Semantics for Abstract Interpretation-Based Static Analyzes of Temporal Properties*

Damien Massé

LIX, École Polytechnique, Palaiseau, France,
masse@lix.polytechnique.fr,
http://www.lix.polytechnique.fr/~masse/

Abstract. Analyzing programs with abstract interpretation often requires to use a combination of forward and backward analyzes. However, the forward analyzes are mostly reachability analyzes, which are often not precise enough when we want to check other temporal properties. In this paper we propose to combine transition systems with temporal properties (with backward modalities) to get "extended" transition systems. We define forward and backward semantics for these systems, and give indications for their combination. These semantics, which use set of trees, can be abstracted to make more precise forward and backward analyzes of programs. The combinations of these analyzes are expected to give better results than the current abstract analyzes.

1 Introduction

Most program analyzes are either "forward" or "backward". "Forward" analyzes start with initial states and abstract traces from these states, simulating possible behaviors of the program. "Backward" analyzes start with final states and simulate the reverse execution of the program. As sets of states (and even more set of traces) are often infinite, and cannot be exactly represented, abstractions are used to represent several states with one *abstract* state. Manipulation on abstract states are done by "abstract functions". *Abstract interpretation* [2, 6] is a very general framework to get abstractions and abstract functions and to prove the correctness of an abstract analysis. Fully automatic abstractions to infinite domains, with widening and narrowing [2, 5], may be the best solution when the analysis must be without any human interaction.

Analyzes with these abstractions can give imprecise results, sometimes not precise enough to prove the property we want to check (or its negation). In this case, we can inject the result (sound but imprecise) in another analysis, improving the results the analysis would have given. We can do that with a forward and a backward analysis, and repeat the process several times. This combination of analyzes (first described in [2]) gives sound and precise results. However, most of the time, the forward analysis is a reachability analysis. This analysis is

* This work was supported in part by the RTD project IST-1999-20527 DAEDALUS of the European IST FP5 program.

effective when the goal is to detect possible run-time errors (or potential termination). When the goal is to find initial states satisfying more complex temporal properties, however, this reachability analysis is still usable [10], but the results do not seem precise enough to really help the backward analysis. This problem appears because the forward analysis is independent of the temporal property being checked. In this paper we propose to use an extension of transition systems we call "extended transition systems", to combine temporal properties and transition systems. We define a forward semantics for these extended systems, based on sets of trees instead of set of traces, which will be more precise and give a precise concrete semantics for abstract interpretation. We define also a backward semantics, and study combinations of backward and forward analyzes in this new framework.

The first section of this paper will show a small example to explain the motivation of this work. In the second section, we will introduce notations and properties on sets of free trees, which will be used afterward. Then we will define what we call "extended transition systems", and present forward and backward maximal execution semantics for these systems. Then we will show methods to produce an extended transition system from a program (represented as a transition system) and a temporal property. Finally, we will discuss on the combination of forward and backward analyzes by abstraction of the different concrete semantics.

2 Motivations

We will start by the very simple program in imperative language presented in Fig. 1.

(1) Initial state: $I: \; x = y = \text{uninit.}$

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

(2)

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

(3)

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

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

Fig. 1. Simple program

`input` describes a non-deterministic function controllable by the user, `random` a non-deterministic, incontrollable function.

We want to know if, from the initial state (program point (1), `x` and `y` uninitialized), 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 analyzes, in the framework of abstract interpretation, for this kind of temporal property was described in [10]. 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 will use interval analysis [4], which is quite imprecise but very fast (precise relational analyzes like polyhedrons [8] are too expensive for large programs).

The result is given in the table (we start by a forward analysis) described in Fig. 2.

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:	$[0, 1]$	$[0, 1]$

Fig. 2. Analysis of the program of Fig. 1. Each line describes the result for a program point and a variable (as the analysis is non-relational). The column "Forward" gives the results for the reachability analysis, which the column "Combination" give the combination of these results with a backward analysis. This column is the fixpoint of the analysis.

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 [13] would be sufficient, but more complex cases would need complex relational analyzes. Thus we try another approach.
- 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 on the point (2), `y` can take each 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 from a transition relation and a set of initial states. This semantics, which gives 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), and none of those has all its leaves satisfying $x = 1$. Thus, there is no way to force $x = 1$ in program point (4).

We can modify the program by swapping the `random` command and the `input` command. The new program is given in Fig. 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 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".

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

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

(2)

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

(3)

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

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

Fig. 3. Modified program.

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 by some definitions and notations about trees (or, more accurately, about free trees).

3 Trees

Transition systems traditionally use sets of traces. 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 domain of the semantics of extended transition systems. The following section gives some definitions and notations about this domain.

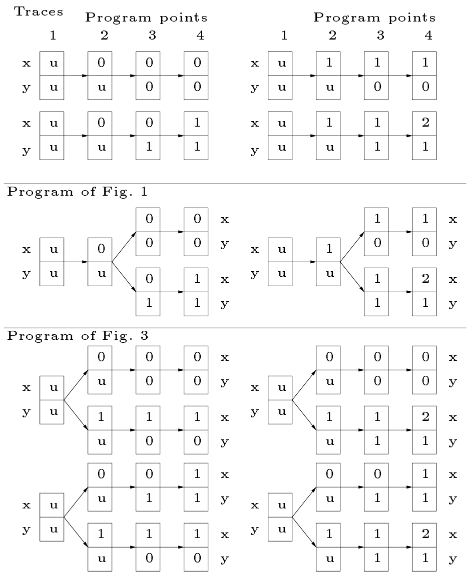

Fig. 4. Traces generated by the programs of Fig. 1 and Fig. 3. Each state is described by its program point (given by its column) and the value of x and y. The traces which describe the same "strategy" for the user are put together in a tree for both program.

3.1 Free trees

We will note Σ the set of states, Σ^* the set of finite sequences 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 extended to sets of sequences in the following way:

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

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

Definition 1 (Free tree). *A* free tree *labeled 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.

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

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 maxima 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 labeled by Σ, and $\Theta = \wp(\mathcal{T})$.

Definition 2 (Subtrees). *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 3 (Well-founded tree). *A free tree t labeled by Σ is said to be well-founded iff $branch(t) \subseteq \Sigma^*$.*

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

A tree with finite arity is well-founded if and only if it is finite. However, in this article we will deal mostly with infinite arity, and the distinction between finite and well-founded must be made (an infinite but well-founded tree is presented in Fig. 5). We will note $\mathcal{T}_{WF}$ the set of well-founded trees labeled by Σ.

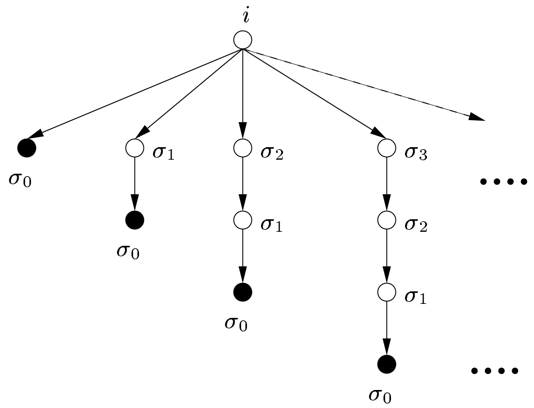

Fig. 5. An infinite well-founded tree. $\Sigma = \{i, \sigma_n \mid n \geq 0\}$, and $t = \{i\sigma_k\sigma_{k-1}\ldots\sigma_l \mid 0 \leq l \leq k\}$ The depth of the tree is unbounded, but it has no infinite branch.

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. – *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:*

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

– *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:*

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

3.2 Orders on set of trees

$\subseteq$ is an partial order on $\mathcal{T}$. However, this order is too imprecise: we will define a "prefix" order $\preceq_\mathcal{T}$ on trees (as in [9]), such that $t \preceq_\mathcal{T} t'$ if t' extends t only on leaves of t:

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

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, in general, $\theta \subseteq \theta' \Rightarrow \theta' \sqsubseteq \theta$).

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: the branches of the trees are 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 1. *We note:*

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

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

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

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

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 Extended transition system

4.1 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 5 (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.*

We will use the functional form for extended transition relations. We can interpret an extended transition system in two ways: first as a program 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 approaches 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 6 (Elementary trees). *τ being an extended transition relation, we denote by* $\mathrm{elem}(\tau)$ *the set of trees:*

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

$\mathrm{elem}(\tau)$ gives a graphical description of the extended transition relation τ (see Fig. 6 for example).

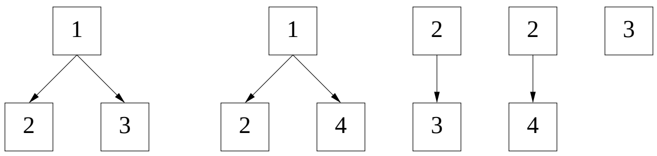

Fig. 6. The elementary trees of the extended transition system $\langle \Sigma, \tau \rangle$ with $\Sigma = \{1, 2, 3, 4\}$ and $\tau(1) = \{\{2, 3\}, \{2, 4\}\}$, $\tau(2) = \{\{3\}, \{4\}\}$, $\tau(3) = \{\emptyset\}$, $\tau(4) = \emptyset$.

4.2 Forward semantics

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

Definition 7 (Forward operator). *The forward operator F of an extended transition system $\langle \Sigma, \tau \rangle$ is:*

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

$F(\theta)$ 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 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 8 (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:*

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

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

With traces, 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 sets of incomparable trees. We need a condition on τ:

Proposition 2. *If an extended transition system $\langle \Sigma, \tau \rangle$ satisfies:*

$$\forall \sigma \in \Sigma :\ \emptyset \in \tau(\Sigma) \implies \tau(\sigma) = \{\emptyset\}, \tag{1}$$

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

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 ($\mathrm{lfp}_a^{\sqsubseteq} F$ is the $\sqsubseteq$-fixpoint of F greater than or equal to a):

Definition 9 (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 $\mathcal{I} = \{\{i\} \mid i \in I\}$.

An example of the maximal forward semantics is given in Fig. 7.

4.3 Backward semantics

There are two possible approaches for the backward semantics: a least fixpoint, which gives only 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 afterward.

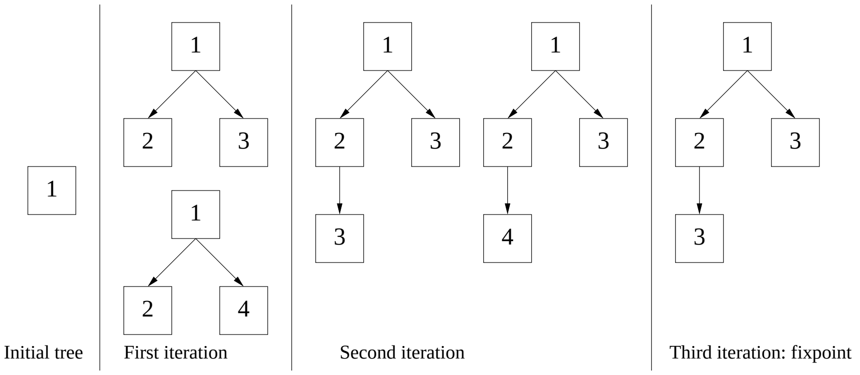

Fig. 7. The iterations and the maximal forward semantics of the extended transition system given in Fig. 6, with $I = \{1\}$.

Definition 10 (Backward operator). *The backward operator B of an extended transition system $\langle \Sigma, \tau \rangle$ is:*

$$\begin{aligned} B : \Theta &\to \Theta \\ \theta &\mapsto \{t \in \theta \ \mid \ \sigma = root(t) \\ &\quad\ \exists S \in \tau(\sigma), \exists t_s \ \in \ \theta \textit{ for all } s \in S \textit{ with } root(t_s) = s, \\ &\quad\ t = \sigma.\{t_s\}\}. \end{aligned}$$

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

To define the maximal trace semantics (as in [7]), 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 $\tau(\sigma) = \emptyset$ are more like error states). Thus we note:

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

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

Definition 11 (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 be not reached after ω iterations. This may appear in the case of unbounded non-determinism.

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

Proposition 3.

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

Definition 12 (Maximal backward semantics). *The maximal backward semantics is defined as:*

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

The following theorem displays the links between backward and forward semantics.

Theorem 1. *We have, for all $I \subseteq \Sigma$, with $\mathcal{T}(I)$ being the set of trees rooted by an element of I:*

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

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

Cousot [3, 7] define the maximal traces 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 [3]. 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. *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 subtrees in $\mathcal{B}^m_{WF}$).

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.

5 Making extended transition system

Our idea is to create an extended transition system from a program and a temporal property we want to prove. A program is represented by a classical transition system (Σ_0, τ_0). The transformation depends on the temporal property. In this section, we present some examples.

5.1 Distinction between two non-determinisms

The temporal properties expressed here are quite easy, with only one fixpoint. Using the μ-calculus formalism, the temporal property is written

$$\Phi = \alpha\, X.(A \vee B \wedge \Diamond X \vee C \wedge \Box X)$$

with α being either μ or ν.

This 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 formulas express 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 satisfy Φ, then the state satisfy Φ). States satisfying C are "fated" non-deterministic states: we cannot choose the successor. States which satisfy neither A, 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 created $\langle \Sigma, \tau \rangle$ is defined as follows, with $\Sigma = \Sigma_0$:

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

τ satisfies (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 $\alpha = \mu$ (resp. with $\alpha = \nu$).

Theorem 2. *An initial state satisfies Φ (with $\alpha = \mu$) if and only it is the root of a well-founded tree in $\mathcal{F}^m(I)$.*

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

5.2 CTL

A CTL formula has several nested fixpoints [1]. We may simulate the results of these fixpoints with only one fixpoint by extending the set of states. Let us recall the basic CTL operators (the negation is only used for atomic predicates):

$$\begin{aligned}
\phi = p \mid \neg p \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \mathbf{AX}\phi \mid \mathbf{EX}\phi \mid \mathbf{AF}\phi \mid \mathbf{EF}\phi \\
\mid \mathbf{AG}\phi \mid \mathbf{EG}\phi \mid \mathbf{A}(\phi_1 \mathbf{U} \phi_2) \mid \mathbf{E}(\phi_1 \mathbf{U} \phi_2) \mid \mathbf{A}(\phi_1 \mathbf{R} \phi_2) \mid \mathbf{E}(\phi_1 \mathbf{R} \phi_2).
\end{aligned}$$

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. 8. Let Ψ be the set of subformulas 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 satisfy Φ if and only if (i, Φ) is the root of a tree in $\mathcal{F}^m(I \times \Phi) \cap \mathcal{T}^{T_\Psi}$.

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

Fig. 8. Definition of τ for CTL.

6 Combination

One of the basic goal of using extended transition systems is to make automatic abstractions, and to use the combination between forward and backward analyzes. The principle of the combination is to use the result of the previous analysis at each iteration of the new 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 see in the concrete domain. As the approximations are over-approximations[2], the operations must be made so that we still get

[2] The combination of analyzes 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.

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.

6.1 Using backward results in forward analysis

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

Theorem 3. *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:

$$\mathcal{F}^m(I) = \operatorname{lfp}_{\mathcal{I}}^{\sqsubseteq} \lambda X.(F(X) \downarrow \mathcal{B}^m)$$
$$\mathcal{F}^m(I) \cap \mathcal{T}_{WF} = \operatorname{lfp}_{\mathcal{I}}^{\sqsubseteq} \lambda X.(F(X) \downarrow \mathcal{B}^m_{WF})$$

6.2 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 don't appear together with other tree. 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.

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

Theorem 4. *With* $\mathcal{H} = \{\mathit{subtrees}(t) \mid t \in \mathcal{F}^m(I)\}$*, we have:*

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

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:

$$\forall u \in t, \exists v \in t,\ u \preceq v \vee v \preceq u,\ \text{such that } t_{[v]} \in \theta \tag{2}$$

To see what may be θ graphically, we just remove an "upper part" of t without removing a complete branch. We get a set of trees which satisfy the property (2) (see Fig. 9 for an example). This result is expressed in the following theorem:

Theorem 5. *We define a "slice" θ of a tree t as a set of subtrees of t which satisfies (2). We denote $Slices(t)$ the set of all slices of a tree t. Then, with*

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

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 general. Finding a smaller suitable set is a part of future developments.

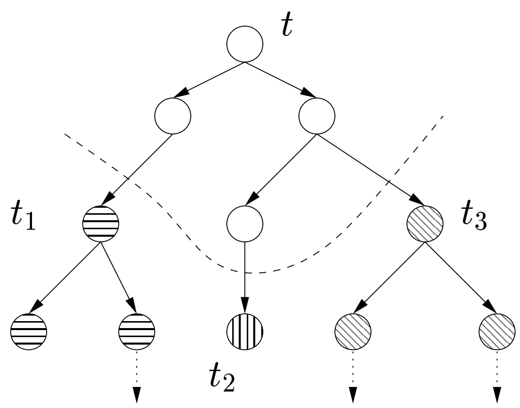

Fig. 9. An example of a set of trees ($\{t_1, t_2, t_3\}$) which satisfies (2) with respect to the tree t.

7 Conclusion

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, more than the backward semantics, use a complex order on set of trees, and complex operations. Deriving analyzes from these semantics may be hard. However, our goal is not to prove properties directly from the forward analyzes, but to collect more informations which will help the backward analyzes.

Much work about approximations of sets of trees for abstract interpretation was done by Mauborgne [11, 12]. Even if this work deals with trees with finite arity, it is expected to be very helpful to abstract structures on sets of trees.

Acknowledgments

I wish to thank the anonymous referees, as well as Radhia and Patrick Cousot, Francesco Logozzo, Charles Hymans and Jean Goubault-Larrecq for their comments and suggestions.

References

1. E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT press, 1999.
2. 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.
3. 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.
4. 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.
5. 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.
6. P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, August 1992.
7. 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.
8. 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.
9. Panagiotis Manolios and Richard J. Trefler. Safety and liveness in branching time. In *Logic in Computer Science*, pages 366–, 2001.
10. 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.
11. Laurent Mauborgne. *Representation of Sets of Trees for Abstract Interpretation*. PhD thesis, École Polytechnique, 1999.
12. Laurent Mauborgne. An incremental unique representation for regular trees. *Nordic Journal of Computing*, 7(4):290–311, 2000.
13. 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-padoII.pdf.