Well-Structured Transition Systems Everywhere !

A. Finkel and Ph. Schnoebelen

Lab. Specification and Verification, ENS de Cachan & CNRS UMR 8643, 61 av. Pdt Wilson, 94235 Cachan Cedex, FRANCE

Abstract

Well-structured transition systems (WSTS's) are a general class of infinite state systems for which decidability results rely on the existence of a well-quasi-ordering between states that is compatible with the transitions.

In this article, we provide an extensive treatment of the WSTS idea and show several new results. Our improved definitions allow many examples of classical systems to be seen as instances of WSTS's.

Key words: infinite systems; verification; well-quasi-ordering

1 Introduction

Verification of infinite-state systems. Formal verification of programs and systems is a very active field for both theoretical research and practical developments, especially since impressive advances in formal verification technology proved feasible in several realistic applications from the industrial world. The highly successful model-checking approach for finite systems [16] suggested that a working verification technology could well be developed for systems with an infinite state space.

This explains the considerable amount of work that has been devoted in recent years to this "verification of infinite state systems" field, with a surprising wealth of positive results [50,26]. The field now has its own conference.

Well-structured transition systems. A very interesting development in this field is the introduction of *well-structured transition systems* (WSTS's). These are transition systems where the existence of a well-quasi-ordering over the infinite set of states ensures the termination of several algorithmic methods. WSTS's are an abstract generalization of several specific structures and they allow general decidability results that can be applied to Petri nets, lossy channel systems, and many more. (Of course, WSTS's are not intended as a general explanation of all the decidability results one can find for specific models.)

Finkel [29,30,32] was the first to propose a definition of WSTS (actually several variant definitions). His insights came from the study of Petri nets where several decidability results rely on a monotonicity property (transitions firable from marking M are firable from any larger marking) and Dickson's lemma (inclusion between markings of a net is a well-ordering). He mainly investigated the decidability of termination, boundedness and coverability-set problems. He applied the idea to several classes of fifo nets and of CFSM's (see Section 9).

Independently, Abdulla *et al.* [1,2] later proposed another definition. Their insights came from their study of lossy-channel systems and other families of analyzable infinite-state systems (e.g. integral relational automata [20]). They mainly investigated covering, inevitability and simulation problems. They applied the idea to timed networks [6] and lossy systems.

Later, Kushnarenko and Schnoebelen [43] introduced WSTS's with *downward* compatibility, motivated by some analysis problems raised by Recursive-Parallel Programs.

Some criticisms. Though the two earlier lines of work had a great unifying power, they still suffered from some defects (these defects are also present in [43]).

- Both Finkel and Abdulla *et al.* proposed definitions aiming at a specific algorithm or two, hence their WSTS concept is burdened with *unnecessary structure*. Finkel was interested in coverability-sets, so that his definition included e.g. complex continuity requirements. Abdulla *et al.* were interested in simulation with a finite state system, so that their definition included e.g. labels on transitions.
- More generally, both definitions are *conservative* for no good reason we can think of. As a consequence, both proposals end up surprisingly short of simple examples of their "general" concept.
- Both definitions mix up *structural* and *effectiveness* issues while in reality the effectiveness requirements are quite variable, depending on which algorithm one is talking about. As a result their definition becomes more complex and restrictive when more decision methods for WSTS's are found.

- Several proofs in these earlier papers are quite messy or tedious. In part this is due to the unnecessarily complex definitions and to a lack of classifying work. As a consequence some potentially enlightening connections are hard to notice.
- These early works do not tackle the fundamental question of "how / when can one turn a given (family of) transition system into WSTS ?", i.e. how can one find a compatible well-ordering ?

Our contribution. This article is both a *survey* of earlier works on the WSTS idea, and a presentation of our proposal for *a new conceptual framework*, together with several new results or new extensions of earlier results.

More precisely,

- We propose and investigate a cleaner, more general definition of WSTS, generalizing the early insights of Finkel, Abdulla *et al.*, Kushnarenko and Schnoebelen.
- We separate structural and effectiveness issues and only add effectiveness hypothesis when and where they are needed for a decision method.
- We classify the decision methods into two main families: set-saturation and tree-saturation methods.
- We give five main decidability results. Except for Theorem 5.5, they never appeared in such a general framework. Both Theorem 3.6, made possible by our Definition 3.2, and Theorem 4.8, made possible by the key notion of stuttering compatibility, have much more applications than their ancestors.
- We give a quite large collection (summarized in Fig. 9, page 31) of system models that can be fitted into the WSTS framework. These examples come from various fields of computer science. Many have not been noticed earlier. Several of them use original and innovative well-orderings. Roughly half of them are WSTS's only in our new, generalized, definition.
- Finally, we ask how and when a given transition system can be given a well-structure. We offer surprisingly general answers (e.g. our Ubiquity Theorem) for the more liberal definition.

Outline of the article. This article is divided into three parts. In Part I we introduce the fundamentals concepts underlying WSTS's (section 2) and we describe the two main families of decision methods for WSTS's: set-saturation methods (section 3) and tree-saturation methods (section 4). We conclude this part with downward-WSTS's (section 5).

Part II is devoted to examples of WSTS's. We successively visit Petri nets and their extensions (section 6), string rewrite systems (section 7), process algebra (section 8), communicating automata (section 9) and a few less classical operational models of computation (section 10). All these (families of) models are found to be well-structured in natural ways. This is a strong point in favor of our claim that we propose a more interesting definition of WSTS's.

Finally, Part III is concerned with the passage from TS's to WSTS's. We investigate when and how does there exist well-quasi-orderings that can provide a well-structure to a given transition system.

Part I: Fundamentals of WSTS's

Part I presents the technical core of the WSTS idea. As a rule, examples and illustrations have been postponed until Part II.

2 Basic notions

2.1 Well-quasi-orderings

Recall that a quasi-ordering (a *qo*) is any reflexive and transitive relation $\leq$. We let $x < y$ denote $x \leq y \not\leq x$. A partial ordering (a *po*, an "ordering", ...) is an antisymmetric qo. Any qo induces an equivalence relation ($x \equiv y$ iff $x \leq y \leq x$) and gives rise to a po between the equivalence classes.

We now need a few results from the theory of well-orderings (see also e.g. [44,40]).
Definition 2.1. *A* well-quasi-ordering *(a* wqo*) is any quasi-ordering $\leq$ (over some set X) such that, for any infinite sequence $x_0, x_1, x_2, \ldots$ in X, there exists indexes $i < j$ with $x_i \leq x_j$.*

Hence a wqo is well-founded, i.e. it admits no infinite strictly decreasing sequence $x_0 > x_1 > x_2 > \cdots$
Lemma 2.2. (Erdös & Rado) *Assume $\leq$ is a wqo. Then any infinite sequence contains an infinite increasing subsequence: $x_{i_0} \leq x_{i_1} \leq x_{i_2} \ldots$ (with $i_0 < i_1 < i_2 \ldots$).*

Proof. Consider an infinite sequence and the set $M = \{i \in \mathbb{N} \mid \forall j > i, x_i \not\leq x_j\}$. M cannot be infinite, otherwise it would lead to an infinite subsequence contradicting the wqo hypothesis. Thus M is bounded and any x_i with i beyond M can start an infinite increasing subsequence. $\square$

Given $\leq$ a quasi-ordering, an *upward-closed set* is any set $I \subseteq X$ such that $y \geq x$ and $x \in I$ entail $y \in I$. To any $x \in X$ we associate $\uparrow x \stackrel{\text{def}}{=} \{y \mid y \geq x\}$. It is upward-closed. A *basis* of an upward-closed I is a set I^b such that $I = \bigcup_{x \in I^b} \uparrow x$. Higman investigated ordered sets with the *finite basis property.*

Lemma 2.3. **[40]** *If $\leq$ is a wqo, then any upward-closed I has a finite basis.*

Proof. The set of minimal elements of I is a basis because $\leq$ is well-founded. It only contains a finite number of non-equivalent elements otherwise they would make an infinite sequence contradicting the wqo assumption. $\square$

Lemma 2.4. *If $\leq$ is a wqo, any infinite increasing sequence $I_0 \subseteq I_1 \subseteq I_2 \subseteq \cdots$ of upward-closed sets eventually stabilizes, i.e. there is a $k \in \mathbb{N}$ such that $I_k = I_{k+1} = I_{k+2} = \ldots$*

Proof. Assume we have a counter-example. We extract an infinite subsequence where inclusion is strict: $I_{n_0} \subsetneq I_{n_1} \subsetneq I_{n_2} \subsetneq \cdots$. Now, for any $i > 0$, we can pick some $x_i \in I_{n_i} \setminus I_{n_{i-1}}$. The well-quasi-ordering hypothesis means that the infinite sequence of x_i's contains an increasing pair $x_i \leq x_j$ for some $i < j$. Because x_i belongs to an upward-closed set I_{n_i} we have $x_j \in I_{n_i}$, contradicting $x_j \notin I_{n_{j-1}}$. $\square$

2.2 Transition systems

A *transition system* (TS) is a structure $\mathcal{S} = \langle S, \rightarrow, \ldots \rangle$ where $S = \{s, t, \ldots\}$ is a set of *states*, and $\rightarrow \subseteq S \times S$ is any set of *transitions*. TS's may have additional structure like initial states, labels for transitions, durations, causal independence relations, etc., but in this paper we are only interested in the state-part of the behaviors.

We write $Succ(s)$ (resp. $Pred(s)$) for the set $\{s' \in S \mid s \rightarrow s'\}$ of immediate *successors* of s (resp. $\{s' \in S \mid s' \rightarrow s\}$ the immediate *predecessors*). A state with no successor is a *terminal state*. A *computation* is a maximal sequence $s_0 \rightarrow s_1 \rightarrow s_2 \cdots$ of transitions.

We write $\xrightarrow{n}$ (resp. $\xrightarrow{+}$, $\xrightarrow{=}$, $\xrightarrow{*}$) for the n-step iteration of the transition relation $\rightarrow$ (resp. for its transitive closure, for its reflexive closure, for its reflexive and transitive closure). Hence $\xrightarrow{1}$ is $\rightarrow$. We use similar notation for $Succ$ and $Pred$, so that for $\alpha \in \{+, =, *, 0, 1, 2, \ldots\}$, $Succ^{\alpha}(s)$ is $\{s' \mid s \xrightarrow{\alpha} s'\}$.

$\mathcal{S}$ is *finitely branching* if all $Succ(s)$ are finite. We restrict our attention to finitely branching TS's.

2.3 Well-structured transition systems

Definition 2.5. *A* well-structured transition system *(WSTS) is a TS* $\mathcal{S} = \langle S, \rightarrow, \leq \rangle$ *equipped with a qo* $\leq \subseteq S \times S$ *between states such that the two following conditions hold:*

(1) well-quasi-ordering: *$\leq$ is a wqo, and*
(2) compatibility: *$\leq$ is (upward) compatible with $\rightarrow$, i.e. for all $s_1 \leq t_1$ and transition $s_1 \rightarrow s_2$, there exists a sequence $t_1 \xrightarrow{*} t_2$ such that $s_2 \leq t_2$.*

Thus compatibility states that $\leq$ is a weak simulation relation *à la* R. Milner.

See Figure 1 for a diagrammatic presentation of compatibility where we quantify universally over solid lines and existentially over dashed lines. Several

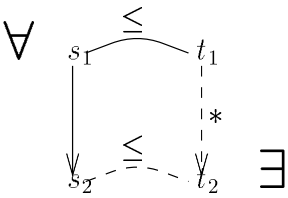


Fig. 1. (Upward) compatibility

families of formal models of processes give rise to WSTS's in a natural way, e.g. Petri nets when inclusion between markings is used as the well-ordering. In Part II, we shall see many more examples.

3 Set-saturation methods

We speak of *set-saturation methods* when we have methods whose termination relies on Lemma 2.4. In this section, we illustrate the idea with the backward reachability method for the covering problem, generalizing a result from [1].

Other examples of the set-saturation family are the algorithm for simulation by a finite state system (from [1]), and the algorithm for the sub-covering problem (from [43]).

The algorithm studied in [5] is essentially the algorithm from [1] for the covering problem, described as a symbolic method for reachability. Revesz's procedure [53] is another set-saturation method based on wqo's, but we don't see how to state it with a WSTS point of view.

Assume $\mathcal{S} = \langle S, \rightarrow, \leq \rangle$ is a WSTS and $I \subseteq S$ is a set of states. Backward reachability analysis involves computing $Pred^*(I)$ as the limit of the sequence $I_0 \subseteq I_1 \subseteq \ldots$ where $I_0 \stackrel{\text{def}}{=} I$ and $I_{n+1} \stackrel{\text{def}}{=} I_n \cup Pred(I_n)$. The problem with such a general approach is that termination is not guaranteed. For WSTS's, this can be solved when I is upward-closed:

Proposition 3.1. *If $I \subseteq S$ is an upward-closed set of states, then $Pred^*(I)$ is upward-closed.*

Proof. Assume $s \in Pred^*(I)$. Then $s \xrightarrow{*} t$ for some $t \in I$. If now $s' \geq s$ then upward-compatibility entails that $s' \xrightarrow{*} t'$ for some $t' \geq t$. Then $t' \in I$ and $s' \in Pred^*(I)$. □

$Pred^*(I)$ can be computed if we make a few decidability assumptions:

Definition 3.2. *A WSTS has* effective pred-basis *if there exists an algorithm accepting any state $s \in S$ and returning $pb(s)$, a finite basis of $\uparrow Pred(\uparrow s)$.*

Note that Definition 3.2 is distinct from the requirement for a basis of $Pred(\uparrow s)$ used in [1]. Our definition is necessary for the generalized Theorem 3.6 we aims at.

Now assume that $\mathcal{S}$ is a WSTS with effective pred-basis. Pick I^b a finite basis of I and define a sequence $K_0, K_1, \ldots$ of sets with $K_0 \stackrel{\text{def}}{=} I^b$, and $K_{n+1} \stackrel{\text{def}}{=} K_n \cup pb(K_n)$. Let m be the first index such that $\uparrow K_m = \uparrow K_{m+1}$. Such an m must exist by Lemma 2.4.

Lemma 3.3. $\uparrow K_m = \uparrow \bigcup_{i \in \mathbb{N}} K_i$.

Proof. This is not a consequence of Lemma 2.4 but rather of

$$\uparrow Y = \uparrow Y' \text{ implies } \uparrow pb(Y) = \uparrow pb(Y').$$

which relies on the definition of pb and the distributivity property of $Pred$ and $\uparrow$w.r.t. union. □

Lemma 3.4. $\uparrow \bigcup K_i = Pred^*(I)$.

Proof. Use induction over n and show that

$$K_n \subseteq \uparrow K_n \subseteq Pred^*(I) \ (= \uparrow Pred^*(I))$$

On the other hand, the definition of pb entails $\uparrow Pred^n(I) \subseteq \uparrow K_n$, so that

$$Pred^*(I) \subseteq \bigcup_{i \in \mathbb{N}} \uparrow K_i \subseteq \uparrow \bigcup_{i \in \mathbb{N}} K_i \subseteq \uparrow Pred^*(I).$$

□

Proposition 3.5. *If $\mathcal{S}$ is a WSTS with (1) effective pred-basis and (2) decidable $\leq$, then it is possible to compute a finite basis of $Pred^*(I)$ for any upward-closed I given via a finite basis.*

Proof. The sequence $K_0, K_1, \ldots$ can be constructed effectively (each K_n is finite and *pb* is effective). The index m can be computed because the computability of $\leq$ entails the decidability of "$\uparrow K = \uparrow K'$?" for finite sets K and K'. Finally, K_m is a computable finite basis of $Pred^*(I)$. □

The *covering problem* is to decide, given two states s and t, whether starting from s it is possible to cover t, i.e. to reach a state $t' \geq t$.

The covering problem is often called the "control-state reachability problem" when S has the form $Q \times D$ (for Q a finite set of so-called "control states" and D an infinite set of data values) and $(q, d) \leq (q', d')$ entails $q = q'$ and $d \leq d'$.
Theorem 3.6. *The covering problem is decidable for WSTS's with (1) effective pred-basis and (2) decidable $\leq$.*

Proof. Thanks to Proposition 3.5, one can compute K, a finite basis of $Pred^*(\uparrow t)$. It is possible to cover t starting from s iff $s \in \uparrow K$. By decidability of $\leq$, it is possible to check whether $s \in \uparrow K$. □

Variants of this problem can be decided in the same way. E.g. deciding whether t can be covered from all states in a given upward-closed I. Or from all states in a downward-closed $D = S \setminus I$ (this requires WSTS's with intersection-effectiveness, i.e. there is an algorithm computing $inter(s, s')$, a finite basis of $\uparrow s \cap \uparrow s'$).

4 Tree-saturation methods

We speak of *tree-saturation methods* when we have methods representing (in some way) all possible computations inside a finite tree-like structure. In this section, we illustrate the idea with the Finite Reachability Tree and its several applications to termination, inevitability, and boundedness problems.

Other examples of the tree-saturation idea is the algorithm for simulation of a finite state systems (from [1]), and the algorithm for coverability-sets (from [29]).

We assume $\mathcal{S} = \langle S, \rightarrow, \leq \rangle$ is a WSTS.

4.1 Finite reachability tree

Definition 4.1. **[32]** *For any $s \in S$, $FRT(s)$, the* Finite Reachability Tree *from s, is a directed unordered tree where nodes are labeled by states of S. Nodes are either* dead *or* live*. The root node is a live node n_0, labeled by s (written $n_0 : s$). A dead node has no child node. A live node $n : t$ has one children $n' : t'$ for each successor $t' \in Succ(t)$. If along the path from the root $n_0 : s$ to some node $n' : t'$ there exists a node $n : t$ ($n \neq n'$) such that $t \leq t'$, we say that n* subsumes *n', and then n' is a dead node. Otherwise, n' is live.*

Thus leaf nodes in $FRT(s)$ are exactly (1) the nodes labeled with terminal states, and (2) the subsumed nodes. See Part II for examples.
Lemma 4.2. *$FRT(s)$ is finite (hence the name).*

Proof. The wqo property ensures that all paths in $FRT(s)$ are finite because an infinite path would have to contain a subsumed node. Finite branching and König's lemma conclude the proof. □

With finiteness, we observe that $FRT(s)$ is effectively computable if $\mathcal{S}$ has (1) a decidable $\leq$, and (2) effective $Succ$ (i.e. the $Succ$ mapping is computable).

The *construction* of $FRT(s)$ does not require compatibility between $\leq$ and $\rightarrow$. However, when we have compatibility, $FRT(s)$ contains, in a finite form, sufficient information to answer several questions about computations paths starting from s. For a start, we have
Lemma 4.3. *Any computation starting from s has a finite prefix labeling a maximal path in $FRT(s)$.*

Proof. Obvious. □

Further results need slightly restricted notions of compatibility: transitive compatibility and stuttering compatibility.

4.2 Transitive and stuttering compatibility

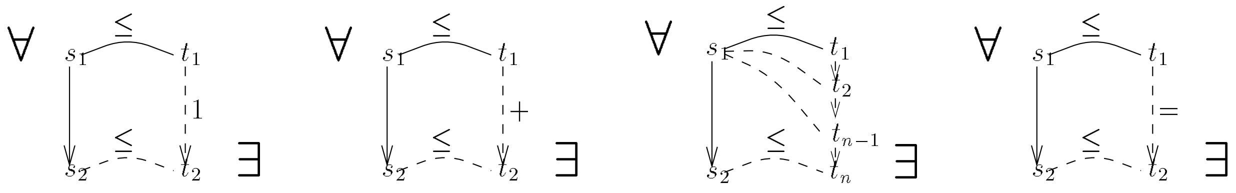
Definition 4.4. *A WSTS $\mathcal{S}$ has* strong compatibility *if for all $s_1 \leq t_1$ and transition $s_1 \rightarrow s_2$, there exists a transition $t_1 \rightarrow t_2$ with $s_2 \leq t_2$.*

$\mathcal{S}$ has transitive compatibility *if for all $s_1 \leq t_1$ and transition $s_1 \rightarrow s_2$, there exists a* non-empty *sequence $t_1 \rightarrow t_2 \rightarrow \cdots \rightarrow t_n$ with $s_2 \leq t_n$.*

$\mathcal{S}$ *has* stuttering compatibility *if for all* $s_1 \leq t_1$ *and transition* $s_1 \rightarrow s_2$*, there exists a* non-empty *sequence* $t_1 \rightarrow t_2 \rightarrow \cdots \rightarrow t_n$ *with* $s_2 \leq t_n$ *and* $s_1 \leq t_i$ *for all* $i < n$*.*

$\mathcal{S}$ *has* reflexive compatibility *if for all* $s_1 \leq t_1$ *and transition* $s_1 \rightarrow s_2$*, either* $s_2 \leq t_1$ *or there exists a transition* $t_1 \rightarrow t_2$ *with* $s_2 \leq t_2$*.*

See Figure 2 for a diagrammatic presentation of these refinements of compatibility.



(a) strong (b) transitive (c) stuttering (d) reflexive

Fig. 2. Transitive and stuttering compatibility

Strong compatibility (also called "1-1 compatibility") is inspired from classical strong simulation and the other forms of compatibility we use are more general than this.

Reflexive compatibility is strong compatibility for $\xrightarrow{=}$.

Transitive compatibility is slightly less general than the "reflexive-and-transitive" compatibility we used in Def. 2.5. Finkel's notion of "3-structured systems" [32] uses transitive compatibility in a framework where labels of transitions are taken into account.

Stuttering compatibility, introduced in [43], is less general than transitive compatibility. The name comes from "stuttering" [15] (also "branching" [36]) bisimulation. Both are more general than strong compatibility.

In practice, the strong, "1-1", notion used by Abdulla *et al.* is much more limited than appears at first sight. Clearly, their motivation was the decidability of simulation with a finite-state system. Unfortunately most examples do not have strong compatibility, so that for e.g. Lossy Channel Systems they have to modify the semantics of the model.

More generally, when $\mathcal{S} = \langle S, \rightarrow, \leq \rangle$ is a WSTS, then $\mathcal{S}^* \stackrel{\text{def}}{=} \langle S, \xrightarrow{*}, \leq \rangle$ has strong, "1-1", compatibility, but it is not necessarily a WSTS. $\mathcal{S}^*$ is in general not finitely branching. Worse, when effectiveness issues are taken into account, $\mathcal{S}^*$ needs not have effective *Succ* or pred-basis even when $\mathcal{S}$ has. Finally, the inevitability properties investigated in [1] do not translate from $\xrightarrow{*}$ (or $\xrightarrow{+}$) to $\rightarrow$.

4.3 Termination

Assume $\mathcal{S}$ is a WSTS with transitive compatibility.

Proposition 4.5. *$\mathcal{S}$ has a non-terminating computation starting from s iff $FRT(s)$ contains a subsumed node.*

Proof. ($\Rightarrow$): Consider a non-terminating computation. A finite prefix labels a path in $FRT(s)$ (Lemma 4.3). The last node of this path is a leaf node, not labeled with a terminal state, hence a subsumed node.

($\Leftarrow$): If $n_2 : t_2$ is the leaf node subsumed by $n_1 : t_1$, we have $s \xrightarrow{*} t_1 \xrightarrow{+} t_2$ with $t_1 \leq t_2$. Transitive compatibility allows to infer the existence of some $t_2 \xrightarrow{+} t_3$ with $t_2 \leq t_3$. Repeating this reasoning, we build an infinite computation starting from s. □

Hence we have

Theorem 4.6. *Termination is decidable for WSTS's with (1) transitive compatibility, (2) decidable $\leq$, and (3) effective Succ.*

4.4 Eventuality properties

Assume $\mathcal{S}$ is a WSTS with stuttering compatibility.

Proposition 4.7. [43] *Assume I is upward-closed. There exists a computation starting from s where all states are in I iff $FRT(s)$ has a maximal path where all nodes are labeled with states in I.*

Proof. ($\Rightarrow$): Use Lemma 4.3.

($\Leftarrow$): Assume that $n_0 : t_0, \ldots, n_k : t_k$ is a maximal path in $FRT(s)$ with all labels in I. If n_k is a live node, then $t_0 \rightarrow t_1 \rightarrow \cdots t_k$ is a computation and we are done. If n_k is a dead node, then we display an infinite computation ($s =$) $s_0 \rightarrow s_1 \rightarrow \cdots$ where all states are greater (w.r.t. $\leq$) than one of the t_i's, and thus belong to I.

We define the s_i's inductively, starting from $s_0 \stackrel{\text{def}}{=} s$ ($= t_0$). Assume we have already built $s_0, \ldots, s_n$. We have $s_n \geq t_i$ for some $i \leq k$. There are two cases:

- $i < k$: then $t_i \to t_{i+1}$. Because of stuttering compatibility, there exists a sequence $s_n \to \cdots \to s_m$ $(m > n)$ with $s_n, \ldots, s_{m-1} \geq t_i$ and $s_m \geq t_{i+1}$. We use them to lengthen our sequence up to s_m.
- $i = k$: then, because n_k is dead, $t_j \leq t_k$ for some $j < k$. Thus $t_j \leq s_n$, so that we are back to the previus case and can lengthen our sequence.

□

Now we can generalize a Theorem from [1].

The *control-state maintainability problem* is to decide, given an initial state s and a finite set $Q = \{t_1, \ldots, t_m\}$ of states, whether there exists a computation starting from s where all states cover one of the t_i's. The dual problem, called the *inevitability problem*, is to decide whether all computations starting from s eventually visit a state not covering one of the t_i's. Concrete examples abound. See Part II. E.g. for Petri nets, we can ask whether a given place will inevitably be emptied.

Theorem 4.8. *The control-state maintainability problem and the inevitability problem are decidable for WSTS's with (1) stuttering compatibility,(2) decidable $\leq$, and (3) effective Succ.*

Proof. Thanks to Proposition 4.7, the control-state maintainability problem reduces to checking whether $FRT(s)$ has a maximal path with all labels in $\uparrow Q$. □

4.5 Strict compatibility

Finkel [32] also considered WSTS's with *strict compatibility.* Strict compatibility is a stronger form of compatibility.

Definition 4.9. *A WSTS $\mathcal{S}$ has* strict compatibility *if for all $s_1 < t_1$ and transition $s_1 \to s_2$, there exists a sequence $t_1 \xrightarrow{*} t_2$ with $s_2 < t_2$.*

Note that strict compatibility already requires normal non-strict compatibility by assuming that $\mathcal{S}$ is a WSTS. When $\leq$ is a partial ordering, strict compatibility alone entails non-strict compatibility. We adopted the more general definition for situations where $\leq$ is a quasi-ordering.

Strict compatibility means that from strictly larger states it is possible to reach strictly larger states. See Figure 3 for a diagrammatic presentation. Of course the concept can be combined with transitive, stuttering, ... compatibility.

When we have strict compatibility, the Finite Reachability Tree contains infor-

$$\forall\ \begin{tikzcd} s_1 \arrow[r, dash, bend left, "<"] \arrow[d] & t_1 \arrow[d, dashed, "*"] \\ s_2 \arrow[r, dash, dashed, bend right, "<"] & t_2 \end{tikzcd}\ \exists \qquad \text{and} \qquad \forall\ \begin{tikzcd} s_1 \arrow[r, dash, bend left, "\leq"] \arrow[d] & t_1 \arrow[d, dashed, "*"] \\ s_2 \arrow[r, dash, dashed, bend right, "\leq"] & t_2 \end{tikzcd}\ \exists$$

Fig. 3. Strict compatibility

mation pertaining to the finiteness of the number of reachable states. Assume that $\mathcal{S} = \langle S, \rightarrow, \leq \rangle$ is a WSTS with strict transitive compatibility. Further, assume that $\leq$ is a partial ordering (not a quasi-ordering).

Proposition 4.10. *For any $s \in S$, $Succ^*(s)$ is infinite iff $FRT(s)$ contains a leaf node $n : t$ subsumed by an ancestor $n' : t'$ with $t' < t$.*

Proof. ($\Leftarrow$:) If $n : t$ is subsumed by $n' : t'$ then $\mathcal{S}$ admits an infinite computation $s \xrightarrow{*} t_0 \xrightarrow{+} t_1 \xrightarrow{+} t_2 \ldots$ with $t_i < t_{i+1}$ for all $i = 0, 1, 2, \ldots$. This computation is easily built inductively by picking $t_0 = t$ and $t_1 = t'$. Then, strict transitive compatibility allows us to deduce, from $t_{i-1} \xrightarrow{+} t_i$ and $t_{i-1} < t_i$ the existence of a $t_i \xrightarrow{+} t_{i+1}$ with $t_i < t_{i+1}$. Then the t_i's are all distinct and $Succ^*(s)$ is infinite.

($\Rightarrow$:) Assume $Succ^*(s)$ is infinite. We first show that there exists a computation starting from s without any loop, i.e. *where all states are distinct.* For this, we cannot simply remove loops from an infinite computation as this may well result into a finite prefix only. So we rather consider the (finitely branching) tree of all prefixes of computations. Now prune this tree by removing all prefixes with a loop. Because any reachable state can be reached without a loop, the pruned tree still contains an infinite number of prefixes. Now König's lemma gives us an infinite computation with no loop.

We now apply Lemma 4.3 to this computation: this provides a node $n : t$ subsumed by $n' : t'$ with $t \neq t'$. Hence $t' < t$ because $\leq$ is a partial ordering. $\square$

Now we can restate a Theorem from [32].

The *boundedness problem* is to decide, given a TS $\mathcal{S}$ and some state $s \in S$, whether $Succ^*(s)$, the "set of reachable states", is finite.

Theorem 4.11. *The boundedness problem is decidable for WSTS's with (1) strict transitive compatibility, (2) a decidable $\leq$ which is a partial ordering, and (3) computable $Succ$.*

Proof. We can apply Prop. 4.10 so that it is enough to build $FRT(s)$ and inspect it for a subsumed node with strict subsumption. This can be done when $Succ$ and $\leq$ are effective. $\square$

5 Downward-WSTS's

There also exists a notion of downward-WSTS, first introduced in [43] for the RPPS model. We generalize it to

Definition 5.1. *A* downward-WSTS *is a TS $\mathcal{S} = \langle S, \rightarrow, \leq \rangle$ equipped with a qo $\leq \subseteq S \times S$ between states such that the two following conditions hold:*

(1) ***well-quasi-ordering:*** *$\leq$ is a wqo, and*
(2) ***downward-compatibility:*** *$\leq$ is downward-compatible with $\rightarrow$, i.e. for all $s_1 \geq t_1$ and transition $s_1 \rightarrow s_2$, there exists a sequence $t_1 \xrightarrow{*} t_2$ such that $s_2 \geq t_2$.*

See Figure 4 for a diagrammatic presentation of downward-compatibility. Downward-WSTS's have been less investigated, partly because only a few

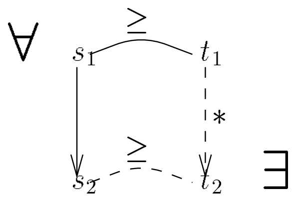


Fig. 4. Downward compatibility

recent models give rise naturally to WSTS's with downward-compatibility. In Part II, we shall see several examples.

Assume $\mathcal{S} = \langle S, \rightarrow, \leq \rangle$ is a downward-WSTS with reflexive compatibility and K, K' are two sets of states.

Lemma 5.2. *$\uparrow K \subseteq \uparrow K'$ implies $\uparrow Succ^{=}(K) \subseteq \uparrow Succ^{=}(K')$.*

Proof. (Recall that $Succ^{=}(K)$ is $K \cup Succ(K)$.) Assume $s \in \uparrow Succ^{=}(K)$. Then there exist $s_1 \in K$ and t_1 with $s_1 \xrightarrow{=} t_1 \leq s$. Because $\uparrow K \subseteq \uparrow K'$, there is a $s_2 \in K'$ with $s_2 \leq s_1$. Because $\mathcal{S}$ is a downward-WSTS with reflexive compatibility, there exists a $s_2 \xrightarrow{=} t_2$ with $t_2 \leq t_1$. Hence $t_2 \leq s$. Now $t_2 \in Succ^{=}(K')$ entails $s \in \uparrow Succ^{=}(K')$. $\square$

Now assume $s \in S$ and define a sequence $K_0, K_1, \ldots$ of sets with $K_0 \stackrel{\text{def}}{=} \{s\}$, and $K_{n+1} \stackrel{\text{def}}{=} K_n \cup Succ(K_n)$. Let m be the first index such that $\uparrow K_m = \uparrow K_{m+1}$. Such an m must exist by Lemma 2.4.

Lemma 5.3. *$\uparrow K_m = \uparrow \bigcup_{i \in \mathbb{N}} K_i = \uparrow Succ^*(s)$.*

Proof. The first equality is a direct consequence of Lemma 5.2, the second follows from the definition of the K_i's. $\square$

Proposition 5.4. *If S is a downward-WSTS with (1) reflexive compatibility, (2) effective Succ, and (3) decidable $\leq$, then it is possible to compute a finite basis of $\uparrow Succ^*(s)$ for any $s \in S$.*

Proof. We proceed as with Prop. 3.5, The sequence $K_0, K_1, \ldots$ can be constructed effectively (each K_i is finite and $Succ$ is effective). The index m can be computed by computability of $\leq$. Finally, K_m is a computable finite basis of $\uparrow Succ^*(s)$. $\square$

The *sub-covering problem* is to decide, given two states s and t, whether starting from s it is possible to be covered by t, i.e. to reach a state $t' \leq t$.

Theorem 5.5. *The sub-covering problem is decidable for downward-WSTS's with (1) reflexive compatibility, (2) effective Succ and (3) decidable $\leq$.*

Proof. Thanks to Proposition 5.4, one can compute K, a finite basis of $\uparrow Succ^*(s)$. It is possible to be covered by t starting from s iff $t \in \uparrow K$. By decidability of $\leq$, it is possible to check whether $t \in \uparrow K$. $\square$

Part II: The ubiquity of WSTS's

In this second part we review several fundamental computational models and discover instances of WSTS's in these frameworks.

In general, all the well-structured systems we mention enjoy the effectiveness requirements assumed e.g. by Theorems 3.6, 4.6, 4.8 and 5.5. We will not state this explicitly every time. In fact, we mainly state the few exceptions, all of them occurring when we use a non-trivial $\leq$ for which decidability is lost.

6 The well-structure of Petri nets

Petri nets are a well-known model of concurrent systems. See [51,52] for a general presentation.

Formally a net $N = \langle P_N, T_N, F_N \rangle$ has a finite set P_N of *places*, a finite set T_N of *transitions* (with $P_N \cap T_N = \emptyset$) and a flow matrix $F_N : (P_N \times T_N \cup T_N \times P_N) \to \mathbb{N}$. Figure 5 contains an example net. The configurations of a net N are markings, which can be seen as P_N-indexed vectors of non-negative integers, or as multisets of places. The marking M_0 depicted in Figure 5 is denoted $\{p_1, p_1, p_2, p_3\}$ or $p_1^2 p_2 p_3$.

The simplest ordering between markings is inclusion: $M \subseteq M'$ when $M(p) \leq M'(p)$ for every place. That it is a wqo is known as Dickson's lemma [24].

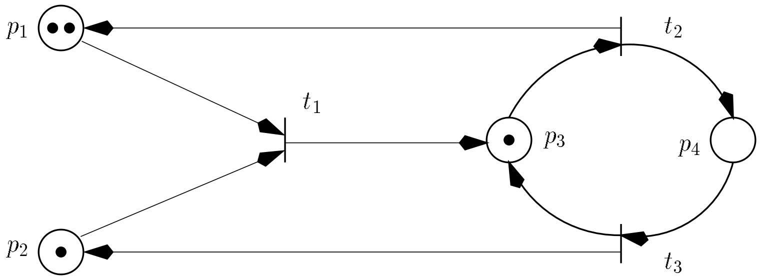


Fig. 5. A Petri net

Petri nets with inhibitory arcs extend the basic model with special "inhibitory" arcs (also called "zero-test" arcs) that forbid (inhibit) the firing of a given transition when a given place is not empty.

Petri nets with transfer arcs [38,22,25] extend the basic model with special "transfer" arcs. Here transitions fire as usual but their effect is richer: the transfer arcs say whether the full content of some place must be transfered (added) to some other place.

Petri nets with reset arcs [38,22,25] extend the basic model with special "reset arcs" telling how the firing of some transitions resets (empties) some places.

Self-modifying nets [54] are Petri nets where the weight on arcs is not a constant anymore. Rather it is an expression evaluating into a linear combination (with non-negative coefficients) of the current contents of the places. *Post self-modifying nets* are self-modifying nets where the self-modifying extension is only allowed on "post" arcs (arcs from transitions to places).

In all these extensions, reachability becomes undecidable [8,54]. However

Theorem 6.1. *Using the inclusion ordering,*

1. *Petri nets are WSTS's with strong strict compatibility,*
2. *Petri nets with transfer arcs are WSTS's with strong strict compatibility,*
3. *Petri nets with reset arcs are WSTS's with strong compatibility,*
4. *post self-modifying nets are WSTS's with strong strict compatibility.*

Proof. Obvious. □

So that e.g. covering is decidable for them ! Covering is a classical problem in the Petri net field. It was known to be decidable since [42]. As noted in [1], a byproduct of Theorem 3.6 is a backward-based algorithm for the covering problem in Petri nets. This also applies to the three extensions (transfer arcs, reset arcs, post self-modifying nets) we mentioned. Essentially the same algorithm is used in [9] for the covering problem in Petri nets with reset arcs. It also applies to all generalized nets where transitions invoke "good increasing recursive functions" [35].

As far as we know, all implemented algorithms for this problem use Karp and Miller's coverability tree, or the coverability graph, or some such quite complex forward-based method. These methods cannot be generalized to all extensions (e.g. it fails for reset arcs [25]).

Other orderings can turn Petri nets into WSTS's. Assume $N = \langle P, T, F, M_0 \rangle$ is a marked net (a net with a given initial marking M_0). Say a place $p \in P$ is *unbounded* if there are reachable (from M_0) markings with an arbitrarily large number of tokens in p. Separate bounded and unbounded places and write $P = P_b \cup P_{nb}$. Usually, one sees places in P_b as "control places" and places in P_{nb} as "data places" or "counter places".

Now define the ordering

$$M \ll M' \overset{\text{def}}{\Leftrightarrow} \begin{cases} M(p) = M'(p) & \text{for all } p \in P_b, \\ M(p) \leq M'(p) & \text{for all } p \in P_{nb}. \end{cases}$$

This is a well-ordering *over the set of reachable markings*. So that, if we associate to a marked net $\langle N, M_0 \rangle$ a transition system $\mathcal{S}_{N,M_0}$ containing only the reachable markings we get

Proposition 6.2. *$\langle \mathcal{S}_{N,M_0}, \ll \rangle$ is a WSTS.*

This works for all the extensions like post self-modifying nets, etc. we mentioned earlier. However, the well-ordering is only decidable when we can tell effectively which places of the net are bounded. This can be done for Petri nets and for post self-modifying nets. (For nets with reset arcs and nets with transfer arcs, telling whether a given place p is bounded is not decidable [25]).

The *partial bounded reachability problem* is, given a marked net N, M_0 and a marking M, to tell whether from M_0 it is possible to reach an M' with $M'(p) = M(p)$ for all $p \in P_b$.

Theorem 6.3. *The partial bounded reachability problem is decidable for Petri nets and post self-modifying nets.*

Proof. The partial bounded reachability problem is an instance of the covering problem for $\langle \mathcal{S}_{N,M_0}, \ll \rangle$. □

The most surprising aspect of this result is the relative simplicity of the algorithmic notions that are involved.

Inhibitory arcs can be handled if we have no synchronization. BPP nets, short for *Basic Parallel Processes*, are nets where the *pre-set* of transitions is reduced to a single place [21,50].

Theorem 6.4. *With the inclusion ordering, BPP's with inhibitory arcs are downward-WSTS's with reflexive compatibility.*

Proof. Assume $M_1 \supseteq M'_1$ and $M_1 \xrightarrow{t} M_2$. If t is firable in M'_1 then $M'_1 \xrightarrow{t} M'_2$ and $M_2 \supseteq M'_2$. If t is not firable in M'_1 then this cannot be caused by an inhibitory arc because $M_1 \supseteq M'_1$. Hence M'_1 does not contain $pre(t)$. But $pre(t)$ is some place p, so that $M_1 - \{p\} \supseteq M'_1$. Now M_2 is $M_1 - \{p\} + post(t)$ and $M_2 \supseteq M'_1$. $\square$

7 The well-structure of string rewrite systems

Context-Free Grammars (CFG) are a special kind of string rewrite systems. Formally a CFG is a tuple $G = \langle N_G, T_G, R_G \rangle$ where N_G (the non-terminal symbols) and T_G (the terminal symbols) are disjoint alphabets and where, writing Σ_G for $N_G \cup T_G$, $R_G \subseteq N_G \times \Sigma_G^*$ is a finite set of production rules of the form $Z \rightarrow w$. See [10] for details.

As an example, we consider $G = \langle \{S, X, Y\}, \{a, b\}, R_G \rangle$ where R_G, the set of rules, is given as

$$\begin{aligned} S &\rightarrow YX \mid b \\ X &\rightarrow S \\ Y &\rightarrow a \end{aligned}$$

The rules in R_G induce a notion of rewrite step: if $Z \rightarrow w$ is in R_G then $uZv \rightarrow_G uwv$ for any u, v. Usually, we are interested in derivation sequences that start from a given, so-called axiom, non-terminal, and end up with a word in T_G^*.

In our previous example, a possible derivation for the terminal word ab is

$$S \rightarrow_G YX \rightarrow_G aX \rightarrow_G aS \rightarrow_G ab \tag{1}$$

If instead of focusing on the language generated by G, we emphasize the rewrite steps, then G gives rise to a transition system $\mathcal{S}_G$ where states are words in Σ_G^* and (1) now is a computation of $\mathcal{S}_G$, starting from S.

Several natural orderings can be defined between words:

embedding: a word u embeds into a word v (also u is a subword of v), written $u \preccurlyeq v$, iff u can be obtained by erasing letters from v.
left-factor: a word u is a left-factor (also, a prefix) of a word v, written $u \leq_{\mathrm{lf}} v$, iff v is some uw.
Parikh: $u \leq_{\mathrm{P}} v$ iff a permutation of u is a subword of v.
subset: $u \subseteq v$ iff any symbol in u is in v.

$\preccurlyeq$, $\leq_{\mathrm{lf}}$ are po's while $\leq_{\mathrm{P}}$, and $\subseteq$ are qo's. Assuming a finite alphabet, $\preccurlyeq$ is a wqo (Higman's Lemma) while $\leq_{\mathrm{lf}}$ is not. Being larger than $\preccurlyeq$, both $\leq_{\mathrm{P}}$ and $\subseteq$ are wqo's.

Theorem 7.1. *For any context-free grammar G,*
1. $\langle \mathcal{S}_G, \preccurlyeq \rangle$ is a WSTS with strong strict compatibility,
2. $\langle \mathcal{S}_G, \leq_{\mathrm{P}} \rangle$ is a WSTS with strong strict compatibility,
3. $\langle \mathcal{S}_G, \subseteq \rangle$ is a downward-WSTS with reflexive-transitive compatibility.

Proof. Left as an easy exercise. □

When it comes to applications, the precise choice of which ordering we consider is quite relevant because many decidable properties for WSTS's (e.g. coverability) are expressed in terms of the ordering itself. When a choice is possible, using a larger ordering will often yield less information but more efficient algorithms.

We illustrate this compromise on two different well-structured views of context-free grammars. Fig. 6 displays $FRT(XY)$ for $\langle \mathcal{S}_G, \preccurlyeq \rangle$. With the subword viewpoint, $FRT(XY)$ has 15 nodes. When we switch to the $\langle \mathcal{S}_G, \leq_{\mathrm{P}} \rangle$ view (see Fig. 7) $FRT(XY)$ is a subtree of the previous tree and only has 12 nodes.

There exist many extensions of CFG's which remain partly analyzable. One example are the *permutative grammars* [46], i.e. grammars where context-sensitive permutative rules like "$xS \to Sx$" are allowed (between any pair of symbols).

Theorem 7.2. *For any permutative grammar G, $\langle \mathcal{S}_G, \preccurlyeq \rangle$ is a downward-WSTS with reflexive compatibility.*

Proof. Left as an easy exercise. □

A machine model related to CFG's are *stack automata* (also *pushdown pro-*

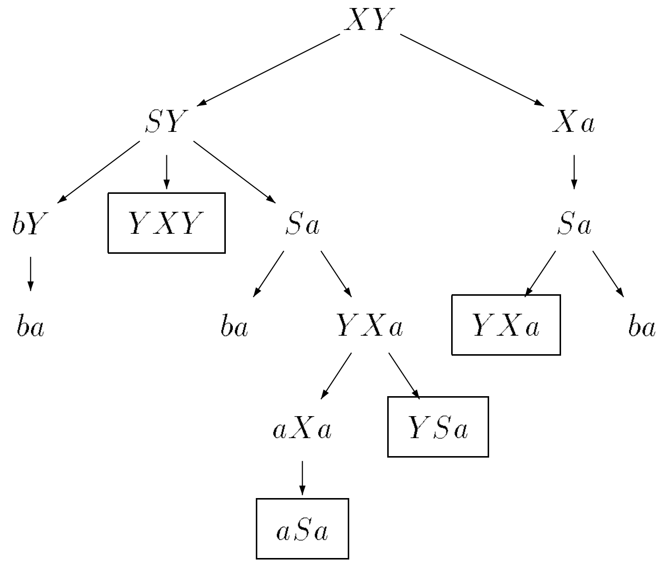


Fig. 6. $FRT(XY)$ in $\langle \mathcal{S}_G, \preccurlyeq \rangle$. Subsumed nodes are boxed.

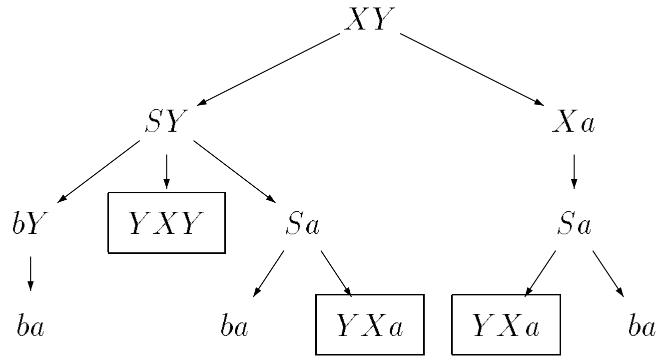


Fig. 7. $FRT(XY)$ in $\langle \mathcal{S}_G, \leq_{\mathrm{P}} \rangle$. Subsumed nodes are boxed.

cesses). Configurations have the form $\langle q, w \rangle$ where q is a control-state and $w \in \Sigma^*$ is a stack-content. Quasi-orderings between words lead to quasi-orderings between configurations of a stack automaton. E.g. with

$$\langle q, w \rangle \leq_{\mathrm{lf}} \langle q', w' \rangle \overset{\mathrm{def}}{\Leftrightarrow} q = q' \text{ and } w \leq_{\mathrm{lf}} w'.$$

For stack-automata, the $\leq_{\mathrm{lf}}$ ordering is compatible with transitions but it is not a wqo (unless we restrict the set of configurations). $\preccurlyeq$ is a wqo but it is not compatible with transitions (unless we consider subclasses of stack automata).

8 The well-structure of Basic Process Algebra

Milner's CCS [49] is the paradigmatic process algebra. Recently, several fragments of CCS with good decidability properties have been investigated [21,50]. Here we focus on BPA.

BPA, short for *Basic Process Algebra*, is a subset of CCS first studied in [11] where only prefixing, non-deterministic choice, sequential composition and guarded recursion are allowed. Here is an example BPA declaration:

$$\Delta : \quad \begin{aligned} X &:= aYX + bX + c \\ Y &:= bXX + a \end{aligned}$$

and a possible derivation is

$$X \xrightarrow{a}_\Delta YX \xrightarrow{b}_\Delta XXX \xrightarrow{c}_\Delta XX \xrightarrow{c}_\Delta \cdots \tag{2}$$

BPA systems can be seen as CFG's with head-rewriting (we lose head-rewriting when we replace sequential composition by parallel composition, yielding BPP's). Here the states of $\mathcal{S}_\Delta$ are words in N_Δ^* because the symbols in T_Δ are not stored in the state.

Because of the head-rewriting strategy, BPA systems do not have transitions compatible with word-embedding. E.g. if in the previous example, we consider $Y \preccurlyeq XYX$ and step $Y \xrightarrow{b}_\Delta XX$, we cannot find some v' with $XYX \rightarrow_\Delta v'$ and $XX \preccurlyeq v'$. (Here as with stack automata, the left-factor ordering is compatible with head-rewriting but it is not a wqo.)

However, if we restrict ourselves to Normed BPA (a class first introduced in [11]) we can find a well-structure. Formally, a BPA process is *normed* if it admits a terminating behavior. A BPA declaration is normed if all its processes are normed. From a CFG viewpoint, this corresponds to grammars in Greibach normal form and *without useless productions*. E.g. our example Δ above is a normed BPA declaration.

With Normed BPA, the difficulty with head-rewriting can be circumvented.

Theorem 8.1. *For a Normed BPA declaration Δ, $\langle \mathcal{S}_\Delta, \preccurlyeq \rangle$ is a WSTS with stuttering compatibility.*

Proof. Assume $u \preccurlyeq v$ and $u \rightarrow_\Delta u'$. Thus $u \neq \epsilon$. u has the form Xu_1 and u' is some wu_1 where $X \rightarrow_\Delta w$.

Because $u \preccurlyeq v$, v has the form $Y_1 \ldots Y_m X v_1$ with $u_1 \preccurlyeq v_1$. With normedness, there must exist sequences $\sigma_1, \ldots, \sigma_m$ of transitions such that $Y_i \xrightarrow{\sigma_i}_\Delta \epsilon$ for $i = 1, \ldots, m$. Then there exists a sequence

$$Y_1 \ldots Y_m X v_1 \xrightarrow{\sigma_1}_\Delta Y_2 \ldots Y_m X v_1 \xrightarrow{\sigma_2}_\Delta \cdots \xrightarrow{\sigma_m}_\Delta X v_1 \rightarrow_\Delta w v_1$$

so that stuttering compatibility is established. □

Without any normedness hypothesis, a well-structured view of BPA processes is still possible:

Theorem 8.2. *For any BPA declaration* Δ, $\langle \mathcal{S}_\Delta, \preccurlyeq \rangle$ *is a downward-WSTS with reflexive compatibility.*

Proof. Left as an easy exercise. □

9 The well-structure of Communicating Finite State Machines

A *Communicating Finite State Machine* (CFSM) [12,14] can be seen as a Finite State Automaton (FSA) $\langle Q, \Sigma, \ldots \rangle$ equipped with a collection $c_1, \ldots, c_n$ of n fifo channels. A transition of the FSA is labeled with a *send action* (e.g. $q \stackrel{c_i!a}{\rightarrow} q'$) or a *receive action* (e.g. $q \stackrel{c_i?a}{\rightarrow} q'$).

A CFSM C naturally gives rise to a transition system $\mathcal{S}_C$: a configuration of $\mathcal{S}_C$ is some $s = \langle q, w_1, \ldots, w_n \rangle$ where $q \in Q$ is a control state of the FSA, and each $w_i \in \Sigma^*$ is a word describing the current content of channel c_i. In configuration $s = \langle q, w_1, \ldots, w_n \rangle$, transition $q \stackrel{c_i!a}{\rightarrow} q'$ is possible, reaching $s' = \langle q', w_1, \ldots, w_{i-1}, w_i.a, w_{i+1}, \ldots, w_n \rangle$, a new configuration where the control state is now q' and where the sent symbol a has been appended after w_i. In s, transition $q \stackrel{c_i?a}{\rightarrow} q'$ is only possible if channel c_i contains an a in first position, i.e. if w_i is some $a.w'$. Then we reach $\langle q', w_1, \ldots, w_{i-1}, w', w_{i+1}, \ldots, w_n \rangle$.

Fig. 8 shows an example where P_1 and P_2 are two different automata communicating via two fifo channels. This gives a CFSM C_{P_1,P_2} if we see P_1 and P_2 as one single global FSA.

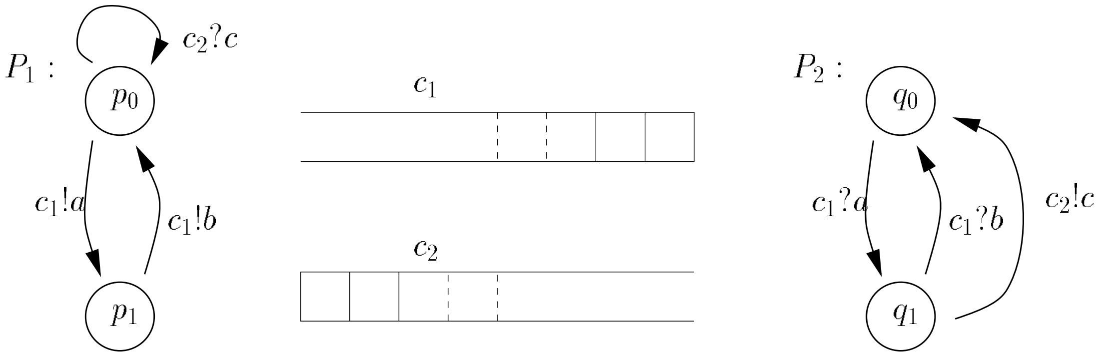


Fig. 8. A Communicating Finite State Machine

A possible behavior for this example is

$$\langle p_0q_0,\epsilon,\epsilon\rangle \stackrel{c_1!a}{\to} \langle p_1q_0,a,\epsilon\rangle \stackrel{c_1!b}{\to} \langle p_0q_0,a.b,\epsilon\rangle \stackrel{c_1?a}{\to} \langle p_0q_1,b,\epsilon\rangle \stackrel{c_2!c}{\to} \langle p_0q_0,b,c\rangle \to \cdots \tag{3}$$

Because the channels are unbounded, $\mathcal{S}_C$ is in general an infinite TS.

Fifo nets [48] are an extension of Petri nets where every place contains a fifo queue of messages (instead of the usual tokens). Basically, fifo nets are CFSM's with an explicit parallel structure and more general synchronization primitives.

Several orderings between configuration are derived from orderings between words. E.g.

$$\langle q,w_1,\ldots,w_n\rangle \preccurlyeq \langle q',w'_1,\ldots,w'_n\rangle \stackrel{\text{def}}{\Leftrightarrow} \begin{cases} q=q' \text{ and} \\ w_i \preccurlyeq w'_i \text{ for } i=1,\ldots,n \end{cases}$$

and similarly for $\leq_{\text{lf}}$. $\leq_{\text{lf}}$ is quite natural but it is still not compatible with the transitions of a CFSM (unlike the pushdown automata case) and it is not a wqo. $\preccurlyeq$ and $\leq_{\text{P}}$ are wqo's but they are not compatible with the transitions.

It turns out that both CFSM's and fifo nets are Turing-powerful [14,48]. Hence there cannot exist a general effective well-structure for CFSM's.

However, for many classes of CFSM's, there exist interesting decidable problems. The rest of this section considers different subclasses and their well-structure.

9.1 Free choice fifo nets, completely specified protocols and lossy channel systems

These three models have many similarities. They are very useful when modeling systems assuming unsafe communication links, e.g. the alternating bit protocol.

Free choice fifo nets [34] are a subclass of fifo nets that are free from any deadlock caused by the order of messages in the queues. Formally, in a free choice fifo net, whenever a place p has more than one possible output transition (this corresponds to a receive action in CFSM speak) then all these transitions only depend on p. This ensures that any message in a fifo queue may eventually be output without deadlock [34].

Completely specified protocols [33] are CFSM's in which every control state q admits the receive actions $q \stackrel{c_i?a}{\rightarrow} q$ for every action a. This action effectively "looses" a without changing the current configuration.

Lossy channel systems [3,4] are CFSM's with a modified semantics allowing the loss of messages: in any configuration the system may loose any symbol from any channel. I.e. any transition $\langle q, w_1, \ldots, w_n \rangle \rightarrow \langle q, w_1, \ldots, w_i', \ldots, w_n \rangle$ is possible when w_i' is obtained by removing one symbol from w_i.

Theorem 9.1. *With the $\preccurlyeq$ (subword) ordering, free choice fifo nets, completely specified protocols, and lossy channel systems are WSTS with stuttering compatibility.*

Proof. Left as an easy exercise. □

9.2 CFSM's with insertion errors

Cécé *et al.* introduced CFSM's with *insertion errors* [19]. These are CFSM's with a modified behavior: at any time, arbitrary symbols (noise) can be inserted anywhere in the channels. These too can be seen as well-structured systems, but not as easily as lossy channel systems.

First, when we consider $\rightarrow^{-1}$, the transition relation backward, CFSM's with Insertion Errors are exactly lossy channel systems. This view can be useful for reachability analysis, and it helps understand why [19] considered forward analysis on CFSM's with Insertion Errors, rather than the usual backward analysis based on iterated pred-basis.

Another possibility uses forward transitions and the natural subword ordering:

Theorem 9.2. *For C, a CFSM with Insertion Errors, $\langle \mathcal{S}_C, \preccurlyeq \rangle$ is a downward-WSTS with stuttering compatibility.*

Proof. Left as an easy exercise. □

9.3 Monogeneous CFSM's

More involved orderings may be used. For example, consider C_{P_1,P_2} from Fig. 8. It has transitions compatible with $\sqsubseteq$, defined by

$$\langle p_i q_j, w_1, w_2\rangle \sqsubseteq \langle p_{i'} q_{j'}, w_1', w_2'\rangle \overset{\text{def}}{\Leftrightarrow} \begin{cases} p_i = p_{i'} \text{ and } q_j = q_{j'}, \text{ and} \\ w_1' \in w_1.(ab)^* \text{ if } i = i' = 0, \text{ and} \\ w_1' \in w_1.(ba)^* \text{ if } i = i' = 1, \text{ and} \\ w_2' \in w_2.c^* \end{cases}$$

This variation around the left-factor ordering is not a well-ordering in general, but it is a well-ordering on $Q_1 \times Q_2 \times [(ab)^* + (ab)^* a] \times c^*$, a set containing all the reachable states of C_{P_1,P_2}. The same approach can be generalized to all Monogeneous CFSM's [27,28].

Given a sequence σ of transitions firable from a configuration $s = \langle q, w_1, \ldots, w_n\rangle$, we write $out(\sigma)$ for the vector $(u_1, \ldots, u_n)$ of sequences of messages output (sent) by σ. Then $s.out(\sigma)$ denotes $\langle q, w_1 u_1, \ldots, w_n u_n\rangle$.

We define an ordering $\ll$ between configurations with

$$s \ll s' \overset{\text{def}}{\Leftrightarrow} s.out(\sigma) \leq_{\text{lf}} s'.out(\sigma) \text{ for all sequences } \sigma \text{ firable from } s.$$

Monogeneous fifo nets are fifo nets where the send actions follow a certain regularity. See [27,31] for details. $\ll$ is a (decidable) wqo over the reachable states of monogeneous fifo nets. It has strong compatibility.

Theorem 9.3. [28] *With $\ll$, monogeneous fifo nets (and CFSM's) are WSTS's with strong compatibility.*

Proof. Omitted. □

9.4 Other families

Cécé [17] used a complicated ordering to show that Synchronizable CFSM [37] are WSTS's with strict compatibility. It is also possible to show that half-duplex CFSM's [18] are well-structured.

10 Miscellaneous models

There exist several other examples of WSTS's that we do not present at any length.

Some families are trivial, and we only mention them to show that WSTS's generalize many things:

- Finite state systems are WSTS's with strong strict compatibility just by taking equality as a wqo.
- More interestingly, all systems where an infinite set of states can be partitioned into a finite number of equivalence classes can be turned into WSTS's if the notion of equivalence one assumes enjoys the compatibility requirements.

  For instance, several variants of bisimulation have the compatibility requirements we mentioned. By definition, this applies to the data-independent systems of [41]. Similarly, timed-automata [7] and some hybrid automata [39] can be seen as WSTS's.

Some families are issued from models less well-known than Petri nets or context-free grammars:

- The integral relational automata of [20] are some kind of counter machine where the contents of counters can be compared and moved around but no addition or subtraction is allowed. The can be seen as some kind of WSTS's though they are infinitely branching because of input actions.
- The Recursive Parallel Program Schemes of [43,45] are some kind of nets with a restricted form of synchronization and where markings have a hierarchical, tree-like, structure. Two different wqo's turn them into WSTS's or downward-WSTS's.
- Using the fact that (several different notions of) embedding between graphs are wqo's, it is possible to find WSTS's in the field of graph-grammars and graph-rewriting systems [23].

Other families of WSTS's are obtained by applying simple general restrictions or structuring modifications to well-known models:

- **The lossy system idea.** It can be applied to many models: Lossy Turing Machines, Lossy Counter Machines [47,13], etc. Admittedly, a CFSM with lossy buffers is more realistic than a Turing Machine with lossy memory.
- **The home-state idea.** An *home-state* is a state that can be reached from any reachable state. If the initial state s_0 of some $\mathcal{S}$ is a home state, then the trivial ordering $S \times S$ is a wqo with stuttering compatibility because any step $s_1 \rightarrow s_2$ can be mimicked from any other state s'_1 simply with

$s'_1 \xrightarrow{*} s_0 \xrightarrow{*} s_1 \rightarrow s_2$. If we take a less trivial wqo, e.g. control-state equality, we have a WSTS with transitive compatibility.

Clearly, by adding a simple general "$s \rightarrow s_0$" transition, it is possible to modify most models so that their initial state is a home state, turning the model into a resetable variant that is a WSTS. The behavior of the resetable variant has obvious relations with the behavior of the previous untouched system (agreed, the resetable variant does not terminate).

We shall let the reader ponder on an interesting exercise: what can be inferred by trying to apply our five decidability results on Resetable Turing Machines ?

Part III: From transition systems to well-structured transition systems

11 WSTS's everywhere

Clearly, the several examples we presented in Part II followed a common scenario: we investigated well-known operational models of computation, and exhibited wqo's that enjoyed one form or compatibility or another. Often the wqo was suggested by the precise nature of the states (words, tuples of integers, ...) and then compatibility sometimes relied on restrictions on the possible transitions. Sometimes the wqo was quite surprising.

In this section, we would like to discuss in more general terms the problem of turning a given TS into a WSTS.

This is a quite natural and interesting question. No mention of this question is made in [1]. Finkel only discussed it in his habilitation thesis [28] where he showed that there is a largest compatible well-ordering and that this ordering is not decidable between marking of Petri nets. However, his analysis suffers from the problems we raised in the introduction of this article so that no clear and general answer is offered.

Our main result in this section is
Theorem 11.1. (Ubiquity.) *For any TS $\mathcal{S}$, there is a wqo $\leq_T$ such that $\langle \mathcal{S}, \leq_T \rangle$ is a WSTS with strict strong compatibility.*

Proof. Consider $\mathcal{S} = \langle S, \rightarrow \rangle$ and, for $s \in S$, define $T(s)$ as the length of a longest computation starting from s (such a longest computation exists because our TS's are finitely branching). $T(s)$ belongs to $\bar{\mathbb{N}} \stackrel{\text{def}}{=} \mathbb{N} \cup \{\omega\}$. Define $s \leq_T s'$ as $T(s) \leq T(s')$. $\leq_T$ is a wqo (induced by the natural wqo over $\bar{\mathbb{N}}$).

There remains to check strict strong compatibility. So consider $T(s_1) \leq T(s'_1)$ and $s_1 \to s_2$ If $T(s'_1) = \omega$ then s'_1 admits a non-terminating computation and there is a transition $s'_1 \to s'_2$ with $T(s'_2) = \omega$. Otherwise $T(s'_1) = n \geq T(s_1)$ and $T(s_2) < T(s_1)$ so that $n > 0$ and there exists a $s'_1 \to s'_2$ with $T(s'_2) = n - 1$. In both cases $T(s_2) \leq T(s'_2)$. Additionally, if $T(s_1) < T(s'_1)$ then $T(s_2) < T(s'_2)$. ☐

Furthermore $\leq_T$ is canonical in the following sense:

Proposition 11.2. *If $\langle \mathcal{S}, \leq \rangle$ is a WSTS with transitive compatibility, then $\leq \subseteq \leq_T$.*

Proof. Assume $\langle \mathcal{S}, \leq \rangle$ is a WSTS with transitive compatibility and $s_1 \leq t_1$. Given any sequence $s_1 \to s_2 \ldots \to s_n$, transitive compatibility implies the existence of some $t_1 \xrightarrow{+} t_2 \ldots \xrightarrow{+} t_n$. Then clearly $T(s_1) \leq T(t_1)$. ☐

Theorem 11.1 tells us that all TS's can be well-structured. Even TS's issued from formalisms with Turing power. Because the TS associated to a Turing Machine has finite branching and effective $Succ$, the $\leq_T$ wqo cannot be decidable in general, otherwise the decidability results for effective WSTS's would apply. Indeed we have

Theorem 11.3. *Assume $\mathcal{S}$ has effective $Succ$. Then $\leq_T$ is decidable among the states of $\mathcal{S}$ iff termination is decidable for states of $\mathcal{S}$.*

Proof. ($\Rightarrow$): use Theorem 4.6.

($\Leftarrow$): Assume termination is decidable and consider a state s. If s does not terminate, then $T(s) = \omega$. If s terminates, it is easy to compute $T(s) \in \mathbb{N}$ using the effectiveness of $Succ$ (and finite branching). Hence T is computable, so that $\leq_T$ is decidable. ☐

Finally, the $\leq_T$ wqo suffers from two main drawbacks:

- it is only computable when termination is decidable,
- it lacks expressive power.

Of course, the first inconvenient is shared with all compatible wqo's with transitive compatibility.

The second inconvenient calls for more comments. Assume $\leq_T$ is decidable for some TS $\mathcal{S}$. Pick two states s and t. Then, using Theorem 4.8 we can decide whether starting from s, the system inevitably reach a state not covered by t. This means, "inevitably reach a state farther to termination than t" (in which case s itself was already not covered by t). Using Theorem 3.6, we can tell whether, starting from s, it is possible to cover t. Again "covering t" is not very meaningful. It is much more meaningful to cover a marking of a Petri net (or a configuration of a CFSM) with the $\subseteq$ (or $\preccurlyeq$) ordering than with the $\leq_T$ ordering.

Except for Theorem 4.6 (termination), all the general decidability theorems we gave in this article involve properties defined in term of the $\leq$ ordering. They are mainly interesting when $\leq$ itself is a rich quasi-ordering, reflecting some structure of the configurations of the system under scrutiny. Note that the well-structuring wqo's we gave in Part II were all inspired by the structure of the configurations.

12 Labeled transition systems

Theorem 11.1 is possible because we considered TS's without labels. Here we briefly consider labeled TS's.

Definition 12.1. *A* labeled transition system *(LTS) is a structure* $\mathcal{S} = \langle S, L, \rightarrow, \ldots\rangle$ *where* $L = \{a, b, \ldots\}$ *is a set of labels and* $\rightarrow \subseteq S \times L \times S$ *is a set of transitions.*

For LTS's it is possible to restrict the WSTS idea so that compatibility preserves labels. This is what [1] and some proposals in [32] assume. As explained in our introduction, we think this viewpoint should be seen as a restriction of the general framework we set up in this article.

Definition 12.2. *A* well-structured LTS (WSLTS) with strong compatibility *is a LTS* $\mathcal{S} = \langle S, L, \rightarrow, \leq\rangle$ *equipped with a qo* $\leq \subseteq S \times S$ *between states such that the two following conditions hold:*

(1) well-quasi-ordering: *$\leq$ is a wqo, and*
(2) strong compatibility: *$\leq$ is compatible with $\rightarrow$, i.e. for all $s_1 \leq t_1$ and transition $s_1 \xrightarrow{a} s_2$, there exists a transition $t_1 \xrightarrow{a} t_2$ such that $s_2 \leq t_2$.*

Here again our definition does not mix effectiveness and structural issues. It is possible to have less restrictive compatibility that still preserves labels but we will not pursue this here.

Here, compatibility in Definition 12.2 really states that $\leq$ is a simulation in the classical LTS understanding [49]. Indeed, if we use $\sqsubseteq$ to denote the simulation quasi-ordering, and assume $\langle \mathcal{S}, \leq\rangle$ is a WSLTS with strong compatibility, then $\leq \subseteq \sqsubseteq$. Furthermore assuming that $\mathcal{S}$ is a LTS, we have

Theorem 12.3. *There exists a $\leq$ such that $\langle \mathcal{S}, \leq \rangle$ is a WSLTS with strong compatibility iff $\sqsubseteq$ is a wqo over the states of $\mathcal{S}$.*

Proof. If $\langle \mathcal{S}, \leq \rangle$ is a WSLTS then $\leq \subseteq \sqsubseteq$ but any qo containing a wqo is itself a wqo. Reciprocally, if $\sqsubseteq$ is a wqo over the states of $\mathcal{S}$, it is possible to pick $\leq \stackrel{\text{def}}{=} \sqsubseteq$ and get a WSLTS. $\square$

Observe that $\leq_T$ and $\sqsubseteq$ agree in the unlabeled case. In the labeled case, $\sqsubseteq$ is not always a wqo so that Theorem 12.3 can be used to prove that a given LTS admits no well-structured view, independently of effectiveness issues.

For example, consider the BPA declaration

$$\Delta : \quad \begin{aligned} X &:= a + aXa \\ Y &:= bXb \end{aligned}$$

Here the set of states reachable from Y contains all configurations $a^n b$ for $n > 0$ and, over such states, $\sqsubseteq$ coincide with $\preccurlyeq$. Clearly it is not a wqo and, consequently, $\mathcal{S}_\Delta$ cannot be turned into a WSLTS with strong compatibility.

13 Conclusion

In this article we proposed a definition of well-structured transition systems that is both simpler and more general than the earlier proposals of Finkel and Abdulla *et al.*. Simplicity and generality come essentially from a clear separation of structural and effectiveness issues.

The benefits of this new approach are multiple: simpler proofs, more general theorems, more instances of WSTS's. Regarding this last point, we would like to stress again that both Finkel's and Abdulla *et al.*'s earlier definitions did not lead to that many examples of WSTS's. On the other hand, we easily exhibited instances of the generalized WSTS notion in several operational models of computation. Figure 9 classifies the examples we gave. Of course many more families of models exist and our list is (probably) far from exhaustive. We welcome any additional example that readers could provide.

Directions for future work are numerous, some of them linked to methodological and practical issues, some more theoretical. Clearly the issue of finding well-structuring wqo's for given systems is very important. We just started investigating it in Part III.

	upward-WSTS	downward-WSTS	strict	non-strict	strong	reflexive	stuttering	transitive	
Petri nets	●		●	○	●	○	○	○	Petri Nets and their extensions
post self-modifying nets	●		●	○	●	○	○	○	
Petri nets with transfer arcs	●		●	○	●	○	○	○	
Petri nets with reset arcs	●			●	●	○	○	○	
BPP with inhibitory arcs		●		●		●			
CFSM's with Lossy Channels	●			●			●	○	Communicating Finite State Machines
CFSM's with insertion errors		●		●			●	○	
Free-choice fifo nets	●			●			●	○	
Monogeneous CFSM's	●		●	○	●	○	○	○	
Synchronizable CFSM's	●		●	○	●	○	○	○	
BPA (Basic Process Algebra)		●		●		●			Process Algebras
BPP (Basic Parallel Process)	●		●	○	●	○	○	○	
Normed BPA	●			●			●	○	
Context-free grammars	●		●	○	●	○	○	○	String Rewriting
Permutative grammars		●		●		●			
RPPS Schemes		●		●		●			Misc. models
RPPS Schemes with $\preccurlyeq_\perp$	●			●			●	○	
Integral Relational Automata	●			●	●	○	○	○	
Timed Automata	●			●	●	○	○	○	
Essentially finite systems	●			●	●	○	○	○	Generic WSTS's
Lossy systems	●			●			●	○	
Resetable systems	●			●			●	○	

(A ● indicates presence, a ○ is presence implied by a stronger ●.)

Fig. 9. Some families of well-structured transition systems

Another general direction is to look for new decidability results. In this article we did not describe all existing results. We omitted the decision method for simulation with a finite system (see [1]) or the algorithm for the coverability set (see [29]). Still we believe there is much room for new decidability results. In all likelihood such results will have to make some additional hypothesis or another, which may differ from the hypothesis we used in this article.

References

[1] P. A. Abdulla, K. Čerāns, B. Jonsson, and T. Yih-Kuen. General decidability theorems for infinite-state systems. In *Proc. 11th IEEE Symp. Logic in Computer Science (LICS'96), New Brunswick, NJ, USA, July 1996*, pages 313–321, 1996.

[2] P. A. Abdulla, K. Čerāns, B. Jonsson, and T. Yih-Kuen. Algorithmic analysis of programs with well quasi-ordered domains, 1997. To appear in Information and Computation.

[3] P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *Proc. 8th IEEE Symp. Logic in Computer Science (LICS'93), Montreal, Canada, June 1993*, pages 160–170, 1993.

[4] P. A. Abdulla and B. Jonsson. Undecidability of verifying programs with unreliable channels. In *Proc. 21st Int. Coll. Automata, Languages, and Programming (ICALP'94), Jerusalem, Israel, July 1994*, volume 820 of *Lecture Notes in Computer Science*, pages 316–327. Springer, 1994.

[5] P. A. Abdulla and B. Jonsson. Ensuring completeness of symbolic verification methods for infinite-state systems, 1997. To appear in Theor. Comp. Sci.

[6] P. A. Abdulla and B. Jonsson. Verifying networks of timed processes. In *Proc. Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98), Lisbon, Portugal, March 1998*, volume 1384 of *Lecture Notes in Computer Science*, pages 298–312. Springer, 1998.

[7] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[8] T. Araki and T. Kasami. Some decision problems related to the reachability problem for Petri nets. *Theoretical Computer Science*, 3(1):85–104, 1977.

[9] A. Arnold and M. Latteux. Récursivité et cônes rationnels fermés par intersection. *Calcolo*, XV(IV):381–394, 1978.

[10] J.-M. Autebert, J. Berstel, and L. Boasson. Context-free languages and pushdown automata. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 3, pages 111–174. Springer, 1997.

[11] J. C. M. Baeten, J. A. Bergstra, and J. W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. In *Proc. Parallel Architectures and Languages Europe (PARLE'87), Eindhoven, NL, June 1987, vol. II: Parallel Languages*, volume 259 of *Lecture Notes in Computer Science*, pages 94–111. Springer, 1987.

[12] G. von Bochmann. Finite state description of communication protocols. *Computer Networks and ISDN Systems*, 2:361–372, 1978.

[13] A. Bouajjani and R. Mayr. Model checking lossy vector addition systems. In *Proc. 16th Ann. Symp. Theoretical Aspects of Computer Science (STACS'99), Trier, Germany, Mar. 1999*, volume 1563 of *Lecture Notes in Computer Science*, pages 323–333. Springer, 1999.

[14] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

[15] M. C. Browne, E. M. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59(1–2):115–131, 1988.

[16] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[17] G. Cécé. *Etat de l'art des techniques d'analyse des automates finis communicants.* Rapport de DEA, Université de Paris-Sud, Orsay, France, September 1993.

[18] G. Cécé and A. Finkel. Programs with quasi-stable channels are effectively recognizable. In *Proc. 9th Int. Conf. Computer Aided Verification (CAV'97), Haifa, Israel, June 1997*, volume 1254 of *Lecture Notes in Computer Science*, pages 304–315. Springer, 1997.

[19] G. Cécé, A. Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, 1995.

[20] K. Čerāns. Deciding properties of integral relational automata. In *Proc. 21st Int. Coll. Automata, Languages, and Programming (ICALP'94), Jerusalem, Israel, July 1994*, volume 820 of *Lecture Notes in Computer Science*, pages 35–46. Springer, 1994.

[21] S. Christensen, Y. Hirshfeld, and F. Moller. Decidable subsets of CCS. *The Computer Journal*, 37(4):233–242, 1994.

[22] G. Ciardo. Petri nets with marking-dependent arc cardinality: Properties and analysis. In *Proc. 15th Int. Conf. Applications and Theory of Petri Nets, Zaragoza, Spain, June 1994*, volume 815 of *Lecture Notes in Computer Science*, pages 179–198. Springer, 1994.

[23] B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, vol. B*, chapter 5, pages 193–242. Elsevier Science Publishers, 1990.

[24] L. E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with r distinct prime factors. *Amer. Journal Math.*, 35:413–422, 1913.

[25] C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In *Proc. 25th Int. Coll. Automata, Languages, and Programming (ICALP'98), Aalborg, Denmark, July 1998*, volume 1443 of *Lecture Notes in Computer Science*, pages 103–115. Springer, 1998.

[26] J. Esparza. More infinite results. In *Proc. 1st Int. Workshop on Verification of Infinite State Systems (INFINITY'96), Pisa, Italy, Aug. 30–31, 1996*, volume 5 of *Electronic Notes in Theor. Comp. Sci.* Elsevier Science Publishers, 1997.

[27] A. Finkel. About monogeneous fifo Petri nets. In *Proc. 3rd European Workshop on Applications and Theory of Petri Nets, Varenna, Italy, Sep. 1982*, pages 175–192, 1982.

[28] A. Finkel. *Structuration des Systèmes de Transitions. Applications au Contrôle du Parallelisme par Files FIFO.* Thèse de Docteur d'Etat, Université de Paris-Sud, Orsay, France, June 1986.

[29] A. Finkel. A generalization of the procedure of Karp and Miller to well structured transition systems. In *Proc. 14th Int. Coll. Automata, Languages, and Programming (ICALP'87), Karlsruhe, FRG, July 1987*, volume 267 of *Lecture Notes in Computer Science*, pages 499–508. Springer, 1987.

[30] A. Finkel. Well structured transition systems. Research Report 365, Lab. de Recherche en Informatique (LRI), Univ. Paris-Sud, Orsay, August 1987.

[31] A. Finkel. A new class of analyzable CFSMs with unbounded FIFO channels. In *Prelim. Proc. 8th IFIP WG 6.1 Int. Symp. Protocol Specification, Testing and Verification, Atlantic City, New Jersey, USA, June 1988*, 1988.

[32] A. Finkel. Reduction and covering of infinite reachability trees. *Information and Computation*, 89(2):144–179, 1990.

[33] A. Finkel. Decidability of the termination problem for completely specificied protocols. *Distributed Computing*, 7:129–135, 1994.

[34] A. Finkel and A. Choquet. Fifo nets without order deadlock. *Acta Informatica*, 25(1):15–36, 1987.

[35] A. Finkel, P. McKenzie, and C. Picaronny. A well-structured framework for analysing Petri nets extensions. Research Report LSV-99-2, Lab. Specification and Verification, ENS de Cachan, Cachan, France, February 1999.

[36] R. J. van Glabbeek and W. P. Weijland. Branching time and abstraction in process algebra. In G. X. Ritter, editor, *Information Processing 89*, pages 613–618. North-Holland, August 1989.

[37] M. G. Gouda and L. E. Rosier. Synchronizable networks of communicating finite state machines. Unpublished manuscript, 1985.

[38] B. Heinemann. Subclasses of self-modifying nets. In *Applications and Theory of Petri Nets (Selected Papers from the First and Second European Workshop, Starsbourg, France, Sep. 1980, Bad Honnef, Germany, Sep. 1981)*, pages 187–192. Springer, 1982.

[39] T. A. Henzinger. Hybrid automata with finite bisimulations. In *Proc. 22nd Int. Coll. Automata, Languages, and Programming (ICALP'95), Szeged, Hungary, July 1995*, volume 944 of *Lecture Notes in Computer Science*, pages 324–335. Springer, 1995.

[40] G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc. (3)*, 2(7):326–336, 1952.

[41] B. Jonsson and J. Parrow. Deciding bisimulation equivalences for a class of non-finite-state programs. *Information and Computation*, 107(2):272–302, 1993.

[42] R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969.

[43] O. Kouchnarenko and Ph. Schnoebelen. A model for recursive-parallel programs. In *Proc. 1st Int. Workshop on Verification of Infinite State Systems (INFINITY'96), Pisa, Italy, Aug. 1996*, volume 5 of *Electronic Notes in Theor. Comp. Sci.* Elsevier, 1997.

[44] J. B. Kruskal. The theory of well-quasi-ordering: A frequently discovered concept. *J. Combinatorial Theory, Series A*, 13(3):297–305, 1972.

[45] O. Kushnarenko and Ph. Schnoebelen. A formal framework for the analysis of recursive-parallel programs. In *Proc. 4th Int. Conf. Parallel Computing Technologies (PaCT'97), Yaroslavl, Russia, Sep. 1997*, volume 1277 of *Lecture Notes in Computer Science*, pages 45–59. Springer, 1997.

[46] E. Mäkinen. On permutative grammars generating context-free languages. *BIT*, 25:604–610, 1985.

[47] R. Mayr. Lossy counter machines. Tech. Report TUM-I9830, Institut für Informatik, TUM, Munich, Germany, October 1998.

[48] G. Memmi and A. Finkel. An introduction to FIFO nets–monogeneous nets: A subclass of FIFO nets. *Theoretical Computer Science*, 35(2–3):191–214, 1985.

[49] R. Milner. *Communication and Concurrency.* Prentice Hall Int., 1989.

[50] F. Moller. Infinite results. In *Proc. 7th Int. Conf. Concurrency Theory (CONCUR'96), Pisa, Italy, Aug. 1996*, volume 1119 of *Lecture Notes in Computer Science*, pages 195–216. Springer, 1996.

[51] J. L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice Hall Int., 1981.

[52] W. Reisig. *Petri Nets. An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985.

[53] P. Z. Revesz. A closed form for Datalog queries with integer order. In *Proc. 3rd Int. Conf. Database Theory (ICDT'90), Paris, France, Dec. 1990*, volume 470 of *Lecture Notes in Computer Science*, pages 187–201. Springer, 1990.

[54] R. Valk. Self-modifying nets, a natural extension of Petri nets. In *Proc. 5th Int. Coll. Automata, Languages, and Programming (ICALP'78), Udine, Italy, Jul. 1978*, volume 62 of *Lecture Notes in Computer Science*, pages 464–476. Springer, 1978.