From Well Structured Transition Systems to Program Verification

Alain Finkel
Université Paris-Saclay, ENS Paris-Saclay, CNRS,
Laboratoire Spécification et Vérification, 91190, Gif-sur-Yvette, France.
Institut Universitaire de France.
finkel@lsv.fr

Abstract: We describe the use of the theory of WSTS for verifying programs.

1 Preliminaries

A relation $\leq \subseteq X \times X$ over a set X is a *quasi-ordering* if it is reflexive and transitive, and a *partial ordering* if it is antisymmetric as well. It is *well-founded* if it has no infinite descending chain. A quasi-ordering $\leq$ is a *well-quasi-ordering* (resp. *well partial order*), *wqo* (resp. *wpo*) for short, if for every infinite sequence $x_0, x_1, \cdots \in X$, there exist $i < j$ such that $x_i \leq x_j$. This is strictly stronger than being well-founded.

One example of well-quasi-ordering is the componentwise ordering of tuples over $\mathbb{N}$. More formally, $\mathbb{N}^d$ is well-quasi-ordered by $\leq$ where, for every $\mathbf{x}, \mathbf{y} \in \mathbb{N}^d$, $\mathbf{x} \leq \mathbf{y}$ if and only if $\mathbf{x}(i) \leq \mathbf{y}(i)$ for every $i \in [d]$. We extend $\mathbb{N}$ to $\mathbb{N}_\omega \stackrel{\text{def}}{=} \mathbb{N} \cup \{\omega\}$ where $n \leq \omega$ for every $n \in \mathbb{N}_\omega$. $\mathbb{N}_\omega^d$ ordered componentwise is also well-quasi-ordered. Let Σ be a finite alphabet. We write Σ^* to denote the set of finite words over Σ. For every $u, v \in \Sigma^*$, we write $u \sqsubseteq v$ if u is a subword of v, i.e. u can be obtained from v by removing zero, one or multiple letters. Σ^* is well-quasi-ordered by $\sqsubseteq$.

2 Well Structured Transition Systems

2.1 Well structured transition systems: wqo and monotony

An *ordered (labeled) transition system* is a triple $(X, \xrightarrow{\Sigma}, \leq)$ such that $(X, \xrightarrow{\Sigma})$ is a *(labeled) transition system* and $\leq$ is a quasi-ordering. An ordered transition system $\mathscr{S}$ is a *well structured transition system (WSTS)* if $\leq$ is a well-quasi-ordering and $\mathscr{S}$ is *monotone*, i.e. for all $x, x', y \in X$ and $a \in \Sigma$ such that $x \xrightarrow{a} y$ and $x' \geq x$, there exists $y' \in X$ such that $x' \xrightarrow{*} y'$ and $y' \geq y$. Many other types of monotonicities were defined in the literature (see [13]), but, for our purposes, we only need to introduce strong monotonicities. We say that $\mathscr{S}$ has *strong monotonicity* if for all $x, x', y \in X$ and $a \in \Sigma$, $x \xrightarrow{a} y$ and $x' \geq x$ implies $x' \xrightarrow{a} y'$ for some $y' \geq y$. We say that $\mathscr{S}$ has *strong-strict monotonicity*[1] if it has strong monotonicity and for all $x, x', y \in X$ and $a \in \Sigma$, $x \xrightarrow{a} y$ and $x' > x$ implies $x' \xrightarrow{a} y'$ for some $y' > y$.

Theorem 1. *[10, 13, 2] Termination, boundedness, control-state reachability and coverability are decidable for effective WSTS with strong-strict monotony.*

[1] Strong-strict monotonicity should not be confused with strong *and* strict monotonicities. Here strongness and strictness have to hold at the *same* time.

L. Fribourg and M. Heizmann (Eds.): VPT/HCVS 2020
EPTCS 320, 2020, pp. 44–49, doi:10.4204/EPTCS.320.3

There are two main techniques for proving these decidability results: backward and forward analysis. The backward coverability algorithm allows to compute the finite basis of the set of all predecessors of the upward closure of a state. The forward coverability algorithm computes the finite reduced reachability tree and the finite (extended) Karp-Miller tree (under supplementary hypothesis): these two forward algorithms operate with inductive downward closed invariants.

2.2 A short story of well structured transition systems

Well structured transition systems (initially called *structured transition systems* in [10]) were initially defined and studied as monotone transition systems equipped with a well-quasi-ordering on their set of states. Termination was shown decidable for *well structured transition systems* with *transitive* monotonicity, while boundedness was shown decidable for well structured transition systems with *strict* monotonicity in [10]. For a subclass of finitely branching labeled well structured transition systems with strong-strict monotonicity, now called *very well structured transition systems* in [5], a generalization of the Karp-Miller algorithm was shown to compute their coverability sets [10, 5]. In [2], the coverability problem was shown to be decidable for a subclass of well structured transition systems, i.e. *labeled* well structured transition systems with *strong monotonicity* [2, Def. 3.4] and satisfying an additional *effective hypothesis*: the existence of an algorithm to compute the finite set $min(\mathrm{Pre}\,(\uparrow s))$ of minimal elements of $\mathrm{Pre}\,(\uparrow s)$, where $\mathrm{Pre}\,(\uparrow s)$ is the set of immediate predecessors of the upward-closure $\uparrow s$ of a state s. In [13], mathematical properties were distinguished from effective properties, and the coverability problem was shown decidable for the *entire* class of well structured transition systems satisfying the similar additional *effective hypothesis* that there exists an algorithm to compute the finite set $min(\uparrow \mathrm{Pre}\,(\uparrow s))$, i.e., the hypotheses of transitions labeling and strong monotonicity made in [2] turned out to be superfluous.

Today, following the presentation of [13], what is *mathematically* known as *well structured transition systems* (or shortly *well structured systems*) is exactly the original class of *structured transition systems* [10]; and necessary effective hypotheses are added for obtaining decidability of properties such as termination, control-state reachability, coverability and boundedness.

3 From Programs to Well Structured Transition Systems

3.1 The general method

Given a program P and a safety property ϕ, let's describe two steps for verifying that P satisfies ϕ by using WSTS:

1. The first step is to build a transition system $(S, \rightarrow)$ associated with (P, ϕ). This is well known as the operationnal semantics of the program and we are used to this. But the problem is the hudge size of the associated transition system. In general we will define and compute an abstraction of the original program P because we may (and must) forget some useless parts of the program that have no effect on property ϕ. A kind of such activities is the (static and dynamic) slicing that computes parts of the program that may modify a set of variables and this computation can be done with a small cost. There exist other techniques to build abstractions of the program that produce smaller and tractable programs. We have also to translate the property ϕ on P into a state-property ϕ_S in $(S, \rightarrow)$ (sometimes a formula in a logic) that would be decidable for WSTS.
2. The second step is to look for an ordering $\leq$ having these two desired properties (monotony and well ordering), i.e., such that $(S, \rightarrow, \leq)$ is WSTS. Let us recall that the termination ordering makes

of each transition system a WSTS [13] but this ordering is undecidable so the obtained WSTS is not effective and we cannot deduce the decidability of usual properties. If we find such decidable ordering $\leq$, we just verify whether $(S, \rightarrow, \leq)$ satisfies the state-property ϕ_S. To make this verification, one usually reduces ϕ_S to a coverability property in $(S, \rightarrow, \leq)$.

3.2 What can you do when you can't find a monotone well ordering ?

Let us analyse two cases that are not directly translatable into WSTS.

3.2.1 We found a well ordering which is not strongly monotone

Let us consider the case in which we found a well ordering $\leq$ but $(S, \rightarrow, \leq)$ is unfortunately not strongly monotone. Apart from the usual well ordering on integers (Dickson), there exist many well orderings on different kinds of sets: let us enumerate, the multiset ordering, the subword ordering on finite words (Higman), the homeomorphic embedding on finite trees (Kruskal), the minor ordering (Robertson & Seymour) on finite graphs,...etc. These orderings can be often extended to the infinite. With Jean Goubault-Larrecq, we define in [11] an algebra allowing the composition of well orderings by many operators like finite cartesian product.

Let us consider a counter machine M. Recall that the usual ordering on positive integers (which extends to vectors of integers) is well (Dickson Lemma) but it is not (strongly) monotone on general counters machines because the guards containing tests to zero are typically not monotone. We may change the original machine into another one which will be a WSTS. We may change the operations and/or the states.

A first drastic action is to remove the tests to zero; another possibility is to replace tests to zero by resets (or by transfers). The new machine M_{new} is now monotone, hence machine M_{new} is a WSTS (for the usual ordering) that over-approximates the original counter machine M. If M_{new} never meets a bad state then one may deduce the same for M. Other properties like termination, boundedness, non-reachability are also preserved by monotonic abstraction [3].

We may change the states by abstracting them modulo an equivalence relation $\equiv$ or even with an ordering. One may also look for a computable abstraction $(S', \rightarrow', \leq')$ of $(S, \rightarrow, \leq)$ where $S' = (S/\equiv)$ and $\leq' = (\leq / \equiv)$ are an abstraction of $(S, \leq)$ such that the new transition relation $\rightarrow'$ (between abstract states in S') is monotone with respect to $\leq'$ which must be still well and then $(S', \rightarrow', \leq')$ is a WSTS. The Abstract Interpretation [7] could be completed in the direction to produce WSTSs.

Another way is to consider general non monotone models and to test if a particular instance of the model is strongly monotone. This question is decidable, for example, for Presburger counter machine [14].

3.2.2 We found a strongly monotone ordering which is not well

A first possibility is use algorithms in WSTS as semi-algorithms in strongly monotone transition systems. But there is another way. The ordering which is not well on the considered set of states could be well on the subset of reachable states. In general, the reachability set is not computable but in some cases, it is possible to compute an overapproximation of the reachability set on which the ordering is well.

Another way is to consider general strongly monotone non-well ordered transition systems and to test if a particular ordering is well. This question is decidable, for example, for orderings defined by Presburger formulas (Presburger orderings) (see [14] for the decidability for orderings in $\mathbb{N}$).

4 Examples

4.1 Programs with integers

Many programs can be modeled as counter machines (for example programs with lists [4]). Presburger counter machines (PCM) are a general model that allows to express guards and operations as Presburger formulas. It is clear that PCM contain Minsky machines and, as an immediate consequence, all non-trivial properties are undecidable for PCM. Let us now illustrate some notions introduced in step 2 of the strategy described before. Let $M = (Q, ...)$ be a Presburger counter machine with a set finite set Q of control-states and d counters. Let us first consider the most natural well ordering $\leq$ on integers that we classically extend on vectors as follows: let $\preceq \stackrel{\text{def}}{=} =_Q \times \leq^d$ where $=_Q$ is the equality on the finite set Q and $\leq^d$ is the vector ordering component by component. By Dickson Lemma, we know that $\preceq$ is still well. We cannot directly decide whether M is strongly monotone for $\preceq$ but we may decide the strong monotony property for M because both the description of M and of the strong monotony property can be expressed as Presburger formulas [14]. If M is strongly monotone for $\preceq$, we may use the WSTS theory. In the case where M is not strongly monotone for $\preceq$, we may use the following (non-terminating) semi-algorithm that enumerates Presburger formulas $\psi_1, \psi_2, ..., \psi_n, ...$ representing well orderings $\leq_1, \leq_2, ..., \leq_n, ...$ on $\mathbb{N}^d$ and test, for all n, whether M is strongly $\leq_n$-monotone. If there exists an integer $n \geq 1$ such that $\leq_n$ is well and strongly monotone on M, then the termination of the previous semi-algorithm is insured. But if there don't exist such n, this enumeration will never terminate and then it don't provide an algorithm to decide whether there exists a strongly monotone Presburger well ordering for M. Let us define the class of *existentially (strongly) well structured* Presburger counter machines as follows:

Definition 4.1. A Presburger counter machine M is *existentially well structured (resp. existentially strongly well structured)* if there exists a Presburger well ordering that is monotone (resp. strongly monotone) for M.

Coverability and other properties (see Theorem 1) are decidable for existentially well structured PCMs. We may prove that the monotony property is undecidable [14] for PCM of dimension one (and for Minsky machines of dimension 2) with the usual well ordering on integers and we conjecture that the *existentially well structured problem* (i.e., whether a PCM is existentially well structured) is also undecidable. Another natural (and still open) question is then to know whether the *existential strongly well structured problem* is decidable for PCMs.

4.2 Communication protocols

Let us consider a distributed program composed of a finite set of processes (finite automata, pushdown processes,...) that exchanges messages through fifo channels. We know that queue automata also called fifo machines (i.e., a finite automaton that communicates with an unique fifo buffer also called a *bi-directional* fifo channel) may simulate Turing machines and counter machines [15] and this is still true for two finite automata communicating through *one-directional* fifo channels [6]. Let us consider, for simplifying notations, fifo machines (a single sequential control-graph) $M = (Q, ...)$ communicating with d channels and the most natural ordering on words, adapted to the fifo behavior, say the prefix ordering $\leq_{prefix}$ that is extended as previously by $\preceq_{prefix} \stackrel{\text{def}}{=} =_Q \times \leq^d_{prefix}$. Unfortunatly this ordering is not monotone neither well (except in the trivial case where the channel alphabets are reduced to an unique letter). The subword ordering $\sqsubseteq$ on finite words is well (Higman's Theorem) and its classical extension $\preceq_\sqsubseteq \stackrel{\text{def}}{=} =_Q \times \sqsubseteq^d$ is also well but it is not monotone on fifo machines ; however, $\preceq_\sqsubseteq$ is monotone on fifo machines with other semantics (like lossy, insertion), hence such non-perfect fifo machines are WSTS

for the extended subword ordering. These kind of non-perfect fifo machines over-approximates original perfect fifo machines and we may apply the monotonic abstraction described previously in Section 3.

4.3 Other programs

There exist many other illustrations of the power of WSTS to verify programs like hardware design, multithreaded programs, distributed systems. Let's quote programs with pointers and the use of graphs and orderings on graphs (subgraph ordering and minor ordering) to model the state of the memory [1], parameterized verification of distributed algorithms [8], programs with time constraints (timed Petri nets), cryptographic protocols [9], broadcast protocols,...etc.

References

[1] Parosh Aziz Abdulla, Muhsin Atto, Jonathan Cederberg & Ran Ji (2009): *Automated Analysis of Data-Dependent Programs with Dynamic Memory*. In Zhiming Liu & Anders P. Ravn, editors: *Automated Technology for Verification and Analysis, 7th International Symposium, ATVA 2009, Macao, China, October 14-16, 2009. Proceedings*, *Lecture Notes in Computer Science* 5799, Springer, pp. 197–212. Available at `https://doi.org/10.1007/978-3-642-04761-9_16`.

[2] Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson & Yih-Kuen Tsay (2000): *Algorithmic Analysis of Programs with Well Quasi-ordered Domains*. *Inf. Comput.* 160(1-2), pp. 109–127, doi:`10.1006/inco.1999.2843`.

[3] Parosh Aziz Abdulla, Giorgio Delzanno, Noomene Ben Henda & Ahmed Rezine (2009): *Monotonic Abstraction: on Efficient Verification of Parameterized Systems*. *Int. J. Found. Comput. Sci.* 20(5), pp. 779–801. Available at `https://doi.org/10.1142/S0129054109006887`.

[4] Sébastien Bardin, Alain Finkel, Étienne Lozes & Arnaud Sangnier (2006): *From Pointer Systems to Counter Systems Using Shape Analysis*. In Ramesh Bharadwaj, editor: *Proceedings of the 5th International Workshop on Automated Verification of Infinite-State Systems (AVIS'06)*, Vienna, Austria. Available at `http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/BFLS-AVIS-06.pdf`.

[5] Michael Blondin, Alain Finkel & Jean Goubault-Larrecq (2017): *Forward Analysis for WSTS, Part III: Karp-Miller Trees*. In Satya Lokam & R. Ramanujam, editors: *Proceedings of the 37th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'17)*, *Leibniz International Proceedings in Informatics* 93, Leibniz-Zentrum für Informatik, Kanpur, India, pp. 16:1–16:15, doi:`10.4230/LIPIcs.FSTTCS.2017.16`. Available at `https://hal.archives-ouvertes.fr/hal-01736704/`.

[6] Daniel Brand & Pitro Zafiropulo (1983): *On Communicating Finite-State Machines*. *J. ACM* 30(2), pp. 323–342. Available at `https://doi.org/10.1145/322374.322380`.

[7] Patrick Cousot & Radhia Cousot (1977): *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints*. In Robert M. Graham, Michael A. Harrison & Ravi Sethi, editors: *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, ACM, pp. 238–252. Available at `https://doi.org/10.1145/512950.512973`.

[8] Giorgio Delzanno & Jan Stückrath (2014): *Parameterized Verification of Graph Transformation Systems with Whole Neighbourhood Operations*. In Joël Ouaknine, Igor Potapov & James Worrell, editors: *Reachability Problems - 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings*, *Lecture Notes in Computer Science* 8762, Springer, pp. 72–84. Available at `https://doi.org/10.1007/978-3-319-11439-2_6`.

[9] Emanuele D'Osualdo & Felix Stutz (2019): *Decidable Inductive Invariants for Verification of Cryptographic Protocols with Unbounded Sessions*. *CoRR* abs/1911.05430. Available at `http://arxiv.org/abs/1911.05430`.

[10] Alain Finkel (1987): *A generalization of the procedure of Karp and Miller to well structured transition system*. In Thomas Ottmann, editor: *Proceedings of the 14th International Colloquium on Automata, Languages and Programming (ICALP'87)*, *Lecture Notes in Computer Science* 267, Springer-Verlag, Karlsruhe, Germany, pp. 499–508, doi:10.1007/3-540-18088-5_43. Available at http://www.lsv.fr/Publis/PAPERS/PDF/F-icalp87.pdf.

[11] Alain Finkel & Jean Goubault-Larrecq (2009): *Forward Analysis for WSTS, Part I: Completions*. In Susanne Albers & Jean-Yves Marion, editors: *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, *LIPIcs* 3, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, pp. 433–444. Available at https://doi.org/10.4230/LIPIcs.STACS.2009.1844.

[12] Alain Finkel & Philippe Schnoebelen (1998): *Fundamental Structures in Well-Structured Infinite Transition Systems*. In Claudio L. Lucchesi & Arnaldo V. Moura, editors: *Proceedings of the 3rd Latin American Symposium on Theoretical Informatics (LATIN'98)*, *Lecture Notes in Computer Science* 1380, Springer, Campinas, Brasil, pp. 102–118, doi:10.1007/BFb0054314. Available at http://www.lsv.ens-cachan.fr/Publis/PAPERS/PS/FinSch-latin98.ps.

[13] Alain Finkel & Philippe Schnoebelen (2001): *Well-Structured Transition Systems Everywhere!* *Theoretical Computer Science* 256(1-2), pp. 63–92, doi:10.1016/S0304-3975(00)00102-X. Available at http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/FinSch-TCS99.pdf.

[14] Ekanshdeep Gupta & Alain Finkel (2019): *The well structured problem for Presburger counter machines*. In Arkadev Chattopadhyay & Paul Gastin, editors: *Proceedings of the 39th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'19)*, Leibniz International Proceedings in Informatics, Leibniz-Zentrum für Informatik, Bombay, India, pp. 41:1–41:15, doi:10.4230/LIPIcs.FSTTCS.2019.41. Available at https://drops.dagstuhl.de/opus/frontdoor.php?source_opus=11603.

[15] Bernard Vauquelin & Paul Franchi-Zannettacci (1980): *Automates a File*. *Theor. Comput. Sci.* 11, pp. 221–225. Available at https://doi.org/10.1016/0304-3975(80)90047-X.