Algorithmic Aspects of WQO (Well-Quasi-Ordering) Theory

Part II: Algorithmic Applications of WQOs

Philippe Schnoebelen

LSV, CNRS & ENS Cachan

Chennai Mathematical Institute, Jan. 2017

Based on joint work with Sylvain Schmitz, Prateek Karandikar, K. Narayan Kumar, Alain Finkel, ..

Lecture notes & exercises available via `www.lsv.ens-cachan.fr/~phs`

If you missed Part I

$(X, \leqslant)$ is a well-quasi-ordering (a wqo) if any infinite sequence $x_0, x_1, x_2 \ldots$ over X contains an increasing pair $x_i \leqslant x_j$ (for some $i < j$)

Examples.

1. $(\mathbb{N}^k, \leqslant_\times)$ is a wqo (Dickson's Lemma)
   where, e.g., $(3,2,1) \leqslant_\times (5,2,2)$ but $(1,2,3) \not\leqslant_\times (5,2,2)$
2. $(\Sigma^*, \leqslant_*)$ is a wqo (Higman's Lemma)
   where, e.g., $abc \leqslant_* bacbc$ but $cba \not\leqslant_* bacbc$

Objectives for today's course:

- See algorithms that rely on wqos: verification of WSTS's
- Reduce complexity analysis to bounds on bad sequences

IF YOU MISSED PART I

$(X, \leqslant)$ is a well-quasi-ordering (a wqo) if any infinite sequence $x_0, x_1, x_2 \ldots$ over X contains an increasing pair $x_i \leqslant x_j$ (for some $i < j$)

Examples.

1. $(\mathbb{N}^k, \leqslant_\times)$ is a wqo (Dickson's Lemma)
   where, e.g., $(3,2,1) \leqslant_\times (5,2,2)$ but $(1,2,3) \not\leqslant_\times (5,2,2)$
2. $(\Sigma^*, \leqslant_*)$ is a wqo (Higman's Lemma)
   where, e.g., $abc \leqslant_* bacbc$ but $cba \not\leqslant_* bacbc$

Objectives for today's course:

- See algorithms that rely on wqos: verification of WSTS's
- Reduce complexity analysis to bounds on bad sequences

Outline for Part II

- Well-structured transition systems (WSTS's)
- Deciding Termination
- Deciding Coverability
- (in lecture notes only:) other wqo-based algorithms: other termination proofs, relevance logic, Karp-Miller trees, ..

All of these are actual examples of algorithms that terminate thanks to wqo-theoretical arguments

Question for Part III. terminate in how many steps exactly?

Outline for Part II

- Well-structured transition systems (WSTS's)
- Deciding Termination
- Deciding Coverability
- (in lecture notes only:) other wqo-based algorithms: other termination proofs, relevance logic, Karp-Miller trees, ..

All of these are actual examples of algorithms that terminate thanks to wqo-theoretical arguments

Question for Part III. terminate in how many steps exactly?

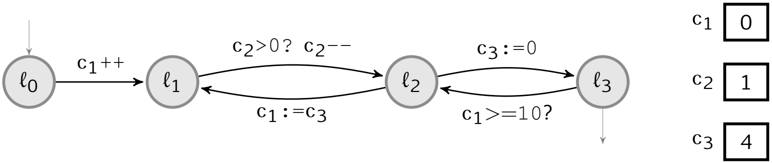
WSTS: Well-structured transition systems

In program verification, wqo's appear prominently under the guise of WSTS.

Def. A WSTS is a system $(S, \to, \leqslant)$ where

1. $(S, \to)$ with $\to \subseteq S \times S$ is a transition system
2. the set of states $(S, \leqslant)$ is wqo, and
3. the transition relation is compatible with the ordering (also called "monotonic"): $s \to t$ and $s \leqslant s'$ imply $s' \to t'$ for some $t' \geqslant t$

Some WSTS's: Monotonic Counter Machines



A run of M: $(\ell_0,0,1,4) \to (\ell_1,1,1,4) \to (\ell_2,1,0,4) \to (\ell_3,1,0,0)$

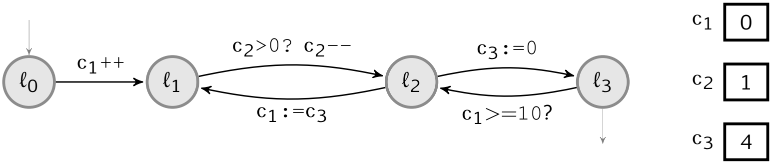
Ordering states: $(\ell_1,0,0,0) \leqslant (\ell_1,0,1,2)$ but $(\ell_1,0,0,0) \not\leqslant (\ell_2,0,1,2)$.
This is wqo as a product of wqo's: $(Loc,=) \times (\mathbb{N}^3,\leqslant_\times)$

Compatibility: easily checked when guards are upward-closed and assignments are monotonic functions of the variables.

NB. Other updates can be considered as long as they are monotonic.
Extending guards require using a finer ordering.

Question. How does this compare to Minsky (counter) machines?

SOME WSTS'S: MONOTONIC COUNTER MACHINES



A run of M: $(\ell_0,0,1,4) \to (\ell_1,1,1,4) \to (\ell_2,1,0,4) \to (\ell_3,1,0,0)$

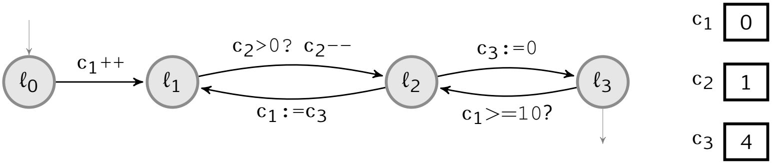
Ordering states: $(\ell_1,0,0,0) \leqslant (\ell_1,0,1,2)$ but $(\ell_1,0,0,0) \not\leqslant (\ell_2,0,1,2)$.
This is wqo as a product of wqo's: $(Loc,=) \times (\mathbb{N}^3,\leqslant_\times)$

Compatibility: easily checked when guards are upward-closed and assignments are monotonic functions of the variables.

NB. Other updates can be considered as long as they are monotonic.
Extending guards require using a finer ordering.

Question. How does this compare to Minsky (counter) machines?

SOME WSTS'S: MONOTONIC COUNTER MACHINES



A run of M: $(\ell_0,0,1,4) \to (\ell_1,1,1,4) \to (\ell_2,1,0,4) \to (\ell_3,1,0,0)$

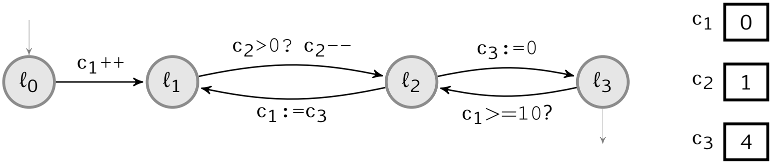
Ordering states: $(\ell_1,0,0,0) \leqslant (\ell_1,0,1,2)$ but $(\ell_1,0,0,0) \not\leqslant (\ell_2,0,1,2)$.
This is wqo as a product of wqo's: $(Loc,=) \times (\mathbb{N}^3,\leqslant_\times)$

Compatibility: easily checked when guards are upward-closed and assignments are monotonic functions of the variables.

NB. Other updates can be considered as long as they are monotonic. Extending guards require using a finer ordering.

Question. How does this compare to Minsky (counter) machines?

Some WSTS's: Monotonic Counter Machines



A run of M: $(\ell_0,0,1,4) \to (\ell_1,1,1,4) \to (\ell_2,1,0,4) \to (\ell_3,1,0,0)$

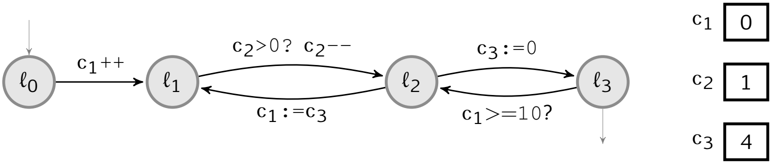
Ordering states: $(\ell_1,0,0,0) \leqslant (\ell_1,0,1,2)$ but $(\ell_1,0,0,0) \nleqslant (\ell_2,0,1,2)$.
This is wqo as a product of wqo's: $(Loc,=) \times (\mathbb{N}^3,\leqslant_\times)$

Compatibility: easily checked when guards are upward-closed and assignments are monotonic functions of the variables.

NB. Other updates can be considered as long as they are monotonic.
Extending guards require using a finer ordering.

Question. How does this compare to Minsky (counter) machines?

Some WSTS's: Monotonic Counter Machines



A run of M: $(\ell_0,0,1,4) \to (\ell_1,1,1,4) \to (\ell_2,1,0,4) \to (\ell_3,1,0,0)$

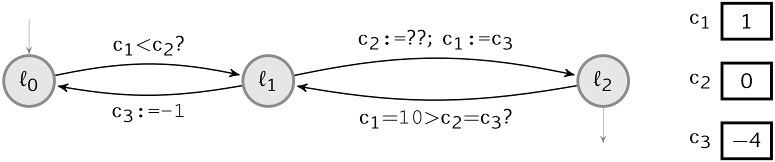
Ordering states: $(\ell_1,0,0,0) \leqslant (\ell_1,0,1,2)$ but $(\ell_1,0,0,0) \not\leqslant (\ell_2,0,1,2)$.
This is wqo as a product of wqo's: $(Loc,=) \times (\mathbb{N}^3,\leqslant_\times)$

Compatibility: easily checked when guards are upward-closed and assignments are monotonic functions of the variables.

NB. Other updates can be considered as long as they are monotonic.
Extending guards require using a finer ordering.

Question. How does this compare to Minsky (counter) machines?

SOME WSTS'S: RELATIONAL AUTOMATA



Guards: comparisons between counters and constants

Updates: assignments with counter values, constants, & "??"

One does not use $\leqslant_\times$ to compare states!! Rather

$$(a_1,\ldots,a_k)\leqslant_{\text{sparse}}(b_1,\ldots,b_k)$$

$$\overset{\text{def}}{\Leftrightarrow} \forall i,j=1,\ldots,k:\big(a_i\leqslant a_j \text{ iff } b_i\leqslant b_j\big)\wedge\big(|a_i-a_j|\leqslant|b_i-b_j|\big)\,.$$

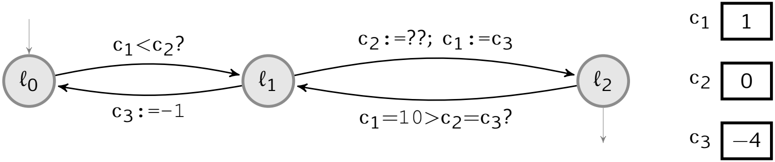
Fact. $(\mathbb{Z}^k,\leqslant_{\text{sparse}})$ is wqo

Compatibility: We use

$$(\ell,a_1,\ldots,a_k)\leqslant(\ell',b_1,\ldots,b_k) \overset{\text{def}}{\Leftrightarrow}$$

$$\ell=\ell'\wedge(a_1,\ldots,a_k,-1,10)\leqslant_{\text{sparse}}(b_1,\ldots,b_k,-1,10)\,.$$

Some WSTS's: Relational automata



Guards: comparisons between counters and constants

Updates: assignments with counter values, constants, & "??"

One does not use $\leqslant_\times$ to compare states!! Rather

$$(a_1,\ldots,a_k) \leqslant_{\text{sparse}} (b_1,\ldots,b_k)$$

$$\overset{\text{def}}{\Leftrightarrow} \forall i,j = 1,\ldots,k : \big(a_i \leqslant a_j \text{ iff } b_i \leqslant b_j\big) \land \big(|a_i - a_j| \leqslant |b_i - b_j|\big)\,.$$

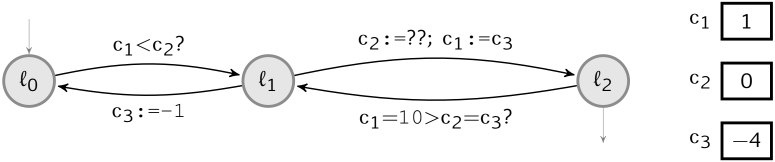
Fact. $(\mathbb{Z}^k, \leqslant_{\text{sparse}})$ is wqo

$$(\ell, a_1,\ldots,a_k) \leqslant (\ell', b_1,\ldots,b_k) \overset{\text{def}}{\Leftrightarrow}$$

Compatibility: We use

$$\ell = \ell' \land (a_1,\ldots,a_k,-1,10) \leqslant_{\text{sparse}} (b_1,\ldots,b_k,-1,10)\,.$$

SOME WSTS'S: RELATIONAL AUTOMATA



Guards: comparisons between counters and constants

Updates: assignments with counter values, constants, & "??"

One does not use $\leqslant_\times$ to compare states!! Rather

$$(a_1,\dots,a_k) \leqslant_{\text{sparse}} (b_1,\dots,b_k)$$

$$\overset{\text{def}}{\Leftrightarrow} \forall i,j=1,\dots,k : \big(a_i \leqslant a_j \text{ iff } b_i \leqslant b_j\big) \wedge \big(|a_i - a_j| \leqslant |b_i - b_j|\big)\,.$$

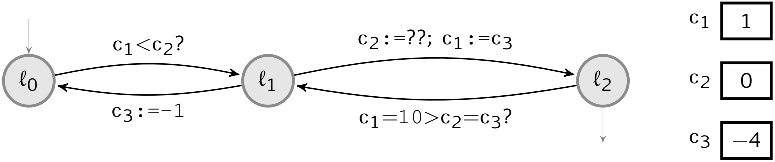
Fact. $(\mathbb{Z}^k, \leqslant_{\text{sparse}})$ is wqo

$$(\ell, a_1,\dots,a_k) \leqslant (\ell', b_1,\dots,b_k) \overset{\text{def}}{\Leftrightarrow}$$

Compatibility: We use

$$\ell = \ell' \wedge (a_1,\dots,a_k,-1,10) \leqslant_{\text{sparse}} (b_1,\dots,b_k,-1,10)\,.$$

SOME WSTS'S: RELATIONAL AUTOMATA



Guards: comparisons between counters and constants
Updates: assignments with counter values, constants, & "??"

One does not use $\leqslant_\times$ to compare states!! Rather

$$(a_1,\ldots,a_k) \leqslant_{\text{sparse}} (b_1,\ldots,b_k)$$
$$\stackrel{\text{def}}{\Leftrightarrow} \forall i,j = 1,\ldots,k : \big(a_i \leqslant a_j \text{ iff } b_i \leqslant b_j\big) \wedge \big(|a_i - a_j| \leqslant |b_i - b_j|\big) .$$

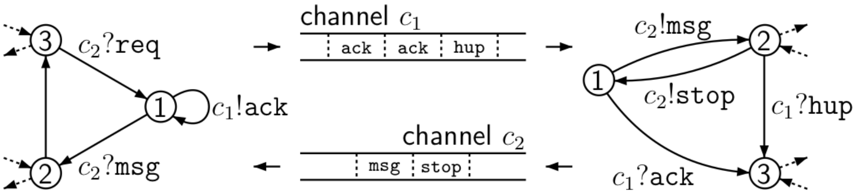
Fact. $(\mathbb{Z}^k, \leqslant_{\text{sparse}})$ is wqo

$$(\ell, a_1,\ldots,a_k) \leqslant (\ell', b_1,\ldots,b_k) \stackrel{\text{def}}{\Leftrightarrow}$$

Compatibility: We use

$$\ell = \ell' \wedge (a_1,\ldots,a_k,-1,10) \leqslant_{\text{sparse}} (b_1,\ldots,b_k,-1,10) .$$

Some WSTS's: LCS / Lossy Channel Systems



A configuration $\sigma = (\ell_1, \ell_2, w_1, w_2)$ with $w_i \in \Sigma^*$.
E.g., $w_1 = \texttt{hup.ack.ack}$.

Reliable steps: $\sigma \rightarrow_{\text{rel}} \rho$ read in front of channels, write at end (FIFO)

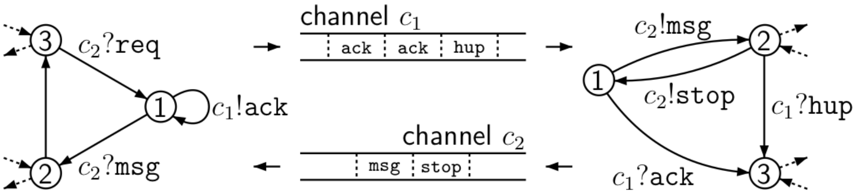
Lossy steps: messages may be lost nondeterministically

$$\sigma \rightarrow \sigma' \overset{\text{def}}{\Leftrightarrow} \sigma \sqsupseteq \rho \rightarrow_{\text{rel}} \rho' \sqsupseteq \sigma' \text{ for some } \rho, \rho'$$

where $(S, \sqsubseteq)$ is the wqo $(Loc_1, =) \times (Loc_2, =) \times (\Sigma^*_{c_1}, \leqslant_*) \times (\Sigma^*_{c_2}, \leqslant_*)$

A model useful for concurrent protocols but also timed automata, metric temporal logic, products of modal logics, ...

SOME WSTS'S: LCS / LOSSY CHANNEL SYSTEMS



A configuration $\sigma = (\ell_1, \ell_2, w_1, w_2)$ with $w_i \in \Sigma^*$.
E.g., $w_1 = \mathtt{hup.ack.ack}$.

Reliable steps: $\sigma \to_{\text{rel}} \rho$ read in front of channels, write at end (FIFO)

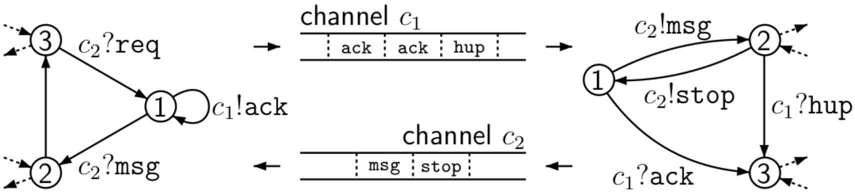
Lossy steps: messages may be lost nondeterministically

$$\sigma \to \sigma' \overset{\text{def}}{\Leftrightarrow} \sigma \sqsupseteq \rho \to_{\text{rel}} \rho' \sqsupseteq \sigma' \text{ for some } \rho, \rho'$$

where $(S, \sqsubseteq)$ is the wqo $(Loc_1, =) \times (Loc_2, =) \times (\Sigma^*_{c_1}, \leqslant_*) \times (\Sigma^*_{c_2}, \leqslant_*)$

A model useful for concurrent protocols but also timed automata, metric temporal logic, products of modal logics, ...

Some WSTS's: LCS / Lossy Channel Systems



A configuration $\sigma = (\ell_1, \ell_2, w_1, w_2)$ with $w_i \in \Sigma^*$.
E.g., $w_1 = \mathtt{hup.ack.ack}$.

Reliable steps: $\sigma \to_{\text{rel}} \rho$ read in front of channels, write at end (FIFO)

Lossy steps: messages may be lost nondeterministically

$$\sigma \to \sigma' \overset{\text{def}}{\Leftrightarrow} \sigma \sqsupseteq \rho \to_{\text{rel}} \rho' \sqsupseteq \sigma' \text{ for some } \rho, \rho'$$

where $(S, \sqsubseteq)$ is the wqo $(Loc_1, =) \times (Loc_2, =) \times (\Sigma^*_{c_1}, \leqslant_*) \times (\Sigma^*_{c_2}, \leqslant_*)$

A model useful for concurrent protocols but also timed automata, metric temporal logic, products of modal logics, ...

Termination

Termination is the question, given a TS $(S, \to, \ldots)$ and a state $s_{\text{init}} \in S$, whether there are no infinite runs starting from s_{init}

Lem. [Finite Witnesses for Infinite Runs]
A WSTS $(S, \to, \leqslant)$ 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. $s_i \leqslant s_j$

Coro. One can decide Termination for a WSTS by enumerating all finite runs from s_{init} and reject when/if a good sequence is found.

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

Termination

Termination is the question, given a TS $(S, \rightarrow, \ldots)$ and a state $s_{\text{init}} \in S$, whether there are no infinite runs starting from s_{init}

Lem. [Finite Witnesses for Infinite Runs]
A WSTS $(S, \rightarrow, \leqslant)$ 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 $\stackrel{\text{def}}{\Leftrightarrow}$ there exist $i < j$ s.t. $s_i \leqslant s_j$

Coro. One can decide Termination for a WSTS by enumerating all finite runs from s_{init} and reject when/if a good sequence is found.

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

Termination

Termination is the question, given a TS $(S, \to, \ldots)$ and a state $s_{\text{init}} \in S$, whether there are no infinite runs starting from s_{init}

Lem. [Finite Witnesses for Infinite Runs]
A WSTS $(S, \to, \leqslant)$ 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.

Coro. One can decide Termination for a WSTS by enumerating all finite runs from s_{init} and reject when/if a good sequence is found.

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

Termination

Termination is the question, given a TS $(S, \rightarrow, \ldots)$ and a state $s_{\text{init}} \in S$, whether there are no infinite runs starting from s_{init}

Lem. [Finite Witnesses for Infinite Runs]
A WSTS $(S, \rightarrow, \leqslant)$ 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.

Coro. Termination is co-r.e.
Since it is also r.e. (for finitely branching systems), it is decidable.

Coro. One can decide Termination for a WSTS by enumerating all finite runs from s_{init} and reject when/if a good sequence is found.

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

Termination

Termination is the question, given a TS $(S, \to, \ldots)$ and a state $s_{\text{init}} \in S$, whether there are no infinite runs starting from s_{init}

Lem. [Finite Witnesses for Infinite Runs]
A WSTS $(S, \to, \leqslant)$ 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. $s_i \leqslant s_j$

Coro. One can decide Termination for a WSTS by enumerating all finite runs from s_{init} and reject when/if a good sequence is found.

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

Coverability (in practice: Safety)

Coverability is the question, given $(S, \to, \dots)$, a state s_{init} and a target state t, whether there is a run $s_{\text{init}} \to s_1 \to s_2 \cdots \to s_n$ with $s_n \geqslant t$.

This is equivalent to having a pseudo-run $s_{\text{init}}, s_1, \dots, s_n$ with $s_n \geqslant t$, where a pseudo-run is a sequence of pseudo-steps $s_{i-1} \to s_i' \geqslant s_i$.

Picture $$\overbrace{s_0 \to s_1' \geqslant s_1}^{\text{1st pseudo-step}} \underbrace{\to s_2' \geqslant s_2}_{\text{2nd pseudo-step}} \to \cdots \geqslant \cdots \overbrace{s_{n-1} \to s_n' \geqslant s_n}^{\text{last pseudo-step}} \geqslant t$$

Lem. [Finite Witnesses for Covering] There is a pseudo-run $s_{\text{init}}, \dots, s_n$ covering t **iff** there is a minimal pseudo-run $s_0 \to\geqslant \cdots \to\geqslant s_{n'} = t$ from some $s_0 \leqslant s_{\text{init}}$ to t such that $s_{n'}, s_{n'-1}, \dots, s_0$ is a bad sequence.

NB. a pseudo-run $s_0, \dots, s_{n'}$ is minimal $\overset{\text{def}}{\Leftrightarrow}$ for all $0 \leqslant i < n'$, s_i is a minimal (pseudo) predecessor of s_{i+1}.

Coro. one can decide Coverability by enumerating all pseudo-runs ending in t (backward-chaining!) that are minimal & bad sequences.

Coverability (in practice: Safety)

Coverability is the question, given $(S, \to, \ldots)$, a state s_{init} and a target state t, whether there is a run $s_{\mathrm{init}} \to s_1 \to s_2 \cdots \to s_n$ with $s_n \geqslant t$.

This is equivalent to having a pseudo-run $s_{\mathrm{init}}, s_1, \ldots, s_n$ with $s_n \geqslant t$, where a pseudo-run is a sequence of pseudo-steps $s_{i-1} \to s'_i \geqslant s_i$.

Picture

$$\overbrace{s_0 \to s'_1 \geqslant s_1}^{\text{1st pseudo-step}} \underbrace{\to s'_2 \geqslant s_2}_{\text{2nd pseudo-step}} \to \cdots \geqslant \cdots \overbrace{s_{n-1} \to s'_n \geqslant s_n}^{\text{last pseudo-step}} \geqslant t$$

Lem. [Finite Witnesses for Covering] There is a pseudo-run $s_{\mathrm{init}}, \ldots, s_n$ covering t **iff** there is a minimal pseudo-run $s_0 \to\geqslant \cdots \to\geqslant s_{n'} = t$ from some $s_0 \leqslant s_{\mathrm{init}}$ to t such that $s_{n'}, s_{n'-1}, \ldots, s_0$ is a bad sequence.

NB. a pseudo-run $s_0, \ldots, s_{n'}$ is minimal $\stackrel{\text{def}}{\Leftrightarrow}$ for all $0 \leqslant i < n'$, s_i is a minimal (pseudo) predecessor of s_{i+1}.

Coro. one can decide Coverability by enumerating all pseudo-runs ending in t (backward-chaining!) that are minimal & bad sequences.

COVERABILITY (IN PRACTICE: SAFETY)

Coverability is the question, given $(S, \to, \ldots)$, a state s_{init} and a target state t, whether there is a run $s_{\text{init}} \to s_1 \to s_2 \cdots \to s_n$ with $s_n \geqslant t$.

This is equivalent to having a pseudo-run $s_{\text{init}}, s_1, \ldots, s_n$ with $s_n \geqslant t$, where a pseudo-run is a sequence of pseudo-steps $s_{i-1} \to s'_i \geqslant s_i$.

Picture
$$\overbrace{s_0 \to s'_1 \geqslant s_1}^{\text{1st pseudo-step}} \!\!\!\!\underbrace{\to s'_2 \geqslant s_2}_{\text{2nd pseudo-step}} \to \cdots \geqslant \cdots \overbrace{s_{n-1} \to s'_n \geqslant s_n}^{\text{last pseudo-step}} \geqslant t$$

Lem. [Finite Witnesses for Covering] There is a pseudo-run $s_{\text{init}}, \ldots, s_n$ covering t **iff** there is a minimal pseudo-run $s_0 \to\geqslant \cdots \to\geqslant s_{n'} = t$ from some $s_0 \leqslant s_{\text{init}}$ to t such that $s_{n'}, s_{n'-1}, \ldots, s_0$ is a bad sequence.

NB. a pseudo-run $s_0, \ldots, s_{n'}$ is minimal $\overset{\text{def}}{\Leftrightarrow}$ for all $0 \leqslant i < n'$, s_i is a minimal (pseudo) predecessor of s_{i+1}.

Coro. one can decide Coverability by enumerating all pseudo-runs ending in t (backward-chaining!) that are minimal & bad sequences.

Coverability (in practice: Safety)

Coverability is the question, given $(S,\to,\ldots)$, a state s_{init} and a target state t, whether there is a run $s_{\text{init}} \to s_1 \to s_2 \cdots \to s_n$ with $s_n \geqslant t$.

This is equivalent to having a pseudo-run $s_{\text{init}}, s_1, \ldots, s_n$ with $s_n \geqslant t$, where a pseudo-run is a sequence of pseudo-steps $s_{i-1} \to s'_i \geqslant s_i$.

Picture
$$\overbrace{s_0 \to s'_1 \geqslant s_1}^{\text{1st pseudo-step}} \to s'_2 \geqslant s_2 \to \cdots \geqslant \cdots \overbrace{s_{n-1} \to s'_n \geqslant s_n}^{\text{last pseudo-step}} \geqslant t$$
$$s_1 \to s'_2 \geqslant s_2 \text{ : 2nd pseudo-step}$$

Lem. [Finite Witnesses for Covering] There is a pseudo-run $s_{\text{init}}, \ldots, s_n$ covering t **iff** there is a minimal pseudo-run $s_0 \to\geqslant \cdots \to\geqslant s_{n'} = t$ from some $s_0 \leqslant s_{\text{init}}$ to t such that $s_{n'}, s_{n'-1}, \ldots, s_0$ is a bad sequence.

NB. a pseudo-run $s_0, \ldots, s_{n'}$ is minimal $\overset{\text{def}}{\Leftrightarrow}$ for all $0 \leqslant i < n'$, s_i is a minimal (pseudo) predecessor of s_{i+1}.

Coro. one can decide Coverability by enumerating all pseudo-runs ending in t (backward-chaining!) that are minimal & bad sequences.

COMPLEXITY ANALYSIS

The two algorithms we have seen guess a finite sequence $s_0, s_1, \ldots, s_\ell$ that is bad (for Coverability) or almost bad (for non-Termination) and check that they are indeed correct witnesses.

We can give a complexity upper bound in $(\text{CO})\text{NTIME}(f(n))$ or $(\text{CO})\text{NSPACE}(f(n))$ if we can bound the size of the sequence —in practice: bound its length ℓ— as a function of the input $(S, \to, \leqslant), s_{\text{init}}, t, ..$

This is the topic for next course . . .

COMPLEXITY ANALYSIS

The two algorithms we have seen guess a finite sequence $s_0, s_1, \ldots, s_\ell$ that is bad (for Coverability) or almost bad (for non-Termination) and check that they are indeed correct witnesses.

We can give a complexity upper bound in $(\text{CO})\text{NTIME}(f(n))$ or $(\text{CO})\text{NSPACE}(f(n))$ if we can bound the size of the sequence —in practice: bound its length ℓ— as a function of the input $(S, \rightarrow, \leqslant), s_{\text{init}}, t, ..$

This is the topic for next course . . .

Complexity analysis

The two algorithms we have seen guess a finite sequence $s_0, s_1, \ldots, s_\ell$ that is bad (for Coverability) or almost bad (for non-Termination) and check that they are indeed correct witnesses.

We can give a complexity upper bound in $(\textsc{co})\textsc{NTime}(f(n))$ or $(\textsc{co})\textsc{NSpace}(f(n))$ if we can bound the size of the sequence —in practice: bound its length ℓ— as a function of the input $(S, \rightarrow, \leqslant), s_{\text{init}}, t, ..$

This is the topic for next course . . .