Safety for Branching Time Semantics *

A. Bouajjani† J.C. Fernandez† S. Graf† C. Rodriguez‡ J. Sifakis†

Abstract

We study in a first part of this paper safety and liveness properties for any given program semantics. We give a topological definition of these properties using a *safety preorder*. Then, we consider the case of branching time semantics where a program is modeled by a set of infinite computation trees modulo bisimulation. We propose and study a safety preorder for this semantics based on simulation and dealing with silent actions. We focus on regular safety properties and characterize them by both tree-automata and formulas of a branching time logic. We show that verifying safety properties on trees reduces to simulation testing.

1 Introduction

The properties of parallel systems may be classified according to the type of behaviors they describe. Several classes of properties are distinguished such as safety, liveness, fairness, termination or recurrence properties.

Such a classification allows structuring a program specification into several components; each of these components may be expressed in an appropriate formalism and verified using a specific verification method.

A classification into safety and liveness properties was first proposed by Lamport in [Lam77]. There, these classes were intuitively defined by :

- a safety property asserts that something bad never happens,
- a liveness property asserts that something good eventually happens.

These classes have been widely studied for *linear time semantics* where a program is modeled by a set of infinite sequences of states (also called computation sequences). Several formal definitions of safety and liveness have been proposed in this case, e.g., in [Lam85], [Sis85], [AS87], [AL88], [MP89]. One of them (see for example [MP89] or [AL88]) defines safety (resp. liveness) properties as the *closed* (resp. *dense*) sets in the Cantor topology on the set of infinite sequences of states. An interesting problem is to know how an arbitrary property can be expressed in terms of safety and liveness properties. Using the topological definition given above, Alpern and Schneider show that any ω-regular property is the intersection of a safety and a liveness property [AS87]. This means that, in this case, it is always possible to decompose a program specification into a safety and a liveness part and deal with each of them separately. However, it has been shown that for more expressive classes, e.g., as ω-context-free properties, this can not be done in general [CL89].

Characterizations of ω-regular safety and liveness properties have been given in different program specification formalisms : automata on infinite sequences and fragments of propositional linear time temporal logics [Sis85], [LPZ85], [AS87]. It has also been shown that the proof methods associated with safety and liveness are different. The verification of a safety property is based on an invariant argument whereas the verification of a liveness property is based on a well-foundedness argument [MP84].

In this paper we study safety and liveness properties for *branching time semantics* where a program is modeled by a set of *infinite computation trees* modulo a bisimulation equivalence.

First, we study the problem of defining formally the classes of safety and liveness properties for such semantics. For this purpose, we generalize the topological approach used for the linear time semantics. We argue that defining safety and liveness properties on trees reduces to finding a preorder on trees such that, in the topology induced by this preorder, the safety properties are the closed sets and the liveness properties

*This work was partially supported by ESPRIT Basic Research Action "SPEC"

†LGI-IMAG, IMAG Campus, BP 53X, 38041 GRENOBLE cedex, FRANCE; e-mail: {bouajjan, fernand, graf, sifakis}@imag.imag.fr

‡VERILOG Rhône-Alpes, Forum du Pré Milliet, 38330 MONTBONNOT SAINT-MARTIN, FRANCE; e-mail: crodrig@imag.imag.fr

are the dense sets. We give some minimal requirements on such a preorder and it turns out that a suitable preorder is the *simulation* relation [Mil71].

Furthermore, we are interested in properties not sensitive to *abstraction*, that means that two equivalent programs (verifying the same properties) remain equivalent after abstraction. Therefore, we combine the simulation preorder with an abstraction criterion allowing to abstract away from all the silent actions, i.e., actions which are considered irrelevant. We call *safety preorder* the resulting preorder and *safety equivalence* the equivalence induced by it. We show that the safety equivalence is weaker than branching bisimulation [GW89] and observational equivalence [Mil80]. We also show that the safety preorder is a precongruence for the standard constructors of parallel programs and give a complete axiomatization for it on *regular trees*, i.e., trees which are unfoldings of finite-state labeled transition systems.

We consider as properties regular sets of trees, i.e., sets definable by finite-state (Rabin) tree-automata (RTA for short) [Rab69], [Rab72]. We recall that the class of RTA definable sets contains all the classes of sets definable by either Büchi, Muller or Streett tree-automata [B$\ddot{\text{o}}$62], [Mul63], [Rab69], [Rab70], [Str82]. It has been shown also that RTA are more expressive than all the commonly used specification formalisms for finite-state parallel programs [VW86], [Wol89] and that they are expressively equivalent to the branching time propositional μ-calculus [Koz83], [Niw88]. We show that in the class of RTA definable sets we have also the nice fact that any property is the intersection of a safety and a liveness property.

In practice, it is often sufficient to consider only safety properties. For example, most of service properties of communication protocols are safety properties. Especially, due to the use of nondeterministic choice for scheduling, most of liveness properties fail to be valid. In order the liveness properties to be verified, a concrete finite scheduler has to be modeled and time constraints are introduced. This gives a less abstract specification but it allows to replace some liveness by bounded eventualities which are safety properties.

We focus on the class of regular safety properties on infinite trees and consider the problem of their characterization by both tree-automata and formulas of a branching time logic.

First, we define a kind of tree-automata which we call safety-recognizers (SR for short) and show that they define exactly closed regular sets of trees. Then, we define a branching time logic called BSL (Branching Time Safety Logic) which is a fragment of the μ-calculus. We show that it is adequate for the safety equivalence and that it is expressively equivalent to the class of SR definable sets of trees. We give also a complete deductive system for this logic. In particular, we establish that any branching time safety property can be expressed by a graph modulo simulation. This allows to reduce the problem of verifying a safety property to a simulation problem between regular trees.

The remainder of this paper is organized as follows. In section 2, we introduce a *semantical* characterization of safety and liveness properties for any given program semantics using a safety preorder on the computations. In section 3, we propose a safety preorder for bisimulation based branching time semantics and study its features. In section 4, we consider the class of RTA-definable properties and give a characterization of safety properties by tree-automata. In section 5, we give a logical characterization of these properties. Finally, we discuss in the conclusion the choice of an appropriate safety preorder.

For lack of space, standard proofs are omitted and will be given in the full paper.

2 Safety and Liveness Properties

In this section, we study the notions of safety and liveness properties for any program semantics. We propose a general definition of these classes and show some general results concerning them.

Let us first fix the framework in which we carry out this study. We denote by $\mathcal{P}$ a term algebra used to describe programs. A program semantics is usually defined by a congruence relation on $\mathcal{P}$. In fact, in order to define a congruence on $\mathcal{P}$, we often consider a congruence on a class $\mathcal{M}$ of program models and establish a homomorphism μ from $\mathcal{P}$ to $\mathcal{M}$. Examples of classes of program models are labeled transition systems and event structures [Win83]. In order to define a congruence on $\mathcal{M}$, we introduce the notion of *computations* generated by a model. Let $\mathcal{C}$ be the set of computations generated by the elements of $\mathcal{M}$. First, we consider an equivalence relation $\simeq$ on $\mathcal{C}$ which we generalize to $2^{\mathcal{C}}$ by considering that for any sets of computations C_1 and C_2, $C_1 \simeq C_2$ if and only if $\forall x_1 \in C_1.\ \exists x_2 \in C_2.\ x_1 \simeq x_2$ and conversely). Then, we define a homomorphism χ from $\mathcal{M}$ to $2^{\mathcal{C}}$ associating with any model a set of computations closed under $\simeq$ and such that $\simeq$ is a congruence on $\chi(\mu(\mathcal{P}))$.

For example, if program models are labeled transition systems (LTS for short), we can consider as computations either sequences or trees. This gives rise respectively to linear and branching time semantics.

In linear time semantics, the set of computation sequences of a LTS may be defined as the set of its maximal paths and an equivalence on sequences may be simply equality. In this case, two programs are equivalent if and only if their models have the same sets of maximal paths. This equivalence is usually called *maximal trace equivalence*. More interesting equivalences can be chosen. For example, in order to deal with abstraction, one can consider the *stuttering trace equivalence*: two sequences are equivalent if and only if one of them can be obtained from the other by removing or adding finite repetitions. However, it is well known that maximal trace equivalences are not congruences for the restriction operator [Mil80].

In branching time semantics, the set of computations of a LTS is the set of trees obtained by unfolding it, starting from its initial states. In this case, any operation on models is defined as a generalization on sets of an operation on trees. Thus, a branching time semantics can be defined by a congruence on computation trees. Several such congruences have been defined, most of them based on bisimulation by incorporating abstraction criteria [Mil80], [GW89].

For a given program semantics, a property is a union of congruence classes of programs. Thus, congruent programs verify the same properties. In fact, if the semantics is defined as described above, we can define a property as a union of equivalence classes of computations; a program π satisfies a property P if and only if all the computations generated by its model belong to the property, i.e., $\chi(\mu(\pi)) \subseteq P$.

Having fixed what is a property for a given semantics, we are interested now in defining safety and liveness properties. Formal definitions of safety and liveness properties have been given for linear time semantics [Lam85], [Sis85], [AS87], [AL88], [MP89]. One of them (see for example [MP89]) defines safety properties as *closed* sets and liveness properties as *dense* sets in the Cantor topology on the set of finite or infinite computation sequences. We propose here a general topological definition of safety and liveness properties based on a preorder on computations. To this end, we need the following preliminary definitions.

First we consider a preorder defined as the limit of a decreasing family of preorders. We consider also the equivalences induced by these preorders.

Definition 2.1 *(limit preorder) Let $\mathcal{U}$ be a set and consider $\{\ll_k\}_{k\in I\!N}$ a family of decreasing preorders on $\mathcal{U}$. Consider also the family of the induced equivalences $\{\cong_k\}_{k\in I\!N}$ such that $\cong_k \;=\; \ll_k \cap \gg_k$, for any $k \in I\!N$. We take $\ll = \bigcap_{k\in I\!N} \ll_k$ and $\cong = \bigcap_{k\in I\!N} \cong_k$.*

Then, we define the notion of $\ll$-closure.

Definition 2.2 *($\ll$-closure) A subset S of $\mathcal{U}$ is $\ll$-closed if and only if $\forall x \in S.\ \forall x' \in \mathcal{U}.\ x' \ll x \Rightarrow x' \in S$. The $\ll$-closure of a subset S of $\mathcal{U}$ is the smallest $\ll$-closed superset of S in $\mathcal{U}$.*

Now, we define the notions of limit and limit-closure.

Definition 2.3 *(limit) Let $S = \{x_i\}_{i\in I\!N}$ be a subset of $\mathcal{U}$ and $x \in \mathcal{U}$. The set S* converges *to x if and only if*

$$\forall k.\ \exists j.\ \forall i \geq j.\ x_i \cong_k x$$

We say that x is a limit *of S.*

Notice that if a set S converges to x and x' then necessarily $x \;\cong\; x'$. Thus, the limit of a set is unique up to $\cong$.

Definition 2.4 *(closure by limit) A subset of $\mathcal{U}$ is limit-closed if and only if it contains all the limits of its subsets. The limit-closure of a subset S of $\mathcal{U}$ is the smallest limit-closed superset of S in $\mathcal{U}$.*

Now, we define a notion of closure which is both $\ll$-closure and limit-closure. We show then that it allows to define a topology on $\mathcal{U}$.

Definition 2.5 *(closed and open sets) A subset of $\mathcal{U}$ is* closed *if and only if it is $\ll$-closed and limit-closed. As usually, an open set is the complement of a closed set.*

Notice that any closed and any open set is closed under the equivalence $\cong$.

Proposition 2.1 *(Topology induced by $\ll$)*
The set of closed sets defines a topology on $\mathcal{U}$, which we denote by $\mathcal{I}(\mathcal{U}, \ll)$.

Proof: It can be shown that the empty set and the full set are closed sets, the intersection of closed sets is a closed set and the finite union of closed sets is a closed set. We give here only the proof of the last fact, i.e., that the union of two closed sets is a closed set. Let S_1 and S_2 be two closed sets. It is obvious that $S_1 \cup S_2$ is $\ll$-closed. We show that $S_1 \cup S_2$ is limit-closed. We consider a set $X = \{x_i\}_{i \in \mathbb{N}} \subseteq S_1 \cup S_2$ which converges to x and show that x is in $S_1 \cup S_2$. Let $Y = \{y_i\}_{i \in \mathbb{N}} = X \cap S_1$ and $Z = \{z_i\}_{i \in \mathbb{N}} = X \cap S_2$. We have to consider two cases:

- Y is finite and Z is finite,
  We have then X is finite, thus, since X has a limit, X converges to an $x' \in X \subseteq S_1 \cup S_2$. Since $S_1 \cup S_2$ is closed under $\cong$ (S_1 and S_2 are closed under $\cong$) and $x \cong x'$, we have $x \in S_1 \cup S_2$.

- Y is infinite or Z is infinite,
  We have $\forall k.\ \exists j.\ \forall i \geq j.\ x_i \cong_k x$. For a given k, let j be such that $\forall i \geq j.\ x_i \cong_k x$. Without loss of generality, consider that Y is infinite. We have then $\{x_i\}_{i \geq j} \cap Y \neq \emptyset$ and thus, $\exists j'.\ \forall i \geq j'.\ y_i \cong_k x$. We conclude that Y converges to x and since S_1 is closed, we have $x \in S_1$.

□

We recall the notions of topological closure and denseness.

Definition 2.6 *(Topological closure) For a given topology on a set $\mathcal{U}$, the* closure *of a set S, which we denote by $\overline{S}$, is the smallest closed set containing S. A set S is* dense *if and only if $\overline{S} = \mathcal{U}$.*

Now, we can give the definition of safety and liveness properties with respect to a limit preorder $\ll$ on $\mathcal{C}$ defined as in definition 2.1.

Definition 2.7 *(Safety and Liveness) Let $\simeq$ be the equivalence on $\mathcal{C}$ defining a program semantics and $\ll$ a limit preorder on $\mathcal{C}$ such that $\simeq \subseteq \ll$. A property is a safety (resp. liveness) property (with respect to $\mathcal{C}$ and $\ll$) if and only if it is a closed (resp. dense) set in the topology $\mathcal{I}(\mathcal{C}, \ll)$.*
We denote by $\Sigma(\mathcal{C}, \ll)$ (resp. $\Lambda(\mathcal{C}, \ll)$) the class of safety (resp. liveness) properties.

To capture the intuitive definition of safety and liveness, the preorder $\ll$ has to express a notion like "*is a restriction of*" or "*is less defined than*". For instance, for linear time semantics, where $\mathcal{U}$ is the set of finite or infinite sequences on an alphabet Σ, a suitable preorder is defined by considering the following family of preorders :

- $\ll_0 = \mathcal{U} \times \mathcal{U}$,

- $\forall k \in \mathbb{N}.\ \forall \sigma_1, \sigma_2 \in \mathcal{U}$.
  $\sigma_1 \ll_{k+1} \sigma_2$ if and only if $(\forall a \in \Sigma.\ \sigma_1 = a.\sigma_1' \Rightarrow (\sigma_2 = a.\sigma_2'$ and $\sigma_1' \ll_k \sigma_2'))$.

It is easy to see from the definition above that $\Sigma(\mathcal{C}, \ll)$ is closed under intersection and finite union, that $\Lambda(\mathcal{C}, \ll)$ is closed under union but not under intersection and that neither $\Sigma(\mathcal{C}, \ll)$ nor $\Lambda(\mathcal{C}, \ll)$ are closed under complementation. Furthermore, the empty set is a safety property and the only property which is both a safety and a liveness is the set $\mathcal{C}$ ($\mathcal{C}$ is not empty).

It is also easy to deduce from this definition that $\ll$ is exactly the preorder induced by safety properties on computations. This derives from the fact that a safety property is by definition $\ll$-closed.

Proposition 2.2 $\forall x_1, x_2 \in \mathcal{C}.\ x_1 \ll x_2$ *if and only if* $\forall S \in \Sigma(\mathcal{C}, \ll).\ x_2 \in S \Rightarrow x_1 \in S$. □

Let $\cong$ be the equivalence induced by the preorder $\ll$, i.e., $\cong = \ll \cap \gg$. It is straightforward to deduce from the proposition above that $\cong$ coincides with the equivalence induced by the safety properties on computations.

It is more interesting to verify if this proposition remains true when we consider models instead of computations. It is straightforward to show the left to right direction using the proposition above. Thus, two $\cong$-equivalent models verify the same safety properties. However, the other direction is not true in general. To see this, let for example m_1 and m_2 be two models such that $\chi(m_1) = \overline{\chi(m_2)}$. By definition of the topological closure, we have $\chi(m_1) \subseteq S$ if and only if $\chi(m_2) \subseteq S$ for any safety property S, but $\chi(m_1) \ll \chi(m_2)$ only if $\chi(m_2)$ is limit-closed. In general, we have the following proposition.

Proposition 2.3 *(Adequacy)*

- $\forall m_1, m_2 \in \mathcal{M}.\ \chi(m_1) \ll \chi(m_2)$ *implies* $\forall S \in \Sigma(\mathcal{C}, \ll).\ \chi(m_2) \subseteq S \Rightarrow \chi(m_1) \subseteq S$.
- $\forall m_1, m_2 \in \mathcal{M}.$ ($\chi(m_2)$ *is limit-closed and* $\forall S \in \Sigma(\mathcal{C}, \ll).\ (\chi(m_2) \subseteq S \Rightarrow \chi(m_1) \subseteq S)$) *implies* $\chi(m_1) \ll \chi(m_2)$.

□

Notice that in particular, when χ associates with any model a limit-closed set of computations, $\cong$ is exactly the equivalence induced by safety properties on program models. For example, the set of maximal paths of a LTS is limit-closed. However, if we introduce some notion of fairness in the models, for example by considering as program models Büchi-like automata, the set of computations associated with a model (the language of the automaton) is clearly not limit-closed in general.

The proposition above shows that the semantics we need when we are interested by just safety properties can be defined by $\cong$. For this reason, we give to $\ll$ the name of *safety preorder* and to $\cong$ the name of *safety equivalence*. However, in order to be able to define a semantics based on $\cong$, we require that $\cong$ induces a congruence on $\chi(\mu(\mathcal{P}))$, or equivalently, that $\ll$ induces a precongruence on this class. Notice that for branching time semantics this means simply that $\ll$ has to be a precongruence.

Now, having defined safety and liveness properties, an interesting question is how an arbitrary property can be expressed in terms of safety and liveness. For example, in the linear time semantics case, Alpern and Schneider have shown in [AS87] that in the class of ω-regular properties, any property is the intersection of a safety and a liveness property. So, let us see how this result can be stated in our general frame. We define first the notion of *sl-separability.*

Definition 2.8 *(sl-separability) A property is* sl-separable *if and only if it is the intersection of a safety and a liveness property. We say that a class of properties is sl-separable if and only if all its properties are sl-separable.*

Thus, a property is sl-separable means that it can be split into a safety and a liveness part and each of these parts can be dealt with separately. In general, for any class of properties it is interesting to know if it is sl-separable or at least to find its largest sl-separable subset. A sufficient condition for sl-separability is expressed by the following proposition.

Proposition 2.4 *Any class of properties which is closed under union, complementation and topological closure is sl-separable.*

Proof: The proof is analogous to the one given in [AS87]. Take a semantics where $\mathcal{C}$ is the set of computations and let $\mathcal{E}$ be a class of properties for this semantics satisfying the hypothesis of the proposition. For any property P in $\mathcal{E}$, consider the sets $S = \overline{P}$ and $L = (\mathcal{C} - \overline{P}) \cup P$. These sets are clearly in $\mathcal{E}$. Furthermore, it is easy to see that they are respectively closed and dense and that $P = S \cap L$. □

In particular, for linear time semantics the class of properties corresponding to ω-regular languages is sl-separable since it satisfies the condition of the proposition above. However, the class of ω-context-free languages is not closed under complementation [CG77] and thus, the proposition above does not apply. In fact, it has been shown that this class is not sl-separable and some finest sufficient conditions for sl-separability of ω-context-free languages are given in [CL89].

In the sequel, we focus on bisimulation based branching time semantics. We have seen in this section that a safety preorder for such a semantics has to be a precongruence preserving bisimulation and allowing to capture the intuitive definition of safety and liveness properties. We claim that such a preorder is the simulation preorder combined with an abstraction criterion. We propose in the following section a safety preorder which preserves the observational congruence and the branching bisimulation.

3 Safety Preorder on Computation Trees

In this section, we define the safety preorder on computation trees and study its features. Throughout this paper we are only interested in regular computation trees which are unfoldings of finite LTSs. We define first an algebra of such computation trees.

3.1 Syntax

Let A be a finite vocabulary and τ a symbol not belonging to A. We call the elements of A *visible actions*, τ silent action and we take $A_\tau = A \cup \{\tau\}$. Let Ξ a set of variables. The language $\mathcal{L}(A_\tau, \Xi)$ of terms is given by the following grammar, where $x \in \Xi$ and $\alpha \in A_\tau$:

$$t ::= Nil \mid x \mid \alpha : t \mid t + t \mid t \times t \mid rec\ x.\ t$$

We use $\sum$ and $\prod$ respectively for finite summation and finite product. The notion of bound and free occurrence of variables are as in the first order predicate calculus, by considering *rec* as a quantifier. A term is closed if there is no variable occurring free in it. Let $\vec{x} = (x_0, \cdots, x_n)$ denote either a tuple or the set of its components. We use $t(\vec{x})$ to stand for the term t with free variables $\vec{x}$.

A free occurrence of a x in t is guarded if it occurs within some subterm $a : t'$ of t where $a \in A$, otherwise it is unguarded. If a term t contains an unguarded occurrence of x, we write $t \rhd x$. A term $rec\ x.\ t$ is guarded if is x guarded in t. A term t is guarded if every subterm $rec\ x.\ t'$ of t is guarded.

Let $\mathcal{T}(A_\tau, \Xi)$ the sublanguage of the guarded and closed terms. Terms of $\mathcal{T}(A_\tau, \Xi)$ represent computation trees, where *Nil* is the empty tree, $+$ is the non-deterministic choice operator, $\times$ is the synchronous product operator and *rec* is the recursion operator.

We denote by $\mathcal{L}(A, \Xi)$ (resp.$\mathcal{T}(A, \Xi)$) the sublanguage of $\mathcal{L}(A_\tau, \Xi)$ (resp. $\mathcal{T}(A_\tau, \Xi)$) of terms without any occurrence of τ.

3.2 Operational semantics

We give an operational semantics [Plo81] for the terms of $\mathcal{L}(A_\tau, \Xi)$ by defining a set of binary relations $\xrightarrow{\alpha} \subseteq \mathcal{L}(A_\tau, \Xi) \times \mathcal{L}(A_\tau, \Xi)$, for any $\alpha \in A_\tau$, as the least relations satisfying the following rules, where $t, t_i, r, r_i \in \mathcal{L}(A_\tau, \Xi), a \in A$ and $\alpha \in A_\tau$:

prefix $\quad \alpha : t \xrightarrow{\alpha} t$

choice $\quad \dfrac{t_1 \xrightarrow{\alpha} r_1}{t_1 + t_2 \xrightarrow{\alpha} r_1} \qquad \dfrac{t_2 \xrightarrow{\alpha} r_2}{t_1 + t_2 \xrightarrow{\alpha} r_2}$

product $\quad \dfrac{t_1 \xrightarrow{a} r_1, t_2 \xrightarrow{a} r_2}{t_1 \times t_2 \xrightarrow{a} r_1 \times r_2} \qquad \dfrac{t_1 \xrightarrow{\tau} r_1}{t_1 \times t_2 \xrightarrow{\tau} r_1 \times t_2} \qquad \dfrac{t_2 \xrightarrow{\tau} r_2}{t_1 \times t_2 \xrightarrow{\tau} t_1 \times r_2}$

recursion $\quad \dfrac{t[rec\ x.\ t/x] \xrightarrow{\alpha} r}{rec\ x.\ t \xrightarrow{\alpha} r}$

Let us recall that a LTS is a quadruple (Q, A_τ, ρ, Q_0) where Q is a set of states, $\rho \subseteq Q \times A_\tau \times Q$ is a labeled transition relation and $Q_0 \subseteq Q$ is the set of initial states.

With a term $t \in \mathcal{T}(A_\tau, \Xi)$, we associate the LTS $S_t = (\mathcal{T}(A_\tau, \Xi), A_\tau, \{\xrightarrow{\alpha}\}_{\alpha \in A_\tau}, \{t\})$.

3.3 Safety preorder

We define a safety preorder on computation trees as a simulation with an abstraction criterion. We recall the definition of a simulation.

Notation Let $\lambda \subseteq A_\tau^*$, and let $t, r \in \mathcal{L}(A_\tau, \Xi)$. We write $t \xrightarrow{\lambda} r$ if and only if:
$\exists u_1 \cdots u_n \in \lambda.\ \exists t_1, \cdots, t_{n-1} \in \mathcal{L}(A_\tau, \Xi).\ t \xrightarrow{u_1} t_1 \xrightarrow{u_2} t_2 \cdots t_i \xrightarrow{u_{i+1}} t_{i+1} \cdots t_{n-1} \xrightarrow{u_n} r.$

Definition 3.1 *Let Π be a family of disjoint languages of finite words on A_τ. For each $R \subseteq \mathcal{L}(A_\tau, \Xi)^2$, we define :*

$$\begin{aligned}\Psi_\Pi(R) = \{&(t_1, t_2) \in \mathcal{L}(A_\tau, \Xi)^2 \mid \\ &\forall \lambda \in \Pi.\ \forall r_1.\ (t_1 \xrightarrow{\lambda} r_1 \Rightarrow \exists r_2.\ (t_2 \xrightarrow{\lambda} r_2\ and\ (r_1, r_2) \in R))\ and \\ &t_1 \rhd X\ \Rightarrow\ t_2 \rhd X\}\end{aligned}$$

We define inductively a family of preorders $\{\sqsubseteq_k^\Pi\}_{k\in I\!N}$ *by* $\sqsubseteq_0^\Pi = \mathcal{L}(A_\tau,\Xi)^2$ *and* $\forall k \geq 0.\ \sqsubseteq_{k+1}^\Pi = \Psi_\Pi(\sqsubseteq_k^\Pi)$. *The* simulation preorder *for* Π *is defined by* $\sqsubseteq^\Pi = \bigcap_{k=0}^{\infty} \sqsubseteq_k^\Pi$. *Consider also the equivalence relations* $\approx_k^\Pi = \sqsubseteq_k^\Pi \bigcap \sqsupseteq_k^\Pi$ *for any* $k \geq 0$. *The* simulation equivalence *for* Π *is defined by* $\approx^\Pi = \bigcap_{k=0}^{\infty} \approx_k^\Pi$.

Different simulation preorders can be defined. The choice of a class Π (i.e., a partial partition of A_τ^*) corresponds to the choice of an *abstraction criterion* on the actions [Bou85]. The strong simulation preorder is defined by $\Pi = \{\{\alpha\} \mid \alpha \in A_\tau\}$. Of course, in this case there is no abstraction since τ-actions play no special role. However, we would like to abstract away from all the τ-actions. Interesting criteria for this purpose can be defined by considering either the class $\Delta = \{\tau^\star a \mid a \in A\}$ or $\Omega = \{\tau^\star a \tau^\star \mid a \in A\}$ or $\Delta' = \Delta \cup \{\tau^\star\}$ or $\Omega' = \Omega \cup \{\tau^\star\}$. However, contrary to the case of bisimulations, we can show that the simulation preorders $\sqsubseteq^\Delta$, $\sqsubseteq^\Omega$, $\sqsubseteq^{\Delta'}$ and $\sqsubseteq^{\Omega'}$ are the same using the following lemma.

Lemma 3.1 $\forall \Pi \in \{\Delta, \Omega, \Delta', \Omega'\}.\ \forall t, t' \in \mathcal{L}(A_\tau,\Xi).\ t \xrightarrow{\tau^*} t' \Rightarrow t' \sqsubseteq^\Pi t$. □

Proposition 3.1 $\sqsubseteq^\Delta = \sqsubseteq^\Omega = \sqsubseteq^{\Delta'} = \sqsubseteq^{\Omega'}$. □

Henceforth, we denote by $\sqsubseteq$ (resp. $\approx$) the preorder $\sqsubseteq^\Delta$ (resp. the equivalence $\approx^\Delta$) called *safety preorder* (resp. *safety equivalence*). Now, we study some properties of $\sqsubseteq$. First, we can show that it is a precongruence on the term language $\mathcal{L}(A_\tau,\Xi)$. First, we show that all the operators of the algebra are monotonic with respect to $\sqsubseteq$. We need some intermediate lemmas.

Lemma 3.2 $\forall k \in I\!N.\ \forall t, s_1, s_2 \in \mathcal{L}(A_\tau,\Xi).\ s_1 \sqsubseteq_k s_2 \Rightarrow t[s_1/x] \sqsubseteq_k t[s_2/x]$

Proof: (Induction on $k \in I\!N$). Suppose that $\forall t, s_1, s_2 \in \mathcal{L}(A_\tau,\Xi).\ s_1 \sqsubseteq_k s_2 \Rightarrow t[s_1/x] \sqsubseteq_k t[s_2/x]$ and $s_1 \sqsubseteq_{k+1} s_2$. If $t[s_1/x] \xrightarrow{\tau^* a} t'$ then either $t \xrightarrow{\tau^* a} r$ and $t' = r[s_1/x]$, or $t \rhd x \ \wedge\ s_1 \xrightarrow{\tau^* a} r_1$ and $t' = r_1$, or $t \xrightarrow{\tau^* a} r$ and $t \rhd x$ and $s_1 \xrightarrow{\tau^* a} r_1$ and $t' = r[s_1/x] \times r_1$.
In the first case, $t[s_2/x] \xrightarrow{\tau^* a} r[s_2/x]$ and $r[s_1/x] \sqsubseteq_k r[s_2/x]$ by induction hypothesis.
In the second case, since $s_1 \sqsubseteq_{k+1} s_2$, $\exists r_2.\ s_2 \xrightarrow{\tau^* a} r_2 \ \wedge\ r_1 \sqsubseteq_k r_2$.
In the last case, since $s_1 \sqsubseteq_{k+1} s_2$, $\exists r_2.\ s_2 \xrightarrow{\tau^* a} r_2 \ \wedge\ r_1 \sqsubseteq_k r_2$. Thus, $t[s_2/x] \xrightarrow{\tau^* a} r[s_2/x] \times r_2$. By induction hypothesis, $r[s_2/x] \times r_1 \sqsubseteq_k r[s_2/x] \times r_2$ and $r[s_1/x] \sqsubseteq_k r[s_2/x]$ and thus, $r[s_1/x] \times r_1 \sqsubseteq_k r[s_2/x] \times r_1$. Then by transitivity, $r[s_1/x] \times r_1 \sqsubseteq_k r[s_2/x] \times r_2$.
Thus, in all cases $\exists t$" such that $t[s_2/x] \xrightarrow{\tau^* a} t$" $\wedge\ t' \sqsubseteq_k t$". □

Similarly, we can prove the following lemma.

Lemma 3.3 $\forall k \in I\!N.\ \forall s, t_1, t_2 \in \mathcal{L}(A_\tau,\Xi).\ t_1 \sqsubseteq_k t_2 \Rightarrow t_1[s/x] \sqsubseteq_k t_2[s/x]$ □

Corollary 3.1 $\forall k \in I\!N.\ \forall s_1, s_2, t_1, t_2 \in \mathcal{L}(A_\tau,\Xi).\ s_1 \sqsubseteq_k s_2 \ \wedge\ t_1 \sqsubseteq_k t_2 \Rightarrow t_1[s_1/x] \sqsubseteq_k t_2[s_2/x]$. □

Proposition 3.2 *(precongruence)*
$\sqsubseteq$ *is a precongruence and* $\approx$ *is a congruence on* $\mathcal{L}(A_\tau,\Xi)$. □

Proof: We have to prove $\forall k \in I\!N.\ \forall t, t_1, t_2 \in \mathcal{L}(A_\tau,\Xi)$ and $\forall \alpha \in A_\tau$, if $t_1 \sqsubseteq_k t_2$, then

$$\begin{array}{rcl} \alpha : t_1 & \sqsubseteq_k & \alpha : t_2 \\ t_1 + t & \sqsubseteq_k & t_2 + t \\ t_1 \times t & \sqsubseteq_k & t_2 \times t \\ rec\ x.\ t_1 & \sqsubseteq_k & rec\ x.\ t_2 \end{array}$$

We give only the proof of the last property by induction on $k \in I\!N$. Notice that, by definition of the operational semantics of the terms, for any terms $t, t' \in \mathcal{L}(A_\tau,\Xi)$, $rec\ x.\ t \xrightarrow{\tau^* a} t'$ if and only if
$t[rec\ x.\ t/x] \xrightarrow{\tau^* a} t'$. Suppose that $t_1 \sqsubseteq_k t_2 \Rightarrow rec\ x.\ t_1 \sqsubseteq_k rec\ x.\ t_2$ and $t_1 \sqsubseteq_{k+1} t_2$.

(i) First, we show that $t_1[rec\ x.\ t_1/x] \sqsubseteq_{k+1} t_1[rec\ x.\ t_2/x]$:

If $t_1[rec\ x.\ t_1/x] \xrightarrow{\tau^* a} t'$, then necessarily $t_1 \xrightarrow{\tau^* a} t'_1$ with $t' = t'_1[rec\ x.\ t_1/x]$. Then,
$t_1[rec\ x.\ t_2/x] \xrightarrow{\tau^* a} t'_1[rec\ x.\ t_2/x]$. Now, we have $t_1 \sqsubseteq_{k+1} t_2$. Thus, by induction hypothesis,
$rec\ x.\ t_1 \sqsubseteq_k rec\ x.\ t_2$. By the lemma 3.2, $t'_1[rec\ x.\ t_1/x] \sqsubseteq_k t'_1[rec\ x.\ t_2/x]$.
Thus, we have $t_1[rec\ x.\ t_1/x] \sqsubseteq_{k+1} t_1[rec\ x.\ t_2/x]$.

(ii) By lemma 3.3, we have
$t_1[rec\ x.\ t_2/x] \sqsubseteq_{k+1} t_2[rec\ x.\ t_2/x]$. Then, by transitivity of $\sqsubseteq_{k+1}$, we have
$t_1[rec\ x.\ t_1/x] \sqsubseteq_{k+1} t_2[rec\ x.\ t_2/x]$.

□

The term language $\mathcal{L}(A_\tau, \Xi)$ can be extended by adding the asynchronous parallel operator, the restriction operator and the relabeling operator [Mil80], [Win83], [BK85]. We can shown that $\sqsubseteq$ is still a precongruence.

Proposition 3.3 *$\sqsubseteq$ is a precongruence for the asynchronous parallel, restriction and relabeling operators.* □

Let us compare now the safety equivalence with some other equivalences. We denote by $\sim_b$ strong bisimulation [Par81], $\sim_o$ observational equivalence [Mil80], [Mil89], $\sim_{bb}$ branching bisimulation [GW89], [GV90] and $\sim_{wt}$ weak maximal trace equivalence, i.e., for any two terms t et t', $t \sim_{wt} t'$ if and only if $WT(t) = WT(t')$, where for any term t, $WT(t)$ is the set of sequences of visible actions $a_1.a_2 \cdots$ such that there exists a maximal derivation $t \xrightarrow{\tau^*} t_1 \xrightarrow{a_1} t'_1 \xrightarrow{\tau^*} t_2 \xrightarrow{a_2} t'_2 \xrightarrow{\tau^*} t_3 \cdots$.

Proposition 3.4 $\sim_b \subset \sim_{bb} \subset \sim_o \subset \approx \subset \sim_{wt}$. □

3.4 A Deductive System for the Safety Preorder

We present a sound and complete axiomatization of the safety preorder on $\mathcal{L}(A_\tau, \Xi)$. Let $\preceq$ be the relation defined by the deductive system below where we omit the usual rules stating that $\preceq$ is a precongruence and the rules for change of bound variables. An equation $t = t'$ stand for two inequations $t \preceq t'$ and $t' \preceq t$.

The Deductive system Ax_1 : For $t, t_1, t_2, t_3 \in \mathcal{L}(A_\tau, \Xi)$ and $a, b \in A$ and $x \in \Xi$, we have:

(I_1) $\quad Nil \preceq t$

(S_1) $\quad (t_1 + t_2) + t_3 = t_1 + (t_2 + t_3)$
(S_2) $\quad t_1 + t_2 = t_2 + t_1$
(S_3) $\quad t + t = t$
(S_4) $\quad t + Nil = t$

(P_1) $\quad (t_1 \times t_2) \times t_3 = t_1 \times (t_2 \times t_3)$
(P_2) $\quad t_1 \times t_2 = t_2 \times t_1$
(P_3) $\quad t \times t = t$
(P_4) $\quad t \times Nil = Nil$
(P_5) $\quad a : t_1 \times a : t_2 = a : (t_1 \times t_2)$
(P_6) $\quad a : t_1 \times b : t_2 = Nil$ if $a \neq b$
(P_7) $\quad t_1 \times (t_2 + t_3) = (t_1 \times t_2) + (t_1 \times t_3)$

(τ_1) $\quad \tau : t = t$

(R_1) $\quad recx.\ t = t[recx.\ t/x]$
(R_2) $\quad t_1 \preceq t_2[t_1/x] \Rightarrow t_1 \preceq recx.\ t_2$ provided x guarded in t_2
(R_3) $\quad t_2[t_1/x] \preceq t_1 \Rightarrow recx.\ t_2 \preceq t_1$ provided x guarded in t_2
(R_4) $\quad recx.\ (x \times t + t') = recx.\ t'$

We write $\vdash_{Ax_1} t_1 \preceq t_2$, (resp . $\vdash_{Ax_1} t_1 = t_2$) when $t_1 \preceq t_2$ (resp. $t_1 = t_2$) can be proved by the deductive system Ax_1. We use simply the symbol $\vdash$ if there is no ambiguity about the considered deductive system.

Theorem 1 *(Soundness) If* $\vdash t_1 \preceq t_2$ *then* $t_1 \sqsubseteq t_2$. *Also, if* $\vdash t_1 = t_2$ *then* $t_1 \approx t_2$. $\Box$

The completeness proof follows the completeness proof given by Milner in [Mil89] for observational congruence. The differences are first, that we are dealing with a simulation instead of a bisimulation, and second, that we consider in addition to the operators he considers the $\times$-operator (synchronous product) and show that it can be eliminated even occurring in a recursion. This is due essentially to its idempotence (axiom P_3).

We recall briefly the notations and the terminology introduced in [Mil89]. Let $\vec{X} = (X_0, \cdots, X_n)$ and $\vec{W} = (W_0, \cdots, W_m)$ be disjoint sets of variables. Let $\vec{H} = (H_0, \cdots, H_n)$ be terms with free variables in $\vec{X} \cup \vec{W}$. A *system of equations* $S = (\vec{E}, \vec{X}, \vec{W}, E_0)$ is a set of formal equations $\vec{E} : \vec{X} = \vec{H}$. We call $\vec{X}$ the formal variables of E and say that E has free variables *in* $\vec{W}$. $E_0 : X_0 = H_0$ is the *leading equation* and X_0 is the *distinguished variable*. We say that $\vec{E} : \vec{X} = \vec{H}$ is *standard* if each H_i takes the form

$$\sum_j a_{ij} : X_j + \sum_k (\prod_l W_{ikl} \times \sum_m a_{ikm} : X_l)$$

where the W_{ikl} are elements of $\vec{W}$ and all actions are in A. As in [Mil89], we say that t *provably satisfies* $\vec{E} : \vec{X} = \vec{H}$ if and only if there are terms $\vec{t} = (t_0, \cdots, t_n)$ such that $t \equiv t_0$ and $\vdash \vec{t} = \vec{H}[\vec{t}/\vec{X}]$. We take $\equiv$ as the syntactic identity. The following propositions are used for the proof of the completeness.

Proposition 3.5 *(τ-elimination) For any term t in $\mathcal{L}(A_\tau, \Xi)$ there exists a term t' in $\mathcal{L}(A, \Xi)$ i.e., without τ-actions, such that $\vdash t = t'$.* $\Box$

Proposition 3.6 *For every term t in $\mathcal{L}(A_\tau, \Xi)$, there exists a guarded term t' in $\mathcal{L}(A, \Xi)$ such that $\vdash t = t'$.* $\Box$

The propositions above say that every term of $\mathcal{L}(A_\tau, \Xi)$ is provable equivalent to a guarded term without τ-actions. This is due to axioms τ_1 and R_4.

Proposition 3.7 *(Equational characterization) Every guarded term $t \in \mathcal{L}(A, \Xi)$ with free variables in $\vec{W}$ provably satisfies a standard equation set $\vec{E} : \vec{X} = \vec{H}$ with free variables in $\vec{W}$.* $\Box$

Proposition 3.8 *Let $t \in \mathcal{L}(A, \Xi)$ provably satisfy S and $t' \in \mathcal{L}(A, \Xi)$ provably satisfy S', where both S and S' are standard sets of equations, and let $t \sqsubseteq t'$. Then there is a standard set of equations $\vec{E} : \vec{X} = \vec{H}$, constructed from S and S', provably satisfied by t and such that $\exists \vec{t'} = (t'_0, \cdots, t'_n)$ with $t' \equiv t'_0$ and $\vec{H}[\vec{t'}/\vec{X}] \preceq \vec{t'}$.* $\Box$

Proposition 3.9 *(Unique solution of equations) If S is a equation set $\vec{E} : \vec{X} = \vec{H}$ with free variables in $\vec{W}$, then there is a term $t \in \mathcal{L}(A, \Xi)$ which provably satisfies S. Moreover, if there are terms $\vec{t'} = (t'_0, \cdots, t'_n)$ (with free variables in $\vec{W}$) such that $\vdash \vec{H}[\vec{t'}/\vec{X}] \preceq \vec{t'}$ then $\vdash t \preceq t'_0$.* $\Box$

From this proposition, it can be deduced that the solution of any equation set is unique up to $=$.

Theorem 2 *(Completeness) For any terms t and t' if $t \sqsubseteq t'$, then $\vdash t \preceq t'$.*

Proof: With any guarded terms $t, t' \in \mathcal{L}(A_\tau, \Xi)$, such that $t \sqsubseteq t'$, we associate two guarded terms $u, u' \in \mathcal{L}(A, \Xi)$ such that $\vdash t = u$ and $\vdash t' = u'$ by propositions 3.5 and 3.6. Then, we associate respectively with u and u' a standard equation set $\vec{E} : \vec{X} = \vec{H}$ and $\vec{E'} : \vec{X'} = \vec{H'}$ by proposition 3.7. There is a standard equation set $\vec{E''} : \vec{X''} = \vec{H''}$, by proposition 3.8, such that $\exists \vec{u} = (u_0, \cdots, u_n)$ and $\vec{u'} = (u'_0, \cdots, u'_n)$, with $u_0 \equiv u$ and $u'_0 \equiv u'$, and $\vdash \vec{u} = \vec{H''}[\vec{u}/\vec{X''}]$ and $\vdash \vec{H''}[\vec{u'}/\vec{X''}] \preceq \vec{u'}$. Finally, by proposition 3.9 and unicity of the solution for an equation set, we have $\vdash u \preceq u'$. $\Box$

4 Automata Based Characterization of Safety Properties

We consider as program specifications regular sets of computation trees modulo strong bisimulation. This corresponds to bisimulation closed RTA-definable sets [Rab69], [Rab72].

We study the closure of this class under topological closure in the topology induced by $\sqsubseteq$. We show the sl-separability of this class and give a characterization of regular safety properties in terms of automata.

Notice that we consider here only computation trees without τ-actions since they can always be eliminated (see proposition 3.5). For technical reasons, we represent computation trees by *Kripke trees* (KT) where the labels are on the states and not on the transitions. It is straightforward to transform a KT into a computation tree and conversely. Therefore, we give a definition of Rabin tree-automata which is not exactly standard: we consider automata recognizing bisimulation closed sets of KT with possibly finite paths and with non bounded but finite branching degree.

Definition 4.1 *(Kripke tree) A* Kripke tree *is a tuple* $K = (Q, A, q_0, \longrightarrow, \pi)$ *where* Q *is a countable set of states,* q_0 *is the initial state,* $\longrightarrow \subseteq Q \times Q$ *is a transition relation without cycles such that* $|\{q' \in Q \mid q \longrightarrow q'\}| < \omega$ *and* $\pi : Q \longrightarrow A$ *is a labeling function of* Q *by* A.

Definition 4.2 *(Rabin tree-automaton)*
A Rabin tree-automaton *is a tuple* $R = (A, W, w_0, \rho, \Omega, F)$ *where* W *is a finite set of states,* w_0 *is the initial state and* $\rho \subseteq W \times A \times 2^W$ *is a "branching" transition relation,* Ω *is a finite set of pairs* $(L_i, U_i) \in 2^W \times 2^W$ *and* $F \subseteq W$ *is a set of final states.*
A Kripke tree $K = (Q, A, q_0, \longrightarrow, \pi)$ *is* accepted *by* R *if and only if* $\exists \lambda : Q \longrightarrow W$ *such that;*

(1) $\lambda(q_0) = w_0$

(2) $\forall q \in Q.\ (\lambda(q), \pi(q), \{\lambda(q_1), \cdots, \lambda(q_n)\}) \in \rho$ *where* $\{q_1, \cdots, q_n\} = \{q' \in Q \mid q \longrightarrow q'\}$.

(3-1) *For any* σ *an infinite path in* K *starting in* q_0, *there exists* $(L, U) \in \Omega$ *such that* $Inf(\lambda(\sigma)) \cap L \neq \emptyset$ *and* $Inf(\lambda(\sigma)) \cap U = \emptyset$,

(3-2) *For any* $\sigma = q_0 \cdots q_n$ *a finite path in* K, $\lambda(q_n) \in F$,

where $\lambda(q_0.q_1 \cdots) = \lambda(q_0).\lambda(q_1) \cdots$ *and* $Inf(w_0.w_1 \cdots) = \{w \in W \mid \forall i \in I\!N.\ \exists j > i.\ w_j = w\}$.

Despite the fact that our definition is quite different from the standard one, the class of sets definable by our Rabin tree-automata is closed under union, intersection and complementation [Rab69], [Rab72], [Sao86].

We introduce now automata which we call *Safety Recognizers* (SR for short) and show that they define exactly regular safety properties with respect to $\sqsubseteq$.

Definition 4.3 *(Safety recognizer)*
A safety recognizer is a tuple $S = (A, W, w_0, \rho)$ *where* W *is a finite set of states,* w_0 *is the initial state and* $\rho \subseteq W \times A \times 2^W$. *A Kripke tree* $K = (Q, A, q_0, \longrightarrow, \pi)$ *is accepted by* S *if and only if* $\exists \lambda : Q \longrightarrow W$ *such that:*

(1) $\lambda(q_0) = w_0$

(2) $\forall q \in Q.\ \exists \Gamma \subseteq W.\ (\lambda(q), \pi(q), \Gamma) \in \rho$ *and* $\{\lambda(q_1), \cdots, \lambda(q_n)\} \subseteq \Gamma$
where $\{q_1, \cdots q_n\} = \{q' \in Q \mid q \longrightarrow q'\}$.

Proposition 4.1 *Any SR-definable set is RTA-definable.*

Proof: Let $S = (A, W, w_0, \rho)$ be a SR. An equivalent RTA is $(A, W, w_0, \rho', \Omega, F)$ where $\Omega = \{(W, \emptyset)\}$, $F = W$ and ρ' is the smallest relation ρ verifying:
$\forall w \in W.\ \forall a \in A.\ (w, a, \Gamma) \in \rho \ \Rightarrow\ \forall \Gamma' \subseteq \Gamma.\ (w, a, \Gamma') \in \rho'$. □

Proposition 4.2 *Any SR-definable set is closed.* □

Proposition 4.3 *The closure of a RTA-definable set is SR-definable.*

Proof: Let P be the set defined by the RTA $R = (A, W, w_0, \rho, \Omega, F)$. Consider the SR obtained from R in the following manner:

1 Remove from R all the states that accept the empty language (the emptiness problem of a RTA is decidable [Rab72], [HR72]). Let W' be the remaining states and ρ' the transition relation modified accordingly : a transition (w, a, Γ) is removed if there exists a state in Γ which is not in W'. The resulting automaton $R' = (A, W', w_0, \rho', \Omega', F')$, where Ω' and F' are the projections of Ω and F on W', defines also the set P.

2 Take $S = (A, W', w_0, \rho')$.

We show that any tree defined by S is in $\overline{P}$, i.e., for any tree t defined by S, we have $t \sqsubseteq t'$ where t' is the limit of a set of trees in P. Let t be a tree defined by S with a labeling λ (see definition 4.3). For any $k \geq 0$, we denote by t_k the full subtree of t of depth k. Each node, and in particular each leaf, of t_k is labeled with λ by a state in W' which accepts a tree in R' by definition. Thus, we can prolong t_k to a tree t'_k in P (each node of t is prolonged in the same way in all the trees t'_k for $k \geq 0$). It easy to verify that the obtained set of trees t'_k in P converges to a tree t' such that $t \sqsubseteq t'$.

Conversely, since S defines a closed set containing P, and $\overline{P}$ is the smallest closed set containing P, S defines $\overline{P}$. □

From propositions 4.2 and 4.3 we obtain the following theorem on SR's expressiveness.

Theorem 3 *(expressiveness)*
The class of SR-definable sets is exactly the class or regular safety properties with respect to $\sqsubseteq$. □

Finally, we give the theorem which states the sl-separability of the class of regular properties on computation trees.

Theorem 4 *(sl-separability)*
The class of RTA-definable properties is sl-separable.

Proof: By propositions 4.3 and 4.1, the class of RTA-definable sets is closed under topological closure. Furthermore, it is closed under union and complementation. Thus, it is sl-separable by proposition 2.4. □

5 The Branching Time Safety Logic BSL

In this section, we give a branching time logic defining exactly the class of safety properties with respect to $\sqsubseteq$.

5.1 Syntax

A formula of $\mathcal{U}(A, \Xi)$ is either $\mathbf{F}$ or defined by the following grammar:

$$\phi ::= Nil \mid \mathbf{T} \mid x \mid B : \phi \mid \phi \oplus \phi \mid \phi \ \wedge \ \phi \mid \phi \ \vee \ \phi \mid \nu x.\ \phi$$

where $x \in \Xi, B \subseteq A$. We denote by $\mathcal{F}(A, \Xi)$ the closed and guarded formulas, where the notions of free variable, closed formula and guardedness are the same as the ones defined in section 3.1 for terms.

5.2 Semantics

The semantics of $\mathcal{U}(A, \Xi)$ is defined by associating with any formula ϕ a set of terms $[[\phi]](\vec{s}) \in 2^{\mathcal{T}(A_\tau, \Xi)}$ for a valuation $\vec{s}$ for the variables occurring free in ϕ, where a valuation is a tuple $\vec{s} = (s_0, \cdots, s_n)$ of closed formulas different from $\mathbf{F}$:

$$[[\phi]](\vec{s}) = \bigcap_{k \in \mathbb{N}} [[\phi]]_k(\vec{s})$$

where $[[\phi]]_0(\vec{s}) = \mathcal{T}(A_\tau, \Xi)$ and for any $k > 0$, $[[\phi]]_k(\vec{s})$ is inductively defined by

- $[[Nil]]_k(\vec{s}) = \{t \in \mathcal{T}(A_\tau, \Xi) \mid I(t) = \emptyset\}$ where $I(t) = \{a \in A \mid \exists t'.\ t \xrightarrow{\tau^* a} t'\}$
- $[[\mathbf{F}]]_k(\vec{s}) = \emptyset$
- $[[\mathbf{T}]]_k(\vec{s}) = \mathcal{T}(A_\tau, \Xi)$
- $[[x_i]]_k(\vec{s}) = [[s_i]]$
- $[[B : \phi]]_k(\vec{s}) =$

$$\{t \in \mathcal{T}(A_\tau, \Xi) \mid \forall a \in A.\ \forall t' \in \mathcal{T}(A_\tau, \Xi).\ t \xrightarrow{\tau^* a} t' \Rightarrow (t' \in [[\phi]]_{k-1}(\vec{s}) \text{ and } a \in B)\}$$

- $[[\phi_1 \oplus \phi_2]]_k(\vec{s}) = \{t \in \mathcal{T}(A_\tau, \Xi) \mid \exists t_1 \in [[\phi_1]]_k(\vec{s}).\ \exists t_2 \in [[\phi_2]]_k(\vec{s}).\ t \sim_b t_1 + t_2\}$
- $[[\phi_1 \ \wedge \ \phi_2]]_k(\vec{s}) = [[\phi_1]]_k(\vec{s}) \cap [[\phi_2]]_k(\vec{s})$
- $[[\phi_1 \ \vee \ \phi_2]]_k(\vec{s}) = [[\phi_1]]_k(\vec{s}) \cup [[\phi_2]]_k(\vec{s})$
- $[[\nu x.\ \phi]]_k(\vec{s}) = [[\phi[\nu x.\ \phi/x]]]_k(\vec{s})$.

Notice that for any formula ϕ different from $\mathbf{F}$, $Nil \in [[\phi]]_k(\vec{s})$.

Now, we define an implication relation between formulas in the following manner : for any two formulas ϕ and ψ, we say that ϕ *implies* ψ if and only if $[[\phi]](\vec{s}) \subseteq [[\psi]](\vec{s})$ for any valuation $\vec{s}$.

5.3 A Deductive System for BSL

We give a sound and complete axiomatization for the implication relation on guarded formulas. Analogous to section 3.4, we define a relation $\preceq$ by the deductive system below where we omit the rules stating that $\preceq$ is a precongruence and the rules for change of bound variables and the propositional calculus rules. An equation $\phi = \phi'$ stands for two implications $\phi \preceq \phi'$ and $\phi' \preceq \phi$.

The Deductive System Ax_2 **:** For $\phi, \phi_1, \phi_2, \phi_3$ guarded formulas of $\mathcal{U}(A, \Xi)$ and $B, B_1, B_2 \subseteq A$ and $x \in \Xi$, we have:

(I_1) $\quad Nil \preceq \phi$ provided ϕ different from $\mathbf{F}$
(I_2) $\quad \mathbf{F} \preceq Nil$

(S_1) $\quad (\phi_1 \oplus \phi_2) \oplus \phi_3 = \phi_1 \oplus (\phi_2 \oplus \phi_3)$
(S_2) $\quad \phi_1 \oplus \phi_2 = \phi_2 \oplus \phi_1$
(S_3) $\quad \phi \oplus \phi = \phi$
(S_4) $\quad \phi \oplus Nil = \phi$
(S_5) $\quad \phi \oplus \mathbf{T} = \mathbf{T}$
(S_6) $\quad \mathbf{T} = A : \mathbf{T}$

(T_1) $\quad (B_1 : \phi_1) \ \wedge \ (B_2 : \phi_2) = (B_1 \cap B_2) : (\phi_1 \ \wedge \ \phi_2)$
(T_2) $\quad (B_1 : \phi) \oplus (B_2 : \phi) = (B_1 \cup B_2) : \phi$
(T_3) $\quad \emptyset : \phi = Nil$
(T_4) $\quad (B : \phi_1) \oplus (B : \phi_2) = B : (\phi_1 \ \vee \ \phi_2)$
(T_5) $\quad \phi_1 \oplus (\phi_2 \ \wedge \ \phi_3) = (\phi_1 \oplus \phi_2) \ \wedge \ (\phi_1 \oplus \phi_3)$
(T_6) $\quad \phi_1 \oplus (\phi_2 \ \vee \ \phi_3) = (\phi_1 \oplus \phi_2) \ \vee \ (\phi_1 \oplus \phi_3)$
(T_7) $\quad \phi_1 \ \wedge \ (\phi_2 \oplus \phi_3) = (\phi_1 \ \wedge \ \phi_2) \oplus (\phi_1 \ \wedge \ \phi_3)$

(R_1) $\quad \nu x.\ \phi = \phi[\nu x.\ \phi/x]$
(R_2) $\quad \phi \preceq \psi(\phi) \Rightarrow \phi \preceq \nu x.\ \psi$
(R_3) $\quad \phi \succeq \psi(\phi) \Rightarrow \phi \succeq \nu x.\ \psi$

We write $\vdash_{Ax_2} \phi_1 \preceq \phi_2$, (resp. $\vdash_{Ax_2} \phi_1 = \phi_2$) when $\phi_1 \preceq \phi_2$ (resp. $\phi_1 \preceq \phi_2$ and $\phi_2 \preceq \phi_1$) can be proved by the deductive system Ax_2. We use simply the symbol $\vdash$ when there is no ambiguity on the concerned deductive system.

Theorem 5 *(Soundness) If* $\vdash \phi_1 \preceq \phi_2$ *then* ϕ_1 *implies* ϕ_2. $\square$

We can prove the completeness of the deductive system by following the steps of the completeness proof of theorem 2 by replacing the proposition 3.7 by the following:

Proposition 5.1 *(Equational characterization) For each guarded formula $\phi \in \mathcal{U}(A,\Xi)$ different from* **F**, *there exists an equation set $\vec{E} : \vec{X} = \vec{H}$ such that:*

- $\vec{\phi} = \vec{H}[\vec{\phi}/\vec{X}]$
- *the leading equation is of the form* $\bigvee_i [\bigoplus_j a_{ij} : X_j + \bigoplus_k (\prod_l W_{ikl} \times \bigoplus_m a_{ikm} : X_l)]$,
- *each other equation is of the form* $\bigoplus_j a_{ij} : X_j + \bigoplus_k (\prod_l W_{ikl} \times \bigoplus_m a_{ikm} : X_l)$.

Proof: By structural induction, in the same way as in the proof of proposition 3.7. Notice that the disjunction operator may only appear in the leading equation due to the axiom T_4. □

Theorem 6 *(Completeness)*
Let be ϕ, ϕ' be guarded formulas in $\mathcal{U}(A,\Xi)$. If ϕ implies ϕ' then $\vdash \phi \preceq \phi'$. □

5.4 Adequacy and Expressiveness

We show that BSL characterizes exactly safety properties with respect to $\sqsubseteq$ and thus, that it is adequate for safety equivalence, i.e., any formula of BSL represents a union of $\approx$-classes. Furthermore, we show that any formula of BSL different from **F** has a largest model (LTS) with respect to $\sqsubseteq$ satisfying it. Thus, any safety property can be represented by a LTS up to simulation and verifying any safety property reduces to simulation testing.

First, we consider a subclass of $\mathcal{U}(A,\Xi)$ isomorphic to $\mathcal{L}(A,\Xi)$. Formulas of this class are called in *standard form.*

Definition 5.1 *A formula is called in* standard form *if it is different from* **F**, *it does not contain any disjunction operator and as prefixing operators only singleton sets are used.*

As a corollary of proposition 5.1, we have the following lemma.

Lemma 5.1 *Any guarded formula of $\mathcal{U}(A,\Xi)$ different from* **F** *is equivalent to a disjunction of formulas in standard form.* □

We establish now the link between guarded terms of $\mathcal{L}(A,\Xi)$ and guarded formulas of $\mathcal{U}(A,\Xi)$ in standard form.

Definition 5.2 *Let Ψ be the function from guarded terms of $\mathcal{L}(A,\Xi)$ to guarded formulas of $\mathcal{U}(A,\Xi)$, defined inductively by :*

- $\Psi_{\mathrm{Nil}} = \mathrm{Nil}$.
- $\Psi_x = x$.
- $\Psi_{a:t} = \{a\} : \Psi_t$.
- $\Psi_{t_1+t_2} = \Psi_{t_1} \oplus \Psi_{t_2}$.
- $\Psi_{t_1 \times t_2} = \Psi_{t_1} \wedge \Psi_{t_2}$.
- $\Psi_{\mathrm{rec}\ x.\,t} = \nu x.\ \Psi_t$

Proposition 5.2 (i) *Let be $\phi \in \mathcal{F}(A,\Xi)$ in standard form. Then there exists a guarded term t such that $\Psi_t = \phi$*

(ii) *Let t_1, t_2 be guarded terms of $\mathcal{L}(A,\Xi)$. $\vdash_{Ax_1} t_1 \preceq t_2 \Leftrightarrow \vdash_{Ax_2} \Psi_{t_1} \preceq \Psi_{t_2}$*

Proof:

(i) Since to any operator on formulas in standard form corresponds an operator on terms.

(ii) Since the image by Ψ of the deductive system Ax_1 is a subset of the deductive system Ax_2.

□

Proposition 5.3 $\forall t \in \mathcal{T}(A,\Xi).\ \forall t' \in \mathcal{T}(A_\tau,\Xi).\ t' \in [[\Psi_t]] \Leftrightarrow t' \sqsubseteq t.$ □

We recall that a finite-state program model (LTS) can be represented by a finite set of terms corresponding to the computation trees which are obtained by unfolding the model starting in its initial states. A model *satisfies* a formula ϕ if and only if all its terms are in $[[\phi]]$. We show that for any guarded and closed BSL formula different from $\mathbf{F}$, the set of models satisfying it has a largest element with respect to the preorder $\sqsubseteq$ extended to sets of terms.

Proposition 5.4 *(largest model satisfying a formula)*
$\forall \phi \in \mathcal{F}(A,\Xi).\ \phi \neq \mathbf{F}.\ \exists t^1_\phi, \cdots, t^n_\phi \in \mathcal{T}(A,\Xi).\ \forall t \in \mathcal{T}(A_\tau,\Xi).\ t \in [[\phi]] \ \Leftrightarrow\ \exists i \in [1,n].\ t \sqsubseteq t^i_\phi.$

Proof: By lemma 5.1 and propositions 5.2(i) and 5.3. □

Theorem 7 *(expressiveness)*
A set is BSL-definable if and only if it is SR-definable.

Proof:
(if direction) : We associate with any SR $S = (A, W, w_0, \rho)$ the equation set:

$$\{X_w = \bigvee_{(w,a,\{w_1,\cdots,w_n\})\in\rho} a : \bigoplus_{i=1}^{n} X_{w_i} \mid w \in W\}.$$

We can prove by induction on the number of equations that for any such equation set there exists a BSL formula satisfying it.

(only if direction) : By proposition 5.4, for any BSL formula there exists a largest model satisfying it, say $\{t^1_\phi, \cdots, t^n_\phi\}$. Any term in this model represents a regular computation tree which can be transformed to a KT (see definition 4.1). This KT, since it is regular, is the unfolding of a finite-state structure which can be viewed as a SR and a finite union of SR's is a SR. □

The theorem above says that BSL characterizes exactly the class of regular safety properties. Thus, by proposition 5.4, any safety property is definable by a LTS up to simulation. Furthermore, by proposition 2.2, BSL is adequate for safety equivalence.

Corollary 5.1 *(adequacy for safety equivalence)*
$\forall t_1, t_2 \in \mathcal{T}(A_\tau,\Xi).\ t_1 \approx t_2$ *if and only if* $\forall \phi \in \mathcal{F}(A,\Xi).\ t_1 \in [[\phi]] \ \Leftrightarrow\ t_2 \in [[\phi]].$ □

Notice that the adequacy of BSL for the safety equivalence remains true when we consider finite models (LTS) instead of computation trees (see proposition 2.3).

6 Conclusion

We have proposed a safety preorder for branching time semantics based on simulation and studied the class of safety properties it defines. It turned out that this class coincides with the intuitive notion of safety properties.

The preorder we have chosen is not sensitive to *deadlock*, i.e., reachability of a state in which no visible action can be performed. Indeed, a tree with a finite path may be equivalent to a tree without finite paths. Thus, if the specifications require a system to be deadlock free, this has to be verified separately.

A deadlock sensitive simulation is *completed simulation* [vG90], where a deadlock state does not simulate any non deadlock state. However, it is easy to see that this relation is not a precongruence for the restriction operator and therefore also for parallel operators incorporating restriction as e.g., the synchronous product.

A deadlock sensitive safety preorder which is a precongruence for the operators we consider in this paper can be based on *ready simulation* [vG90] with abstraction, where two related states have the same visible initial actions.

The abstraction criterion we have chosen (the visible transitions are of the form $\tau^* a$ where a is a visible action) has several advantages in the contex of simulation:

- It allows to define a safety equivalence weaker than observational equivalence,
- It is very interesting in practice. It allows to eliminate all τ-transitions. Furthermore, this can be done locally during the generation of a program model. In fact, the algorithm is analogous to the one for the elimination of ϵ-transitions on finite-state automata.

Minimization and comparison up to safety equivalence are implemented in Aldébaran [Fer89], a tool for the verification of communicating systems represented by LTS's. Aldébaran has been used to compare efficiently protocols with their specifications up to safety equivalence; e.g. the results of the verification of a reliable atomic multicast protocol in the Application Layer can be found in [FM90].

Acknowlegments:

We thank N. Halbwachs for helpful discussions and Ph. Schnoebelen for his comments on a draft of this paper.

References

[AL88] M. Abadi and L. Lamport. The existence of refinement mappings. SRC 29, Digital Equipement Corporation, August 1988.

[AS87] B. Alpern and F.B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2:117–126, 1987.

[B$\ddot{6}$2] J.R. Büchi. On a decision method in restricted second order arithmetic. In Nagel et al., editor, *Logic, Methodology and Philosophy of Sciences*. Stantford Univ. Press, 1962.

[BK85] J. A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *TCS*, 37 (1), 1985.

[Bou85] G. Boudol. Notes on algebraic calculi of processes. In *Logics and Models for Concurrent Systems*. Springer Verlag, 1985. Nato ASI Series F (13).

[CG77] R. Cohen and A. Gold. Theory of ω-languages. *J. Comput. System Sci.*, 15:169–208, 1977.

[CL89] S. Chaudhuri and R.E. Ladner. Safety and liveness of ω-context-free languages. Technical Report 88-11-04, Department of Computer Sciences and Engineering, University of Washington, Seatle, Washington, 1989.

[Fer89] J. C. Fernandez. Aldébaran: A tool for verification of communicating processes. Tech. report Spectre C14, LGI-IMAG Grenoble, 1989.

[FM90] J. Fernandez and L. Mounier. Verification bisimulations on the fly. In *Proceedings of the Third International Conference on Formal Description Techniques FORTE'90 (Madrid, Spain)*, pages 91–105. North-Holland, November 1990.

[GV90] Jan Friso Groote and Frits Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. CS-R 9001, Centrum voor Wiskunde en Informatica, Amsterdam, January 1990.

[GW89] R.J. Van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics (extended abstract). CS-R 8911, Centrum voor Wiskunde en Informatica, Amsterdam, 1989.

[HR72] R. Hossley and C. Rackoff. The emptiness problem for automata on infinite trees. In *Proc. 13th IEEE Symp. on Switching and Automata Theory*, 1972.

[Koz83] D. Kozen. Results on the propositional μ-calculus. In *Theoretical Computer Science*. North-Holland, 1983.

[Lam77] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, SE-3(2):125–143, 1977.

[Lam85] L. Lamport. Logical foundation. In M. Paul and H.J. Siegert, editors, *Distributed Systems– Methods and Tools for Specification, LNCS 190*. Springer Verlag, 1985.

[LPZ85] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Conference on Logics of Programs, LNCS 194*. Springer Verlag, 1985.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *Second Int. Joint Conf. on Artificial Intelligence, BCS*, pages 481–489, 1971.

[Mil80] R. Milner. A calculus of communication systems. In *LNCS 92*. Springer Verlag, 1980.

[Mil89] R. Milner. A complete axiomatization for observational congruence of finite-state behaviours. *Information and Computation*, 81:227–247, 1989.

[MP84] Z. Manna and A. Pnueli. Adequate proof principles for invariance and liveness properties of concurrent programs. *Science of Computer Programming*, 32, 1984.

[MP89] Z. Manna and A. Pnueli. The anchored version of the temporal framework. In J.W. De Bakker, W.P. De Roover, and G. Rozenberg, editors, *Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency, LNCS 354*. Springer Verlag, 1989.

[Mul63] E.D. Muller. Infinite sequences and finite machines. In *4th IEEE Ann. Symp. on Switching Circuit and Logical Design*, pages 3–16, 1963.

[Niw88] D. Niwinski. Fixed points vs. infinite generation. In *Proc. of Third. Symp. on Logic in Computer Science*. Computer Society Press, 1988.

[Par81] D. Park. Concurrency and automata on infinite sequences. In *5th GI-Conference on Theorical Computer Science*. Springer Verlag, 1981. LNCS 104.

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Lecture notes, Aarhus University, 1981.

[Rab69] M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141, 1969.

[Rab70] M.O. Rabin. Weakly definable relations and special automata. In Y. Bar Hillel, editor, *Proc. Symp. Math. Logic and Foundations of Set Theory*. North-Holland, 1970.

[Rab72] M.O. Rabin. Automata on infinite objects and church's problem. In *Proc. Regional AMS Conf. Series in Mathemamtics*, number 13, 1972.

[Sao86] A. Saoudi. Variétés d'automates descendants d'arbres infinis. *TCS*, 43, 1986.

[Sis85] A. P. Sistla. On characterization of safety and liveness. In *Proc. 4th. Symp. Princ. of Dist. Comp.*, pages 39–48. ACM, 1985.

[Str82] R.S. Streett. Propositional dynamic logic of looping and converse. *Information and Control*, 54, 1982.

[vG90] R.J. van Glabbeek. The linear time - branching time spectrum. Technical Report CS-R9029, Centre for Mathematics and Computer Science, 1990.

[VW86] M.Y. Vardi and P. Wolper. Automata-theoritic techniques for modal logics of programs. *J. Comp. Sys. Sci.*, 32, 1986.

[Win83] G. Winskel. Synchronization trees. In J. Diaz, editor, *10th ICALP, LNCS 154*, 1983.

[Wol89] P. Wolper. On the relation of programs and computations to models of temporal logic. In *Temporal Logic in Specification, LNCS 398*, 1989.