Boundaries and Efficiency of Verification

Jan Strejček*

Faculty of Informatics, Masaryk University Brno,
Botanická 68a, 602 00 Brno, Czech Republic, strejcek@fi.muni.cz

SUPERVISOR: Mojmír Křetínský, Faculty of Informatics, Masaryk University Brno
KEYWORDS: model checking, linear temporal logic, stuttering, rewrite systems

Abstract. Formal verification of concurrent systems is in the center of interest of many researchers for a long time. Here we present directions and some results of our work in this area, namely rewrite systems with constraints and modifications of well known stuttering principle.

1 Research Area – Main Themes

Formal verification is one of the currently practiced methods for design validation. In contrast with the other methods, verification is able to pronounce that the validated system is correct. On the other hand, the problem of verification algorithms is their complexities: it is not possible to verify *large* systems.

The main theme of our research is to find theoretical as well as practical limits of verification and shift them.

2 Directions of the work

We are interested in two areas of formal verification: equivalence checking and model checking.

2.1 Equivalence checking

The aim of equivalence checking is to decide if two systems are the same with respect to given equivalence. The wide range of used equivalences is presented in [vG90].

One direction of our work is to locate the borders of decidability of these equivalences. We work with the standard classes of systems covered by PRS-hierarchy (see [May97] for details). All considered equivalences are decidable on finite systems. For some classes of infinte systems and some equivalences, it is already known whether the equivalence checking problem is decidable or not, but in some cases it is still an open question.

For example, Figure 1 represents the known results on (un)decidability of strong bisimulation within PRS-hierarchy. The situation between the borders has not been detected so far.

* Author has been partially supported by the FRVŠ, grant No. 601/2002, and by the Grant Agency of Czech Republic, grant No. 201/00/0400.

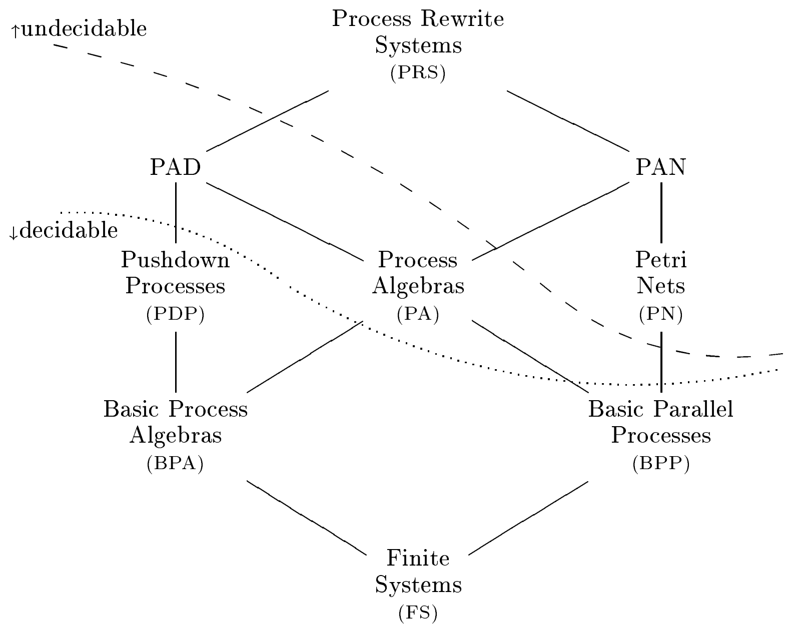

Fig. 1. The PRS-hierarchy and decidability of strong bisimulation.

Another approach (usable even in those cases where the border is already known) is to thicken the PRS-hierarchy and then refine the border. In this way, one can find decision algorithms for more general classes of systems.

2.2 Model checking

The aim of model checking (see e.g. [CGP99]) is to decide if a given system satisfies a given property or not. In the latter case, model checking provides a counterexample, i.e. the behavior of the system violating the property.

The property is usually specified as formula of modal or temporal logic. One of the frequently used logics is *linear temporal logic* (LTL) [Pnu81].

The complexity of the LTL model checking is linear with respect to the size of the system and exponential with respect to the size of the formula. The principal difficulty is that the size of the system is usually very large (due to the problem known as state-space explosion). There are several ways how to deal with this problem, e.g. symbolic model checking, abstraction and compositional techniques, state space reduction.

The idea of state-space reduction methods is to replace the original system by a smaller system such that the new one satisfies the property to be checked if and only if the original system satisfies it. An example of successful reduction method is partial order reduction.

The exponential complexity of model checking with respect to the size of a formula is not usually a crucial problem, as the formula is typically written by user and thus very small. But it can became a real problem when the formula is generated automatically and thus not necessarily so small. In this case the formula minimising algorithms can be very important.

Another direction of our work is to study (fragments of) logics used to denote properties. Such a study can provide solid background for new reduction methods and formula minimising algorithms as well as for specialised model checking algorithms.

3 Results

3.1 Rewrite systems with constraints

The results presented in this subsection have been published as [Str02].

The PRS-hierarchy [May97] has been built with use of rewrite systems with parallel ($t_1 \| t_2$) as well as sequential ($t_1.t_2$) compositions. The classes in the hierarchy correspond to different restrictions on the compositions used on the left and right hand sides of rewrite rules. Some PRS-hierarchy classes correspond to widely known models, e.g. pushdown processes, Petri nets, basic process algebras, etc.

We extend the rewriting formalism by a unit holding a kind of global information which can influence and can be influenced by rewriting. The unit is similar to the *store* used in concurrent constraint programming (CCP - see e.g. [Sar89]). States of the store are described by *constraint system*.

Definition 1. *A* constraint system *is a bounded lattice* $(C, \vdash, \wedge, tt, ff)$*, where* C *is the set of* constraints, $\vdash$ *(called* entailment*) is an ordering on this set,* $\wedge$ *is the* lub *operation, and* tt *(*true*),* ff *(*false*) are the least and the greatest elements of* C *respectively (*$ff \vdash tt$ *and* $tt \neq ff$*).*

A *process rewrite system with finite constraint system* $\mathcal{C} = (C, \vdash, \wedge, tt, ff)$ is defined by initial term t_0, finite set Δ of rewrite rules of the form $(t_1 \xrightarrow{a} t_2, m, n)$, where t_1, t_2 are terms, a is an atomic action, and $m, n \in C \setminus \{ff\}$ are constraints.

Semantics of process rewrite system with finite constraint system (fcPRS) is given by the corresponding labelled transition system. States of the transition system are pairs of process terms and constraints. The initial state is (t_0, tt). The transition relation $\longrightarrow$ is defined as the least relation satisfying the inference rules

$$\frac{(t_1 \xrightarrow{a} t_2, m, n) \in \Delta}{(t_1, o) \xrightarrow{a} (t_2, o \wedge n)} \quad \text{if } o \vdash m \text{ and } o \wedge n \neq ff,$$

$$\frac{(t_1, o) \xrightarrow{a} (t_1', p)}{(t_1 \| t_2, o) \xrightarrow{a} (t_1' \| t_2, p)}, \qquad \frac{(t_1, o) \xrightarrow{a} (t_1', p)}{(t_1.t_2, o) \xrightarrow{a} (t_1'.t_2, p)},$$

where t_1, t_2, t_1' are terms and m, n, o, p are constraints.

The two side conditions in the first inference rule are very close to principles used in CCP. The first one ($o \vdash m$) ensures the rule $(t_1 \xrightarrow{a} t_2, m, n) \in \Delta$ can be used only if the current state of the store o *entails* m (it is similar to *ask(m)* in CCP). The second condition ($o \wedge n \neq ff$) guarantees that the store stays *consistent* after application of the rule (analogous to the consistency requirement when processing *tell(n)* in CCP).

We observed how the presented extension of rewrite formalism changes the expressive power of classes in PRS-hierarchy (w.r.t. strong bisimulation). Our results are summarized in fcPRS-hierarchy presented in Figure 2, where fcXXX means class XXX extended by the store.

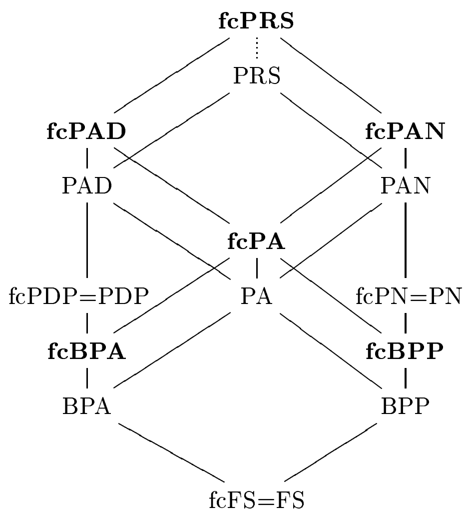

Fig. 2. The fcPRS-hierarchy

The edge between PRS and fcPRS classes in the fcPRS-hierarchy is dotted as we have no proof that the fcPRS class is strictly more expressive (w.r.t. strong bisimulation) than the PRS class.

The fcPRS-hierarchy can be seen as thickened PRS-hierarchy and can be used to refine the borders of decidability of equivalences as mentioned in Subsection 2.1.

Another possible topic for further research is to replace the constraint system with a finite state control, where the evolution of the actual state is determined by a given ordering.

3.2 n-stuttering and syllable stuttering

In this subsection we introduce one generalisation and one modification of well-known stuttering principle introduced in [Lam83].

Syntax of LTL is given by abstract syntax equation

$$\varphi \ ::= \ tt \ \mid \ p \ \mid \ \neg\varphi \ \mid \ \varphi_1 \wedge \varphi_2 \ \mid \ \mathcal{X}\varphi \ \mid \ \varphi_1 \,\mathcal{U}\, \varphi_2$$

where p ranges over a countable set $\Lambda = \{o, p, q, \ldots\}$ of *letters*. We define semantics of LTL in terms of languages over finite words (all of our results carry over to infinite words immediately). Let $\Sigma \subseteq \Lambda$ be a finite alphabet and $w = w(0)w(1)w(2)\ldots w(n-1) \in \Sigma^*$ be a finite word of lenght $|w| = n$. For every $0 \leq i < |w|$ we denote by w_i the i^{th} suffix of w, i.e. the word $w(i)\ldots w(|w|-1)$. Moreover, for all $0 \leq i < |w|$ and $j \geq 1$ such that $i + j \leq |w|$ the symbol $w(i,j)$ denotes the *syllable* (or subword) $w(i)w(i+1)\ldots w(i+j-1)$ of length j. To simplify our notation, whenever we refer to $w(i)$, w_i, or $w(i,j)$, we implicitly impose the condition that the object *exists*.

We use the standard definition of LTL semantics (see e.g. [Pnu81]). Every LTL formula φ defines a language L_φ over Σ consisting of all finite words satisfying φ. A language $L \subseteq \Sigma^*$ is an *LTL property* if $L = L_\varphi$ for some LTL fomula φ.

Let LTL($\mathcal{X}^n$) denote the fragment of LTL containing the formulas with at most n nested $\mathcal{X}$ operators and LTL($\mathcal{U}^m, \mathcal{X}^n$) the fragment of LTL containing the formulas with at most m nested $\mathcal{U}$ operators and n nested $\mathcal{X}$ operators.

In what follows we extend the standard stuttering principle working on LTL($\mathcal{X}^0$) only to whole LTL.

Definition 2. *Let $w \in \Sigma^*$ be a word. A letter $w(i)$ is* n-redundant *if $w(i) = w(i+1) = \ldots = w(i+n+1)$. The* n-canonical form *of w, denoted n:$[w]$, is obtained from w by deleting all n-redundant letters.*

Roughly speaking, a letter is n-redundant if it is followed by at least $n+1$ occurences of the same letter.

A language L is *n-stutter-invariant* iff for every pair $v, w \in \Sigma^*$ such that n:$[v] = n$:$[w]$ it holds that $v \in L \Longleftrightarrow w \in L$.

The stuttering principle says that every LTL($\mathcal{X}^0$) property is 0-stutter-invariant. The following theorem is a direct generalisation of standard stuttering.

Theorem 1. *Every LTL($\mathcal{X}^n$) property is n-stutter-invariant.*

The reverse of this theorem also holds. For $n = 0$ it was originally proven in [PW97].

Theorem 2. *Every n-stutter-invariant LTL property is an LTL($\mathcal{X}^n$) property.*

The theorem below generalises the result formulated for the standard stuttering in [PWW98].

Theorem 3. *Let $n \in \mathbb{N}_0$. To decide if a language (over non-trivial alphabet) given by LTL formula is n-stutter-invariant is PSPACE-hard problem.*

Three previous theorems (and their proofs) provide an algorithm for minimising the number of nested $\mathcal{X}$ operators in given LTL formula.

In following we introduce the new modification of stuttering principle called *syllable stuttering*. As the name indicates, instead of removing repeating letters we remove repeating syllables.

Definition 3. *Let $w \in \Sigma^*$. A syllable $w(i,j)$ is m-*redundant *iff the syllable $w(i+j, m \cdot j)$ equals to $w(i,j)^m$, i.e. the syllable $w(i,j)$ is repeated at least m-times (excluding the original occurrence).*

A language L is *m-syllable-stutter-invariant* iff for every pair $v, w \in \Sigma^*$ such that v equals to w without one m-redundant syllable it holds that $v \in L \Longleftrightarrow w \in L$.

Theorem 4. *Let $m \geq 1$. Every LTL($\mathcal{U}^m, \mathcal{X}^0$) property is m-syllable-stutter-invariant.*

The reverse of the theorem does not hold.

The proofs and some interesting corollaries as well as the common generalisation of k-stuttering and syllable stuttering can be found in [KS02].

The standard stuttering is one of the underlying concepts of partial order reduction methods. We hope that our generalisations offer solid background for some new reduction methods.

References

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[KS02] A. Kučera and J. Strejček. The stuttering principle revisited. 2002. Submitted.

[Lam83] L. Lamport. What good is temporal logic? In R. E. A. Mason, editor, *Proceedings of the IFIP Congress on Information Processing*, pages 657–667, Amsterdam, 1983. North-Holland.

[May97] R. Mayr. Process rewrite systems. *Electronic Notes in Theoretical Computer Science*, 7, 1997.

[Pnu81] A. Pnueli. A temporal logic of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.

[PW97] D. Peled and T. Wilke. Stutter-invariant temporal properties are expressible without the next-time operator. *Information Processing Letters*, 63(5):243–246, 1997.

[PWW98] D. Peled, T. Wilke, and P. Wolper. An algorithmic approach for checking closure properties of temporal logic specifications and omega-regular languages. *Theoretical Computer Science*, 195:183–203, 1998.

[Sar89] V. A. Saraswat. *Concurrent Constraint Programming Languages*. PhD thesis, Computer Science Department, Carnegie Mellon University, 1989.

[Str02] J. Strejček. Rewrite systems with constraints. In L. Aceto and P. Panangaden, editors, *EXPRESS '01: 8th International Workshop on Expressiveness in Concurrency*, volume 52 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2002.

[vG90] R. J. van Glabbeek. The linear time-branching time spectrum. In J. C. M. Baeten and J. W. Klop, editors, *CONCUR '90: 1st International Conference on Concurrency Theory*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer-Verlag, 1990.