WQOs and BQOs in automated program verification

Philippe Schnoebelen

LMF, CNRS & ENS Paris-Saclay
(now visiting CMI in Chennai)

WQO-BQO: What is up? / Lyon / 21-23 Feb 2023

Based on joint work with A. Finkel, S. Schmitz, P. Jančar, P. Chambard, N. Bertrand, P. Karandikar, A. Rabinovich, Ch. Baier, etc.

Outline of the Talk

- Part 1: **Automated program verification?**
- Part 2: **WQOs and Well-Structured Systems (WSTSs)**
- Part 3: **Verifying WSTS**
- Part 4: **Assessing Complexity**

Part 1

Automated program verification?

What is "(Formal) Program Verification"?

"Verification" = Proving that a computer program is "correct", i.e. behaves as announced, has absolutely no bugs.

It is a mathematical proof, about a finite mathematical object, e.g.
— a program;
— an algorithm;
— a protocol / a data type / high-level architecture / abstract program / hybrid system / ...

Proving that the program always terminates, in at most so many steps, using at most so much memory, that it returns a value fulfilling the specification, etc.

Formal verification was
— introduced in the 60s (Dijkstra, Floyd, Hoare, Milner, ..),
— led to "model checking" in the 80s,
— became a requirement for safety critical software (in avionics and other strongly regulated industries).

Pros and cons of formal verification

✓ The proof gives very strong guarantees about a "program" that is almost exactly the real-world object we want to certify.

× Proof can be very difficult to find (or not exist if program has a bug).

× Specifying what has to be proven is hard, error-prone, & never completed.

× Proofs of program correctness have bugs too

✓ Verification can be computer-assisted.

× Proof has to be redone every time you modify/update the program.

✓ Works very well at the right scale: abstract algorithms, protocols, or subroutines/libraries with well-understood interfaces.

✓ Can already boast of some truly incredible achievements, e.g. Xavier Leroy's COMPCERT project.

AND "AUTOMATED" PROGRAM VERIFICATION?

Mostly exists in the form of Model Checking, based on algorithms that automatically prove correctness properties of a program.

Think "computer algebra" or "automated theorem proving" but specialized towards program behaviors and their properties.

Pioneered by Pnueli, Clarke, Emerson, Sifakis in the early 1980s.

Started with finite-state programs:
— Communication protocols, concurrent algorithms, cryptographic protocols, ..
— VLSI designs with huge state space.

Then considered infinite-state programs:
— one cannot hope for a general algorithm.
— huge variety of ad-hoc methods for specific program constructions and properties of interest, looking for best compromises between expressiveness and tractability.

Field is very challenging! E.g. how to prove termination of the $3n + 1$ program, aka Collatz conjecture?

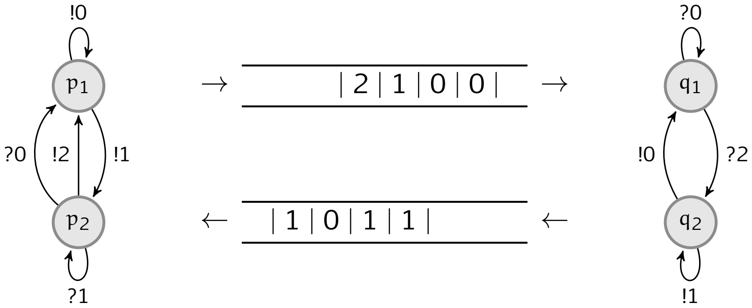
WELL QUASI-ORDERS AND PROGRAM VERIFICATION

- Well-structured systems (WSTSs) are a generic family of models, with infinite but well-quasi-ordered set of states, that admits generic verification algorithms.
- WSTSs invented by Finkel (1987 onward), developed and popularized by Abdulla & Jonsson (1993 onward), Finkel & Schnoebelen (1996 onward), and many others.
- Started with counters, queues, gap-order constraints, etc.
- Still very active these days, with new models (using wqos on graphs, etc.), new algorithms (probabilistic properties, game-theoretical properties, ..) or new applications (data logics, modal logics, etc.) appearing every year.

Part 2

Well-Structured Systems (WSTSs)

EXAMPLE: PRIORITY CHANNEL SYSTEMS (2013)



Unbounded fifo channels (or queues)

$(p_1, q_1, 0012, 1011) \xrightarrow{!1} (p_2, q_1, 00121, 1011)$
$(p_1, q_1, 0012, 1011) \xrightarrow{!1} (p_2, q_1, 00121, 1011) \to (p_2, q_1, 00121, 111)$

messages in transit can supersede messages in front of them if priority is not higher

Operational Semantics = Transition Systems

The behaviour of a Priority Channel System P (more generally, a program) is given by a transition system $\mathcal{S}_P = (S, \rightarrow)$

NB: In general, $\mathcal{S}_P$ is not deterministic: a configuration may have several immediate successors

We are interested in proving properties of paths in $\mathcal{S}_P$

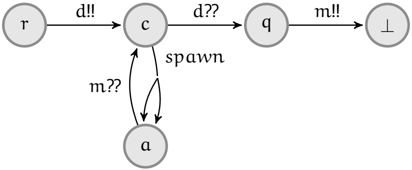
Main questions:

Safety: given $I, Bad \subseteq S$, check that there are no paths from I to Bad. E.g. "deadlock never occurs".

Inevitability: given $I, Good \subseteq S$, check that all maximal paths from I eventually visit $Good$. E.g. "program always terminates".

Example: Broadcast Protocols

Broadcast protocols (Esparza, Finkel, Mayr 1999), aka population protocols, are dynamic & distributed collections of finite-state processes communicating via brodcasts (and rendez-vous, not featured here).

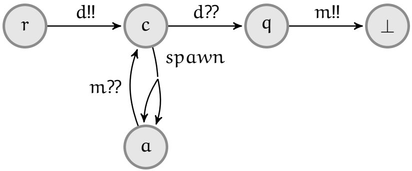


A configuration collects the local states of all processes. E.g., $s = \{c, r, c\}$, also denoted $\{c^2, r\}$.

Steps:

$$\{c^2, q, r\} \xrightarrow{s(pawn)} \{a^2, c, q, r\} \xrightarrow{s} \{a^4, q, r\} \xrightarrow{m} \{c^4, r, \perp\} \xrightarrow{d} \{c, q^4, \perp\}$$

Proving Termination



This protocol has no infinite runs

Proof. Write $s = \{r^{n_1}, q^{n_2}, c^{n_3}, a^*, \perp^*\}$.
In any step $s \to s'$ the triple $\langle n_1, n_2, n_3 \rangle$ decreases in the lexicographic ordering

This is the usual pattern for proofs of termination: one invents a well-founded measure that decreases with every step

Broadcast protocols are well behaved

1. Order the configurations by multiset inclusion, e.g., $\{c, q\} \subseteq \{c^2, r, q\}$

2. Observe that steps are monotonic:

$$s \to t \wedge s \subseteq s' \implies \exists t' : s' \to t' \wedge t \subseteq t'$$

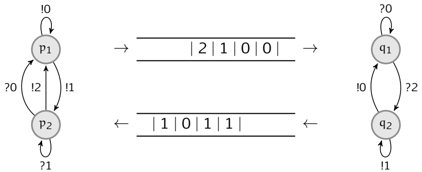
Proof. Case analysis: is $s \to t$ an internal move? or a spawning step? or a broadcasting? or a rendez-vous?

3. Further observe that $(S, \subseteq)$ is wqo

$\Rightarrow$ We say that broadcast protocols are "well structured TS"

Thm. (Finkel, Abdulla, ..)
For such systems termination and safety are decidable

Priority Channel Systems are WSTS



Let $u = p_1..p_k$ be a channel contents.
Write $u \rightarrow_\# u'$ when $u' = p_1..p_{i-1}p_{i+1}..p_k$ and $p_i \leqslant p_{i+1}$.
E.g. $1011 \rightarrow_\# 111$

1. Write $\leqslant_\#$ for the transitive closure of $\rightarrow_\#^{-1}$ and let

$$(p,q,u,v) \leqslant_\# (p',q',u',v') \overset{\text{def}}{\Leftrightarrow} p = q' \wedge q = q' \wedge u \leqslant_\# u' \wedge v \leqslant_\# v'$$

2. Steps are monotonic:

$$s \rightarrow t \wedge s \leqslant_\# s' \implies s' \rightarrow \cdots \rightarrow s \rightarrow t$$

3. Furthermore $(S, \leqslant_\#)$ is a wqo

$\Rightarrow$ PCS are WSTS! We can decide termination and safety.

Part 2

Verifying WSTSs

Deciding Termination for WSTS

Lem. [Finite Witnesses for Infinite Runs]
A WSTS $\mathcal{S}$ has an infinite run from s_{init} **iff** it has a finite run from s_{init} that is a good sequence.

Recall: $s_0, s_1, s_2, \ldots, s_n$ is good $\overset{\text{def}}{\Leftrightarrow}$ there exist $i < j$ s.t. $s_i \leqslant s_j$

Proof. $\Rightarrow$: the infinite run contains an increasing pair
$\Leftarrow$: good finite run $s_0 \xrightarrow{*} s_i \xrightarrow{+} s_j$ can be extended by simulating $s_i \xrightarrow{+} s_j$ from above: $s_j \xrightarrow{+} s_{2j-i}$, then $s_{2j-i} \xrightarrow{+} s_{3j-2i}$, etc.
Corollary. One may decide Termination by enumerating all finite runs from s_{init} until a good sequence is encountered.
If all runs are bad, the enumeration will eventually exhaust them

NB: This requires some minimal effectiveness assumptions on the WSTS, e.g., that the ordering is decidable

Algorithm extends and allows deciding inevitability, finiteness, and regular simulation

DECIDING SAFETY (HERE: COVERABILITY)

Coverability is the question, given $\mathcal{S} = (S, \to, \ldots)$, some initial s_{init} and target t, whether $\mathcal{S}$ has a run $s_{init} \to s_1 \to s_2 \cdots \to s_n$ with $s_n \geqslant t$.

This is equivalent to having a pseudorun $s_{init}, s_1, \ldots, s_n$ with $s_n \geqslant t$, where a pseudorun is a sequence $s_0, s_1, \ldots$ such that for all $i > 0$, there is a step $s_{i-1} \to t_i$ with $t_i \geqslant s_i$.

Picture $\quad s_0 \to t_1 \geqslant s_1 \to t_2 \geqslant s_2 \to \cdots \geqslant \cdots \to t_n \geqslant s_n$

Def. a pseudorun $s_0, \ldots, s_n$ is minimal if for all $0 \leqslant i < n$, s_i is a minimal pseudo predecessor of s_{i+1}.

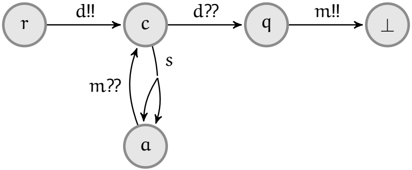
Lem. [Finite Witnesses for Covering]
$\mathcal{S}$ has a pseudorun $s_{init}, \ldots, s_n$ covering t **iff** it has a minimal pseudorun $s_0, s_1, .., t$ from some $s_0 \leqslant s_{init}$ s.t. $t, s_{n-1}, .., s_1, s_0$ is bad

$\Rightarrow$ one decides Coverability by enumerating all minimal pseudoruns ending in t that are bad sequences

Part 3a

Assessing Complexity: Upper Bounds

Broadcast Protocols Take Their Time



"Doubling" run: $\{c^{n}, q, (\perp^*)\} \xrightarrow{s^n} \{a^{2n}, q, (\perp^*)\} \xrightarrow{m} \{c^{2n}, (\perp^*)\}$

Building up: $\{c^{2^0}, q^{n}, r\} \xrightarrow{s^{2^0} m} \{c^{2^1}, q^{n-1}, r\} \xrightarrow{s^{2^1} m} \{c^{2^2}, q^{n-2}, r\} \to$
$\cdots \to \{c^{2^{n-1}}, q, r\} \xrightarrow{s^{2^{n-1}} m} \{c^{2^n}, r\} \xrightarrow{d} \{c^{2^0}, q^{2^n}\}$

Then: $\{c, q, r^{n}\} \xrightarrow{*} \{c, q^{2^n}, r^{n-1}\} \xrightarrow{*} \{c, q^{\text{tower}(n)}\}$

where $\text{tower}(n) \stackrel{\text{def}}{=} \left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\} n$ times $\Rightarrow$ Runs of terminating systems

may have nonelementary lengths

$\Rightarrow$ Running time of generic algorithm verifying termination is not elementary for broadcast protocols

The Fast-Growing Hierarchy

An ordinal-indexed family $(F_\alpha)_{\alpha \in \mathit{Ord}}$ of functions $\mathbb{N} \to \mathbb{N}$

$$F_0(x) \stackrel{\text{def}}{=} x+1 \qquad F_{\alpha+1}(x) \stackrel{\text{def}}{=} \overbrace{F_\alpha(F_\alpha(\dots F_\alpha}^{x+1}(x)\dots))$$

$$F_\omega(x) \stackrel{\text{def}}{=} F_{x+1}(x)$$

gives $F_1(x) \sim 2x$, $F_2(x) \sim 2^x$, $F_3(x) \sim \text{tower}(x)$ and $F_\omega(x) \sim \textsc{Ackermann}(x)$, the first F_α that is not primitive recursive.

$F_\lambda(x) \stackrel{\text{def}}{=} F_{\lambda_x}(x)$ for λ a limit ordinal with a fundamental sequence $\lambda_0 < \lambda_1 < \lambda_2 < \dots < \lambda$.

E.g. $F_{\omega^2}(x) = F_{\omega\cdot(x+1)}(x) = F_{\omega\cdot x+x+1}(x) = \overbrace{F_{\omega\cdot x+x}(F_{\omega\cdot x+x}(..F_{\omega\cdot x+x}}^{x+1}(x)..))$

$\mathscr{F}_\alpha \stackrel{\text{def}}{=}$ all functions computable in time $F_\alpha^{O(1)}$ (very robust).

Complexity Analysis?

When analyzing the termination algorithm, the main question is "how long can a bad sequence be?"

WQO-theory only says that a bad sequence is finite

One can exhibit arbitrarily long bad sequences. E.g. over $(\mathbb{N}^k, \leqslant_\times)$:

— $999, 998, \ldots, 1, 0$

— $(2,2), (2,1), (2,0), (1,999), \ldots, (1,0), (0,999999999), \ldots$

Two tricks: unbounded start element, or unbounded increase in a step

The runs of broadcast protocols don't have unbounded increases, and the starting configuration is the input of our problem

Controlled Bad Sequences

Def. A sequence $x_0, x_1, \ldots$ is controlled $\stackrel{\text{def}}{\Leftrightarrow}$ $|x_i| \leqslant g^i(n_0)$ for all $i = 0, 1, \ldots$

Here the control is the pair (n_0, g) of $n_0 \in \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$.

Fact. For a fixed wqo $(A, \leqslant, |.|)$ and control (n_0, g), there is max length on controlled bad sequences (Kőnig's Lemma again)

Write $L_{g,A}(n_0)$ for this maximum length.

Length Function Theorem for $(\mathbb{N}^k, \leqslant_\times)$ [McAloon 84,Figuiera[2]SS'11]

$L_{g,\mathbb{N}^k}$ is in $\mathscr{F}_{k+m-1}$ when g is in $\mathscr{F}_m$.

APPLICATIONS

Fact. The runs explored by the Termination algorithm are controlled with $|s_{init}|$ and $Succ : \mathbb{N} \to \mathbb{N}$.

Coro. Time/space bound in $\mathscr{F}_{k-1}$ for broadcast protocols with k states, and in $\mathscr{F}_{\omega}$ when k is not fixed.

Fact. The minimal pseudoruns explored by the backward-chaining Coverability algorithm are controlled by $|s_{target}|$ and *Succ*.

Coro. $\cdots$ *same upper bounds* $\cdots$

Thm. [Leroux & Schmitz '19] The algorithm for verification of Vector Addition Systems (or Petri nets) is in $\mathscr{F}_{\omega}$.

This is a general situation:
— WSTS model (or WQO-based algorithm) provides A and g
— WQO-theory provides bounds for $L_{g,A}$
$\Rightarrow$ Complexity upper bounds for WQO-based algorithm

More Length Function Theorems

For finite words with embedding, L_{Σ^*} is in $\mathscr{F}_{\omega^{|\Sigma|-1}}$, and in $\mathscr{F}_{\omega^\omega}$ when alphabet is not fixed [Cichon,..]. Applies e.g. to lossy channel systems.

For sequences over $\mathbb{N}^k$ with embedding, $L_{(\mathbb{N}^k)^*}$ is in $\mathscr{F}_{\omega^{\omega^k}}$, and in $\mathscr{F}_{\omega^{\omega^\omega}}$ when k is not fixed [S.S.]. Applies e.g. to timed-arc Petri nets.

For finite words with priority ordering, L_{Σ^*} is in $\mathscr{F}_{\varepsilon_0}$. Applies e.g. to priority channel systems and higher-order LCS.

Bottom line: one can provide definite complexity upper bounds for WQO-based algorithms

Some research goals: more varied/complex wqos (powerset, restricted families of graphs, ..) & analysis of complex algorithms

Part 3b

Assessing Complexity: Lower Bounds

What About Lower Bounds?

Q. Are the upper bounds for Termination and Coverability optimal?

In the case of broadcast protocols:

The upper bound is tight for the algorithms we presented

But there may exist better algorithms (as with VASS, e.g.)

One can prove that the Termination and Coverability problems are F_ω-hard, hence F_ω-complete, for broadcast protocols [Urquhart'99,..]

and F_{ω^ω}-complete for lossy channel systems [ChambartS'08], $F_{\omega^{\omega^\omega}}$-complete for timed-arc Petri nets [HaddadSS'12], F_{ϵ_0}-complete for priority channel systems [HaaseSS'13]

These results/characterizations have applications outside verification: WSTS models are often used for decidability (or hardness) of problems in logic.

Proving F_α-Hardness

The four hardness results we just mentioned have all been proved using the same techniques:

One shows how the WSTS model can weakly compute F_α and its inverse F_α^{-1}. (Recall: broadcast protocol computing tower function)

Encode initial ordinals in $(S, \leqslant)$ & implement Hardy computations in $\mathcal{S}$.

Hardy computations: $(\alpha+1, x) \mapsto (\alpha, x+1)$ and $(\lambda, x) \mapsto (\lambda_x, x)$.

Main technical issue: robustness

— One easily guarantee $s \leqslant t \Rightarrow \alpha(s) \leqslant \alpha(t)$ but this does not guarantee $F_{\alpha(s)}(x) \leqslant F_{\alpha(t)}(x)$ required for weak computation of F_α.

— We need $s \leqslant t \Rightarrow \alpha(s) \sqsubseteq \alpha(t)$, using an ad-hoc stronger relation $\alpha \sqsubseteq \beta$ that entails $F_\alpha(x) \leqslant F_\beta(x)$.

Conclusion & Executive Summary

- Automated Program Verification is not just a dream, or just a theoretical concept.
- Programs with well-quasi-ordered state space have decidable verification problems.
- The complexity of these problems can often be measured precisely. A good guide here is given by the maximal order type of the state space.
- These results and techniques opened a new section in the Complexity Zoo, see “Complexity Hierarchies Beyond Elementary” [Schmitz ’16].
- Many extensions and developments I did not mention: forward algorithms and topological completions of WQO, etc.

Thank you!

Any questions?

(except *Did I miss the part where you mentioned BQOs?!*)