphi_2 &\longleftrightarrow \mu X.(\varphi_2 \vee (\varphi_1 \wedge \Box X)), \\
\mathbf{E}\varphi_1\mathbf{R}\varphi_2 &\longleftrightarrow \nu X.(\varphi_2 \wedge (\varphi_1 \vee \Diamond X)), \\
\mathbf{A}\varphi_1\mathbf{R}\varphi_2 &\longleftrightarrow \nu X.(\varphi_2 \wedge (\varphi_1 \vee \Box X)).
\end{aligned}$$

2.3.2 $A\mu$-calculus

Many properties are not expressible in CTL or μ-calculus. To prove the generality of our approach, we will use a game logic: the *alternating time μ-calculus* ($A\mu$) [AHK97]. The properties are defined on *alternating transition systems*:

Definition 2.3.1 (Alternating transition system [AHK97]) *An* alternating transition system *is a tuple* $\langle \mathcal{P}, \Sigma, \Delta \rangle$*, with* $\mathcal{P}$ *a set of* players, Σ *a set of* states*, and* $\Delta = \{\delta_i : \Sigma \to \wp(\wp(\Sigma)) \mid i \in \mathcal{P}\}$ *a set of* transition functions.

Informally, an 'alternating transition systems works the following way: when the system is in state σ, each player a must choose a set $\Sigma_a \in \delta_a(\Sigma)$, and the successor of the state σ must lie in $\bigcap_{a \in \mathcal{P}} {\Sigma_a}^2$.

In this definition of an alternating transition system, we change the usual notation to keep Σ as the set of states, as for standard transition systems. Also, we do not include the set of atomic propositions and their interpretation.

The equivalent of the predecessor operators *pre* and $\widetilde{pre}$ for alternating transition systems are the *controllable and uncontrollable predecessor relations*, defined in [HMMR00]. In the general case, letting $I \in \wp(\mathcal{P}) \setminus \{\emptyset\}$ be a team of players, they are defined as:

$$\sigma \in CPre_I(S) \quad \text{iff} \quad \exists (\tau_i \in \delta_i(\sigma))_{i \in I}, \forall (\tau_i \in \delta_i(\sigma))_{i \notin I}. \bigcap_{i \in \mathcal{P}} \tau_i \subseteq S$$

$$\sigma \in UPre_I(S) \quad \text{iff} \quad \forall (\tau_i \in \delta_i(\sigma))_{i \in I}, \exists (\tau_i \in \delta_i(\sigma))_{i \notin I}. \bigcap_{i \in \mathcal{P}} \tau_i \subseteq S$$

Thus, $\sigma \in CPre_I(S)$ means that, when the system is in state σ, the team I can force the next state to be in S, whereas $\sigma \in UPre_I(S)$ means that in state σ, the team I cannot force the game outside S. Of course, if there is only one player, these two operators are equivalent to *pre* and $\widetilde{pre}$.

We can also define a *post* operator for alternating transition systems, representing the possible successor states of a set of states:

$$post(S) = \bigcup_{\sigma \in S} \left(\bigcap_{a \in \mathcal{P}} \bigcup \delta_a(\sigma) \right) \tag{2.2}$$

Alternating-time μ-calculus

$A\mu$ formulas are generated by the grammar:

$$\varphi ::= p \mid \neg p \mid x \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \langle\langle I \rangle\rangle \bigcirc \varphi_1 \mid [\![I]\!] \bigcirc \varphi_1 \mid (\mu x.\varphi_1) \mid (\nu x.\varphi_2)$$

$p \in \Pi$ are atomic propositions, variables x are in a set X, and teams I are in $\wp(\mathcal{P})$.

Letting $\mathcal{E} : X \to \wp(\Sigma)$ map each variable to a set of states, a formula φ defines

[2] In general, it is assumed that the intersection is always a singleton, so the transition function is nonblocking and "deterministic".

a set of states $[\![\varphi]\!]_{\mathcal{E}}$ as follows:

$$\begin{array}{l}
[\![p]\!]_{\mathcal{E}} = \pi(p) \\
[\![\neg p]\!]_{\mathcal{E}} = \Sigma \backslash \pi(p) \\
[\![x]\!]_{\mathcal{E}} = \mathcal{E}(x) \\
[\![\varphi_1 \wedge \varphi_2]\!]_{\mathcal{E}} = [\![\varphi_1]\!]_{\mathcal{E}} \cap [\![\varphi_2]\!]_{\mathcal{E}} \\
[\![\varphi_1 \vee \varphi_2]\!]_{\mathcal{E}} = [\![\varphi_1]\!]_{\mathcal{E}} \cup [\![\varphi_2]\!]_{\mathcal{E}} \\
[\![\langle\!\langle I \rangle\!\rangle \bigcirc \varphi_1]\!]_{\mathcal{E}} = \mathit{CPre}_I([\![\varphi_1]\!]_{\mathcal{E}}) \\
[\![[\![I]\!] \bigcirc \varphi_1]\!]_{\mathcal{E}} = \mathit{UPre}_I([\![\varphi_1]\!]_{\mathcal{E}}) \\
[\![\mu x.\varphi_1]\!]_{\mathcal{E}} = \mathrm{lfp}\ \lambda\rho.[\![\varphi_1]\!]_{\mathcal{E}[x \mapsto \rho]} \\
[\![\nu x.\varphi_1]\!]_{\mathcal{E}} = \mathrm{gfp}\ \lambda\rho.[\![\varphi_1]\!]_{\mathcal{E}[x \mapsto \rho]}
\end{array}$$

When φ is closed, we write $[\![\varphi]\!]$ instead of $[\![\varphi]\!]_{\mathcal{E}}$.

When Σ is finite, it is possible to compute $[\![\varphi]\!]$ using the operators CPre_I and UPre_I. Model-checking algorithms on $A\mu$-calculus formulas use this principle.

2.4 Abstract interpretation

When the set of states is finite, model-checking algorithms can be used to check temporal formulas. But when the set of states is infinite, or too large, these algorithms are not usable. In fact, most temporal properties are simply not decidable for all programs. Therefore, an analyzer must use approximations, and sometimes be unable to prove or refute the property.

A good program analyzer should be efficient (with a good trade-off between precision and cost) and safe (all its answers must be correct). *Abstract interpretation* [CC77, Cou78, CC92a, CC92b] is a theory which is aimed to provide safe and efficient analyses for real programs (cf. for example [BCC+02] for an application of abstract interpretation to real-time critical software).

Abstract interpretation is based on the idea of approximations between concrete and abstract domains (e.g. domains of properties). Though we do not intend to show here the whole framework of abstract interpretation, we present some results we will use in the following chapters.

2.4.1 Galois connection and fixpoint approximations

We just give a few well-known results on Galois connections and fixpoint approximations.

Definition 2.4.1 (Galois connection [CC77]) *Let* $P^\flat(\sqsubseteq^\flat, \bot^\flat, \top^\flat, \sqcap^\flat, \sqcup^\flat)$ *and* $P^\sharp(\sqsubseteq^\sharp, \bot^\sharp, \top^\sharp, \sqcap^\sharp, \sqcup^\sharp)$ *be two complete lattices. Then* $\alpha \in P^\flat \to P^\sharp$ *and* $\gamma \in P^\sharp \to P^\flat$ *define a* Galois connection *(denoted* $P^\flat \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P^\sharp$*) if and only if:*

$$\forall X \in P^\flat, \forall Y \in P^\sharp, \alpha(X) \sqsubseteq^\sharp Y \iff X \sqsubseteq^\flat \gamma(Y).$$

Theorem 2.4.2 (Fixpoint abstract approximation [Cou78, CC92a]) *Let* $(P^\flat, \sqsubseteq^\flat, \bot^\flat, \top^\flat, \sqcap^\flat, \sqcup^\flat)$ *and* $(P^\sharp, \sqsubseteq^\sharp, \bot^\sharp, \top^\sharp, \sqcap^\sharp, \sqcup^\sharp)$ *be two complete lattices related by a Galois connection* $P^\flat \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P^\sharp$*, and* $F^\flat \in P^\flat \to P^\flat$*,* $F^\sharp \in P^\sharp \to P^\sharp$ *be two monotonic functions.*

If $F^\sharp$ *and* $F^\flat$ *satisfy*

$$\forall X \in P^\flat, \alpha \circ F^\flat(X) \sqsubseteq^\sharp F^\sharp \circ \alpha(X),$$

then:

$$\begin{array}{l} \alpha(\mathrm{lfp}\, F^\flat) \sqsubseteq^\sharp \mathrm{lfp}\, F^\sharp, \\ \alpha(\mathrm{gfp}\, F^\flat) \sqsubseteq^\sharp \mathrm{gfp}\, F^\sharp. \end{array}$$

Widening and narrowing operators

The fixpoint approximation theorem is used to compute upper approximations of concrete fixpoints in the abstract domain. To accelerate the convergence of this computation (or to make it possible when the lattice does not satisfy the ascending chain condition), we may use widenings and narrowings [CC77].

Though most of the results presented in this thesis concerning soundness of abstract analysis are proved without considering widenings and narrowings, in general these operators aimed to accelerate convergence can be used without loss of soundness.

Backward and forward abstract analyses

From these results, $\mathcal{F}_I$ and $\mathcal{B}_F$ can be approximated:

Proposition 2.4.3 (Abstract analyses [Cou81]) *Let* $\langle \Sigma, \tau \rangle$ *be a transition system, and* $(P^\sharp, \sqsubseteq^\sharp, \bot^\sharp, \top^\sharp, \sqcap^\sharp, \sqcup^\sharp)$ *a complete lattice related to* $\wp(\Sigma)$ *by a Galois connection* $\wp(\Sigma) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P^\sharp$*.*

Let $pre^\sharp$ *and* $post^\sharp$ *be two monotonic operators such that:*

$$\forall X \subseteq \Sigma, \begin{array}{l} \alpha \circ pre(X) \sqsubseteq^\sharp pre^\sharp \circ \alpha(X), \\ \alpha \circ post(X) \sqsubseteq^\sharp post^\sharp \circ \alpha(X). \end{array}$$

Then:

$$\begin{array}{l} \mathcal{F}_I \subseteq \gamma(\mathrm{lfp}\, \lambda X.(\alpha(I) \sqcup^\sharp post^\sharp(X))), \\ \mathcal{B}_F \subseteq \gamma(\mathrm{lfp}\, \lambda X.(\alpha(F) \sqcup^\sharp pre^\sharp(X))). \end{array}$$

Using this result, we can construct approximations of the forward and backward collecting semantics.

Fixpoint combination

To check unreachability properties (i.e. to verify that a program will not reach error states F), an efficient approach is to approximate $\mathcal{F}_I \cap \mathcal{B}_F$.

To approximate $\mathcal{F}_I \cap \mathcal{B}_F$, one could intersect the approximated fixpoints. However, a better approximation has been proposed in [Cou78]. It uses the *combination* of both abstract analyses, each one being used to enhance the result of the other. Since we intend, in chapter 3, to extend this result, we give first one lemma we intend to exploit, before the standard result.

Lemma 2.4.4 ([CC92a]) *Let* $P^\flat(\sqsubseteq^\flat, \bot^\flat, \top^\flat, \sqcap^\flat, \sqcup^\flat)$ *and* $P^\sharp(\sqsubseteq^\sharp, \bot^\sharp, \top^\sharp, \sqcap^\sharp, \sqcup^\sharp)$ *be complete lattices with a Galois connection* $P^\flat \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P^\sharp$, $\mathcal{F}^\flat \in P^\flat \to P^\flat$ *and* $\mathcal{B}^\flat \in P^\flat \to P^\flat$ *be two monotonic functions, and let*

$$\mathcal{L}^\flat = \mathrm{gfp}\, \lambda Z.(Z \sqcap^\flat \mathcal{F}^\flat(Z) \sqcap^\flat \mathcal{B}^\flat(Z)).$$

If $\mathcal{F}^\sharp \in P^\sharp \to P^\sharp$ *and* $\mathcal{B}^\sharp \in P^\sharp \to P^\sharp$ *are monotonic and satisfy* $\alpha \circ \mathcal{F}^\flat \sqsubseteq^\sharp \mathcal{F}^\sharp \circ \alpha$ *and* $\alpha \circ \mathcal{B}^\flat \sqsubseteq^\sharp \mathcal{B}^\sharp \circ \alpha$, *then, with* $\mathcal{L}^\sharp = \mathrm{gfp}\, \lambda Z.(Z \sqcap^\sharp \mathcal{F}^\sharp(Z) \sqcap^\sharp \mathcal{B}^\sharp(Z))$, *we have:*

$$\mathcal{L}^\flat \subseteq^\flat \gamma(\mathcal{L}^\sharp).$$

This approximation of $\mathcal{L}^\flat$ is the best approximation we can obtain with $F^\sharp$, $B^\sharp$, $\sqcap^\sharp$ and $\sqcup^\sharp$ [CC79a].

When $\mathcal{F}^\flat = \lambda Z.\mathrm{lfp}\, \lambda X.(Z \cap (I \cup post(X)))$ and $\mathcal{B}^\flat = \lambda Z.\mathrm{lfp}\, \lambda X.(Z \cap (F \cup pre(X)))$, we have $\mathcal{L}^\flat = \mathcal{F}_I \cap \mathcal{B}_F$ [Cou78]. Hence:

Proposition 2.4.5 (Fixpoint meet approximation [Cou78]) *With the hypothesis of Proposition 2.4.3, we define:*

$$\mathcal{F}^\sharp = \lambda Z.\mathrm{lfp}\, \lambda X.(Z \sqcap^\sharp (\alpha(I) \sqcup^\sharp post^\sharp(X)))$$
$$\mathcal{B}^\sharp = \lambda Z.\mathrm{lfp}\, \lambda X.(Z \sqcap^\sharp (\alpha(F) \sqcup^\sharp pre^\sharp(X)))$$

Then $\alpha(\mathcal{F}_I \cap \mathcal{B}_F) \sqsubseteq^\sharp \mathrm{gfp}\, \lambda Z.(Z \sqcap^\sharp \mathcal{F}^\sharp(Z) \sqcap^\sharp \mathcal{B}^\sharp(Z))$.

This technique is well known and used, for example, in Bourdoncle's analyzer "Syntox" [Bou93]. This is the first example of property-driven analysis, where the specification (the set of "error" states which must be unreachable) is used in the reachability analysis (the forward one).

2.4.2 Closure operators

Another framework to express abstractions is the framework of closure operators [Cou78]. Whereas the Galois connection framework is useful to abstract concrete domains into machine-representable abstract domains, closure operators enable us to study abstractions independently of the abstract domain. More generally, closure operators are useful to modify monotone operators while keeping some

of their properties. As we will use them (specifically lower closure operators) in our presentation of property-driven analysis (chapter 5), we recall here known results about them.

In this section $\langle D, \leq, \vee, \wedge, \bot, \top \rangle$ is a complete lattice.

Definition 2.4.6 (Closure operators) *An operator ρ on D is an upper (resp. lower) closure operator if and only if it is monotonic, extensive*[3] *(resp. reductive*[4]*) and idempotent.*

To clarify notation, when lower closure operators and upper closure operators appear in the same formulas, we use overlines ($\overline{\rho}$) to denote upper closure operators.

The link between Galois connections and closure operators is given in the following proposition:

Proposition 2.4.7 ([Cou81]) *If $D^\sharp$ is a complete lattice, and $D \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} D^\sharp$ is a Galois connection, then $\overline{\rho} = \gamma \circ \alpha$ is an upper closure operator (and $\rho^\sharp = \alpha \circ \gamma$ is a lower closure operator).*

Upper closure operators and lower closure operators are dual notions. Since we intend to use both simultaneously, and lower closure operators are less widely used, we focus this presentation of already known results on lower closure operators.

It is well-known that lower (or upper) closure operators on a complete lattice form a complete lattice:

Theorem 2.4.8 ([Cou78]) *The set $lco\,(D)$ of lower closure operators on D is a complete lattice $\langle lco\,(D), \sqsubseteq, \sqcup, \sqcap, \lambda x.x, \lambda x.\bot \rangle$, with:*

$$\begin{aligned}
\rho \sqsubseteq \rho' &\iff \forall x \in D, \rho(x) \leq \rho'(x), \\
\textstyle\bigsqcup_{i\in\Delta} \rho_i &= \lambda x. \textstyle\bigvee_{i\in\Delta} \rho_i(x), \\
\textstyle\bigsqcap_{i\in\Delta} \rho_i &= \lambda x.\mathrm{lfp}\, \lambda y.(x \wedge \textstyle\bigwedge_{i\in\Delta} \rho_i(y)).
\end{aligned}$$

Dually, we will note $\langle uco\,(D), \sqsubseteq, \sqcup, \sqcap, \lambda x.\top, \lambda x.x \rangle$ the complete lattice of upper closure operators on D.

Closure operators are closely related to Moore families:

Definition 2.4.9 (Moore families) *An* upper *(resp.* lower*)* Moore family *of D is a subset L of D such that $L = \mathcal{M}^u(L) = \{\vee X \mid X \subseteq L\}$ (resp. $L = \mathcal{M}^l(L) = \{\wedge X \mid X \supseteq L\}$).*

Proposition 2.4.10 ([War42, Cou78]) *Any lower closure operator ρ is uniquely determined by the set of its fixpoints $\rho(D)$, which is an upper Moore family. Reciprocally, any upper Moore family L on D defines a lower closure operator $\rho = \lambda x. \bigvee_{y\in L}(y \leq x)$ for which L are exactly its fixpoints.*

[3] i.e. $\rho(x) \geq x$ for all x

[4] i.e. $\rho(x) \leq x$ for all x

$lco\,(D)$ and the lattice of upper Moore families in D are isomorph, with the following correspondences:

$$\begin{aligned}\rho \sqsubseteq \rho' &\iff \rho(D) \subseteq \rho'(D)\\ \bigsqcup_{i\in\Delta} \rho_i &= \mathcal{M}^u\left(\bigcup_{i\in\Delta} \rho(D)\right)\\ \bigsqcap_{i\in\Delta} \rho_i &= \bigcap_{i\in\Delta} \rho(D)\end{aligned}$$

We will use interchangeably lower closure operators and upper Moore families (and dually upper closure operators and lower Moore families). When there is no ambiguity, we will denote $\rho(D)$ by ρ itself.

Soundness and completeness

Closure operators, as abstractions, can be used to approximate fixpoints.

Proposition 2.4.11 (Soundness [CC79b]) *For all $\phi \in D \xrightarrow{m} D$ and $\rho \in lco\,(D)$, we have:*

$$\begin{aligned}&\rho \circ \phi \circ \rho \leq \rho \circ \phi\\ &\rho(\mathrm{lfp}\,\phi) \geq \mathrm{lfp}\,(\rho \circ \phi)\\ &\rho(\mathrm{gfp}\,\phi) \geq \mathrm{gfp}\,(\rho \circ \phi).\end{aligned}$$

The soundness proposition shows that the fixpoint of the abstract operator ($\mathrm{lgfp}\,(\rho \circ \phi)$) is "correct", but less precise than the abstraction of the concrete fixpoint ($\rho(\mathrm{lgfp}\,\phi)$). When the precision is the same, the abstraction (or the closure operator) is said to be *complete*:

Definition 2.4.12 (Completeness) *$\rho \in lco\,(D)$ is said to be* complete *for a monotone operator ϕ iff $\rho \circ \phi \circ \rho \geq \rho \circ \phi$ (and so $\rho \circ \phi \circ \rho = \rho \circ \phi$).*

Whereas soundness is always satisfied for all ϕ, completeness is always relative to an operator (or a set of operators). When we do not state the operator, completeness will be for ϕ.

The following proposition gives the relation between *completeness* and *fixpoint completeness*.

Proposition 2.4.13 (Fixpoint completeness) *Let ρ be a lower closure operator and ϕ a monotone operator.*

1. *If ρ is complete (with respect to ϕ), then it is* gfp-complete *(i.e. $\rho(\mathrm{gfp}\,\phi) = \mathrm{gfp}\,(\rho \circ \phi)$).*
2. *If ρ is complete* and *continuous, then it is* lfp-complete *(i.e. $\rho(\mathrm{lfp}\,\phi) = \mathrm{lfp}\,(\rho \circ \phi)$).*

(we recall that a monotone operator ϕ is continuous (resp. co-continuous) iff for all increasing (resp. decreasing) chains A in D, $\phi(\bigvee A) = \bigvee_{a\in A} \phi(a)$ (resp. $\phi(\bigwedge A) = \bigwedge_{a\in A} \phi(a)$))

For upper closures, it is well-known that completeness implies lfp-completeness. This result was first presented by Cousot and Cousot [CC79b]. However, completeness does *not* imply gfp-completeness. Thus, for lower closure operators, getting completeness is not sufficient to ensure the property of lfp-completeness.

***Example* 2.4.14** *We choose* $D = \wp(\mathbb{Z})$ *and* $\phi = \lambda X.\{0\} \cup \{x + 1 \mid x \in X\}$. *We know that* $\operatorname{lfp}\phi = [0, +\infty[$. *Let* $\rho = \{\emptyset, [0, +\infty[\}$. *Since* ρ *is an upper Moore family,* ρ *defines a lower closure operator, and one can check easily that* ρ *is complete for* ϕ. *However,* $\operatorname{lfp}\rho \circ \phi = \emptyset$, *and* ρ *is not* lfp*-complete.*

Construction of complete closures

In the construction of property-driven analysis, we need fixpoint-complete operators. Hence we detail the known results on the construction of complete (and fixpoint-complete) closures.

The problem was studied by Giacobazzi, Ranzato and Scozzari. We will use the dual of [GRS00, Thm. 5.10]:

Theorem 2.4.15 (Complete closures [GRS00]) *Let* ρ_0 *be a lower closure operator on* D. *If* ϕ *is co-continuous, then:*

$$\begin{aligned} R_\phi &= \lambda\eta.\ \mathcal{M}^u\left(\cup_{a\in\eta} \min(\phi^{-1}(\uparrow a))\right) \\ \textit{and}\quad \mathcal{R}_\phi(\rho_0) &= \operatorname{lfp}^{\sqsubseteq}_{\sqsubseteq}\ \lambda\eta.(\rho_0 \sqcup R_\phi(\eta)) \end{aligned} \tag{2.3}$$

are well defined and $\mathcal{R}_\phi(\rho_0)$ *is the lowest complete lower closure operator greater than* ρ_0[5].

When ϕ is not co-continuous, we cannot apply the min operator in this formula[6]. However, with $\min' : \wp(D) \to \wp(D)$ satisfying:

- $\forall X \subseteq D, \min'(X) \subseteq X$,
- $\forall X \subseteq D, \forall a \in X, \exists a' \in \min'(X)$ s.t. $a' \leq a$,

we can define R_ϕ and $\mathcal{R}_\phi$ as:

$$\begin{aligned} R_\phi &= \lambda\eta.\ \mathcal{M}^u\left(\cup_{a\in\eta} \min'(\phi^{-1}(\uparrow a))\right) \\ \text{and}\quad \mathcal{R}_\phi(\rho_0) &= \operatorname{lfp}^{\sqsubseteq}_{\sqsubseteq}\ \lambda\eta.(\rho_0 \sqcup R_\phi(\rho_0)) \end{aligned} \tag{2.4}$$

Then $\mathcal{R}_\phi(\rho_0)$ is a complete lower closure operator greater than ρ_0 (though it may not be minimal).

[5] $\mathcal{R}_\phi(\rho_0)$ is called as the complete shell of ρ_0.

[6] Because the formula $\forall a \in D, \forall x \in D, x \in \phi^{-1}(\uparrow a) \ \Rightarrow\ \exists y \in \min(\phi^{-1}(\uparrow a)), y \leq x$ does not hold.

From this result, we can construct gfp-complete operators. To construct lfp-complete operators, we need completeness and continuity. We can achieve this goal by using an extensive operator C on $lco\,(D)$ such that $C(lco\,(D))$ includes only continuous operators[7].

Proposition 2.4.16 *Let C be an extensive operator on $lco\,(D)$ such that, for all lower closure operator η, $C(\eta)$ is $\leq$-continuous. Then $C \circ \mathcal{R}_\phi$ (with $\mathcal{R}_\phi$ defined either by equation (2.3) or by equation (2.4)) is an extensive operator, and $\mathrm{luis}\,(C \circ \mathcal{R}_\phi, \rho_0)$ is a continuous complete lower closure operator greater than ρ_0 (thus, it is* lfp-*complete).*

***Example* 2.4.17** *We give three examples for C.*

- *As $\lambda a.a$ is continuous, $C = \lambda\eta.(\lambda a.a)$ satisfies the conditions of the proposition. This example is trivial, but it proves the existence of at least one possible operator for any lattice D.*
- *When $D = \wp\,(\Sigma)$, $\lambda a.(a \cap X)$ for $X \subseteq \Sigma$ is a continuous lower closure operator. Therefore, C defined by $C(\eta) = \lambda a.(a \cap \eta(\Sigma))$ for all η satisfies the conditions. This example is interesting, as C is an upper closure operator and all elements of $C(lco\,(D))$ can be defined by a subset of Σ. Thus, we will be able to use abstractions of $\wp\,(\Sigma)$ to abstract lower closure operators.*
- *Still when $D = \wp\,(\Sigma)$, let S be a finite subset of Σ. Then, for all $\eta \in lco\,(D)$, $C_S(\eta) = \lambda a.(a \cap \eta(a \cup (\Sigma \backslash S)))$ is a continuous lower closure operator greater than η (informally, $C_S(\eta)$ is close to $\lambda a.(a \cap \eta(\Sigma))$ "outside" S and to η "inside" S). Hence C_S satisfies the conditions needed for C (furthermore, it is also an upper closure operator on $lco\,(D)$). Of course, when $S = \emptyset$ we get the previous example, and when $S = \Sigma$, C_S is the identity on $lco\,(\wp\,(\Sigma))$ (all operators are continuous). As the precision increase with the size of S, we can control the loss of precision induced by the operator.*

[7] To our knowledge, this construction was not proposed before, hence this result may be not "already known". But it is the most intuitive solution to construct lfp-complete lower closure operators, and so we present it in this chapter.

Chapter 3

Combination for complex properties

In the previous chapter, we decided to focus on forward temporal properties to express specifications. Hence the set of states satisfying the specification can be expressed by a formula using only backward operators. We can thus easily derive from this formula an abstract backward analysis to compute an over-approximation of this set of states. This is the principle of *abstract model-checking* algorithms which involve abstractions of states [BCM+92, TXJS92, HMMR00]. However, in general this method requires precise abstractions, done by hand, to be precise enough.

A useful amelioration would be to extend the *backward-forward combination* (as it is recalled in section 2.4.1) for this kind of backward analysis.

In this chapter we study the extension of this method to more complex properties, first with one fixpoint, and then with several fixpoints.

3.1 Combination with one fixpoint

In this section, we study the extension of the combination to backward analyses expressed with one fixpoint.

Let Σ be a set of states, and $(\wp(\Sigma), \subseteq, \emptyset, \Sigma, \cap, \cup)$ is the concrete domain of the analysis. $I \subseteq \Sigma$ is the set of initial states of the program. We denote $(P^\sharp, \sqsubseteq^\sharp, \bot^\sharp, \top^\sharp, \sqcap^\sharp, \sqcup^\sharp)$ the abstract domain, related to the concrete domain by a Galois connection $\wp(\Sigma) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P^\sharp$.

To get symmetry between the concrete and the abstract domains, we define $P^\flat = \wp(\Sigma)$, $\sqsubseteq^\flat = \subseteq$, $\bot^\flat = \emptyset$, $\top^\flat = \Sigma$, $\sqcap^\flat = \cap$, $\sqcup^\flat = \cup$.

Backward semantics

From our specification, we derive a monotone backward operator $b^\flat \in \wp(\Sigma) \stackrel{m}{\to} \wp(\Sigma)$ such that the set of states satisfying the specification is equal to $S = \text{lgfp}\, b^\flat$. We define $\mathcal{B}^\flat = \lambda Z.\text{lgfp}\, \lambda X.(Z \cap b^\flat(X))$. Then $\mathcal{B}^\flat$ is a monotone operator, and $S = \mathcal{B}^\flat(\Sigma)$.

An abstract backward operator $b^\sharp$ is associated to $b^\flat$, with the relation:

$$\forall X \in \wp(\Sigma)\,,\ \alpha \circ b^\flat(X) \sqsubseteq^\sharp b^\sharp \circ \alpha(X)$$

With $\mathcal{B}^\sharp = \lambda Z.\text{lgfp}\, \lambda X.(Z \sqcap^\sharp b^\sharp(X))$, we know that

$$\forall X \in \wp(\Sigma)\,,\ \alpha \circ \mathcal{B}^\flat(X) \sqsubseteq^\sharp \mathcal{B}^\sharp \circ \alpha(X)$$

Note that we use upper approximations. This choice is imposed by our goal to apply the combination to approximate *intersections* of semantics. Lower approximations, used in abstract model-checking [HMMR00], can not be used in this case. Since efficient automatic abstractions are mostly upper abstractions, this restriction is not problematic.

Forward semantics

On the other side, we keep the forward semantics as the forward collecting semantics, as it is in the standard backward-forward combination. Thus, if $I \subseteq \Sigma$ is the set of initial states of the analysis:

$$\mathcal{F}^\flat = \lambda Z.\text{lfp}\, \lambda X.(Z \cap (I \cup \mathit{post}(X))).$$

Here, *post* is defined by the equation (2.1) or by the equation (2.2), depending on the nature of the program. To have symmetric notations, we define $f^\flat = \lambda X.(I \cup \mathit{post}(X))$.

As for the backward semantics, we suppose the existence of $f^\sharp$ such that:

$$\forall X \in \wp(\Sigma)\,,\ \alpha \circ f^\flat(X) \sqsubseteq^\sharp f^\sharp \circ \alpha(X),$$

and we define $\mathcal{F}^\sharp = \lambda Z.\text{lfp}\, \lambda X.(Z \sqcap^\sharp f^\sharp(X))$.

Combination

The concrete combination between $\mathcal{B}^\flat$ and $\mathcal{F}^\flat$ is defined as:

$$\mathcal{L}^\flat = \text{gfp}\, \lambda Z.(Z \cap \mathcal{B}^\flat(Z) \cap \mathcal{F}^\flat(Z)).$$

The abstract combination is $\mathcal{L}^\sharp = \text{gfp}\, \lambda Z.(Z \sqcap^\sharp \mathcal{B}^\sharp(Z) \sqcap^\sharp \mathcal{F}^\sharp(Z))$.

We know, by the lemma 2.4.4, that $\alpha(\mathcal{L}^\flat) \sqsubseteq^\sharp \mathcal{L}^\sharp$, and that $\mathcal{L}^\sharp$ is the best approximation of $\mathcal{L}^\flat$ we can obtain with $\mathcal{B}^\sharp$ and $\mathcal{F}^\sharp$. Thus, the combination is useful when we can use $\mathcal{L}^\flat$ to prove the specification.

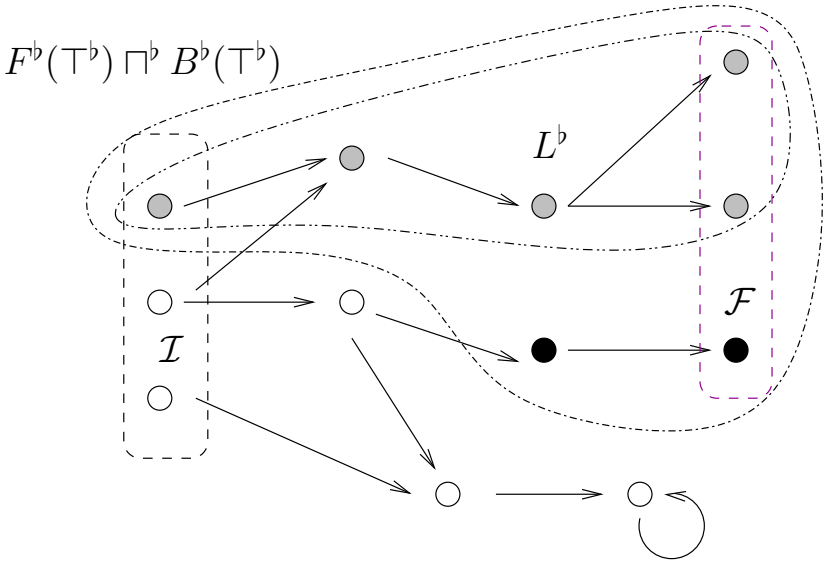


Figure 3.1: Example of transition system where $\mathcal{L}^\flat \neq \mathcal{F}^\flat(\Sigma) \cap \mathcal{B}^\flat(\Sigma)$. Here, $b^\flat = \lambda X.(F \cup \widetilde{pre}(X))$. $\mathcal{L}^\flat$ is then the set of states belonging to a trace of states which satisfy the backward property.

In the "classical" combination, $\mathcal{L}^\flat = \mathcal{B}^\flat(\Sigma) \cap \mathcal{F}^\flat(\Sigma)$. This equality does not hold in general: a state may satisfy the backward property and be reachable from an initial state which does not satisfy the backward property. However, if the goal is only the verification of the specification for the initial states, we do not need such a strict condition.

***Example* 3.1.1** *An example is given in Fig. 3.1. We want to verify that the initial states satisfy $\phi = \mu X.(F \vee \Box X)$. Then $b^\flat = \lambda X.(F \cup \widetilde{pre}(X))$. Some states satisfy ϕ but are not in $\mathcal{L}^\flat$, and the condition $\mathcal{L}^\flat = \mathcal{F}^\flat(\Sigma) \cap \mathcal{B}^\flat(\Sigma)$ is not satisfied. However, in this example, the initial states satisfying ϕ are in $\mathcal{L}^\flat$, hence we can apply the combination to verify the specification* for the initial states. *This is the principle of the extension of the combination.*

3.1.1 Conditions to use the combination

To check the temporal formula, we just need to know the set of initial states which satisfy $\mathcal{B}^\flat(\Sigma)$, that is $I \cap \mathcal{B}^\flat(\Sigma)$. Thus the combination is useful if the equality:

$$I \cap \mathcal{B}^\flat(\Sigma) = I \cap \mathcal{L}^\flat \tag{3.1}$$

is satisfied. Of course, this equality is satisfied in the classical case.

The condition (3.1) is implied by another equality, easier to check:

Lemma 3.1.2 *If*

$$\mathcal{L}^\flat = \mathcal{F}^\flat(\mathcal{B}^\flat(\Sigma)), \tag{3.2}$$

then the equation (3.1) holds.

Proof. As $\mathcal{L}^\flat = \mathcal{F}^\flat(\mathcal{B}^\flat(\Sigma)) = \mathrm{lfp}\,\lambda X.(\mathcal{B}^\flat(\Sigma) \cap (f^\flat(X)))$, it is clear that $\mathcal{L}^\flat \subseteq \mathcal{B}^\flat(\Sigma)$. Moreover, the first iteration of the least fixpoint is $\mathcal{B}^\flat(\Sigma) \cap f^\flat(\emptyset)$, which is equal to $\mathcal{B}^\flat(\Sigma) \cap I$. So we have $\mathcal{B}^\flat(\Sigma) \cap \mathcal{I} \subseteq \mathcal{L}^\flat$, which proves the equation. □

Using the definition of $\mathcal{B}^\flat$ as $\lambda Z.\mathrm{lgfp}\lambda X.(Z \cap b^\flat(X))$, we deduce the following theorem, which is used as the principal condition to extend the combination:

Theorem 3.1.3 *If:*

$$\forall (X, Y) \in \wp(\Sigma)^2, Y \subseteq b^\flat(X) \Leftrightarrow Y \subseteq b^\flat(X \cap post(Y)), \tag{3.3}$$

then the hypothesis of Lemma 3.1.2 holds. Thus, equation (3.1) holds.

Proof. We note that the hypothesis implies:

$$\forall (X, Y) \in \wp(\Sigma)^2, Y \subseteq b^\flat(X) \Leftrightarrow Y \subseteq b^\flat(X \cap f^\flat(Y)). \tag{3.4}$$

We want to prove that $\mathcal{L}^\flat = \mathcal{F}^\flat(\mathcal{B}^\flat(\Sigma))$. Left-to-right inclusion is the consequence of the optimality of $\mathcal{L}^\flat$:

$$\begin{array}{rcl} \mathcal{L}^\flat & = & \mathrm{gfp}\,\lambda Z.(Z \cap \mathcal{F}^\flat(Z) \cap \mathcal{B}^\flat(Z)) \\ & \subseteq & \mathcal{F}^\flat(\mathcal{F}^\flat(\Sigma) \cap \mathcal{B}^\flat(\Sigma)) \cap \mathcal{B}^\flat(\mathcal{F}^\flat(\Sigma) \cap \mathcal{F}^\flat(\Sigma)) \\ & \subseteq & \mathcal{F}^\flat(\mathcal{B}^\flat(\Sigma)) \end{array}$$

Thus, to prove the equality, we need to check that $\mathcal{F}^\flat(\mathcal{B}^\flat(\Sigma))$ is a fixpoint of $\lambda Z.(Z \cap \mathcal{F}^\flat(Z) \cap \mathcal{B}^\flat(Z))$, that is, to prove that $\mathcal{F}^\flat(\mathcal{B}^\flat(\Sigma)) \subseteq \mathcal{F}^\flat(\mathcal{F}^\flat(\mathcal{B}^\flat(\Sigma)))$ and $\mathcal{F}^\flat(\mathcal{B}^\flat(\Sigma)) \subseteq \mathcal{B}^\flat(\mathcal{F}^\flat(\mathcal{B}^\flat(\Sigma)))$.

The former inequality is true because $\mathcal{F}^\flat \circ \mathcal{F}^\flat = \mathcal{F}^\flat$. To prove the latter, we define $\Omega = \mathcal{F}^\flat(\mathcal{B}^\flat(\Sigma))$ and $\Omega' = \mathcal{B}^\flat(\mathcal{F}^\flat(\mathcal{B}^\flat(\Sigma)))$. Here we must distinguish between least and greatest fixpoint for $\mathcal{B}^\flat$.

- if lgfp = lfp, let (X_n), $n \geq 0$ be the (transfinite) iteration sequence starting from $\emptyset$ for $b^\flat$. We will prove that for all $n \geq 0$, $\Omega \cap X_n \subseteq \Omega'$. This is true if $n = 0$, because $X_0 = \emptyset$.

  If n is a successor ordinal, since $X_n = b^\flat(X_{n-1})$, we have $\Omega \cap X_n \subseteq b^\flat(X_{n-1})$. Using the hypothesis of the theorem (equation (3.4)), we obtain $\Omega \cap X_n \subseteq b^\flat(X_{n-1} \cap f^\flat(\Omega \cap X_n))$.

  Moreover, $f^\flat(\Omega \cap X_n) \subseteq f^\flat(\Omega)$, and $X_{n-1} \subseteq \mathcal{B}^\flat(\Sigma)$. Thus:

$$\begin{array}{rcll} X_{n-1} \cap f^\flat(\Omega \cap X_n) & \subseteq & X_{n-1} \cap \mathcal{B}^\flat(\Sigma) \cap f^\flat(\Omega) & \\ & \subseteq & X_{n-1} \cap \Omega & \text{as } \Omega = \mathcal{B}^\flat(\Sigma) \cap f^\flat(\Omega) \end{array}$$

Hence, if the inequality holds for $n-1$:

$$X_{n-1} \cap f^\flat(\Omega \cap X_n) \subseteq \Omega'$$

Thus $\Omega \cap X_n$ is included in $\Omega \cap b^\flat(\Omega')$, which is equal to Ω' by definition of Ω'.

When n is a limit ordinal ($b^\flat$ may be not continuous), if $\Omega \cap X_{n'} \subseteq \Omega'$ for all $n' < n$, then $\Omega \cap X_n = \Omega \cap \bigcup_{n'<n} X_{n'} \subseteq \Omega'$.

Thus, by transfinite induction, $\Omega \cap X_n \subseteq \Omega'$ for all n. As the upper bound of (X_n) is $\mathcal{B}^\flat(\Sigma)$, which includes Ω, we have $\Omega \subseteq \Omega'$.

- if lgfp = gfp, let X_n, $n \geq 0$ be the (transfinite) iteration sequence starting from Σ for $\lambda X.(\Omega \cap b^\flat)$. The limit of X_n is Ω'. $X_1 = \Omega \cap b^\flat(\Sigma) = \Omega$, since $\Omega \subseteq \mathcal{B}^\flat(\Sigma) \subseteq b^\flat(\Sigma)$. Moreover, since $\mathcal{B}^\flat(\Sigma) = b^\flat(\mathcal{B}^\flat(\Sigma))$, $\Omega \subseteq b^\flat(\mathcal{B}^\flat(\Sigma))$, so $\Omega \subseteq b^\flat(\mathcal{B}^\flat(\Sigma) \cap f^\flat(\Omega))$. As $\mathcal{B}^\flat(\Sigma) \cap f^\flat(\Omega) = \Omega$, we have $\Omega \subseteq b^\flat(\Omega)$, and $X_2 = \Omega$. Thus $X_n = \Omega$ for all $n \geq 1$, and $\Omega' = \Omega$.

□

3.1.2 Applications the several specifications

Theorem 3.1.3 gives an interesting condition to prove the correctness of the combination.

Proposition 3.1.4 *We suppose that $\langle \Sigma, \tau \rangle$ is a transition system modelling the program.*

When $b^\flat = \lambda X.(A \cup (B \cap pre(X)) \cup (C \cap \widetilde{pre}(X)))$, the equality (3.1) holds.

Proof. We want to prove that the equation (3.3) is satisfied.

If $Y \subseteq b^\flat(X)$, then, $\forall y \in Y$:

- if $y \in A$, then $y \in b^\flat(X \cap post(Y))$.
- if $y \in B \cap pre(X)$, then $\exists x \in X$ such that $\langle y, x \rangle \in \tau$. Therefore, since $y \in Y$, $x \in post(Y)$, and $y \in B \cap pre(X \cup post(Y))$.

  Thus $y \in b^\flat(X \cap post(Y))$.
- if $y \in C \cap \widetilde{pre}(X)$, then $\forall x \in \Sigma$, $\langle y, x \rangle \in \tau \Rightarrow x \in X$. As $y \in Y$, $\langle y, x \rangle \in \tau \Rightarrow x \in post(Y)$, so $\forall x \in \Sigma$, $\langle y, x \rangle \in \tau \Rightarrow x \in X \cap post(Y)$.

  Thus $y \in C \cap \widetilde{pre}(X \cup post(Y))$, so $y \in b^\flat(X \cap post(Y))$.

Therefore, $Y \subseteq b^\flat(X) \Rightarrow Y \subseteq b^\flat(X \cap post(Y))$. The other side of the equivalence is automatic. Thus the hypothesis of Thm. 3.1.3 is satisfied, and equation (3.1) holds. □

So we can use the backward-forward combination to enhance the verification of properties of the form of: $\overset{\mu}{\nu} X.(A \vee (B \wedge \Diamond X) \vee (C \wedge \Box X))$. These properties are

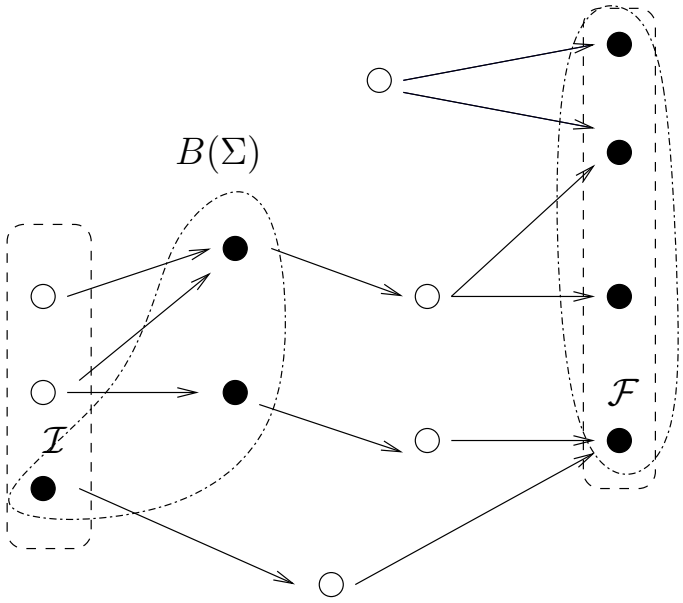


Figure 3.2: Example of specification for which combination with a reachability analysis does not work. $b^\flat = \lambda X.F \cup pre(pre(X))$. Whereas there is a initial state which satisfies the specification, $\mathcal{L}^\flat = \emptyset$.

interesting: they allow to distinguish between different kinds of non-determinism ("controllable" and "uncontrollable" non-determinism), and they include all the basic CTL constructions.

For game logics, a similar result holds:

Proposition 3.1.5 *We suppose that $\langle \mathcal{P}, \Sigma, \Delta \rangle$ is an alternating transition system modelling the program.*

When $b^\flat = \lambda X.(A \cup \bigcup_{I \in \wp(\mathcal{P})}(B_I \cap CPre_I(X)) \cup \bigcup_{I \in \wp(\mathcal{P})}(C_I \cap UPre_I(X)))$, the equality (3.1) holds.

Proof. The proof is essentially the same as for the non-game case. □

Unfortunately, the extension of this technique to the whole μ-calculus does not work: for example, a formula like $\mu X.(F \vee \Diamond\Diamond X)$ leaves "holes" in traces, preventing combination with forward analysis (cf. figure 3.2).

But, from the results we got for specifications with one fixpoint, we can develop an algorithm to extend the combination for the verification of complex specifications with several fixpoints, including the whole CTL.

3.2 Extension to formulas with several fixpoints

3.2.1 Transition system case

Our program is represented by the transition system $\langle \Sigma, \tau \rangle$.

Following the result of proposition 3.1.4, our specification language is the set of μ-calculus formulas generated by the grammar:

$$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Box\varphi \mid \Diamond\varphi \mid \frac{\mu}{\nu} X.(\varphi \vee (\varphi \wedge \Diamond X) \vee (\varphi \wedge \Box X)).$$

It is worth noting that all these formulas are closed, and the defined temporal logic includes CTL. Obviously, the logic does not change if we replace $\frac{\mu}{\nu} X.(\varphi \vee (\varphi \wedge \Diamond X) \vee (\varphi \wedge \Box X))$ with $\frac{\mu}{\nu} X.(\varphi \wedge (\varphi \vee \Diamond X) \wedge (\varphi \vee \Box X))$ in the above grammar.

Our goal is to obtain a "good" upper approximation $\Omega_\varphi(\alpha(I))$ of $\alpha(I \cap [\![\varphi]\!])$, using the backward-forward combination to enhance fixpoint computations. We assume that for all atomic proposition p, we have an upper approximation $[\![p]\!]^\sharp$ of $\alpha([\![p]\!])$[1] (and an upper approximation $[\![\neg p]\!]^\sharp$ of $\alpha([\![\neg p]\!])$).

We need abstractions of *pre*, *post* and $\widetilde{pre}$, and we will respectively denote them by $pre^\sharp$,$post^\sharp$ and $\widetilde{pre}^\sharp$. These abstractions must satisfy, for all $X \subseteq \Sigma$:

$$\begin{array}{rcl} \alpha \circ pre(X) & \sqsubseteq^\sharp & pre^\sharp \circ \alpha(X) \\ \alpha \circ \widetilde{pre}(X) & \sqsubseteq^\sharp & \widetilde{pre}^\sharp \circ \alpha(X) \\ \alpha \circ post(X) & \sqsubseteq^\sharp & post^\sharp \circ \alpha(X) \end{array}$$

Moreover, since we will need to abstract sets of states reachable from other sets, we will denote by $post^*$ and $post^{*\sharp}$ the functions $\lambda X.\mathrm{lfp}\,\lambda Y.(X \cup post(Y))$ and $\lambda X.\mathrm{lfp}\,\lambda Y.(X \sqcup^\sharp post^\sharp(Y))$ respectively. Then, of course, $\alpha \circ post^*(X) \sqsubseteq^\sharp post^{*\sharp} \circ \alpha(X)$ for all $X \subseteq \Sigma$.

To simplify notations we denote by $\mathcal{L}^\natural_{\mathrm{lgfp}}(f, g)$, with $\natural \in \{\flat, \sharp\}$, the expression:

$$\mathrm{gfp}\,\lambda Z.(\mathrm{lfp}\,\lambda X.(Z \sqcap^\natural f(X)) \sqcap^\natural \mathrm{lgfp}\,\lambda X.(Z \sqcap^\natural g(X)))$$

$\mathcal{L}^\natural_{\mathrm{lgfp}}(f, g)$ is the limit of the decreasing chain Z_n defined by $Z_0 = \top^\natural$, $Z_{2n+1} = Z_{2n} \sqcap^\natural \mathrm{lfp}\,\lambda X.(X_{2n} \sqcap^\natural f(X))$ and $Z_{2n+2} = Z_{2n+1} \sqcap^\natural \mathrm{lgfp}\,\lambda X.(X_{2n+1} \sqcap^\sharp g(X))$.

[1] which may be $\alpha([\![p]\!])$, if it is computable.

If φ is a formula, we can now define $\Omega_\varphi \in P^\sharp \to P^\sharp$ as follows:

$$\begin{array}{rcl}
\Omega_p(S) & = & [\![p]\!]^\sharp \sqcap^\sharp S \\
\Omega_{\varphi_1 \wedge \varphi_2}(S) & = & \mathrm{gfp}\, \lambda X.(S \sqcap^\sharp \Omega_{\varphi_1}(X) \sqcap^\sharp \Omega_{\varphi_2}(X)) \\
\Omega_{\varphi_1 \vee \varphi_2}(S) & = & \Omega_{\varphi_1}(S) \sqcup^\sharp \Omega_{\varphi_2}(S) \\
\Omega_{\Box\varphi}(S) & = & \widetilde{pre}^\sharp(\Omega_\varphi(post^\sharp(S))) \\
\Omega_{\Diamond\varphi}(S) & = & pre^\sharp(\Omega_\varphi(post^\sharp(S))) \\
\Omega_{\frac{\mu}{\nu} X.(\varphi_1 \vee (\varphi_2 \wedge \Diamond X) \vee (\varphi_3 \wedge \Box X))}(S) & = & \mathcal{L}^\sharp_{\mathrm{lgfp}}(\ \lambda X.(S \sqcup^\sharp post^\sharp(X)), \\
 & & \quad \lambda Y.(\ \Omega_{\varphi_1}(post^{*\sharp}(S)) \\
 & & \qquad \sqcup^\sharp \Omega_{\varphi_2}(post^{*\sharp}(S)) \sqcap^\sharp pre^\sharp(Y) \\
 & & \qquad \sqcup^\sharp \Omega_{\varphi_3}(post^{*\sharp}(S)) \sqcap^\sharp \widetilde{pre}^\sharp(Y)))
\end{array}$$

It is worth noting that even if we replace $post^{*\sharp}(S)$ by $\top^\sharp$ in the last line, we still have to compute it as the first iteration that leads to $\mathcal{L}^\sharp_{\mathrm{lgfp}}$. However, this replacement may not change the final result of Ω_φ and makes the computation much faster (because the computation of $\Omega_\varphi(\top^\sharp)$ can be simplified). The following theorem is valid with or without the replacement:

Theorem 3.2.1 *For any formula φ, and $I \subseteq \Sigma$:*

$$\alpha(I \cap [\![\varphi]\!]) \sqsubseteq^\sharp \alpha(I) \sqcap^\sharp \Omega_\varphi(\alpha(I))$$

Thus the computation of $\Omega_\phi(\alpha(I))$ (or of an upper approximation of $\Omega_\phi(\alpha(I))$ due to the use of widenings and narrowings) gives a sound approximation of $\alpha(I \cap [\![\varphi]\!])$.

Proof. The proof is by induction on the structure of φ. By monotony of α, it is obvious that $\alpha(I \cap [\![\varphi]\!]) \sqsubseteq^\sharp \alpha(I)$, so we need to prove that $\alpha(I \cap [\![\varphi]\!]) \sqsubseteq^\sharp \Omega_\varphi(\alpha(I))$.

If $\varphi = p$, by monotony of α:

$$\alpha(I \cap [\![p]\!]) \sqsubseteq^\sharp \alpha(I) \sqcap^\sharp \alpha([\![p]\!]) \sqsubseteq^\sharp \alpha(I) \sqcap^\sharp [\![p]\!]^\sharp \sqsubseteq^\sharp \Omega_p(\alpha(I))$$

If $\varphi = \varphi_1 \wedge \varphi_2$:

$$\begin{array}{rcl}
\alpha(I \cap [\![\varphi]\!]) & = & \alpha(I \cap [\![\varphi]\!] \cap [\![\varphi_1]\!] \cap [\![\varphi_2]\!]) \\
 & \sqsubseteq^\sharp & \alpha(I) \sqcap^\sharp \alpha(I \cap [\![\varphi]\!] \cap [\![\varphi_1]\!]) \sqcap^\sharp \alpha(I \cap [\![\varphi]\!] \cap [\![\varphi_2]\!]) \\
 & \sqsubseteq^\sharp & \alpha(I) \sqcap^\sharp \Omega_{\varphi_1}(\alpha(I) \cap [\![\varphi]\!]) \sqcap^\sharp \Omega_{\varphi_2}(\alpha(I \cap [\![\varphi]\!]))
\end{array}$$

Thus $\alpha(I \cap [\![\varphi]\!]) \sqsubseteq^\sharp \mathrm{gfp}\, \lambda X.(\alpha(I) \sqcap^\sharp \Omega_{\varphi_1}(X) \sqcap^\sharp \Omega_{\varphi_2}(X))$, which proves that $\alpha(I \cap [\![\varphi]\!]) \sqsubseteq^\sharp \Omega_\varphi(I)$.

If $\varphi = \varphi_1 \vee \varphi_2$:

$$\begin{array}{rcll}
\alpha(I \cap [\![\varphi]\!]) & = & \alpha(I \cap ([\![\varphi_1]\!] \cup [\![\varphi_2]\!])) & \\
 & = & \alpha(I \cap [\![\varphi_1]\!]) \sqcup^\sharp \alpha(I \cap [\![\varphi_2]\!]) & \text{since } \alpha \text{ is additive [CC77]} \\
 & \sqsubseteq^\sharp & \Omega_{\varphi_1}(\alpha(I)) \sqcup^\sharp \Omega_{\varphi_2}(\alpha(I)) & \\
 & \sqsubseteq^\sharp & \Omega_\varphi(\alpha(I)) &
\end{array}$$

If $\varphi = \Box\varphi_1$, we have $I \subseteq \widetilde{pre}(post(I))$, so:

$$\begin{array}{rcl} \alpha(I \cap [\![\varphi]\!]) & \sqsubseteq^\sharp & \alpha(\widetilde{pre}(post(I) \cap [\![\varphi_1]\!])) \\ & \sqsubseteq^\sharp & \widetilde{pre}^\sharp(\alpha(post(I) \cap [\![\varphi_1]\!])) \\ & \sqsubseteq^\sharp & \widetilde{pre}^\sharp(\Omega_{\varphi_1}(\alpha \circ post(I))) \\ & \sqsubseteq^\sharp & \widetilde{pre}^\sharp(\Omega_{\varphi_1}(post^\sharp \circ \alpha(I))) \\ & \sqsubseteq^\sharp & \Omega_\varphi(\alpha(I)) \end{array}$$

If $\varphi = \Diamond\varphi_1$, using the inequality $I \cap pre([\![\varphi_1]\!]) \subseteq pre(post(I) \cap [\![\varphi_1]\!])$, we can do the same calculus.

If $\varphi = \underset{\nu}{\mu} X.(\varphi_1 \vee (\varphi_2 \wedge \Diamond X) \vee (\varphi_3 \wedge \Box X))$, let us define $h^\flat = \lambda X.([\![\varphi_1]\!] \cup [\![\varphi_2]\!] \cap pre(X) \cup [\![\varphi_3]\!] \cap \widetilde{pre}(X))$. Then $[\![\varphi]\!] = \mathrm{lgfp}\, \lambda X.h^\flat(X)$.

We will use Lemma 2.4.4 with

$$\begin{array}{rcl} f^\flat & = & \lambda X.(I \cup post(X)) \\ f^\sharp & = & \lambda X.(\alpha(I) \sqcup^\sharp post^\sharp(X)) \\ b^\flat & = & \lambda X.(post^*(I) \cap ([\![\varphi_1]\!] \cup [\![\varphi_2]\!] \cap pre(X) \cup [\![\varphi_3]\!] \cap \widetilde{pre}(X))) \\ b^\sharp & = & \lambda X.(\alpha(post^*(I) \cap [\![\varphi_1]\!]) \\ & & \sqcup^\sharp \alpha(post^*(I) \cap [\![\varphi_2]\!]) \sqcap^\sharp pre^\sharp(X) \\ & & \sqcup^\sharp \alpha(post^*(I) \cap [\![\varphi_3]\!]) \sqcap^\sharp \widetilde{pre}^\sharp(X)) \end{array}$$

It is clear that $\alpha \circ b^\flat \circ \gamma \sqsubseteq^\sharp b^\sharp$ and $\alpha \circ f^\flat \circ \gamma \sqsubseteq^\sharp f^\sharp$ (given the standard properties that α is additive and $\alpha \circ \gamma$ is a lower closure operator [CC77]). Thus we have $\alpha(\mathcal{L}^\flat_{\mathrm{lgfp}}(f^\flat, b^\flat)) \sqsubseteq^\sharp \mathcal{L}^\sharp_{\mathrm{lgfp}}(f^\sharp, b^\sharp)$.

First, we prove that $I \cap \mathcal{L}^\flat_{\mathrm{lgfp}}(f^\flat, b^\flat) = I \cap [\![\varphi]\!]$. We have:

$$\mathcal{L}^\flat_{\mathrm{lgfp}}(f^\flat, h^\flat) \subseteq \mathrm{lgfp}\, \lambda X.((\mathrm{lfp}\, \lambda Y.f^\flat(Y)) \cap h^\flat(X)) \subseteq [\![\varphi]\!]$$

Applying theorem 3.1.3 with $h^\flat$, we obtain $I \cap [\![\varphi]\!] = I \cap \mathcal{L}^\flat_{\mathrm{lgfp}}(f^\flat, h^\flat)$, so $I \cap [\![\varphi]\!] = I \cap \mathrm{lgfp}\, \lambda X.((\mathrm{lfp}\, \lambda Y.f^\flat(Y)) \cap h^\flat(X)) = I \cap \mathrm{lgfp}\, \lambda X.b^\flat(X)$.

Then, applying theorem 3.1.3, with $b^\flat$ this time, we obtain: $I \cap \mathrm{lgfp}\, \lambda X.b^\flat(X) = I \cap \mathcal{L}^\flat_{\mathrm{lgfp}}(f^\flat, b^\flat)$, proving the equality. So we proved:

$$\begin{array}{rcl} \alpha(I \cap [\![\varphi]\!]) & = & \alpha(I \cap \mathcal{L}^\flat_{\mathrm{lgfp}}(f^\flat, b^\flat)) \\ & \sqsubseteq^\sharp & \alpha(I) \sqcap^\sharp \alpha(\mathcal{L}^\flat_{\mathrm{lgfp}}(f^\flat, b^\flat)) \\ & \sqsubseteq^\sharp & \alpha(I) \sqcap^\sharp \mathcal{L}^\sharp_{\mathrm{lgfp}}(f^\sharp, b^\sharp) \end{array}$$

Now we need to check that:

$$\mathcal{L}^\sharp_{\mathrm{lgfp}}(f^\sharp, b^\sharp) \sqsubseteq^\sharp \Omega_\varphi(\alpha(I))$$

By the induction hypothesis, we see that $\Omega_\varphi(\alpha(I)) = \mathcal{L}^\sharp_{\mathrm{lgfp}}(f^\sharp, b'^\sharp)$ with $b'^\sharp$ satisfying $b^\sharp \sqsubseteq^\sharp b'^\sharp$, which completes the proof.

□

Hence, we got an algorithm which uses combination for specifications with several fixpoints. In this algorithm, compared with an abstract model-checking algorithm which use only the backward abstract operators, new fixpoints appear for $\wedge$ and $\frac{\mu}{\nu}$. In both cases, we can take more imprecise approximations to reduce the complexity of the analysis (for example, using a very rough approximation of $S \sqcap^\sharp \Omega_{\varphi_1} \sqcap^\sharp \Omega_{\varphi_2}$ and then make only one iteration of the greatest fixpoint to approximate $\Omega_{\varphi_1 \wedge \varphi_2}(S)$). Still, we have the trade-off between precision and speed.

3.2.2 Alternating transition system case

Now our program is modelled by an alternating transition system $\langle \mathcal{P}, \Sigma, \Delta \rangle$. Π is a set of atomic propositions, and $\pi : \Pi \to \wp(\Sigma)$ the interpretation of the elements of Π.

Abstract operators

To verify $A\mu$-formulas, we need to abstract the operators. For each p in Π, let $\pi^\sharp(p)$ be an element of $P^\sharp$ such that $\pi(p) \subseteq \gamma(\pi^\sharp(p))$[2], and $\pi^\sharp(\overline{p})$ an element of $P^\sharp$ such that $\Sigma \backslash \pi(p) \subseteq \gamma(\pi^\sharp(\overline{p}))$.

For each subset I of $\mathcal{P}$, we define the abstract controllable predecessor relation $CPre_I^\sharp \in P^\sharp \to P^\sharp$ and the abstract uncontrollable predecessor relation $UPre_I^\sharp \in P^\sharp \to P^\sharp$. These relations must satisfy, $\forall X \subseteq \Sigma$:

$$\begin{aligned} \alpha \circ CPre_I(X) &\sqsubseteq^\sharp CPre_I^\sharp \circ \alpha(X) \\ \alpha \circ UPre_I(X) &\sqsubseteq^\sharp UPre_I^\sharp \circ \alpha(X) \end{aligned}$$

To make the backward-forward combination, we need an abstract successor operator for forward analysis. This *abstract successor relation* $post^\sharp$ must satisfy:

$$\alpha \circ post(X) \sqsubseteq^\sharp post^\sharp \circ \alpha(X)$$

Again, we define $post^{*\sharp} = \lambda X.\text{lfp}\, \lambda Y.(X \sqcup^\sharp post^\sharp(Y))$. One can easily check that:

$$\alpha \circ post^*(\sigma) \sqsubseteq^\sharp post^{*\sharp} \circ \alpha(\sigma)$$

We consider the closed $A\mu$ formulas φ generated by the grammar:

$$\begin{aligned} \varphi \quad ::= \quad & p \mid \neg p \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \langle\langle I \rangle\rangle \bigcirc \varphi_1 \mid [\![I]\!] \bigcirc \varphi_1 \\ & \mid \frac{\mu}{\nu} x.(\varphi \vee \bigvee_{I \in \wp(\mathcal{P}) \setminus \{\emptyset\}} (\varphi_I \wedge \langle\langle I \rangle\rangle \bigcirc x) \vee \bigvee_{I' \in \wp(\mathcal{P}) \setminus \{\emptyset\}} (\varphi_{I'} \wedge [\![I']\!] \bigcirc x)). \end{aligned}$$

As before, the last term of the grammar can be rewritten exchanging $\vee$ and $\wedge$ without modifying the expressivity of the logic.

[2] If $\alpha(\pi(p))$ is computable, we can take $\pi^\sharp(p) = \alpha(\pi(p))$

As for the non-game case, we can now define, if φ is a formula, $\Omega_\varphi \in P^\sharp \to P^\sharp$ as follows:

$$\begin{aligned}
\Omega_p(S) &= \pi^\sharp(p) \sqcap^\sharp S \\
\Omega_{\neg p}(S) &= \pi^\sharp(\overline{p}) \sqcap^\sharp S \\
\Omega_{\varphi_1 \wedge \varphi_2}(S) &= \mathrm{gfp}\, \lambda X (S \sqcap^\sharp \Omega_{\varphi_1}(X) \sqcap^\sharp \Omega_{\varphi_2}(X)) \\
\Omega_{\varphi_1 \vee \varphi_2}(S) &= \Omega_{\varphi_1}(S) \sqcup^\sharp \Omega_{\varphi_2}(S) \\
\Omega_{\langle\!\langle I \rangle\!\rangle \bigcirc \varphi}(S) &= \mathit{CPre}^\sharp_I(\Omega_\varphi(\mathit{post}^\sharp(S))) \\
\Omega_{[\![I]\!] \bigcirc \varphi}(S) &= \mathit{UPre}^\sharp_I(\Omega_\varphi(\mathit{post}^\sharp(S)))
\end{aligned}$$

$$\Omega_{\mu \atop \nu} x.(\varphi \vee \bigvee_I (\varphi_I \wedge \langle\!\langle I \rangle\!\rangle \bigcirc x) \vee \bigvee_{I'} (\varphi_{I'} \wedge [\![I']\!] \bigcirc x))(S) =$$

$$\begin{array}{lll}
\mathcal{L}^\sharp_{\mathrm{lgfp}}(& \lambda X.(S \sqcup^\sharp \mathit{post}^\sharp(X)), & \\
& \lambda Y.(& \Omega_\varphi(\mathit{post}^{*\sharp}(S)) \\
& & \sqcup^\sharp \bigsqcup^\sharp_I (\Omega_{\varphi_I}(\mathit{post}^{*\sharp}(S)) \sqcap^\sharp \mathit{CPre}^\sharp(Y)) \\
& & \sqcup^\sharp \bigsqcup^\sharp_{I'} (\Omega_{\varphi_{I'}}(\mathit{post}^{*\sharp}(S)) \sqcap^\sharp \mathit{UPre}^\sharp(Y))))
\end{array}$$

Theorem 3.2.2 *For all formulas φ generated by the grammar above, and $I \subseteq Q$:*

$$\alpha(I \cap [\![\varphi]\!]) \sqsubseteq^\sharp \alpha(I) \sqcap^\sharp \Omega_\varphi(\alpha(I))$$

Proof. The proof is essentially the same of the non-game case.

All we need are the equalities $I \cap \mathit{UCPre}_I([\![\varphi_1]\!]) \subseteq \mathit{UCPre}_I(\mathit{post}(I \cap [\![\varphi_1]\!]))$ with $\mathit{UCPre} = \mathit{UPre}$ or CPre, and the equivalence:

$$Y \subseteq b^\flat(X) \Longleftrightarrow Y \subseteq b^\flat(X \cap \mathit{post}(Y))$$

with $b^\flat = \lambda X.(A \cup \bigcup_I (B_I \cap \mathit{CPre}_I(X)) \cup \bigcup_{I'} (C_{I'} \cap \mathit{UPre}_{I'}(X)))$. These properties are quite easy to check. □

3.3 A simple example

We illustrate the combination with a very short and easy example. We will analyse the small non-deterministic program shown in Fig. 3.3.

In this figure, `x`, `n` are integers, `(random in [0,1])` returns a random integer in $[0, 1]$, and `(input in [0,1])` returns a integer in $[0, 1]$ entered by the user (these commands behave in the same way in the transition relation). Control point (0) is the program entry, we differentiate it from control point (1), which is the while loop entry.

With initial condition `x=1` at control point (0), we will try to prove that the user cannot be sure to have `x=0` at control point (9), that is, the initial condition satisfies $\nu x.((\neg A) \wedge (B \vee \Diamond x) \wedge (C \vee \Box x))$ with A meaning that `x=0` at control point (9), C being the set of states at control point (2), and B being the set of states at other control points.

```
(0) { x = 1 }
(1)
    while (n>0) do {
(2)
       if (random in [0,1]=0) then
(3)
          x = x * n;
(4)
       else
(5)
          x = x * (n-1);
(6)
       fi
(7)
       n = n - (input in [0,1]);
(8)
    }
(9)
```

Figure 3.3: A short non-deterministic program.

As we use an upper approximation, we take the negation of the proposition, that is (knowing that $\neg B = C$) : $\mu\, x.(A \vee (B \wedge \Diamond X) \vee (C \wedge \Box X))$. So we must approximate lfp $\lambda x.([\![A]\!] \cup ([\![B]\!] \cap \mathit{pre}(x)) \cup ([\![C]\!] \cap \widetilde{\mathit{pre}}(x)))$.

We will use *interval analysis* [CC76], with the improvement of the results of local decreasing iterations [Gra92] for assignments in the backward analysis.

We must abstract $\mathit{post}(X)$, $\mathit{pre}(X)$ and $\widetilde{\mathit{pre}}(X)$. Abstract operators may be described as systems of semantics equations [Cou78, Cou81]. The program is almost deterministic, and $\widetilde{\mathit{pre}}^\sharp$ is very close to $\mathit{pre}^\sharp$. The differences appear at control points (2) and (7), but we only need to express it at control point (2), with the equation:

$$P_2 = P_3 \sqcap P_5$$

($\sqcap$ being the intersection of abstract environments).

The table of Fig. 3.4 gives the results with a single forward analysis ($\mathcal{F}^\sharp(\top^\sharp)$), a single backward analysis ($\mathcal{B}^\sharp(\top^\sharp)$), the intersection of both analyses ($\mathcal{F}^\sharp(\top^\sharp) \sqcap^\sharp \mathcal{B}^\sharp(\top^\sharp)$), and the first iteration of combination ($\mathcal{B}^\sharp(\mathcal{F}^\sharp(\top^\sharp))$).

The next iteration of the combination will lead to $\emptyset$ everywhere, which is of course the abstract fixpoint $\mathcal{L}^\sharp$. So $\mathcal{L}^\flat = \emptyset$ (which is not equal to $\mathcal{F}^\flat(\top^\flat) \cap \mathcal{B}^\flat(\top^\flat)$). As, for this kind of temporal property, $\mathcal{L}^\flat \cap I = I \cap \mathcal{B}^\flat(\top^\flat)$, we obtained the expected result.

Lab. (var.)	$\mathcal{F}^\sharp(\top^\sharp)$	$\mathcal{B}^\sharp(\top^\sharp)$	$\mathcal{F}^\sharp(\top^\sharp) \sqcap^\sharp \mathcal{B}^\sharp(\top^\sharp)$	$\mathcal{B}^\sharp(\mathcal{F}^\sharp(\top^\sharp))$
(0) x:	$[1]$	$[-\infty, +\infty]$	$[1]$	$\emptyset$
n:	$[-\infty, +\infty]$	$[-\infty, +\infty]$	$[-\infty, +\infty]$	$\emptyset$
(1) x:	$[0, +\infty]$	$[-\infty, +\infty]$	$[0, +\infty]$	$[0]$
n:	$[-\infty, +\infty]$	$[-\infty, +\infty]$	$[-\infty, +\infty]$	$[-\infty, +\infty]$
(2) x:	$[0, +\infty]$	$[-\infty, +\infty]$	$[0, +\infty]$	$[0]$
n:	$[1, +\infty]$	$[-\infty, +\infty]$	$[1, +\infty]$	$[1, +\infty]$
(3) x:	$[0, +\infty]$	$[-\infty, +\infty]$	$[0, +\infty]$	$[0]$
n:	$[1, +\infty]$	$[-\infty, +\infty]$	$[1, +\infty]$	$[1, +\infty]$
(4) x:	$[0, +\infty]$	$[-\infty, +\infty]$	$[0, +\infty]$	$[0]$
n:	$[1, +\infty]$	$[-\infty, +\infty]$	$[1, +\infty]$	$[1, +\infty]$
(5) x:	$[0, +\infty]$	$[-\infty, +\infty]$	$[0, +\infty]$	$[0, +\infty]$
n:	$[1, +\infty]$	$[-\infty, +\infty]$	$[1, +\infty]$	$[1, +\infty]$
(6) x:	$[0, +\infty]$	$[-\infty, +\infty]$	$[0, +\infty]$	$[0]$
n:	$[1, +\infty]$	$[-\infty, +\infty]$	$[1, +\infty]$	$[1, +\infty]$
(7) x:	$[0, +\infty]$	$[-\infty, +\infty]$	$[0, +\infty]$	$[0]$
n:	$[1, +\infty]$	$[-\infty, +\infty]$	$[1, +\infty]$	$[1, +\infty]$
(8) x:	$[0, +\infty]$	$[-\infty, +\infty]$	$[0, +\infty]$	$[0]$
n:	$[0, +\infty]$	$[-\infty, +\infty]$	$[0, +\infty]$	$[0, +\infty]$
(9) x:	$[0, +\infty]$	$[0]$	$[0]$	$[0]$
n:	$[-\infty, +\infty]$	$[-\infty, +\infty]$	$[-\infty, +\infty]$	$[-\infty, +\infty]$

Figure 3.4: Result of the analysis of the program.

3.4 Conclusion

With the results developed in this chapter, we can extend all the advantages of the known combination to large classes of temporal properties. Existing analyses can be modified to include this combination without much trouble. In particular, we can use all state-based abstractions. Furthermore, the algorithms we presented can be simplified to make a trade-off between precision and speed. The gain in precision is significant, especially when we use widenings and narrowings, as it can change the strategy of the widening.

To implement an analyzer which uses this approach, over-approximations of the $\widetilde{pre}$ operator (or equivalent operators in the game logic) are needed. This is not difficult to find an efficient ones for non-relational abstractions. However, with this kind of abstractions, the combination is not very powerful, because we can not take into account the real independence of the random generators (cf. chapter 4). Relational abstractions are very costly, and abstractions of "forced" backward operators (like $\widetilde{pre}$) are hard to develop.

The development of better forward analyses (reachability analysis is not precise enough) is the subject of the following chapters.

Chapter 4

Tree semantics

In the previous chapter, we saw how to combine backward and forward state-based analyses for complex properties. However, the forward analysis used is only a reachability analysis, independent of the property we want to satisfy.

In this chapter, we illustrate with an example why this reachability analysis is not sufficient when non-relational abstractions are used. To solve this problem, a semantical approach is proposed. An extension of transition systems, called "extended transition systems", is defined in order to take the specification into account in the semantics of the program. Then forward and backward semantics (based on sets of trees) for these systems are defined, and the links between them are shown. We study the combination of these semantics in this new framework, and we give examples of constructions of extended transition systems from programs and specifications.

4.1 Limitation of the combination

We will start by the very simple program in an imperative language presented in Fig. 4.1.

In this program, `input` describes a non-deterministic function controllable by

(1) Initial state: $I: \ x = y =$ uninit.

```
x:= input in {0,1}
```

(2)

```
y:= random in {0,1}
```

(3)

```
x:= x+y
```

(4) Final states: $F: \ x = 1$

Figure 4.1: Simple program

Label	Forward	Combination
(1) x:	non-init.	non-init.
y:	non-init.	non-init.
(2) x:	$[0,1]$	$[0,1]$
y:	non-init.	non-init.
(3) x:	$[0,1]$	$[0,1]$
y:	$[0,1]$	$[0,1]$
(4) x:	$[0,2]$	$[1]$
y:	$[0,1]$	$[0,1]$

Figure 4.2: Analysis of the program of Fig. 4.1. Each line describes the result for a program point and a variable (as the analysis is non-relational). The column "Forward" gives the result for the reachability analysis; the column "Combination" gives the combination of this result with a backward analysis. The result of the combination is a fixpoint of both analysis.

the user, `random` a non-deterministic, uncontrollable function, without any notion of probabilities.

We want to know if, from the initial state (program point (1), `x` and `y` uninitialised), the user can go to final states in F. We can describe this property as $R \neq \emptyset$, with:

$$R \;=\; I \cap \operatorname{lfp} \lambda X.(F \cup (\Sigma_3 \cap \widetilde{\operatorname{pre}}(X)) \cup (\Sigma_2 \cap \operatorname{pre}(X)))$$

Σ_i being the set of states at program point (i). R is the set of the initial states from which the user is able to reach a final state.

The combination of backward and forward analyses in the framework of abstract interpretation for this kind of temporal property was described in the previous chapter. If S is the result of the analysis, then $I \cap S \supseteq R$, so that $I \cap S \neq \emptyset$ is a necessary condition for the property $R \neq \emptyset$ to be satisfied. In particular, $I \cap S = \emptyset$ implies that $R = \emptyset$, so that in that case the property is unsatisfiable.

We use the interval analysis [CC76], which is quite imprecise but very fast (precise relational analyses like polyhedra [CH78] are too expensive for large programs).

The result is given in the table (we start by a forward analysis) described in Fig. 4.2.

We get the combination after the first backward analysis, and the result is too imprecise. We can get more informations in two distinct ways:

- We may give a relation between x and y at point (2), describing the information, given by backward analysis, that we must have $0 \leq x + y \leq 1$. This approach would lead to relational domains. In this particular case, octagons [Min01] would be sufficient, but more complex cases would need complex relational analyses. Thus we try another approach.

(1) Initial state: $I: \ x = y =$ non-init.

```
x:= random in {0,1}
```

(2)

```
y:= input in {0,1}
```

(3)

```
x:= x+y
```

(4) Final states: $F: \ x = 1$

Figure 4.3: Modified program.

- It may be better if the information given by the forward analysis enables to give $\emptyset$ already at program point (3) with the backward analysis. As the command `y := random in {0,1}` in the program does not use `x`, the intuition would be not to modify `x` during the analysis of this command, but during the analysis of `x := y+x`. To get this result, we must know that at point (3), `y` can take any value in $\{0, 1\}$ whatever the value of `x`, and regardless of the choices of the user. On the contrary, the value of `x` depends on the choices of the user. This distinction must appear in the forward semantics.

The forward analysis derives from the trace semantics, which is the most general semantics we can have for a transition relation and a set of initial states. This semantics, which gives a set of traces, does not include the context of the execution of the program (here, the difference between `input` and `random`). Our idea is to make several sets of traces, each set describing a "strategy" of the "user" (with the same initial state). Each set of traces forms a tree, so we get a set of trees. With the program used as an example, we get two trees (described in Fig. 4.4), and none of those has all its leaves satisfying $x = 1$. Thus, there is no way to force $x = 1$ at program point (4).

We can modify the program by swapping the `random` command and the `input` command. The new program is given in Fig. 4.3. In this case, the choice of the "user" takes place in two possible situations ($x = 0$ or $x = 1$ at program point (2)). As it has two possibilities for this choice, we have four possible strategies. Therefore, we get four trees (see Fig. 4.4 for detail), each tree expressing a strategy by determining the choices the "user" would make in all situations. One of these trees has all its leaves satisfying $x = 1$, which shows that there is a winning strategy for the "user".

This example is a form of game. However, this approach is not specifically related to games. The goal is to prove a temporal property, and each tree may be seen as a potential "proof" of the property. The presence of a winning strategy is merely the proof of the temporal property.

In the following sections, we develop this approach. We must begin with some definitions and notations about trees (or, more accurately, about free trees).

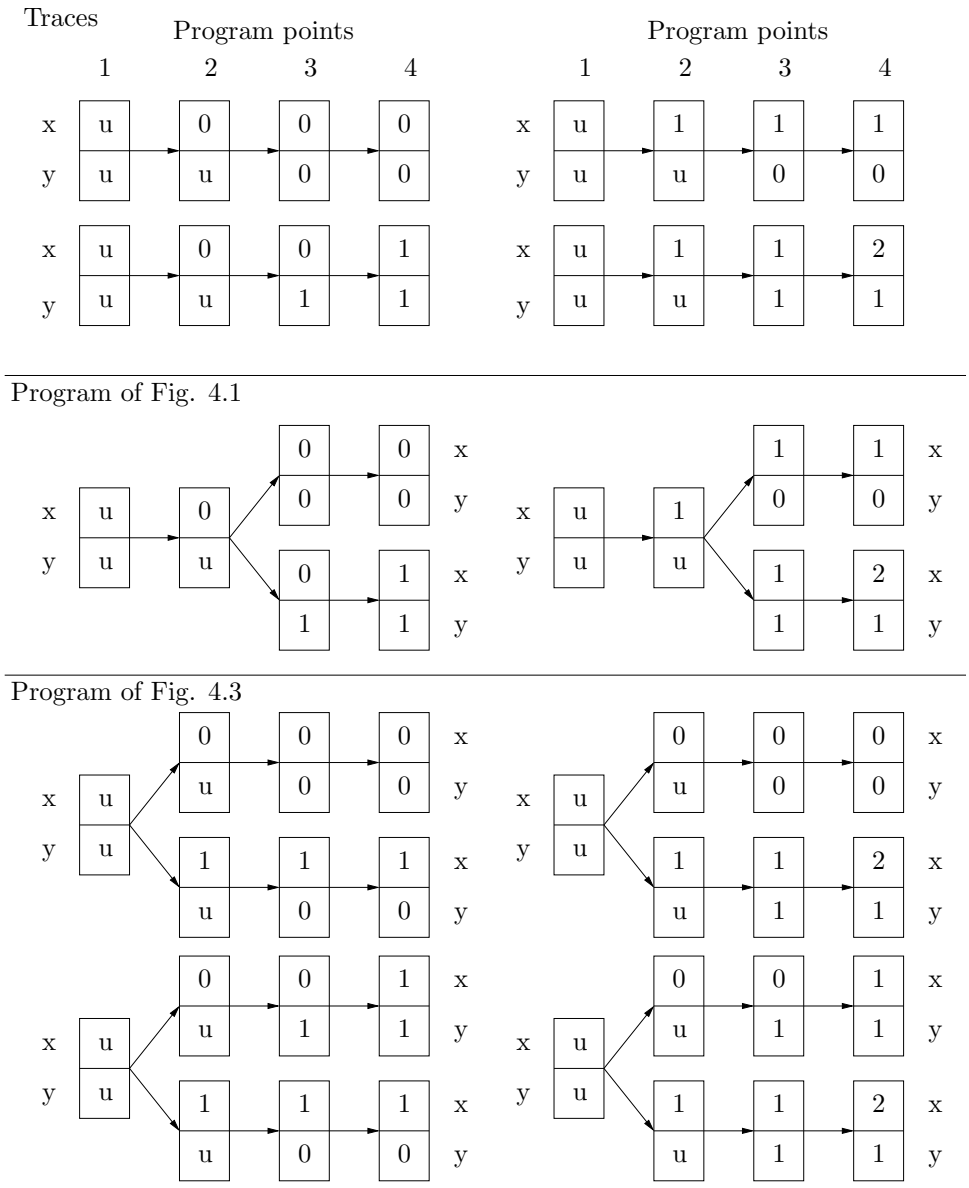


Figure 4.4: Traces generated by the programs of Fig. 4.1 and Fig. 4.3. Each state is described by its program point (given by its column) and the value of x and y. The traces which describe the same "strategy" for the user are put together in a tree for both program. For the program of Fig. 4.3, each trace appears in two trees and we get four trees.

4.2 Trees

Transition systems semantics traditionally use sets of traces [Cou02]. In order to combine the transition relation with the temporal property being checked, we will use sets of free trees. The set of sets of free trees will be the semantical domain for extended transition systems. The following section gives some definitions and notations about this domain.

4.2.1 Free trees

We will note Σ the set of states, Σ^* the set of finite sequences (or traces) of elements of Σ, Σ^ω the set of infinite sequences, and Σ^∞ the union of Σ^* and Σ^ω. ϵ denotes the empty sequence.

We note $\preceq$ the prefix order on Σ^∞. The dot . is the concatenation operator between two sequences. It is lifted to sets of sequences in the following way:

$$\forall u \in \Sigma^*, \forall V \subseteq \Sigma^\infty,\ u.V = \{u.v \mid v \in V\},$$
$$\forall U \subseteq \Sigma^*, \forall V \subseteq \Sigma^\infty,\ U.V = \{u.V \mid u \in U\}.$$

Hence, with the notations used, $u.V \subseteq \Sigma^\infty$ and $U.V \subseteq \wp(\Sigma^\infty)$.

In the following, σ, σ' will always be states, and u, v will always be (possibly empty) sequences. For example, $u.\sigma$ will denote a (non-empty) finite sequence that ends with σ.

Definition 4.2.1 (Free tree) *A* free tree *labelled by Σ is a non-empty prefix-closed subset of Σ^* with only one "root" (sequence of length one)*[1].

All the trees used in this article will be free trees. As we will never use the empty sequence ϵ of a free tree, we may omit it as well when describing a free tree.

t being a tree, we note $\bar{t}$ the closure of t in Σ^∞, $root(t)$ the "root" of t, and $branch(t)$ the set of maximum elements of $\bar{t}$. We will name the elements of $branch(t)$ the *branches* of t. A *leaf* of t is the last element of a finite branch of t. The set of leaves of t will be noted $leaf(t)$.

We denote by $\mathcal{T}$ the set of free trees labelled by Σ, and $\Theta = \wp(\mathcal{T})$.

Definition 4.2.2 (Subtree) *Let t be a tree, and $u.\sigma \in t$, we define the* subtree *of t rooted at $u.\sigma$ as:*

$$t_{[u.\sigma]} = \{\sigma.v \mid u.\sigma.v \in t\}.$$

Then $t_{[u.\sigma]}$ is a tree rooted by σ. The set of subtrees of a tree t is denoted $subtrees(t)$.

Definition 4.2.3 (Well-founded tree) *A free tree t labelled by Σ is said to be well-founded iff $branch(t) \subseteq \Sigma^*$.*

[1] This definition excludes the empty tree $\{\epsilon\}$.

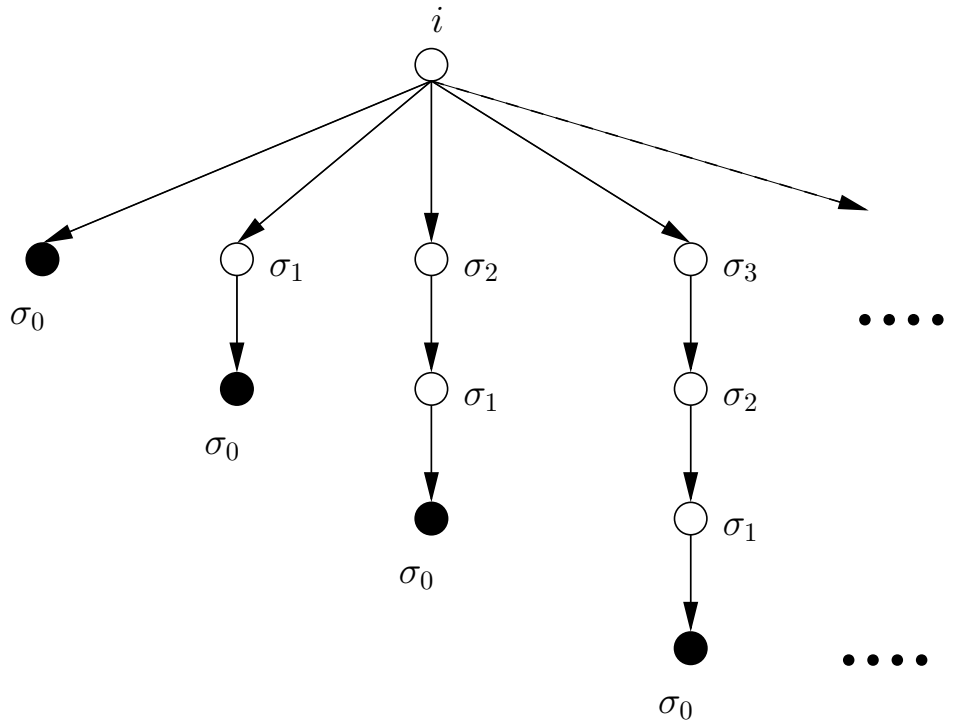


Figure 4.5: An infinite well-founded tree. $\Sigma = \{i, \sigma_n \mid n \geq 0\}$, and $t = \{i\sigma_k\sigma_{k-1}\ldots\sigma_l \mid 0 \leq l \leq k\}$. The depth of the tree is unbounded, but it has no infinite branch.

A well-founded tree does not have any infinitely increasing chains of sequences.

A tree with finite arity is well-founded if and only if it is finite. However, we will deal mostly with infinite arity trees, and the distinction between finite and well-founded must be made (an infinite but well-founded tree is presented in Fig. 4.5). We will note $\mathcal{T}_{WF}$ the set of well-founded trees labelled by Σ.

In order to define some sets of trees by conditions on their infinite branches, or on their well-founded subtrees, we introduce some notations:

Definition 4.2.4 *With $\theta \subseteq \mathcal{T}_{WF}$, we note $\mathcal{T}_\theta$ the set of trees t such that all well-founded subtrees of t are in θ:*

$$\mathcal{T}_\theta = \{t \in \mathcal{T} \mid \mathit{subtrees}(t) \cap \mathcal{T}_{WF} \subseteq \theta\};$$

Definition 4.2.5 *With $S \subseteq \Sigma^\omega$, we note $\mathcal{T}^S$ the set of trees t such that each infinite branch of t has a suffix in S:*

$$\mathcal{T}^S = \{t \in \mathcal{T} \mid \forall u \in \mathit{branch}(t) \cap \Sigma^\omega, \exists v \in S, u = u'.v\}.$$

4.2.2 Orders on sets of trees

$\subseteq$ is a partial order on $\mathcal{T}$. However, this order is too imprecise: we will define a "prefix" order $\preceq_{\mathcal{T}}$ on trees (as in [MT01]), such that $t \preceq_{\mathcal{T}} t'$ if t' extends t only on leaves of t:

$$t \preceq_{\mathcal{T}} t' \iff t \subseteq t' \wedge (\forall u' \in branch(t'), \exists u \in branch(t), u \preceq u').$$

We denote by $\sqsubseteq$ the preorder on Θ defined as:

$$\theta \sqsubseteq \theta' \iff (\forall\, t' \in \theta',\ \exists\, t \in \theta,\ t \preceq_{\mathcal{T}} t')$$

Note that $\emptyset$ is the only supremum in $(\Theta, \sqsubseteq)$ (and $\theta \subseteq \theta' \Rightarrow \theta \sqsupseteq \theta'$).

As $\sqsubseteq$ is a preorder, it is not usable to define semantics as fixpoints of iterative sequences. Rather than quotienting Θ, we will restrict ourselves to a subset of Θ in which $\sqsubseteq$ will be a partial order. The subset used will be the sets of non-comparable trees where all comparable branches of each tree are equal: we can see the branches of the trees as traces, and all traces must be equally complete in all the trees where they appear. With this restriction (stronger than the quotienting), we can define a least upper bound for increasing sequences of sets.

Proposition 4.2.6 *We note:*

$$\Theta_{NC} = \{\theta \in \Theta \mid \forall (t,t') \in \theta^2, \forall u \in branch(t), \quad \forall u' \in branch(t'), \\ u \preceq u' \Rightarrow u = u'\}.$$

And, $(\theta_i)_{i\in\mathbb{N}}$ being an increasing chain of Θ_{NC}, we define $\sqcup(\theta_i)$ as:

$$t \in \sqcup(\theta_i) \iff \exists (t_0, t_1, \ldots) \in \theta_0 \times \theta_1 \times \ldots, \\ t_0 \preceq_{\mathcal{T}} t_1 \preceq_{\mathcal{T}} \ldots \wedge\ t = \cup(t_i).$$

Then $\Theta_{NC}\langle \sqsubseteq, \bot, \sqcup\rangle$ is a complete partial order *(cpo) with $\bot = \{\{\sigma\} \mid \sigma \in \Sigma\}$.*

Proof. First, let us check that $\sqsubseteq$ is a partial order, i.e. it is anti-symmetric.

Let θ, θ' be in Θ_{NC} such that $\theta \sqsubseteq \theta'$ and $\theta' \sqsubseteq \theta$. For all t' in θ', there exists t in θ such that $t \preceq_{\mathcal{T}} t'$. Then there exists t'' in θ' such that $t'' \preceq_{\mathcal{T}} t \preceq_{\mathcal{T}} t'$. Thus $t'' \preceq_{\mathcal{T}} t'$, which means that each branch of t' has a prefix which is a branch of t''. But t'' and t' are both in θ', so all their comparable branches are equal. Hence, all branch of t' is a branch of t'', and since $t'' \subseteq t'$, $t'' = t' = t$. Thus, $\theta' \subseteq \theta$ and, by symmetry, $\theta = \theta'$. So $\sqsubseteq$ is anti-symmetric, it is a partial order.

Now, we must prove that $\sqcup$ is an inductive join.

Let $(\theta_i)_{i\in\mathbb{N}}$ be an increasing chain of Θ_{NC}. It is obvious that $\sqcup(\theta_i) \sqsupseteq \theta_j$ for all j.

Let us prove that $\sqcup(\theta_i)$ is in Θ_{NC}: let t, t' be trees in $\sqcup(\theta_i)$, and $u \in branch(t)$, $u' \in branch(t')$ such that $u \preceq u'$. Since $t \in \sqcup(\theta_i)$, it exists an increasing chain $(t_0, t_1, \ldots) \in \theta_0 \times \theta_1 \times \ldots$ such that $t_0 \preceq_{\mathcal{T}} t_1 \preceq_{\mathcal{T}} \ldots$ and $t = \cup(t_i)$. Similarly, it exists an increasing chain $(t'_0, t'_1, \ldots) \in \theta_0 \times \theta_1 \times \ldots$ such that $t'_0 \preceq_{\mathcal{T}} t'_1 \preceq_{\mathcal{T}} \ldots$ and

$t' = \cup(t'_i)$. Then, for all i, we can find $u_i \in branch(t_i)$ and $u'_i \in branch(t'_i)$ such that $u_i \preceq u$ and $u'_i \preceq u'$. Hence u_i and u'_i are comparable, so they are equal. Furthermore, u (resp. u') is the limit of (u_i). So $u = u'$, which proves that $\sqcup(\theta_i) \in \Theta_{NC}$.

Then, let θ be an element of Θ_{NC} such that $\forall j \in \mathbb{N}, \theta_j \sqsubseteq \theta$.

Let t be a tree of θ. For all $j \in \mathbb{N}$, there exists $t_j \in \theta_j$ such that $t_j \preceq_{\mathcal{T}} t$. We will prove that (t_j) form an increasing chain of trees for $\preceq_{\mathcal{T}}$.

Let i and j be two integers such that $i < j$. Let t^i_j be a tree of θ_i such that $t^i_j \preceq_{\mathcal{T}} t_j$ (it exists because $\theta_i \sqsubseteq \theta_j$). Then $t^i_j \preceq_{\mathcal{T}} t$, and each branch u of t has a prefix u^i_j which is a branch of t^i_j. But u has also a prefix u_j which is a branch of t_i. Since t_i and t^i_j are both in θ_i and u^i_j and u_j are comparable as prefix of the same sequence, $u^i_j = u_j$. As t_i and t^i_j are subset of t, all their branches are prefix of branches of t. Hence $t_i = t^i_j$.

We conclude that $(t_i)_{i\in\mathbb{N}}$ forms a $\preceq_{\mathcal{T}}$-increasing chain of trees. Thus, $t' = \cup(t_i)$ is in $\sqcup(\theta_i)$. It is clear that $t' \preceq_{\mathcal{T}} t$. Thus $\sqcup(\theta_i) \sqsubseteq \theta$, which proves that $\sqcup(\theta_i)$ is the least upper bound of (θ_i). □

We will therefore be able to define semantics as limit of increasing chains for $\sqsubseteq$ in Θ_{NC}. In the next section, we define the extension of the transition systems we want to study, and forward and backward semantics for this extension.

4.3 Extended transition system

4.3.1 Definitions

As a transition relation associates a set of successors to each state, an extended transition relation will associate several sets of successors to each state.

Definition 4.3.1 (Extended transition relation and system) *Σ being a set of states, an extended transition relation τ on Σ is a subset of $\Sigma \times \wp(\Sigma)$, or an element of $\Sigma \rightarrow \wp(\wp(\Sigma))$. An extended transition system is a pair $\langle \Sigma, \tau \rangle$, where τ is an extended transition relation on Σ.*

In general, we will use the functional form for extended transition relations. We can interpret an extended transition system in two ways: first as a game with two players (the user and the machine), such that from a state σ, the user choose the set of potential next states in $\tau(\sigma)$, and the machine arbitrarily takes one state in the chosen set. The second approach is related to logic: each set in $\tau(\sigma)$ is an alternative way (expressed as a set of requirements) to prove σ. In this approach, $\tau(\sigma) = \{\emptyset\}$ means that σ is an "axiom", whereas $\tau(\sigma) = \emptyset$ means that σ is "false".

Each extended transition relation can be described as a set of "elementary" trees of depth 1 or 2, in the following way:

Definition 4.3.2 (Elementary trees) *τ being an extended transition relation, we denote by* $\mathrm{elem}(\tau)$ *the set of trees:*

$$\mathrm{elem}(\tau) = \{\{\sigma,\ \sigma.\sigma' \mid \sigma' \in S\} \mid \sigma \in \Sigma,\ S \in \tau(\sigma)\}$$

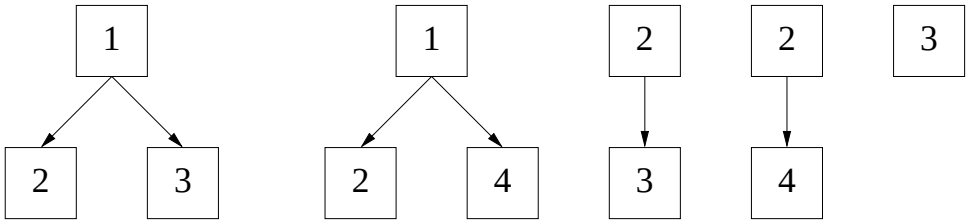


Figure 4.6: The elementary trees of the extended transition system $\langle \Sigma, \tau \rangle$ with $\Sigma = \{1, 2, 3, 4\}$ and $\tau(1) = \{\{2, 3\}, \{2, 4\}\},\ \tau(2) = \{\{3\}, \{4\}\},\ \tau(3) = \{\emptyset\},\ \tau(4) = \emptyset$.

elem(τ) gives a graphical description of the extended transition relation τ (see Fig. 4.6 for example).

4.3.2 Forward semantics

From a set of trees, we make a forward step by appending elementary trees to the leaves of the trees:

Definition 4.3.3 (Forward operator) *The forward operator F of an extended transition system $\langle \Sigma, \tau \rangle$ is:*

$$\begin{array}{rcl} F : \Theta & \to & \Theta \\ \theta & \mapsto & \{\ t' \ \mid \exists t \in \theta, \\ & & \exists s_{u.\sigma} \in \tau(\sigma) \text{ for all } u.\sigma \in branch(t) \text{ s.t.} \\ & & t' = t \cup \{u.\sigma.\sigma' \mid u.\sigma \in branch(t), \sigma' \in s_{u.\sigma}\}\}. \end{array}$$

$F(\theta)$ is created by appending (coherently) an elementary tree to each finite branch of each tree of θ. We can see that F can also "remove" a tree when a leaf σ satisfies $\tau(\sigma) = \emptyset$. F is, of course, a morphism for the union $\cup$. A simple forward semantics of (Σ, τ) is therefore $\mathrm{lfp}\, \lambda X.(\mathcal{I} \cup F(X))$, where $\mathcal{I} = \{\{i\} \mid i \in I\}$ is the set of initial states described as trees:

Definition 4.3.4 (Partial forward semantics) *The partial forward semantics $\mathcal{F}^p(I)$ of an extended transition system $\langle \Sigma, \tau \rangle$ with initial states $I \subseteq \Sigma$ is defined as:*

$$\mathcal{F}^p(I) = \mathrm{lfp}^{\subseteq}_{\emptyset}\, \lambda X.(\mathcal{I} \cup F(X))$$

where $\mathcal{I} = \{\{i\} \mid i \in I\}$.

In the trace semantics point of view, this semantics gives the partial traces of the transition system starting from the initial states.

A more expressive semantics would give only the maximal trees coherent with the transition system. We can achieve this goal by using the order $\sqsubseteq$ on the set Θ_{NC} we defined before. To ensure the stability of Θ_{NC} by F, we need a condition on τ:

Proposition 4.3.5 *If an extended transition system $\langle \Sigma, \tau \rangle$ satisfies:*

$$\forall \sigma \in \Sigma : \ \emptyset \in \tau(\sigma) \implies \tau(\sigma) = \{\emptyset\}, \tag{4.1}$$

then F defines a $\sqsubseteq$-monotonic and extensive function from Θ_{NC} to Θ_{NC}.

Proof. We need to prove that for all $\theta \in \Theta_{NC}$, $F(\theta)$ is in Θ_{NC}.

Let θ be an element of Θ_{NC}, t, t' be trees of $F(\theta)$, and $u.\sigma$ (resp. $u'.\sigma'$) be a branch of t (resp. t'). We suppose that $u.\sigma \preceq u'.\sigma'$. We want to prove that $u.\sigma = u'.\sigma'$.

Let t_0 (resp. t'_0) be an element of θ such that $t \in F(\{t_0\})$ (resp. $t' \in F(\{t'_0\})$). Then, either $u.\sigma$ (resp. $u'.\sigma'$) is a branch of t_0 (resp. t'_0), or u (resp. u') is a branch of t_0 (resp. t'_0). Let us examine the different cases:

- $u.\sigma$ is a branch of t_0 and $u'.\sigma'$ is a branch of t'_0. Since $u.\sigma$ and $u'.\sigma'$ are comparable, and t_0, t'_0 are in θ, $u.\sigma = u'.\sigma'$.
- $u.\sigma$ is a branch of t_0 and u' is a branch of t'_0. Then $u.\sigma = u'$. Since $u.\sigma$ is a branch of t_0 and of t, by definition of F, $\emptyset \in \tau(\sigma)$. Since $u.\sigma$ is a branch of t'_0 and $u.\sigma.\sigma'$ a branch of t', $\tau(\sigma) \neq \{\emptyset\}$, and we have a contradiction.
- u is a branch of t_0 and $u'.\sigma'$ is a branch of t'_0. Then $u = u'.\sigma'$, which is impossible since we supposed that $u.\sigma \preceq u'.\sigma'$.
- u is a branch of t_0 and u' is a branch of t'_0. Then $u = u'$. The only possibility to get $u.\sigma$ and $u'.\sigma'$ comparable is then $\sigma = \sigma'$. Then $u.\sigma = u'.\sigma'$.

Thus, F defines a function from Θ_{NC} to Θ_{NC}. Proving the monotonicity and extensivity of F is easy. □

This condition says that a potential final state is always final. Note that a tree which has a leaf σ with $\tau(\sigma) = \emptyset$ will disappear in the next iteration of F. This is compatible with the order $\sqsubseteq$, which has $\emptyset$ as a supremum.

Under these conditions, we can define a maximal forward semantics on an extended transition system (we recall that $\mathrm{lfp}_a^{\sqsubseteq} F$ is the $\sqsubseteq$-lowest fixpoint of F greater than or equal to a):

Definition 4.3.6 (Maximal forward semantics) *The maximal forward semantics $\mathcal{F}^m(I)$ of an extended transition system $\langle \Sigma, \tau \rangle$ with initial states $I \subseteq \Sigma$ is defined as:*

$$\mathcal{F}^m(I) = \mathrm{lfp}_{\mathcal{I}}^{\sqsubseteq} F$$

with $\mathcal{I} = \{\{i\} \mid i \in I\}$.

We can note that F is $\sqcup$-continuous, so $\mathcal{F}^m(I)$ is reached with at most ω iterations. An example of the maximal forward semantics is given in Fig. 4.7.

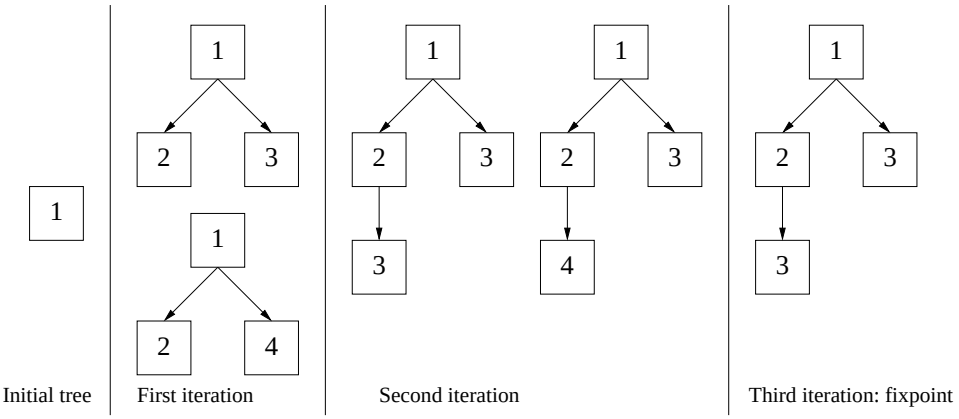


Figure 4.7: The iterations and the maximal forward semantics of the extended transition system given in Fig. 4.6, with $I = \{1\}$.

Coherent trees

Before examining the backward semantics, let us see what is exactly a tree in $\mathcal{F}^m(I)$. We define the notion of *coherent* trees with respect to an extended transition system.

Definition 4.3.7 (Coherent tree) *A tree t is* coherent *with the extended transition system $\langle \Sigma, \tau \rangle$ if:*

$$\forall u.\sigma \in t,\ \exists s \in \tau(\sigma) \text{ such that } u.\sigma.\sigma' \in t \Leftrightarrow \sigma' \in s.$$

We note $\mathcal{C}_\tau$ the set of the trees that are coherent with respect to $\langle \Sigma, \tau \rangle$. A tree is coherent with ϕ when for each node labelled by σ, the set of direct successors of this node is in $\tau(\sigma)$. Each leaf σ of this tree satisfies $\{\emptyset\} \in \tau(\sigma)$.

The link between $\mathcal{C}_\tau$ and $\mathcal{F}^m(I)$ is clear:

Lemma 4.3.8 *$\mathcal{F}^m(I)$ is the set of trees coherent with $\langle \Sigma, \tau \rangle$ the roots of which are in I.*

Proof. We denote by $\mathcal{C}_\tau^I$ the set of trees coherent with $\langle \Sigma, \tau \rangle$ which roots are in I. For all t in $\mathcal{C}_\tau^I$, we note: $t_n = \{u \in t \mid \mathit{length}(t) \leq n\}$ for all $n \geq 1$.

If we denote by $F^n(\mathcal{I})$ the successive iterations of F starting with $\mathcal{I} = \{\{i\} \mid i \in I\}$, it is easy to prove by induction that $t_n \in F^{n-1}(\mathcal{I})$ for all n. Then $t \in \mathcal{F}^m(I)$, and $\mathcal{C}_\tau^I \subseteq \mathcal{F}^m(I)$.

On the other hand, let t be a tree of $\mathcal{F}^m(I)$, and $u.\sigma \in t$. $u.\sigma$ is a branch of a tree t_0 which appears in the iterations of $F^n(\mathcal{I})$ as a "prefix" of t. Let t_1 the element of $F(\{t_0\})$ which is a "prefix" of t too. Then $\exists s \in \tau(\sigma)$ such that $u.\sigma.\sigma' \in t_1 \Leftrightarrow \sigma' \in s$.

As the successive iterations from t_1 do not make the tree "grow" from $u.\sigma$, we are sure that t is coherent in $u.\sigma$.

Thus, $t \in \mathcal{C}^I$, and $\mathcal{C}^I_\tau = \mathcal{F}^m(I)$. □

4.3.3 Backward semantics

There are two possible approaches to define the backward semantics: a least fixpoint, which only gives well-founded trees, or a greatest fixpoint, which gives all maximal trees. Both will be described here. The possibility of using both with a bi-inductive definition, as we can do with traces, will be discussed afterwards.

Definition 4.3.9 (Backward operator) *The backward operator B of an extended transition system $\langle \Sigma, \tau \rangle$ is:*

$$\begin{array}{rcl} B : \Theta & \rightarrow & \Theta \\ \theta & \mapsto & \{t \in \theta \mid \sigma = root(t) \\ & & \exists S \in \tau(\sigma), \exists t_s \in \theta \text{ for all } s \in S \text{ with } root(t_s) = s, \\ & & t = \sigma.(\cup_{s \in S} t_s)\}. \end{array}$$

$B(\theta)$ are the trees created by appending the trees of θ to an elementary tree of τ. B is monotonic and a complete $\cap$-morphism (hence it is co-continuous). However, B is not continuous.

To define the maximal trace semantics (as in [CC92c]), we start from all final states (states which have no successor). Here we will start with states σ such that $\tau(\sigma) = \{\emptyset\}$ (on the contrary, states σ such that $\tau(\sigma) = \emptyset$ intuitively correspond to error states). Thus we note:

$$f_\tau = \{\sigma \in \Sigma \mid \tau(\sigma) = \{\emptyset\}\}$$

With traces, we define the maximal finite trace semantics (with a least fixpoint), and a maximal trace semantics (finite or infinite) with a greatest fixpoint. Here we define the maximal well-founded backward semantics, and the maximal backward semantics.

Definition 4.3.10 (Maximal well-founded backward semantics) *The maximal well-founded backward semantics is defined as:*

$$\mathcal{B}^m_{WF} = \mathrm{lfp}^{\subseteq} \lambda X.(f_\tau \cup B(X)).$$

Note that, as B is not continuous, the least fixpoint may not be reached after ω iterations. This may occur in the case of unbounded non-determinism.

The correspondence between $\mathcal{B}^m_{WF}$ and coherent trees is described by the lemma:

Lemma 4.3.11 $\mathcal{B}^m_{WF} = \mathcal{C}_\tau \cap \mathcal{T}_{WF}$.

Proof. We only give a sketch of the proof.

It is easy to check that $f_\tau \subseteq \mathcal{C}_\tau$, and that if $\theta \subseteq \mathcal{C}_\tau$, then $B(\theta) \subseteq \mathcal{C}_\tau$. Thus $\mathcal{B}^m_{WF} \subseteq \mathcal{C}_\tau$.

Now, let t be a well-founded and coherent tree. We associate to each u in t an ordinal o_u such that:

$$o_u = \begin{cases} 1 & \text{if } u \text{ is a leaf} \\ (\sup_{u.\sigma \in t} o_{u.\sigma}) + 1 & \text{otherwise} \end{cases}$$

The definition of o_u is possible because t is well-founded.

For all ordinals o, we denote $t^o = \{t_{[u]} \mid o_u \leq o\}$. Then we can prove by transfinite induction that for all ordinal o, the o^{th} iteration of $\lambda X.(f_\tau \cup B(X))$ starting from $\emptyset$ contains t^o. Thus, if σ is the root of t, t is in the ${o_\sigma}^{th}$ iteration of $\lambda X.(f_\tau \cup B(X))$. Hence $t \in \mathcal{B}^m_{WF}$. □

The well-founded backward semantics can be defined as a greatest fixpoint, starting only from the set of all well-founded trees. In this case, the fixpoint is reached in ω iterations (at most).

Proposition 4.3.12

$$\mathcal{B}^m_{WF} = \mathrm{gfp}^{\subseteq}_{\mathcal{T}_{WF}} \lambda X.(f_\tau \cup B(X)).$$

Proof. We only give a sketch of the proof.

We want to prove that $\mathrm{gfp}^{\subseteq}_{\mathcal{T}_{WF}} \lambda X.(f_\tau \cup B(X)) = \mathcal{C}_\tau \cap \mathcal{T}_{WF}$. Let (θ_n) be the successive decreasing iterations of $\lambda X.(f_\tau \cup B(X))$ starting from $\mathcal{T}_{WF}$. By induction, we can prove that for all n, the trees of θ_n are "coherent until the depth n". Hence $\mathrm{gfp}^{\subseteq}_{\mathcal{T}_{WF}} \lambda X.(f_\tau \cup B(X)) \subseteq \mathcal{C}_\tau \cap \mathcal{T}_{WF}$.

Furthermore, $\mathcal{C}_\tau \cap \mathcal{T}_{WF}$ is a fixpoint of $\lambda X.(f_\tau \cup B(X))$. Thus $\mathrm{gfp}^{\subseteq}_{\mathcal{T}_{WF}} \lambda X.(f_\tau \cup B(X)) = \mathcal{C}_\tau \cap \mathcal{T}_{WF} = \mathcal{B}^m_{WF}$. □

Definition 4.3.13 (Maximal backward semantics) *The maximal backward semantics is defined as:*

$$\mathcal{B}^m = \mathrm{gfp}^{\subseteq} \lambda X.(f_\tau \cup B(X)).$$

The semantics gives exactly the set of trees coherent with $\langle \Sigma, \tau \rangle$:

Lemma 4.3.14 $\mathcal{B}^m = \mathcal{C}_\tau$.

Proof. Similar as for proposition 4.3.12. □

Using the lemmas linking the semantics and $\mathcal{C}_\tau$, we can display the relationship between backward and forward semantics.

Theorem 4.3.15 *We have, for all $I \subseteq \Sigma$, with $\mathcal{T}(I)$ being the set of trees rooted by an element of I:*

$$\mathcal{B}^m_{WF} \cap \mathcal{T}(I) = \mathcal{F}^m(I) \cap \mathcal{T}_{WF};$$
$$\mathcal{B}^m \cap \mathcal{T}(I) = \mathcal{F}^m(I).$$

Proof. This is the direct consequence of lemmas 4.3.8, 4.3.11 and 4.3.14. □

In [Cou02, CC92c], Cousot and Cousot defines the maximal trace semantics of a transition system as a mix of a greatest and a least fixpoint (the greatest fixpoint being used for infinite traces, and the least fixpoint being used for finite traces). This combination is useful for further abstractions such as potential termination [Cou02]. However, in the case of sets of trees, infinite and finite traces are mixed, and it seems quite hard to define the maximal backward semantics as a combination of a greatest and a least fixpoint. However, we can compute first only well-founded trees with a least fixpoint, and then use a greatest fixpoint on a restricted set of infinite trees defined by the well-founded set previously computed:

Proposition 4.3.16 *We have:*

$$\mathcal{B}^m = \mathrm{gfp}^{\subseteq}_{\mathcal{T}_{\mathcal{B}^m_{WF}}} \lambda X.(f_\tau \cup B(X)).$$

(we recall that $\mathcal{T}_{\mathcal{B}^m_{WF}}$ is the set of trees which have all well-founded subtrees in $\mathcal{B}^m_{WF}$).

Proof. It is sufficient to notice that $\mathcal{T}_{\mathcal{B}^m_{WF}} \supseteq \mathcal{B}^m$. □

Thus, to compute the maximal backward semantics, we first generate the well-founded backward semantics as a least fixpoint, then create the set of all trees with well-founded subtrees in the generated set, and then use a greatest fixpoint.

4.4 Making extended transition system

To use extended transition systems, we have to define them as semantics of real programs. Our idea is to create an extended transition system from a program and a temporal property we want to prove.

In the non-game approach, a program is represented by its small-step semantics expressed as a classical transition system $\langle \Sigma_0, \tau_0 \rangle$. The transformation depends on the temporal property. In this section, we present some examples.

4.4.1 Distinction between two kind of non-determinisms

The temporal properties expressed here are quite easy, with only one fixpoint. Using the μ-calculus formalism, the temporal property we intend to use is defined as:

$$\Phi = \frac{\mu}{\nu} X.(A \vee (B \wedge \Diamond X) \vee (C \wedge \Box X)).$$

This kind of formula includes, of course, all the basic CTL operators (excepted **AX** and **EX**). But we do not intend to combine them (with several fixpoints). However, the formula expresses also some form of game properties, by introducing "fated" and "free" non-determinisms, depending of the current state.

States satisfying A are "final" states. States satisfying B are "free" non-deterministic states: we can choose the successor in order to get A (in other words, if one successor satisfies Φ, then the state satisfies Φ). States satisfying C are "fated" non-deterministic states: we cannot choose the successor. States which satisfy neither A, nor B nor C are errors states. For the sake of simplicity, we identify A with the set of states satisfying A.

The extended transition system $\langle \Sigma, \tau \rangle$ is then defined as follows, with $\Sigma = \Sigma_0$:

$$\begin{array}{lll} \tau(\sigma) = & \emptyset & \text{if } \sigma \notin A \cup B \cup C \\ \tau(\sigma) = & \{\emptyset\} & \text{if } \sigma \in A \\ \tau(\sigma) = & \{\{\sigma'\} \mid \sigma' \in \tau_0(\sigma)\} & \text{if } \sigma \in B \\ \tau(\sigma) = & \{\tau_0(\sigma)\} & \text{if } \sigma \in C \end{array}$$

τ satisfies (4.1). Furthermore, the roots of the backward semantics with a least fixpoint (resp. with a greatest fixpoint) of $\langle \Sigma, \tau \rangle$ are exactly the set of states satisfying Φ with $\frac{\mu}{\nu} = \mu$ (resp. with $\frac{\mu}{\nu} = \nu$).

Theorem 4.4.1 *An initial state satisfies Φ (with $\frac{\mu}{\nu} = \mu$) if and only it is the root of a well-founded tree in $\mathcal{F}^m(I)$.*

An initial state satisfies Φ (with $\frac{\mu}{\nu} = \nu$) if and only it is the root of a tree in $\mathcal{F}^m(I)$.

Proof. It is easy to prove that the computation of the maximal backward semantics (resp. maximal well-founded backward semantics) of the transition system is similar to the backward algorithm computing the formula: if θ_n is the n^{th} iteration of the backward semantics, then $root(\theta_n)$ are the states computed at the n^{th} iteration of the backward algorithm. Using the results of theorem 4.3.15, we prove the proposition. □

4.4.2 Game properties with only one fixpoint

An extended transition system expresses a form of game, where the (two) players do not play simultaneously (first the set of possible successors is chosen, then a successor is picked in this set). To prove that this restriction is not problematic, we show that it is possible to use extended transition systems to check properties on alternating transition systems.

Let $\langle \mathcal{P}, \Sigma, \Delta \rangle$ be an alternating transition system. We want to check a specification described as a $A\mu$-calculus formula of the kind:

$$\Phi = \frac{\mu}{\nu} X.(A \vee \bigvee_{I \in \wp(\mathcal{P}) \setminus \{\emptyset\}} (B_I \wedge \langle\langle I \rangle\rangle \bigcirc x) \vee \bigvee_{I' \in \wp(\mathcal{P}) \setminus \{\emptyset\}} (C_{I'} \wedge [\![I']\!] \bigcirc x)).$$

For the sake of simplicity, we suppose that A and all the B_I and $C_{I'}$ are disjoint.

The extended transition system is then (Σ, τ), with τ defined as follows:

$$\begin{array}{lll}
\tau(\sigma) = & \emptyset & \text{if } \sigma \notin A \cup \bigcup B_I \cup \bigcup C_{I'} \\
\tau(\sigma) = & \{\emptyset\} & \text{if } \sigma \in A \\
\tau(\sigma) = & \{\bigcup_{\forall i \notin I, \tau_i \in \delta_i(\sigma)} \bigcap_{i \in \mathcal{P}} \tau_i \mid \forall i \in I, \tau_i \in \delta_i(\sigma)\} & \text{if } \sigma \in B_I \\
\tau(\sigma) = & \bigotimes_{\forall i \in I, \tau_i \in \delta_i(\sigma)} \{\bigcap_{i \in \mathcal{P}} \tau_i \mid \forall i \notin I, \tau_i \in \delta_i(\sigma)\} & \text{if } \sigma \in C_I
\end{array}$$

where each set of $\bigotimes_{x \in X} \mathcal{E}_x$ is generated by picking exactly one element of each set of $\mathcal{E}_x$ for all x and putting these elements together.

τ satisfies the equation (4.1) (if the alternating transition system is nonblocking), and enables to check Φ:

Theorem 4.4.2 *Let I be the set of initial states, and $\sigma \in I$. σ satisfies Φ (with $\frac{\mu}{\nu} = \mu$) if and only it is the root of a well-founded tree in $\mathcal{F}^m(I)$. Furthermore, σ satisfies Φ (with $\frac{\mu}{\nu} = \nu$) if and only it is the root of a tree in $\mathcal{F}^m(I)$.*

Proof. Again, the backward maximal semantics follows the backward computation of the set of states satisfying Φ. □

4.4.3 CTL

A CTL formula may have several nested fixpoints [CGP99]. We simulate the actions of these fixpoints with only one fixpoint by extending the set of states.

Φ being a formula, we denote $sub(\Phi)$ the set of all sub-formulas in Φ. Then we define Σ as:

$$\Sigma = \Sigma_0 \times sub(\Phi).$$

The extended transition relation τ is defined in Fig. 4.8. Let Ψ be the set of sub-formulas of Φ of the form **AG**, **EG**, **AR**, **ER**. For all ϕ in Ψ, we define $T_\phi = (\Sigma_0 \times \{\phi\})^\omega$, and $T_\Psi = \bigcup_{\phi \in \Psi} T_\phi$. T_Ψ are the infinite sequences where the temporal formula is constant and of the form **G** or **R**. The only infinite branches allowed for a tree "proving" Φ must have a suffix in T_Ψ.

Then an initial state i satisfies Φ if and only if (i, Φ) is the root of a tree in $\mathcal{F}^m(I \times \Phi) \cap \mathcal{T}^{T_\Psi}$. This property can be proved by induction on the structure of Φ.

4.5 Combination

One of the basic goals of using extended transition systems is to make automatic abstractions, and to use the combination between forward and backward analyses. The principle of the combination is to use the result of the previous analysis at each iteration of the next analysis, to get more precise values (which are still sound). Even if the operation is done on the abstract domain, there are underlying operations on the concrete domain. In this section, we examine the possible combinations one can

$$
\begin{array}{rcl}
\tau(\sigma, p) & = & \{\emptyset\} \text{ if } \sigma \text{ satisfy } p \\
\tau(\sigma, p) & = & \emptyset \text{ otherwise} \\
\tau(\sigma, \phi_1 \vee \phi_2) & = & \{\{(\sigma, \phi_1)\}, \{(\sigma, \phi_2)\}\} \\
\tau(\sigma, \phi_1 \wedge \phi_2) & = & \{\{(\sigma, \phi_1), (\sigma, \phi_2)\}\} \\
\tau(\sigma, \mathbf{AX}\phi) & = & \{\{(\sigma', \phi) \mid \sigma' \in \tau_0(\sigma)\}\} \\
\tau(\sigma, \mathbf{EX}\phi) & = & \{\{(\sigma', \phi)\} \mid \sigma' \in \tau_0(\sigma)\} \\
\tau(\sigma, \mathbf{AF}\phi) & = & \{\{(\sigma, \phi)\}, \{(\sigma', \mathbf{AF}\phi) \mid \sigma' \in \tau_0(\sigma)\}\} \\
\tau(\sigma, \mathbf{EF}\phi) & = & \{\{(\sigma, \phi)\}, \{(\sigma', \mathbf{EF}\phi)\} \mid \sigma' \in \tau_0(\sigma)\} \\
\tau(\sigma, \mathbf{AG}\phi) & = & \{\{(\sigma, \phi), (\sigma', \mathbf{AG}\phi) \mid \sigma' \in \tau_0(\sigma)\}\} \\
\tau(\sigma, \mathbf{EG}\phi) & = & \{\{(\sigma, \phi), (\sigma', \mathbf{EG}\phi)\} \mid \sigma' \in \tau_0(\sigma)\} \\
\tau(\sigma, \mathbf{A}(\phi_1 \mathbf{U} \phi_2)) & = & \{\{(\sigma, \phi_2)\}, \{(\sigma, \phi_1), (\sigma', \mathbf{A}(\phi_1 \mathbf{U} \phi_2)) \mid \sigma' \in \tau_0(\sigma)\}\} \\
\tau(\sigma, \mathbf{E}(\phi_1 \mathbf{U} \phi_2)) & = & \{\{(\sigma, \phi_2)\}, \{(\sigma, \phi_1), (\sigma', \mathbf{E}(\phi_1 \mathbf{U} \phi_2))\} \mid \sigma' \in \tau_0(\sigma)\} \\
\tau(\sigma, \mathbf{A}(\phi_1 \mathbf{R} \phi_2)) & = & \{\{(\sigma, \phi_1), (\sigma, \phi_2)\}, \{(\sigma, \phi_2), (\sigma', \mathbf{A}(\phi_1 \mathbf{R} \phi_2)) \mid \sigma' \in \tau_0(\sigma)\}\} \\
\tau(\sigma, \mathbf{E}(\phi_1 \mathbf{R} \phi_2)) & = & \{\{(\sigma, \phi_1), (\sigma, \phi_2)\}, \{(\sigma, \phi_2), (\sigma', \mathbf{E}(\phi_1 \mathbf{R} \phi_2))\} \mid \sigma' \in \tau_0(\sigma)\}
\end{array}
$$

Figure 4.8: Definition of τ for CTL.

see in the concrete domain. As the approximations are over-approximations[2], the operations must be made so that we still get a superset of $\mathcal{F}^m(I)$ (or $\mathcal{F}^m(I) \cap \mathcal{T}_{WF}$) at the end of the computation, in order to keep sound results.

4.5.1 Using backward results in forward analysis

For each iteration of F, we can remove the trees which are not a subset (or "prefix") of a tree obtained in the backward analysis:

Theorem 4.5.1 *With* $\Downarrow: \Theta \times \Theta \longrightarrow \Theta$ *defined as:*

$$\theta_1 \Downarrow \theta_2 = \{t_1 \in \theta_1 \mid \exists t_2 \in \theta_2, t_1 \subseteq t_2\}$$

we have the following results:

$$
\begin{array}{rcl}
\mathcal{F}^m(I) & = & \mathrm{lfp}^{\sqsubseteq}_{\mathcal{I} \Downarrow \mathcal{B}^m} \lambda X.(F(X) \Downarrow \mathcal{B}^m) \\
\mathcal{F}^m(I) \cap \mathcal{T}_{WF} & = & \mathrm{lfp}^{\sqsubseteq}_{\mathcal{I} \Downarrow \mathcal{B}^m_{WF}} \lambda X.(F(X) \Downarrow \mathcal{B}^m_{WF})
\end{array}
$$

[2]The combination of analyses uses over-approximations to check *intersection* of properties (e.g. states which are initial states *and* satisfy the temporal property). To get an equivalent of a lower approximation, we can use the negation of the temporal property.

Proof. Let (θ_i) (resp. (θ'_i)) be the successive iterations of F starting from $\mathcal{I}$ (resp. of $\lambda X.(F(X) \Downarrow \mathcal{B}^m)$ starting from $\mathcal{I} \Downarrow \mathcal{B}^m$). We prove by transfinite induction that for all i, $\theta_i \Downarrow \mathcal{B}^m = \theta'_i$. The right inclusion ($\theta_i \Downarrow \mathcal{B}^m \supseteq \theta'_i$) is a consequence of the reductivity of $\lambda\theta.\theta \Downarrow \mathcal{F}^m(I)$. To prove the left inclusion, we distinguish between ordinals:

If this property is satisfied for an ordinal i, let t be a tree of $\theta_{i+1} \Downarrow \mathcal{B}^m$. Then it exists t' in θ_i such that $t \in F(\{t'\})$, and we can find t_f in $\mathcal{B}^m$ such that $t \subseteq t_f$. Thus, $t' \subseteq t_f$, and $t' \in \theta_i \Downarrow \mathcal{B}^m$ so $t' \in \theta'_i$. Hence $t \in F(\theta'_i) \Downarrow \mathcal{B}^m$.

If ω is a limit ordinal, and the property is satisfied for all $i < \omega$, let $t \in \theta_\omega \Downarrow \mathcal{B}^m$. Then for all $i < \omega$, it exists $t_i \in \theta_i$ such that $t_i \subseteq t$ and $\cup(t_i) = t$, and it exists $t' \in \mathcal{B}^m$ such that $t' \supseteq t$. Then for all i, $t_i \in \theta_i \Downarrow \mathcal{B}^m$, so $t_i \in \theta'_i$. Hence $t \in \theta'_\omega$.

Thus $\mathcal{F}^m(I) = \mathcal{F}^m(I) \Downarrow \mathcal{B}^m = \mathrm{lfp}^{\sqsubseteq}_{\mathcal{I} \Downarrow \mathcal{B}^m} \lambda X.(F(X) \Downarrow \mathcal{B}^m)$. The proof is the same for $\mathcal{F}^m(I) \cap \mathcal{T}_{WF}$. □

***Remark* 4.5.2** *$\lambda\theta.(\theta \Downarrow \mathcal{B}^m)$ is, in fact, a lower closure operator on Θ_{NC}. The proof we shown is the proof of the property of* lfp*-completeness of this operator with respect to $\lambda\theta.\mathcal{I} \sqcup F(\theta)$ (even if this operator is* not *complete). We will see in chapter 5 that this approach is more general.*

4.5.2 Using forward results in backward analysis

Each iteration of the backward analysis gives a set of trees. One possible improvement is to intersect this set with a set of potential trees given by the forward analysis. However, it would be much better to impose some kind of *constraints* on the set of trees given by the iteration. The idea is to remove trees which will not change the final results because they do not appear together with other trees. Formally, from $\mathcal{F}^m(I)$, we would like to get $\mathcal{H} \subseteq \Theta$ such that B' defined as:

$$B'(\theta) = \{t \in B(\theta) \mid \exists H \in \mathcal{H}, t \in H \wedge H \subseteq B(\theta)\}$$

may replace B as the backward operator.

***Remark* 4.5.3** *Here again, $\rho_{\mathcal{H}} = \lambda\theta.\{t \in \theta \mid \exists H \in \mathcal{H}, t \in H \wedge H \subseteq \theta\}$ is a lower closure, and $B' = \rho_{\mathcal{H}} \circ B$. The problem is therefore to find a good lower closure operator to apply to B.*

In order to get $\mathcal{H}$, we need to distinguish the different cases of backward semantics:

With a greatest fixpoint

To get t in $\mathcal{B}^m$, all subtrees of t must be in $\mathcal{B}^m$, at each stage of the iteration. Therefore, we can take for $\mathcal{H}$ the set of all subtrees of each tree in $\mathcal{F}^m(I)$:

Theorem 4.5.4 *With $\mathcal{H} = \{subtrees(t) \mid t \in \mathcal{F}^m(I)\}$, we have:*

$$\mathcal{B}^m = \mathrm{gfp}^{\subseteq} \lambda X.(f_\tau \cup B'(X)).$$

With a least fixpoint

Let $\mathcal{B}$ be an iteration of $\lambda X.(f_\tau \cup B(X))$. Then t will be in $\mathcal{B}^m_{WF}$ if we can find a set θ of subtrees of t in $\mathcal{B}$ which "covers" all the branches of t:

$$\forall u \in t, \exists v \in t,\ u \preceq v \vee v \preceq u, \text{ such that } t_{[v]} \in \theta \tag{4.2}$$

To see what may be θ intuitively, we just remove an "upper part" of t without removing a complete branch. We get a set of trees which satisfy the property (4.2) (see Fig. 4.9 for an example). This result is expressed in the following theorem:

Theorem 4.5.5 *We define a "slice" θ of a tree t as a set of subtrees of t which satisfies (4.2). We denote Slices(t) the set of all slices of a tree t. Then, with*

$$\mathcal{H} = \bigcup_{t \in (\mathcal{F}^m(I) \cap \mathcal{T}_{WF})} \mathit{Slices}(t)$$

we have:

$$\mathcal{B}^m_{WF} = \mathrm{lfp}^{\subseteq}\, \lambda X.(f_\tau \cup B'(X))$$

In practice, this theorem is not usable, $\mathcal{H}$ is too large, and its approximations would be too imprecise.

4.6 Discussion

We have defined new forward and backward concrete semantics for checking temporal properties on programs: from the transition system and the temporal property, we create an extended transition system which combines informations from both, and we give the semantics of this extended transition system. The forward semantics uses a complex order on sets of trees, and complex operations. Deriving analyses from these semantics may be hard. However, our goal is not to prove properties directly from the forward analyses, but to collect more information which will help the backward analyses.

Much work about approximations of sets of trees for abstract interpretation was done by Mauborgne [Mau99, Mau00]. Even if this work deals with trees with finite arity, it is expected to be very helpful to abstract structures on sets of trees.

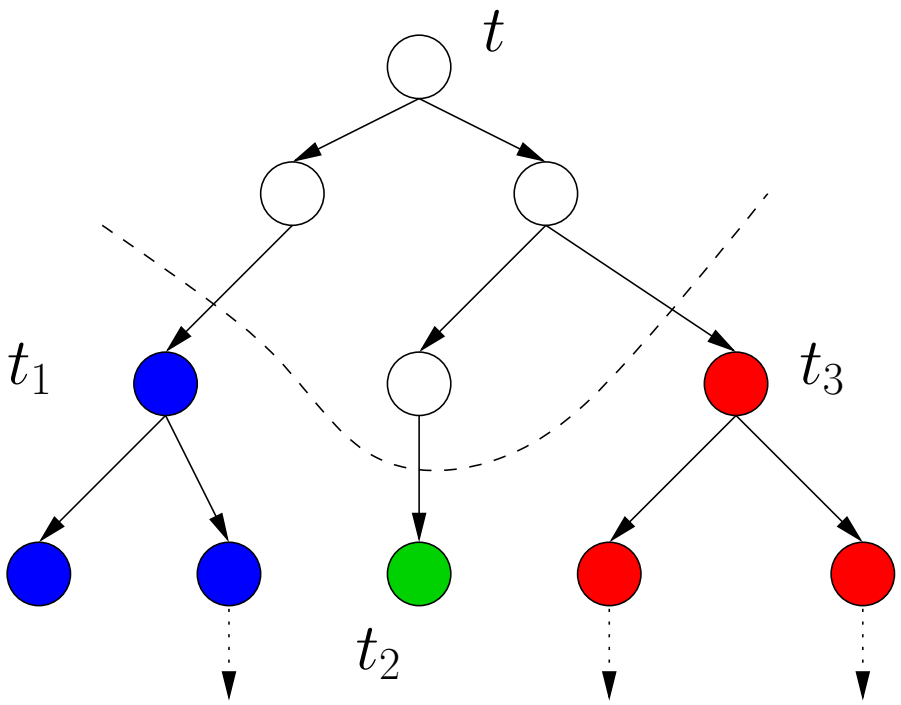


Figure 4.9: An example of a set of trees ($\{t_1, t_2, t_3\}$) which satisfies (4.2) with respect to the tree t.

Chapter 5

Property-driven analysis

As we saw in the previous chapter, the operation of backward-forward combinations in tree semantics involves lower closure operators: proving the soundness of the combination was comparable to the proof of fixpoint-completeness of a lower closure operator with respect to the iteration function.

From this observation, we can develop the idea of *property-driven analysis*: assuming that the verification of the specification is done through the application of a lower closure operator on a fixpoint semantics of the program, we can construct from it another lower closure operator which can restrict the iterations to significant parts (for the checking of the property).

This principle does not look too different from the general principle of abstract interpretation: the goal of the abstraction itself is to restrict the set of properties expressible in order to get a computable approximation which can prove or refute the specification. However, the main point is the possibility to use simultaneously abstraction and property-driven analysis, and even to abstract the construction of the "driver", making it computable while keeping the soundness of the whole analysis. Afterwards, the last stage of the combination is to prove the possibility of reusing the results of each abstract analysis (the initial one, and the abstract construction of "guide") to refine the results of the other.

One important advantage of this approach is that we do not need to develop complex backward and forward semantics before proving the soundness of the combination. Starting from only one semantics and a specification, we derive the other "semantics" and we are sure that the combination between both will be sound. Of course, we can use this approach on tree semantics to keep the expressiveness of sets of trees.

5.1 Concrete description

The first step of the design of a static analysis is the description of the concrete semantics of a program. The semantics is often described as a fixpoint $\mathcal{S} = \text{lgfp}\,\phi$ on a *cpo* D. We suppose that D is a complete lattice, and, for simplicity, that $D = \wp(\Sigma)$, e.g. Σ is a set of states, traces, trees, etc.

Our main hypothesis is that the property we want to prove is expressed by the inclusion of $\mathcal{S}$ in a subset $\mathcal{P}$ of Σ. This hypothesis may seem too strong, but we can modify the concrete semantics in order to satisfy it.

***Example* 5.1.1** *We can describe the program as a transition system $\langle \Sigma, \tau \rangle$, Σ being the set of states. I are the initial states of the program.*

1. *To express that the program will never reach error states E, we can write either:*

$$(\text{lfp}\lambda X.I \cup post(X)) \subseteq \Sigma\backslash E,$$

   or

$$(\text{lfp}\lambda X.E \cup pre(X)) \subseteq \Sigma\backslash I.$$

2. *To express that for all initial states, the program may not go wrong (with the CTL formalism, $\forall i \in I, i \models \mathbf{EG}(\neg error)$), we can only use a backward approach:*

$$(\text{lfp}\lambda X.E \cup \widetilde{pre}(X)) \subseteq \Sigma\backslash I \qquad \textit{where } E \textit{ are the error states.}$$

3. *For complex CTL properties, with more than one fixpoint, we can use extended transition systems. The inclusion of the semantics (either forward or backward) with trees rooted by non-initial states and/or with conditions on their infinite branches is in the studied framework.*

Proving $\mathcal{S} \subseteq \mathcal{P}$ is equivalent to prove $\mathcal{S} \cap (\Sigma\backslash\mathcal{P}) = \emptyset$. This kind of property may be checked in the framework of abstract interpretation: an over-approximation of $\mathcal{S} \cap \mathcal{Q}$ (with $\mathcal{Q} = \Sigma\backslash\mathcal{P}$) is computed and compared with $\emptyset$.

The main idea of the approach is to consider $\rho_0 = \lambda X.X \cap \mathcal{Q}$ as a *lower closure operator* on $\wp(\Sigma)$, and to exploit the results on the construction of fixpoint complete lower closure operators. With the construction of complete closures described in section 2.4.2, we can derive a lower closure ρ greater than ρ_0 which is lgfp-complete for ϕ. Then:

$$\rho_0(\text{lgfp}\,\rho \circ \phi) = \rho_0 \circ \rho(\text{lgfp}\,\phi) = \mathcal{S} \cap \mathcal{Q}$$

The figure (5.1) is an illustration of this result.

Thus, instead of computing an over-approximation of $\text{lgfp}\,\phi$, we can compute an over-approximation of $\text{lgfp}\,\rho \circ \phi$.

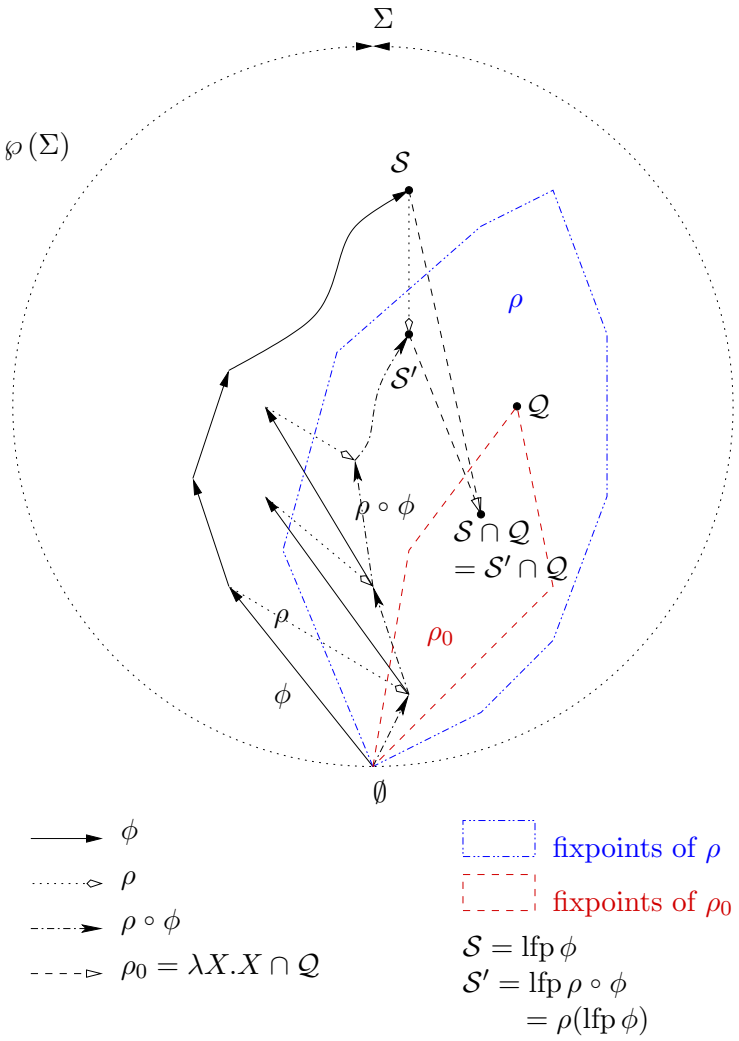


Figure 5.1: Description of the combination, without abstractions. We want to compute $\mathcal{Q} \cap \operatorname{lfp} \phi = \rho_0(\operatorname{lfp} \phi)$. Let $\rho \sqsupseteq \rho_0$ such that ρ is lfp-complete for ϕ. Then $\mathcal{Q} \cap \operatorname{lfp} \rho \circ \phi = \mathcal{Q} \cap \operatorname{lfp} \phi$.

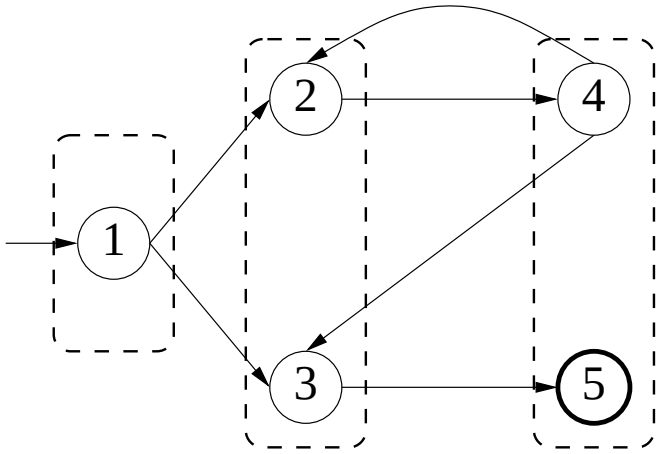


Figure 5.2: Transition system described in example 5.1.2. $\{1\}$, $\{2,3\}$ and $\{4,5\}$ generates $\mathcal{R}_\phi(\rho_0)$

***Example* 5.1.2** *With $\phi = \lambda X.F \cup \widetilde{pre}(X)$, we can use the equation (2.3) since ϕ is co-continuous. With $a \subseteq \Sigma$, we have:*

$$min(\phi^{-1}(\uparrow a)) = \{post(a \backslash F)\}$$

For example, we can choose $\Sigma = \{1,2,3,4,5\}$, $\mathcal{Q} = \{1\}$, $F = \{5\}$ and $\tau = \{(1,2),(1,3),(2,4),(3,5),(4,3),(4,2)\}$.

Then $\mathcal{R}_\phi(\rho_0) = \mathcal{M}(\{\{1\},\{2,3\},\{4,5\}\})$ (cf. Fig. 5.2). The derived analysis $\mathrm{lfp}(\mathcal{R}_\phi(\rho_0)) \circ \phi$ gives $\emptyset$ at the first iteration. Therefore, the computation of $\mathcal{R}_\phi(\rho_0)$ is a kind of "forward analysis" which carries information useful to prove the property we want to check.

5.2 Abstract construction

The results given until now were on the concrete domain, without abstractions. Of course, the lfp-complete lower closure operator is, in general, not computable. However, the goal of our approach is to get a new, computable analysis. Thus we need abstractions.

In this section, we introduce them, both in $\wp(\Sigma)$ (to compute sound approximations in the first analysis) and in $lco(\wp(\Sigma))$ (to compute sound approximations of the complete lower closure operators). We will show that the previous results are still correct with these abstractions, thanks to the usage of lower closure operators[1].

[1]Specifying the property with upper closures would require to use a lower abstraction on the closure domain *and* on the initial concrete domain.

This will prove the correctness of our method.

In the following propositions, R_ϕ is defined with the equation (2.3) if ϕ is co-continuous or with the equation (2.4) otherwise.

The case of a greatest fixpoint is easier, so we present it first.

Proposition 5.2.1 (gfp **analysis)** *Let ϕ be a monotone operator on $\wp(\Sigma)$, and $\mathcal{Q}$ be a subset of Σ. We define $\rho_0 = \lambda X.X \cap \mathcal{Q}$. Let $\overline{\nu} \in uco(\wp(\Sigma))$ and $\Upsilon \in uco(lco(\wp(\Sigma)))$ be (upper) abstractions of $\wp(\Sigma)$ and $lco(\wp(\Sigma))$, respectively. Then, with:*

$$\rho = \operatorname{lfp} \lambda\eta.\Upsilon(\rho_0 \sqcup R_\phi(\eta)),$$

we have $\mathcal{Q} \cap \operatorname{gfp} \phi \subseteq \operatorname{gfp} \overline{\nu} \circ \rho \circ \phi$.

Proof. The soundness property, applied on the computation of ρ, ensures that: $\rho \sqsupseteq \Upsilon(\mathcal{R}_\phi(\rho_0))$ Hence, $\rho \sqsupseteq \mathcal{R}_\phi(\rho_0)$. Thus:

$$\begin{array}{lcll} \operatorname{gfp} \overline{\nu} \circ \rho \circ \phi & \supseteq & \operatorname{gfp} \rho \circ \phi & \\ & \supseteq & \operatorname{gfp} \mathcal{R}_\phi(\rho_0) \circ \phi & \\ & \supseteq & \mathcal{R}_\phi(\rho_0)(\operatorname{gfp} \phi) & \text{since } \mathcal{R}_\phi(\rho_0) \text{ is gfp-complete} \\ & \supseteq & \rho_0(\operatorname{gfp} \phi) & \\ & \supseteq & \mathcal{Q} \cap \operatorname{gfp} \phi & \end{array}$$

□

For lfp analysis, we need to construct a continuous operator. We use the method presented in proposition 2.4.16.

Proposition 5.2.2 (lfp **analysis)** *Let ϕ be a monotone operator on $\wp(\Sigma)$, and $\mathcal{Q}$ be a subset of Σ. We define $\rho_0 = \lambda X.X \cap \mathcal{Q}$. Let C be an extensive operator on $lco(\wp(\Sigma))$ such that for all η, $C(\eta)$ is continuous. Let $\overline{\nu} \in uco(\wp(\Sigma))$ and $\Upsilon \in uco(lco(\wp(\Sigma)))$ be (upper) abstractions of $\wp(\Sigma)$ and $lco(\wp(\Sigma))$, respectively. Then, with:*

$$\begin{array}{l} R = \lambda\eta.\Upsilon \circ C(\operatorname{lfp} \lambda\eta'.\Upsilon(\eta \sqcup R_\phi(\eta'))) \\ \rho = \operatorname{luis}(R, \rho_0), \end{array}$$

we have $\mathcal{Q} \cap \operatorname{lfp} \phi \subseteq \operatorname{lfp} \overline{\nu} \circ \rho \circ \phi$.

Proof. The proof is similar: we just need to show that ρ is greater than a lower closure ρ' which is lfp-complete for ϕ and greater than ρ_0. □

It may seem that ρ would be long to compute, as there are two nesting fixpoints. In practical applications, however, it is probable that we do not need to apply C many times.

These theorems give a method to define a new, "reverse" analysis (since R_ϕ depends on ϕ^{-1}) which can be used to "guide" the first analysis, and thus to enhance its result.

Expressing $R_\phi(\eta)$ is not so hard in practice. When η is generated by a set $\mathcal{A}$ of subsets of Σ (that is, $\eta = \mathcal{M}^u(A)$), we need only to know that $R_\phi(\eta)$ is generated by

$\cup_{X\in\mathcal{A}}\min^?(\phi^{-1}(\uparrow X))$ ($\min^?$ being either min or $\min'$). Therefore, we do not need to keep a representation of the whole Moore family, just of a set of generators.

***Example* 5.2.3** *Starting from an abstraction $\overline{\nu}$ of $\wp(\Sigma)$, we can use $\Upsilon = C = \lambda\eta.(\lambda X.X \cap \overline{\nu} \circ \eta(\Sigma))$ (which can be represented as an element of $\overline{\nu}$). Then the result is the same for* lfp *and* gfp *analysis. To examine the result of this abstraction, we take $\overline{\nu} = \lambda x.x$.*

Then, for all $\eta \in \Upsilon(lco\,(\wp(\Sigma)))$, η satisfies $\eta = \mathcal{M}(\{\{x\} \mid x \in \eta(\Sigma)\})$, so we only have to express $\min'(\phi^{-1}(\uparrow \{x\}))$ to get $R_\phi(\eta)$.

With Σ being a set of states and $\phi = \lambda X.A \cap (F \cup pre(X))$ (that is, $\mathrm{lgfp}\,\phi$ are the states which can go to F or, for the gfp*, loop indefinitely in A), we can use:*

$$\min'(\phi^{-1}(\uparrow \{x\})) = \begin{cases} \emptyset & \text{if } x \notin A \\ \{\emptyset\} & \text{if } x \in F \\ \{\{y\} \mid y \in post(\{x\})\} & \text{otherwise} \end{cases}$$

Then, with $\eta = \lambda X.X \cap Y$ and $\rho_0 = \lambda X.X \cap \mathcal{Q}$,

$$\Upsilon(\rho_0 \sqcup R_\phi(\eta)) = \lambda X.X \cap Y' \quad \text{with } Y' = \mathcal{Q} \cup post(Y \cap (A \backslash F))$$

Thus $\rho = \lambda X.X \cap (\mathrm{lfp}\,\lambda Y.\mathcal{Q} \cup post(Y \cap (A\backslash F)))$. We got the reachability analysis in A (with a slight modification due to F). Using it before the backward analysis is a well-known idea both in abstract interpretation and in model-checking [CC99].

With this abstraction, this is also the best result we can get with $\phi = \lambda X.A \cap (F \cup \widetilde{pre}(X))$.

5.3 The combination

We showed how to construct a reverse analysis from an initial one, and with this reverse analysis restrict the range of the first analysis. However, the backward-forward combination used in abstract interpretation works in both ways: the result of the first analysis is used to get a better reverse result, which we can use in the first analysis, and so on.

Our approach is not symmetrical: the first analysis is on $\wp(\Sigma)$, whereas the second one is on $lco\,(\wp(\Sigma))$. But we can still use the result of the initial analysis, even restricted, in the reverse analysis. This property is given by the following proposition for lfp analysis (the same result holds for gfp analysis):

Proposition 5.3.1 *Let ϕ be a monotone operator on $\wp(\Sigma)$, and $\mathcal{Q}$ be a subset of Σ. Let ρ_1 be a* lfp*-complete lower closure operator for ϕ such that $\mathcal{Q} \cap \mathrm{lfp}\,\phi \subseteq \mathrm{lfp}\,\rho_1 \circ \phi$, and T be a subset of Σ such that $T \supseteq \mathrm{lfp}\,\rho_1 \circ \phi$. We define $\varrho_T \in lco\,(\wp(\Sigma))$ as $\varrho_T = \lambda X.X \cap T$.*

Let C be an extensive and monotone operator on $lco\,(\wp(\Sigma))$ such that for all η, $C(\eta)$ is continuous.

Let $\overline{\nu} \in uco\,(\wp\,(\Sigma))$ *and* $\Upsilon \in uco\,(lco\,(\wp\,(\Sigma)))$ *be (upper) abstractions of* $\wp\,(\Sigma)$ *and* $lco\,(\wp\,(\Sigma))$*, respectively. Then, with:*

$$R^T = \lambda\eta.\Upsilon \circ C(\mathrm{lfp}\,\lambda\eta'.\Upsilon(\varrho_T \sqcap (\eta \sqcup R_\phi(\eta'))))$$
$$\rho^T = \mathrm{lfp}\,\lambda\eta.(\varrho_T \sqcap (\rho_0 \sqcup R^T(\eta))$$

then it exists $\rho_2 \in lco\,(\wp\,(D))$ *such that* ρ_2 *is* lfp*-complete for* ϕ*,* $\rho^T \sqsupseteq \rho_2$ *and* $\mathcal{Q} \cap \mathrm{lfp}\,\phi \subseteq \mathrm{lfp}\,\rho_2 \circ \phi$ *(hence,* $\mathcal{Q} \cap \mathrm{lfp}\,\phi \subseteq \mathrm{lfp}\,\overline{\nu} \circ \rho^T \circ \phi$*).*

With this proposition, we can construct a decreasing sequence (T_n) of elements of $\wp\,(\Sigma)$ greater than $\mathrm{lfp}\,\phi \cap \mathcal{Q}$ (along with a decreasing sequence of lower closure operators (ρ^{T_n}), each ρ^{T_n} being greater than a lfp-complete closure operator ρ_n which satisfies $\mathcal{Q} \cap \mathrm{lfp}\,\phi \subseteq \mathrm{lfp}\,\rho_n \circ \phi$):

$$\begin{array}{ll} T_0 = \Sigma & \\ T_{k+1} = \mathrm{lfp}\,\overline{\nu} \circ \rho^{T_k} \circ \phi & \\ T_\omega = \bigcap_{k<\omega} T_k & \text{for all limit ordinal } \omega \end{array}$$

***Example* 5.3.2** *We continue the example (5.2.3). We have:*

$$\rho^T = \lambda X.X \cap (\mathrm{lfp}\,\lambda Y.T \cap (\mathcal{Q} \cup post(Y \cap (A \backslash F)))).$$

Then, in the sequence (T_n)*,* T_{n+1} *is constructed by doing a forward analysis restricted to* T_n*, then a backward analysis restricted to the result of the forward analysis. This result is similar to the backward-forward combination used in abstract interpretation [CC02], even if the goal is not the same.*

5.4 Abstractions of $lco\,(\wp\,(\Sigma))$

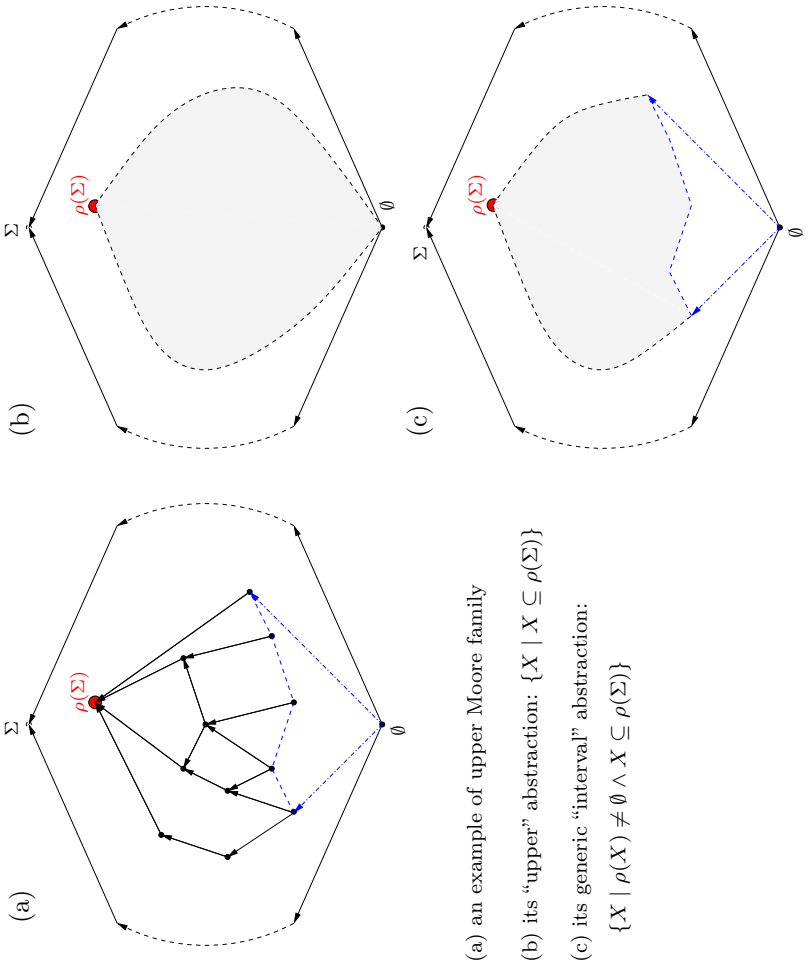
To implement an analyzer based on property-driven checking, we must develop efficient abstractions of $lco\,(\wp\,(\Sigma))$. The first approach is to find abstractions from existing abstractions of $\wp\,(\Sigma)$. As an upper Moore family is an element of $\wp\,(\wp\,(\Sigma))$, this work can be related to the search of abstractions of $\wp\,(\wp\,(\Sigma))$. Following this principle, we will express the abstractions Υ as operators on the lattice of upper Moore families.

5.4.1 "Upper" abstraction

The "upper" abstraction was already given in example (5.2.3): from an abstraction $\overline{\nu} \in uco\,(\wp\,(\Sigma))$, we define $\Upsilon_{\overline{\nu}}(\rho) = \{X \mid X \subseteq \overline{\nu} \circ \rho(\Sigma)\}$.

Then we can make an equivalence between the elements of $\Upsilon_{\overline{\nu}}$ and the elements of $\overline{\nu}$, which helps the calculus.

All these abstractions are less precise than the "generic" upper abstraction, independent of $\overline{\nu}$: $\Upsilon_0(\rho) = \{X \mid X \subseteq \rho(\Sigma)\}$. This abstraction is represented on figure (5.3b).



(a) an example of upper Moore family
(b) its "upper" abstraction: $\{X \mid X \subseteq \rho(\Sigma)\}$
(c) its generic "interval" abstraction: $\{X \mid \rho(X) \neq \emptyset \wedge X \subseteq \rho(\Sigma)\}$

Figure 5.3: Illustration of a lower closure operator as a Moore family (a), its generic "upper" abstraction (b) and "interval" abstraction (c).

5.4.2 "Interval" abstraction

The "interval" abstraction is a combination of an "upper" abstraction and a "lower" one. The "upper" abstraction was presented before. The "lower" would give properties on the lowest elements of the lower closure operator ρ. However, the lowest element of ρ is simply $\emptyset$. Thus, we will try to give properties on the "lower part" of $\rho\backslash\{\emptyset\}$. On the other hand, we forget what is between this "lower part" and the maximum $\rho(\Sigma)$: all elements are possible fixpoints.

Our "generic" abstraction is then:

$$\Upsilon_0(\rho) = \{\emptyset\} \cup \{X \mid \exists (Y,Z) \in \rho^2, Y \neq \emptyset \wedge Y \subseteq X \subseteq Z\}$$

An illustration of this abstraction is given Fig. (5.3c).

Described with closure operators, we see that this abstraction keeps $\rho^{-1}(\{\emptyset\})$ (i.e. $\rho^{-1}(\{\emptyset\}) = (\Upsilon_0(\rho))^{-1}(\{\emptyset\})$):

$$\Upsilon_0(\rho) = \lambda X. \begin{cases} \emptyset & \text{if } X \notin \{Y \mid \rho(Y) \neq \emptyset\} \\ X \cap \rho(\Sigma) & \text{otherwise} \end{cases}$$

***Example* 5.4.1** *We represent the lattice of* $lco\,(D)$ *with* $D = \{\bot, 0, -, +, \top\}$ *in Fig. 5.4, and the "upper" and "interval" abstraction in this lattice.*

With this definition, $\Upsilon_0(\rho)$ is represented by an element of $\wp\,(\Sigma)$ and an element of $\wp\,(\wp\,(\Sigma))$. The first one, $\rho(\Sigma)$, can be easily abstracted. However, the second one ($\{Y \mid \rho(Y) \neq \emptyset\}$) is difficult to approximate. A possible approach would be to choose a (lower) approximation of $\bigcap\{Y \mid \rho(Y) \neq \emptyset\}$. In practical cases, this approach often gives $\emptyset$. Thus, we propose another alternative, which consists in abstracting each element of the set before intersecting them in the abstract domain. The more the abstract intersection keeps informations, the better this alternative will be. Following this principle, we will use a Galois connection for this abstraction: we do not require the abstraction function to be surjective.

Starting from an abstraction of $\wp\,(\Sigma)$ described as a Galois connection $\wp\,(\Sigma) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \Sigma^\sharp$ ($\Sigma^\sharp$ being a complete lattice), we define $\Upsilon_{\alpha,\gamma} \in uco\,(lco\,(\wp\,(\Sigma)))$ as:

$$\Upsilon_{\alpha,\gamma}(\rho) = \lambda X. \begin{cases} \emptyset & \text{if } \alpha(X) \sqsubset^\sharp \sqcap^\sharp \{\alpha(Y) \mid Y \in \rho\} \\ X \cap \gamma \circ \alpha(\rho(\Sigma)) & \text{otherwise} \end{cases} \tag{5.1}$$

We can remark that all elements of $\Upsilon_{\alpha,\gamma}$ can be represented by an element of $\Sigma^\sharp \times \Sigma^\sharp$:

Theorem 5.4.2 $\Upsilon_{\alpha,\gamma} \in uco\,(lco\,(\wp\,(\Sigma)))$, *and* $\Upsilon_{\alpha,\gamma} \sqsubseteq \Upsilon_0$. *Furthermore,* $\Upsilon_{\alpha,\gamma}$ *is associated to the Galois connection:*

$$(lco\,(\wp\,(\Sigma))\,, \sqsubseteq) \underset{\alpha^\bullet}{\overset{\gamma^\bullet}{\leftrightarrows}} (\Sigma^\sharp, \sqsupseteq^\sharp) \times (\Sigma^\sharp, \sqsubseteq^\sharp)$$
$$\alpha^\bullet(\rho) = (\sqcap^\sharp\{\alpha(Y) \mid Y \in \rho\},\, \alpha(\rho(\Sigma)))$$
$$\gamma^\bullet(l,u) = \{X \mid l \sqsubseteq^\sharp \alpha(X) \sqsubseteq^\sharp u\}$$

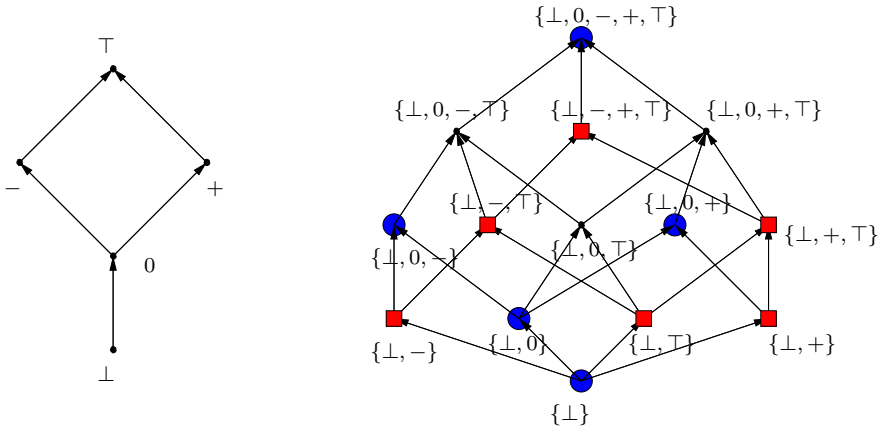


Figure 5.4: The lattice $lco\,(D)$ with $D = \{\bot, 0, -, +, \top\}$. Each lower closure is described by its Moore family. Elements of the "upper" abstraction of $lco\,(D)$ ($\Upsilon(\rho) = \lambda x.(\rho(D) \wedge x)$) are the large discs, while elements of the "interval" abstraction are represented as squares. We can see that the structure of the "upper" abstraction is exactly the structure of D, and that the "interval" abstraction is much more precise. The three closures which are not in the "interval" abstractions are closures which contains 0 and $\top$ without all the elements between ($-$ and $+$). Their image by this abstraction is then the identity closure represented by D.

***Remark* 5.4.3** *If ρ is generated by $\mathcal{A}$, then*

$$\alpha^\bullet(\rho) = (\sqcap^\sharp\{\alpha(Y) \mid Y \in \mathcal{A}\},\ \alpha(\cup\mathcal{A}))$$

Thus we do not need to compute the whole Moore family to get the abstract element.

***Example* 5.4.4** *We choose $\Sigma = \mathbb{Z}$ and $\Sigma^\sharp = \mathbb{Z}^\infty \times \mathbb{Z}^\infty$, with $\alpha(X) = (\min X, \max X)$ and $\gamma(m, M) = \{i \mid m \leq i \leq M\}$ (this is the numerical interval domain [CC77], except that we do not restrict this domain to "true" intervals, where $m \leq M$).*

If ρ is generated by $\{\{n, n+3\} \mid n \geq 1\}$, then $\alpha^\bullet(\rho) = ((+\infty, 4), (1, +\infty))$, and

$$\Upsilon_{\alpha,\gamma}(\rho) = \lambda X. \begin{cases} \emptyset & \text{if } \max X < 4 \\ X \cap [1, +\infty[& \text{otherwise} \end{cases}$$

We can see that this approximation is more precise than the "upper-only" abstraction used in the example (5.2.3).

***Remark* 5.4.5** *As we can see in the example above, using a Galois connection for the abstraction of $\wp(\Sigma)$ is very important: we can use a larger abstract domain $\Sigma^\sharp$, with α non-surjective, to obtain a better precision. On the contrary, using the classical interval domain for $\Sigma^\sharp$ (where all elements (m, M) with $m > M$ are collapsed into $\bot$) would give $(\bot, (1, +\infty))$ for $\alpha^\bullet(\rho)$, which is less precise. Since all closure operators induce a surjective Galois connection, we can not use this framework here.*

5.5 Links with tree semantics

The technique of property-driven checking and the tree semantics described in chapter 4 are not unrelated. We already remarked that fixpoint-complete lower closure operators are used for the combination of backward and forward semantics of extended transition systems (cf. remarks 4.5.2 and 4.5.3). Using the property-driven checking technique on backward semantics of extended transition systems, we can develop more efficient means of combination. On the other hand, it is possible to develop an extended transition system from a monotonic operator ϕ with formulas related to the formulas used to construct the lower closure operator.

Let ϕ be a monotonic operator on a domain $\wp(\Sigma)$. We define an extended transition system $\langle \Sigma, \tau_\phi \rangle$ as:

$$\forall \sigma \in \Sigma,\ \tau\sigma = \min^?(\phi^{-1}(\uparrow \{\sigma\})) \tag{5.2}$$

with $\min^?$ being either min or $\min'$.

Proposition 5.5.1 *If $\mathcal{B}^m_{WF}$ (resp. $\mathcal{B}^m$) is the maximal well-founded backward semantics (resp. maximal backward semantics) of $\langle \Sigma, \tau_\phi \rangle$, then:*

$$\begin{aligned} \mathit{root}(\mathcal{B}^m_{WF}) &= \operatorname{lfp} \phi \\ \mathit{root}(\mathcal{B}^m) &= \operatorname{gfp} \phi \end{aligned}$$

Then the forward maximal semantics of $\langle \Sigma, \tau_\phi \rangle$ starting from trees rooted by $\mathcal{Q}$ is a "reverse" semantics for the analysis of $\mathrm{lgfp}\, \phi \cap \mathcal{Q}$.

***Example* 5.5.2** *Let $\phi = \mathrm{lgfp} X.(A \cup (B \cap pre(X)) \cup (C \cap \widetilde{pre}(X)))$ defined on a standard transition system $\langle \Sigma, \tau_0 \rangle$. When we apply the proposition, we get the extended transition system described in section 4.4.1*[2]*. Hence we re-demonstrate the theorem 4.4.1.*

On the other hand, when ϕ is defined on an alternating transition system $\langle \mathcal{P}, \Sigma, \Delta \rangle$, with one fixpoint, as in the section 4.4.2, the extended transition system generated by our proposition is, in general, "smaller" than the one described in section 4.4.2. Here the result depends on the definition of $\min'$*, whereas the description made in section 4.4.2 does not try to "minimise" the set of successors of each state.*

The tree semantics gives a "concrete vision" of the forward analysis (as a construction of a set of trees), something the lower closure construction can not do. Furthermore, as we keep the history of the computation in the trees, it is more expressive than lower closure on states (to check liveness properties, for example, tree semantics may be more efficient as we can abstract the evolution of the computation).

On the other hand, lower closure operators have some advantages: the domain is well studied and abstractions are easier to develop. Moreover, the computation of the lower closure operator, which involves $\min^?(\phi^{-1}(\uparrow X))$ for any X, can be faster (in the number of iterations) than the computation of the forward tree semantics which involves $\min^?(\phi^{-1}(\uparrow \{\sigma\}))$ for each state (as $\lambda X.\min^?(\phi^{-1}(\uparrow X))$ is in general not monotonic).

***Example* 5.5.3** *Let Σ be a large set with $\{0,1,2,3\} \subseteq \Sigma$. $\mathcal{Q} = \{0\}$, and ϕ is a monotonic and co-continuous operator such that:*

$$\begin{aligned} \min(\phi^{-1}(\uparrow \{0\}) &= \{\{1,2\}\} \\ \min(\phi^{-1}(\uparrow \{1\}) &= \{\emptyset\} \\ \min(\phi^{-1}(\uparrow \{2\}) &= \{\{1\},\{3\}\} \\ \min(\phi^{-1}(\uparrow \{1,2\}) &= \{\{1\}\} \\ \min(\phi^{-1}(\uparrow \{3\}) &= \{...\} \end{aligned}$$

A complete operator ρ for ϕ greater than $\lambda X.\mathcal{Q} \cap X$ is defined by the Moore family:

$$\rho = \mathcal{M}^u\left(\{0\},\{1\},\{1,2\}\right).$$

This operator is reached after three iterations. On the other hand, the forward tree semantics may take many more iterations to compute as it explores the descendants of the state 3*.*

[2]Though *pre* is not co-continuous, we can use here $\min' = \min$.

5.6 Conclusion

The main point of this property-driven verification is the possibility of deriving a new abstract analysis from an initial analysis and the description of the property we want to check. $lco\,(\wp\,(\Sigma))$ is the domain of this new analysis, hence we need to over-abstract lower closure operators. We showed that this operation is possible while keeping the correctness of the analysis, something which would not work with upper closure operators. The efficiency of this approach relies mainly on the efficiency of the abstractions on $lco\,(D)$. Hence, to design an analyzer, we need to develop an abstraction of $lco\,(D)$.

Chapter 6

Design of an analyzer

From the results on property-driven analyses, we developed a small analyzer based on Cousot's Marktoberdorf generic analyzer [Cou99]. In this chapter we describe the new results used to implement our analyzer, and the different choices we made.

6.1 Language, semantics and concrete domain

From the language of the initial analyzer[1], we did only small changes:

- We did not use the uninitialisation error value. Rather, the initial values of variables are defined by the user. Hence, there is only one (arithmetic) error Ω, and the values are $\mathbb{I}_\Omega = [\text{min\_int}, \text{max\_int}] \cup \{\Omega\}$. We will note $\mathbb{I} = [\text{min\_int}, \text{max\_int}]$.
- We add a new arithmetic expression `?  in (`A_1`,`A_2`)`, which gives a random integer between the evaluation of A_1 and A_2 (and an arithmetic error if the former is strictly greater than the latter). This expression is added to make an interesting non-deterministic operation.

Thus, the language uses arithmetic expression Aexp, boolean expression Bexp, command Com and list of commands Seq. The syntax is defined as:

$$
\begin{array}{lcl}
A \in \text{Aexp} & ::= & n \mid \text{x} \mid ?\ \texttt{in}\ [A_1, A_2] \mid un\ A \mid A_1\ bin\ A_2 \\
B \in \text{Bexp} & ::= & \texttt{true} \mid \texttt{false} \mid A_1 = A_2 \mid A_1 < A_2 \\
 & & \mid B_1 \text{ and } B_2 \mid B_1 \text{ or } B_2 \\
C \in \text{Com} & ::= & \texttt{skip} \mid \text{x} := A \mid \texttt{if}\ B\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \texttt{fi} \\
 & & \mid \texttt{while}\ B\ \texttt{do}\ S\ \texttt{od} \\
S \in \text{Seq} & ::= & C \mid C\ ;\ S
\end{array}
$$

[1] An very simple imperative language, with only integers and without functions.

In this definition, n are integers, $x \in \mathbb{V}$ variables, $un \in \{+, -\}$ unary operators and $bin \in \{+, -, *, /, \mathrm{mod}\}$ binary operators.

Because this analyzer is only a prototype, we only check simple temporal properties, with only one fixpoint and both forms of non-determinism. The user can specify the initial and final conditions of the analysis (as symbolic expressions), as well as the nature of the random generators (`input` or `random`), and the fixpoint used in the temporal specification (least fixpoint or greatest fixpoint).

6.2 Abstract domain

We expose briefly each stage of the construction of the abstract domain.

6.2.1 Abstract domain of values

First, we need to abstract $lco\,(\wp\,(\mathbb{I}_\Omega))$. To abstract $lco\,(\wp\,(\mathbb{I}))$, we can use the domain defined in example 5.4.4. We denote $\mathbb{I}_{\mathrm{int}}$ the abstract domain in this example. We recall that an element of $\mathbb{I}_{\mathrm{int}}$ is defined by two pairs of integers $((\mathrm{Mm}, \mathrm{mM}), (\mathrm{mm}, \mathrm{MM}))$[2] such that $\mathrm{mm} \leq \mathrm{Mm} \leq \mathrm{MM}$ and $\mathrm{mm} \leq \mathrm{mM} \leq \mathrm{MM}$ (except for the bottom of $\mathbb{I}_{\mathrm{int}}$ which is represented by $((\mathrm{min\_int}, \mathrm{max\_int}), (\mathrm{max\_int}, \mathrm{min\_int}))$).

To abstract $lco\,(\wp\,(\mathbb{I}_\Omega))$, we define the set $T = \{\mathrm{INI}, \mathrm{ERR}, \mathrm{TOP}\}$, and we describe an abstraction α from $lco\,(\wp\,(\mathbb{I}_\Omega))$ to $lco\,(\wp\,(\mathbb{I})) \times T$. $\alpha_1(\rho), \alpha_2(\rho)$ being the two components of the abstraction, we define:

$$\alpha_1(\rho) = \{X \backslash \{\Omega\} \mid X \in \rho\}$$

$$\alpha_2(\rho) = \begin{cases} \mathrm{INI} & \text{if } \forall X \in \rho, \{\Omega\} \notin X \\ \mathrm{ERR} & \text{if } \rho \neq \{\emptyset\} \ \wedge \ \forall X \in \rho \backslash \{\emptyset\}, \{\Omega\} \in X \\ \mathrm{TOP} & \text{otherwise} \end{cases}$$

$lco\,(\wp\,(\mathbb{I})) \times T$ can be defined as a lattice with the structure defined Fig. 6.1, and α is then the abstraction of a Galois connection.

Our abstract domain of values is then $\mathbb{I}_{\mathrm{int}} \times T$. The intuitive meanings of the values of T are:

- INI means that no error is possible. The lower closure associated ρ satisfies $\rho(X) = \rho(X \backslash \{\Omega\})$ for all $X \subseteq \mathbb{I}_\Omega$.
- ERR means that all elements of ρ, except $\emptyset$, contains Ω. Thus, $\Omega \notin X \Rightarrow \rho(X) = \emptyset$.
- TOP is the "do not know" answer.

We loose the relation between the Ω and the non-error values in a same set in ρ. However, we keep the option "error everywhere", which enables to know whether the error is not avoidable.

[2] mm stands for min min, Mm for max min, mM for min max and MM for max max

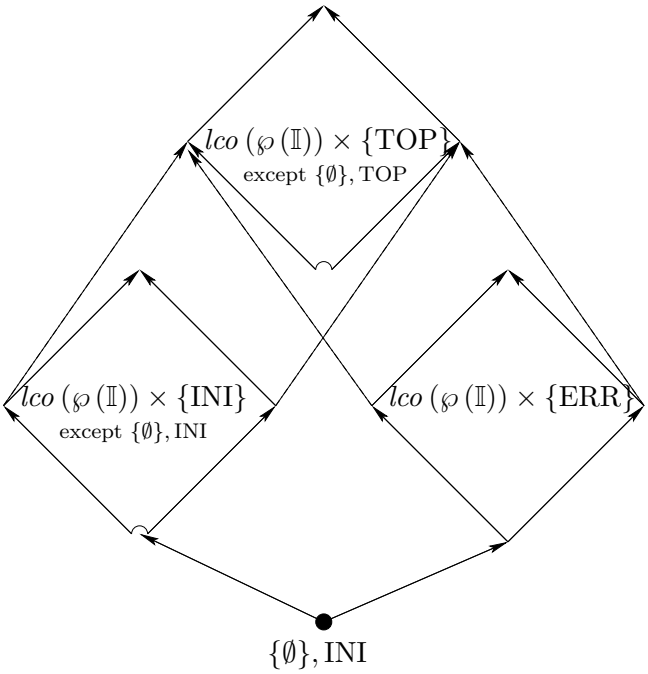


Figure 6.1: Lattice $lco\,(\wp\,(\mathbb{I})) \times T$, divided in three parts, one for each value of T. Note that $(\{\emptyset\}, \text{INI})$ is the global bottom, and that $(\{\emptyset\}, \text{TOP})$ does not appear (it is collapsed with $(\{\emptyset\}, \text{ERR})$).

6.2.2 Abstract environment

$\mathbb{V}$ is the set of variables, and $\mathbb{R} = \mathbb{V} \to \mathbb{I}_\Omega$ are the environments. We need to abstract $lco\,(\wp\,(\mathbb{R}))$.

Non-relational abstraction

The non-relational abstraction abstracts $lco\,(\wp\,(\mathbb{R}))$ to $\mathbb{V} \to lco\,(\wp\,(\mathbb{I}_\Omega))$:

$$\alpha(\rho) = \lambda x.\{\{E(x) \mid E \in \mathcal{E}\} \mid \mathcal{E} \in \rho\}$$

This abstraction keeps no relation at all between variables. More than for abstractions of $\wp\,(\mathbb{R})$, this is problematic: we can not know, in this case, if a random generator is dependent or not on the other values in the memory. To illustrate the problem, we can use the example given section 4.1. To verify the property, we must be sure that, in program point (2), `random` returns 0 and 1 independently of the value of x. This is not possible with a truly independent abstraction.

On the other hand, just knowing the independence of the variables is sufficient. So we can use a very weak relational abstraction here.

Weak relational abstraction: two variables case

First, we give a weak abstraction of $lco\,(\wp\,(\mathbb{I}_\Omega)) \times lco\,(\wp\,(\mathbb{I}_\Omega))$ (like with two variables). Like the non-relational abstraction, we keep an abstract value for each component, But we add a boolean, which expresses the dependence between both components (*true* means that the components may depend from each other). Here, saying that x and y are independent in a lower closure ρ is a Moore family generated by Cartesian products of sets of values.

Hence, the abstract domain is $lco\,(\wp\,(\mathbb{I}_\Omega)) \times lco\,(\wp\,(\mathbb{I}_\Omega)) \times \mathbb{B}$. The concretization of $(\rho_x, \rho_y, \mathit{false})$ is

$$\gamma(\rho_x, \rho_y, \mathit{false}) = \mathcal{M}^u\,(X \times Y \mid X \in \rho_x \wedge Y \in \rho_y)\,.$$

The concretization of $(\rho_x, \rho_y, \mathit{true})$ is $\gamma(\rho_x, \rho_y, \mathit{true}) = \{Z \in \wp\,(\mathbb{I}_\Omega) \times \wp\,(\mathbb{I}_\Omega) \mid \pi_x(Z) \in \rho_x \wedge \pi_y(Z) \in \rho_y\}$ with $\pi_x(Z)$ (resp. $\pi_y(Z)$) the projection of Z to its first (resp. second component).

The concretization of $(\rho_x, \rho_y, \mathit{true})$ corresponds to the concretization of (ρ_x, ρ_y) for the non-relational abstraction. Furthermore for all ρ_x, ρ_y, $\gamma(\rho_x, \rho_y, \mathit{false}) \subseteq \gamma(\rho_x, \rho_y, \mathit{true})$, which means that the "independence" (expressed by the value *false*) carries more information than the "dependence".

***Example* 6.2.1** *To illustrate our abstraction, we restrict the values to $S = \{0, 1\}$. We want to study an abstraction of $lco\,(\wp\,(S \times S))$ to $lco\,(\wp\,(S)) \times lco\,(\wp\,(S)) \times \mathbb{B}$, or, more precisely, we want to describe the meaning of the abstract values. To simplify, we suppose that the first component of the abstract value is $\{\emptyset, \{0, 1\}\}$, and that the second satisfies $\rho(\{0, 1\}) = \{0, 1\}$. Still, we have eight possible cases (four possibilities for the second lower closure, and two possibilities for the boolean), all described in Fig. 6.2, with the order between them.*

When the boolean is false, the values are independent, which means that all the generators of the lower closure are Cartesian products, whereas when the boolean is true we can keep any generator.

The relation is very weak, and its meaning is restricted to the "lower" part of the abstraction, but it is sufficient to keep the independence of the random generators, which was our goal. In particular, the abstract functions will be easier to compute than with stronger relations like octagons.

General case

Our first generic abstract domain is $(\mathbb{V} \to lco\,(\wp\,(\mathbb{I}_\Omega))) \times \wp\,(\wp\,(\mathbb{V}))$, with the concretization:

$$\gamma(f, B) = \{Z \mid \forall x \in \mathbb{V}, \pi_x(Z) \in f(x) \ \wedge\ \forall V \notin B,\, \pi_V(Z) = \times_{x \in V} \pi_x(Z)\}$$

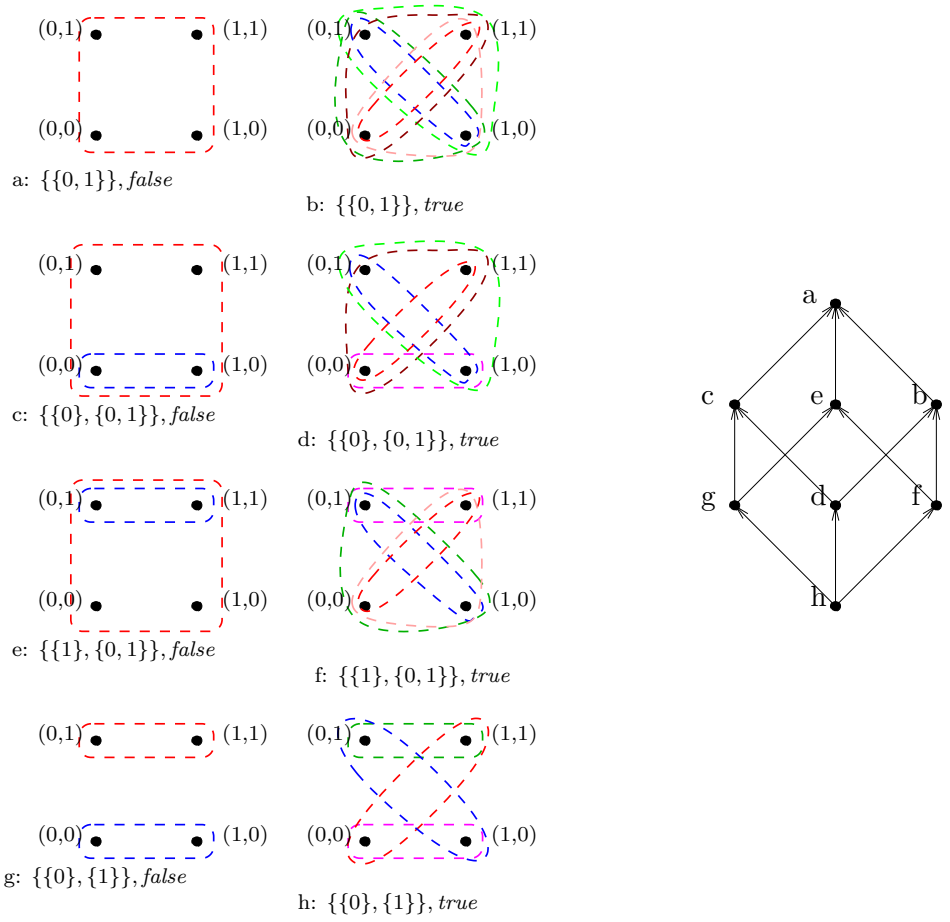


Figure 6.2: The eight cases of example 6.2.1. Each case is described by the value of the boolean, and the generators of the second lower closure operator in the abstract value. For each case, we give the generators of the concretized lower closure. We give also the order between them. We can see that adding the independence boolean (case *false*) restrict greatly the concrete lower closure.

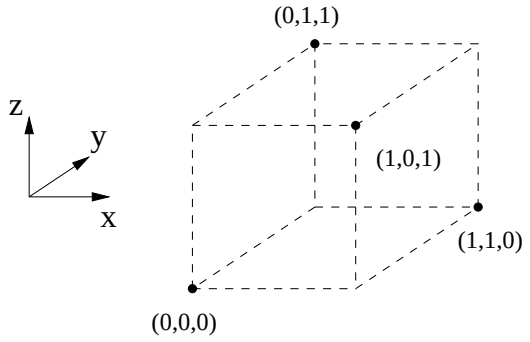


Figure 6.3: An example of a set of tridimensional points which gives Cartesian products when projected in every direction, but is not a Cartesian product.

The principle is to associate a boolean to any subset V of $\mathbb{V}$, describing the idea that these variables are "globally" dependent. If the variables of V are "globally" independent ($V \notin B$), any fixpoint of the lower closure, projected to the space of these variables, must appear as the union of Cartesian products of fixpoints in $f(v)$ for $v \in V$.

Note that three variables may be "independent" when taken by couple without being "globally" independent (a set of points in dimension 3 may look like a Cartesian product when projected in any direction, but not be itself a Cartesian product, cf. Fig 6.3 for an example). Thus we cannot keep only dependence relations between pairs of variables, and construct the "global" dependence function from it.

However, keeping an element of $\wp(\wp(\mathbb{V}))$ is too costly and, in practice, it is not useful[3]. Rather, we keep only a symmetric relation between variables (i.e. a subset of $\wp(\mathbb{V} \times \mathbb{V})$) expressing the dependence of the variables, but such that all sets of variables completely unrelated must be globally independent (thus, this is a subset of the real dependence relations between each couple of variables, but from it we can reconstruct the whole set of dependences). This construction can be viewed as an abstraction, where the concretization function $\gamma_{\mathcal{V}}$ between $\wp(\mathbb{V} \times \mathbb{V})$ and $\wp(\wp(\mathbb{V}))$ is:

$$\gamma_{\mathcal{V}}(R) = \{V \in \wp(\mathbb{V}) \mid \exists (v_1, v_2) \in V^2, (v_1, v_2) \in R\}.$$

Hence, the domain of abstract environments is $(\mathbb{V} \to \mathbb{I}_{\text{int}} \times T) \times \wp(\mathbb{V} \times \mathbb{V})$. Due to this relational abstraction, we do not have the best abstraction function α, but we still have the concretization function γ. Though we cannot use the Galois connection framework, we can use frameworks developed in [CC92b].

[3]It appears to be difficult to infer the dependences with this level of precision.

6.2.3 Abstract domain

Lab is the set of program points, and we want to abstract $lco\,(\wp\,(\text{Lab} \to \mathbb{R}))$. Here we can use a simple non-relational abstraction from $lco\,(\wp\,(\text{Lab} \to \mathbb{R}))$ to $\text{Lab} \to lco\,(\wp\,(\mathbb{R}))$.

This abstraction forgets information on conditionals (which states are related between the branches) and loops. This may be a problem, for example, with the program:

```
x:= ? in [0,1] ;
if (x=0) then x:=0 else x:=1 ;
```

Before the test, there is only one non-empty fixpoint for x ($\{0,1\}$), but after the test, we get two fixpoints ($\{0\},\{1\}$). Hence we loose information. In practice, we think that it is possible to deal with this issue without modifying the abstract domain, by a good implementation of the abstract test during the backward analysis. Here, we can see that both branches of the test are taken. If one branch can not satisfy the specification, the backward analysis must propagate that the specification is not satisfied. An example is given in section 6.3.2.

6.3 Examples

Rather than describing the whole abstract functions of the analyzer, we give some examples of analyses.

6.3.1 First example

This example is the first simple program described in section 4.1. The analysis is given on figure 6.4. The first random generator (for `x`) is controlled by the user, whereas the second (for `y`) is "forced". The initial states are the non-error states of program point (1), and the final states are the states of program point (5) satisfying $x = 1$.

For each program point and each variable, we give:

1. The error status (`ini`, `err` or `top`), corresponding to the values of T.
2. The four integers (mm, Mm, mM, MM). Informally, it means that for each generator, the variable has a value in $[mm, Mm]$ and in $[mM, MM]$.
3. The list of variables which are dependent of this variable. Since the relation of dependence is symmetric, we give it only once.

For example, in program point (3), with only the forward analysis, we get the result:

```
x:[{ ini: (0,1,0,1) }]; y:[{ ini: (0,0,1,1) }]
```

which means informally that `x` is in $\{0,1\}$, `y` is in $\{0,1\}$ and takes all values in $\{0,1\}$ whatever the value of `x` (as they are independent). If the computations were exact, the generators of the concrete lower closure operators would be $\{\{(x:0,y:0),(x:0,y:1)\},\{(x:1,y:0),(x:1,y:1)\}\}$, which are exactly the concretization of the abstract environment. In this case we do not loose information.

In program point (4), the result is:

```
x:[{ ini: (0,1,1,2) }]; y:[{ ini: (0,0,1,1) },x]
```

which means that `x` is in $[0,2]$ and takes at least one value in $[0,1]$ and one value in $[1,2]$, `y` takes all the values of $[0,1]$, and the two variables are dependant. Note that the real generators would be $\{\{(x:0,y:0),(x:1,y:1)\},\{(x:1,y:0),(x:2,y:1)\}$. Hence we loose much information, due to the sum of two variables. However, this is not a problem, thanks to the following backward analysis.

With a backward analysis following the forward analysis, we get $x=1$ in program point (4), and `y` takes all the values in $[0,1]$, which gives the elements $\{(x:1,y:0),(x:1,y:1)\}$, Hence, in program point (3), it yields $\{(x:1,y:0),(x:0,y:1)\}$ which is not possible given the fact that `x` and `y` must be independent. Thus, the application of the lower closure operator gives `BOT` (which represents $\bot$).

A new forward analysis starting from this result would give, of course, $\bot$ everywhere, which proves that the user can force x to be equal to 1 at the end of the program.

6.3.2 Second example

We analyse the example studied in section 3.3. The result of the analysis is given figure 6.5, after one forward analysis, and after one forward followed by one backward analysis. As we get $\bot$ for the initial program point at the end of the backward analysis, making a new forward analysis would give bottom everywhere, which is not useful.

The difficult point in this analysis is the backward computation at program point (2). The analyzer detects that both branches of the analyzer are taken, independently of the variables `x` and `n`. Thus, it computes an intersection of the two environments of program points (3) and (5). Note that, thanks to the forward analysis made before, the result would be the same if we replace

```
if (? in [0;1] = 1) then
```

by

```
a := ? in [0;1]
if (a = 1) then
```

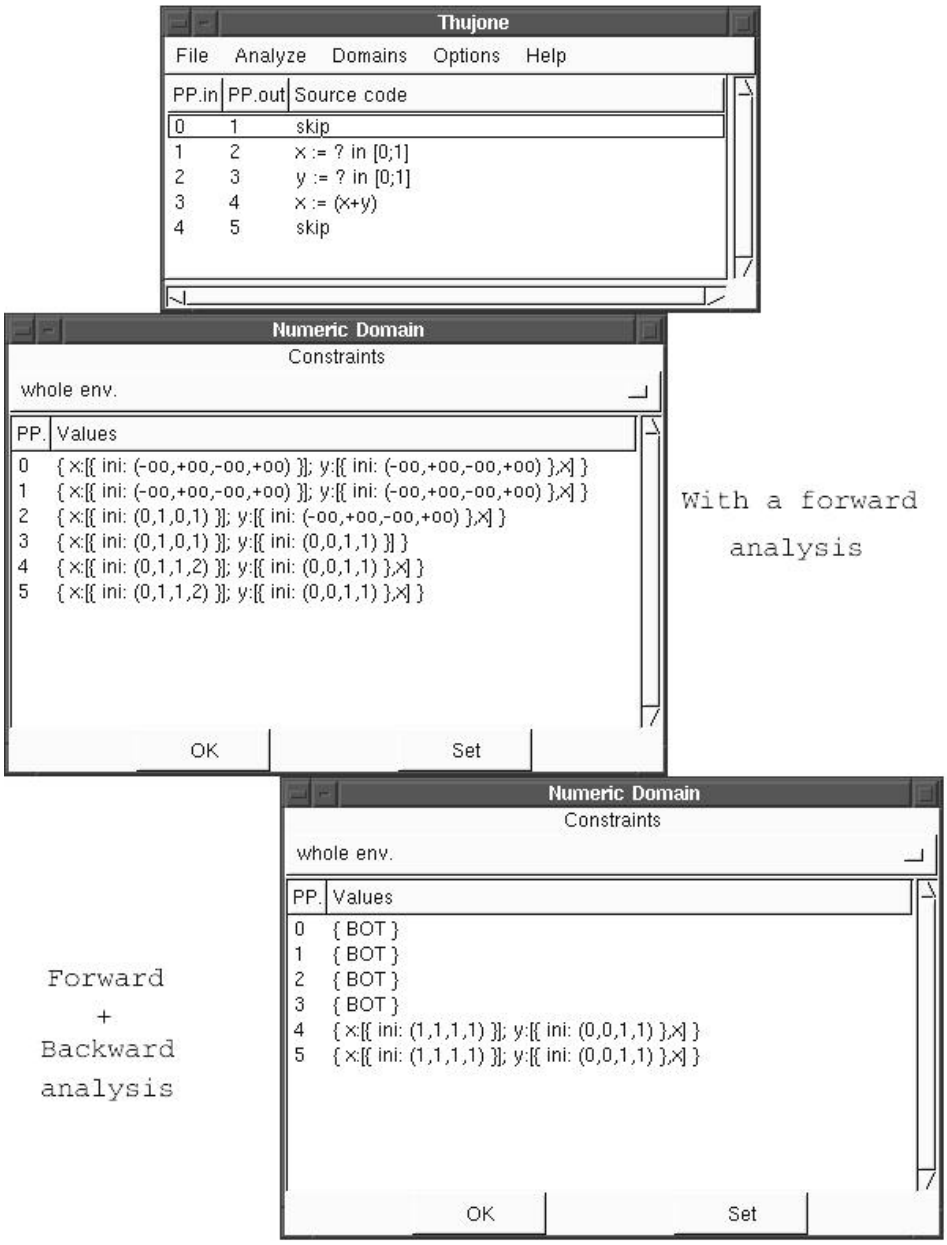


Figure 6.4: Example described in section 6.3.1.

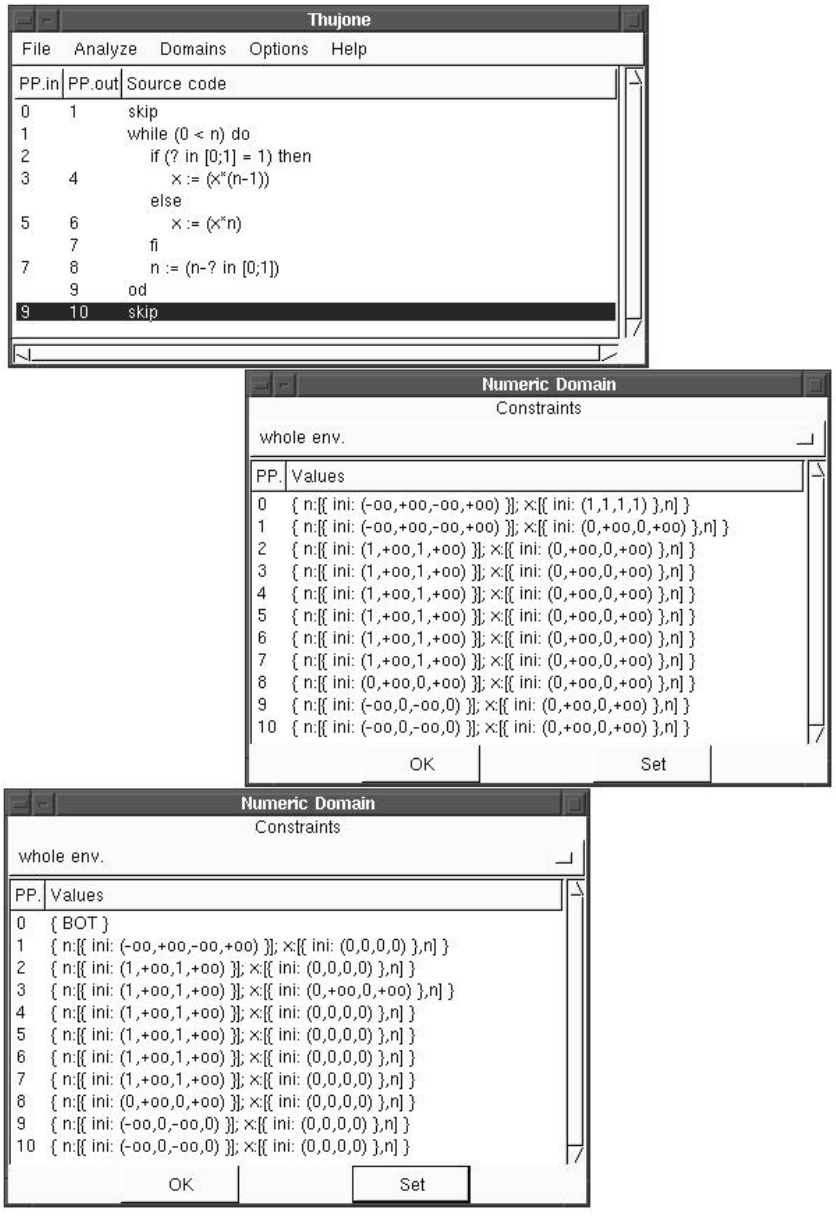


Figure 6.5: Example of section 6.3.2. We give the domain computed after one forward analysis, then after a forward followed by a backward analysis.

6.4 Discussion

We gave a few examples to show the prototype. We need to reduce the loss of informations in some cases (conditionnals in particular). Trace-based partitioning [HT98] may be useful to achieve this goal. We must also extend the class of temporal properties the user can specify.

Chapter 7

Conclusion

Checking complex properties on programs is difficult. Analyses are costly, and often not precise enough to confirm (or refute) the property. We tried to deal with these issues by exploiting as much as possible the interactions between the specification and the given program, even when the abstract domain is not very expressive.

We used three approaches, of varying complexity and efficiency, The extension of the standard combination abstracts the checking of the property (as a backward analysis) and combine it with a reachability analysis. This approach is simple and easy to implement, and appliable to all already existing abstractions (on sets of states), and with a large class of branching-time properties. This method, however, suffers from its imprecise forward analysis which is not specialised.

Including the property to be proved in the semantics to get the tree-based semantics is our second method. We showed how this inclusion is possible for CTL formulas and the forward and backward maximal semantics of the new mathematical objects. However, while tree semantics is very powerful compared to trace semantics, sets of trees are difficult to handle. The implementation of this approach, therefore, would be very difficult.

Our third approach is to include the property in the computation of the fixpoint. We show how to derive a new, reverse analysis from the initial analysis and the specification. The result of this reverse analysis can be used to guide the computation of the initial analysis towards the verification of the specification. Since one analysis is derived from the other, their combination is guaranteed to be sound. Furthermore, we can exploit all the known results on lower closure operators to develop our abstractions. Analyzers based on this approach are easier to implement than analyzers based on tree semantics, but may be less powerful, for example when attempting to check liveness properties.

While we restrict to branching-time temporal properties, we hope that some of those approaches can be extended to other properties. For example, forward and backward semantics for probabilistic programs have been devised [Mon01], but these semantics give uncompatible results and do not seem to combine well. We

hope that the method of property-driven analysis can be adapted to the verification of probabilistic temporal properties.

Bibliography

[AHK97] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 100–109. IEEE Computer Society Press, 1997.

[BCC+02] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software, invited chapter. In T. Mogensen, D.A. Schmidt, and I.H. Sudborough, editors, *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, LNCS 2566, pages 85–108. Springer-Verlag, October 2002.

[BCM+92] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model-checking: 10^{20} states and beyond. *Inf. and Comp.*, 98(2):142 – 170, 1992.

[Bou93] F. Bourdoncle. Abstract debugging of higher-order imperative languages. In *Proceedings of SIGPLAN '93 Conference on Programming Language Design and Implementation*, pages 46–55, 1993.

[CC76] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming*, pages 106–130. Dunod, Paris, France, 1976.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.

[CC79a] P. Cousot and R. Cousot. Constructive versions of Tarski's fixed point theorems. *Pacific Journal of Mathematics*, 82(1):43–57, 1979.

[CC79b] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 1979. ACM Press, New York, NY.

[CC92a] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2–3):103–179, 1992. (The editor of Journal of Logic Programming has mistakenly published the unreadable galley proof. For a correct version of this paper, see `http://www.di.ens.fr/~cousot`.).

[CC92b] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, August 1992.

[CC92c] P. Cousot and R. Cousot. Inductive definitions, semantics and abstract interpretation. In *Conference Record of the Ninthteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 83–94, Albuquerque, New Mexico, January 1992. ACM Press, New York, NY.

[CC99] P. Cousot and R. Cousot. Refining model checking by abstract interpretation. *Automated Software Engineering*, 6(1):69–95, 1999.

[CC00] P. Cousot and R. Cousot. Temporal abstract interpretation. In *Conference Record of the Twentyseventh Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 12–25, Boston, Mass., 2000. ACM Press, New York, NY.

[CC02] P. Cousot and R. Cousot. Software analysis and model checking. In E. Brinksma and K.G. Larsen, editors, *Proceedings of the 14th International Conference on Computer Aided Verification, CAV 2002*, Copenhagen, Denmark, LNCS 2404, pages 37–56. Springer-Verlag Berlin Heidelberg, 27–31 July 2002.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronisation skeletons using branching time temporal logic. In *Proceedings of IBM Workshop on Logics of Programs*, number 131 in Lecture Notes in Computer Science. Springer-Verlag, 1981.

[CGP99] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT press, 1999.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, NY.

[Cou78] P. Cousot. *Méthodes itératives de construction et d'approximation de point fixes d'opérateurs monotones sur un treillis, analyse sémantique des programmes*. Thèse ès sciences mathématiques, University of Grenoble, March 1978.

[Cou81] P. Cousot. Semantic foundations of program analysis. In S.S. Muchnick and N.D. Jones, editors, *Program Flow Analysis: Theory and Applications*, chapter 10, pages 303–342. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.

[Cou99] P. Cousot. The calculational design of a generic abstract interpreter. In M. Broy and R. Steinbrüggen, editors, *Calculational System Design*. NATO ASI Series F. IOS Press, Amsterdam, 1999. Generic Abstract Interpreter available on `http://www.di.ens.fr/~cousot/Marktoberdorf98.shtml`.

[Cou02] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theoretical Computer Science*, 277(1–2):47–103, 2002.

[Gra92] P. Granger. Improving the results of static analyses of programs by local decreasing iterations. In R. K. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science, 12th conference, New Dehli, India*, volume 652 of *Lecture Notes in Computer Science*, pages 68–79. Springer-Verlag, 1992.

[GRS00] R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *Journal of the ACM*, 47(2):361–416, 2000.

[HMMR00] T.A. Henzinger, R. Majumdar, F.Y.C. Mang, and J.-F. Raskin. Abstract interpretation of game properties. In J. Palsberg, editor, *SAS 00: Static Analysis*, volume 1824 of *Lecture Notes in Computer Science*, pages 220–239. Springer-Verlag, 2000.

[HT98] M. Handjieva and S. Tzolovski. Refining static analyses by trace-based partitioning using control flow. In *International Static Analysis Symposium SAS'98*, volume 1503 of *Lecture Notes in Computer Science*, pages 200–214, Pisa, Italy, September 1998. Springer-Verlag, Berlin, Germany.

[Koz83] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, December 1983.

[Mas01] D. Massé. Combining backward and forward analyses of temporal properties. In O. Danvy and A. Filinski, editors, *Proceedings of the Second Symposium PADO'2001, Programs as Data Objects*, volume 2053 of *Lecture Notes in Computer Sciences*, pages 155–172, Århus, Denmark, 21 – 23 May 2001. Springer-Verlag, Berlin, Germany.

[Mas02] D. Massé. Semantics for abstract interpretation-based static analyzes of temporal properties. In M. Hermenegildo, editor, *Proceedings of the Ninth Static Analysis Symposium SAS'02*, volume 2477 of *Lecture Notes in Computer Sciences*, pages 428 – 443, Madrid, Spain, 17 – 20 September 2002. Springer-Verlag, Berlin, Germany.

[Mas03] D. Massé. Property checking driven abstract interpretation-based static analysis. In *Proceedings of the Fourth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'03)*, New York, USA, January 2003. To appear.

[Mau99] Laurent Mauborgne. *Representation of Sets of Trees for Abstract Interpretation*. PhD thesis, École Polytechnique, 1999.

[Mau00] Laurent Mauborgne. An incremental unique representation for regular trees. *Nordic Journal of Computing*, 7(4):290–311, 2000.

[Min01] A. Miné. The octagon abstract domain. In *AST 2001 in WCRE 2001*, IEEE, pages 310–319. IEEE CS Press, October 2001. `http://www.di.ens.fr/~mine/publi/article-mine-ast01.pdf`.

[Mon01] David Monniaux. *Analyse de programmes probabilistes par interprétation abstraite*. Thèse de doctorat, Université Paris IX Dauphine, 2001. Résumé étendu en français. Contents in English.

[MT01] Panagiotis Manolios and Richard J. Trefler. Safety and liveness in branching time. In *Logic in Computer Science*, pages 366–, 2001.

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.

[TXJS92] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-Time Systems. In *7th. Symposium of Logics in Computer Science*, pages 394–406, Santa-Cruz, California, 1992. IEEE Computer Scienty Press.

[War42] M. Ward. The closure operators of a lattice. *Annals Math.*, 43:191–196, 1942.