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
(X, ≤) is a well-quasi-ordering (a wqo) if any infinite sequence x₀, x₁, x₂... over X contains an increasing pair xᵢ ≤ xⱼ (for some i < j)

Examples.
1. (ℕᵏ, ≤ₓ) is a wqo (Dickson’s Lemma)
   where, e.g., (3, 2, 1) ≤ₓ (5, 2, 2) but (1, 2, 3) ≤ₓ (5, 2, 2)

2. (Σ*, ≤*) is a wqo (Higman’s Lemma)
   where, e.g., abc ≤* bacbc but cba ≤* 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

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

Examples.
1. $(\mathbb{N}^k, \preceq_X)$ is a wqo (Dickson’s Lemma)
   where, e.g., $(3, 2, 1) \preceq_X (5, 2, 2)$ but $(1, 2, 3) \npreceq_X (5, 2, 2)$

2. $(\Sigma^*, \preceq_*)$ is a wqo (Higman’s Lemma)
   where, e.g., $abc \preceq_* bacbc$ but $cba \npreceq_* bacbc$

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

**Def.** A WSTS is a system \((S, \rightarrow, \leq)\) where

1. \((S, \rightarrow)\) with \(\rightarrow \subseteq S \times S\) is a transition system
2. the set of states \((S, \leq)\) is wqo, and
3. the transition relation is **compatible with the ordering** (also called “monotonic”): \(s \rightarrow t\) and \(s \leq s’\) imply \(s’ \rightarrow t’\) for some \(t’ \geq t\)
Some WSTS’s: Monotonic Counter Machines

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

Ordering states: \((\ell_1, 0, 0, 0) \preceq (\ell_1, 0, 1, 2)\) but \((\ell_1, 0, 0, 0) \nprec (\ell_2, 0, 1, 2)\).
This is wqo as a product of wqo’s: \((\text{Loc}, =) \times (\mathbb{N}^3, \leq x)\)

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 $M$: $(\ell_0, 0, 1, 4) \rightarrow (\ell_1, 1, 1, 4) \rightarrow (\ell_2, 1, 0, 4) \rightarrow (\ell_3, 1, 0, 0)$

Ordering states: $(\ell_1, 0, 0, 0) \leq (\ell_1, 0, 1, 2)$ but $(\ell_1, 0, 0, 0) \nleq (\ell_2, 0, 1, 2)$. This is wqo as a product of wqo’s: $(\text{Loc}, =) \times (\mathbb{N}^3, \leq_x)$

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 $M$: $(\ell_0, 0, 1, 4) \rightarrow (\ell_1, 1, 1, 4) \rightarrow (\ell_2, 1, 0, 4) \rightarrow (\ell_3, 1, 0, 0)$

**Ordering states:** $(\ell_1, 0, 0, 0) \leq (\ell_1, 0, 1, 2)$ but $(\ell_1, 0, 0, 0) \nleq (\ell_2, 0, 1, 2)$. This is wqo as a product of wqo’s: $(\text{Loc}, =) \times (\mathbb{N}^3, \leq_x)$

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 $M$: $(\ell_0,0,1,4) \rightarrow (\ell_1,1,1,4) \rightarrow (\ell_2,1,0,4) \rightarrow (\ell_3,1,0,0)$

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

**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 $M$: $(\ell_0, 0, 1, 4) \rightarrow (\ell_1, 1, 1, 4) \rightarrow (\ell_2, 1, 0, 4) \rightarrow (\ell_3, 1, 0, 0)$

**Ordering states:** $(\ell_1, 0, 0, 0) \leq (\ell_1, 0, 1, 2)$ but $(\ell_1, 0, 0, 0) \not\leq (\ell_2, 0, 1, 2)$. This is wqo as a product of wqo’s: $(\mathbb{L}oc, =) \times (\mathbb{N}^3, \leq_x)$

**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 \( \leq_x \) to compare states!! Rather

\[
(a_1,\ldots,a_k) \leq_{\text{sparse}} (b_1,\ldots,b_k)
\]

\[
\text{def } \iff \forall i,j = 1,\ldots,k: (a_i \leq a_j \iff b_i \leq b_j) \land (|a_i - a_j| \leq |b_i - b_j|).
\]

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

\[
(l,a_1,\ldots,a_k) \leq (l',b_1,\ldots,b_k) \text{ def } \iff \exists \ell = l' \land (a_1,\ldots,a_k,-1,10) \leq_{\text{sparse}} (b_1,\ldots,b_k,-1,10).
\]

**Compatibility:** We use
**Some WSTS’s: Relational Automata**

**Guards:** comparisons between counters and constants

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

One does not use $\leq$ to compare states!! Rather

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

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

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

$$(\ell, a_1, \ldots, a_k) \leq (\ell', b_1, \ldots, b_k) \iff$$

**Compatibility:** We use

$$\ell = \ell' \land (a_1, \ldots, a_k, -1, 10) \leq_{\text{sparse}} (b_1, \ldots, b_k, -1, 10).$$
** Guards:** comparisons between counters and constants  
** Updates:** assignments with counter values, constants, & “??”

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

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

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

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

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

**Compatibility:** We use

$$\ell = \ell' \land (a_1,\ldots,a_k,-1,10) \leq_{\text{sparse}} (b_1,\ldots,b_k,-1,10).$$
**Guards:** comparisons between counters and constants

**Updates:** assignments with counter values, constants, & “??”

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

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

$$\iff \forall i, j = 1, \ldots, k : (a_i \leq a_j \iff b_i \leq b_j) \land (|a_i - a_j| \leq |b_i - b_j|).$$

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

$$(\ell, a_1, \ldots, a_k) \leq (\ell', b_1, \ldots, b_k) \iff$$

**Compatibility:** We use

$$\ell = \ell' \land (a_1, \ldots, a_k, -1, 10) \leq_{\text{sparse}} (b_1, \ldots, b_k, -1, 10).$$
A **configuration** $\sigma = (\ell_1, \ell_2, w_1, w_2)$ with $w_i \in \Sigma^*$. 
E.g., $w_1 = \text{hup.ack.ack}$.

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

**Lossy steps:** messages may be lost nondeterministically

$$\sigma \rightarrow \sigma' \iff \sigma \sqsubseteq \rho \rightarrow_{\text{rel}} \rho' \sqsubseteq \sigma' \text{ for some } \rho, \rho'$$

where $(S, \sqsubseteq)$ is the wqo $(\text{Loc}_1, =) \times (\text{Loc}_2, =) \times (\Sigma^*_c, \leq_*) \times (\Sigma^*_c, \leq_*)$

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., $w_1 = \text{hup.ack.ack}$.

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

Lossy steps: messages may be lost nondeterministically

$\sigma \rightarrow \sigma' \iff \sigma \sqsubseteq \rho \rightarrow_{\text{rel}} \rho' \sqsubseteq \sigma'$ for some $\rho, \rho'$

where $(S, \sqsubseteq)$ is the wqo $(\text{Loc}_1, =) \times (\text{Loc}_2, =) \times (\Sigma_{c_1}^*, \leq^*) \times (\Sigma_{c_2}^*, \leq^*)$

A model useful for concurrent protocols but also timed automata, metric temporal logic, products of modal logics, ...
A configuration $\sigma = (\ell_1, \ell_2, w_1, w_2)$ with $w_i \in \Sigma^*$.

E.g., $w_1 = \text{hup.ack.ack}$.

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

Lossy steps: messages may be lost nondeterministically

$\sigma \rightarrow \sigma' \iff \sigma \sqsubseteq \rho \rightarrow_{\text{rel}} \rho' \sqsubseteq \sigma'$ for some $\rho, \rho'$

where $(S, \sqsubseteq)$ is the wqo $(\text{Loc}_1, =) \times (\text{Loc}_2, =) \times (\Sigma_{c_1}^*, \leq*) \times (\Sigma_{c_2}^*, \leq*)$

A model useful for concurrent protocols but also timed automata, metric temporal logic, products of modal logics, ...
**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 \(s_{\text{init}}\).

**Lem.** [Finite Witnesses for Infinite Runs]

A WSTS \((S, \rightarrow, \leq)\) has an infinite run from \(s_{\text{init}}\) iff it has a finite run from \(s_{\text{init}}\) that is a good sequence.

**Recall:** \(s_0, s_1, s_2, \ldots, s_n\) is good \(\iff\) there exist \(i < j\) s.t. \(s_i \leq s_j\)

**Coro.** One can decide Termination for a WSTS by enumerating all finite runs from \(s_{\text{init}}\) 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 \(s_{\text{init}}\).

**Lem. [Finite Witnesses for Infinite Runs]**

A WSTS \((S, \rightarrow, \leq)\) has an infinite run from \(s_{\text{init}}\) iff it has a finite run from \(s_{\text{init}}\) that is a good sequence.

**Recall:** \(s_0, s_1, s_2, \ldots, s_n\) is good def there exist \(i < j\) s.t. \(s_i \leq s_j\)

**Coro.** One can decide Termination for a WSTS by enumerating all finite runs from \(s_{\text{init}}\) 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 \(s_{\text{init}}\).

**Lem. [Finite Witnesses for Infinite Runs]**

A WSTS \((S, \rightarrow, \leq)\) has an infinite run from \(s_{\text{init}}\) iff it has a finite run from \(s_{\text{init}}\) that is a good sequence.

**Recall:** \(s_0, s_1, s_2, \ldots, s_n\) is good \(\Longleftrightarrow\) there exist \(i < j\) s.t. \(s_i \leq s_j\)

**Proof.** \(\Rightarrow:\) the infinite run contains an increasing pair

\(\Leftarrow:\) good finite run \(s_0 \stackrel{*}{\rightarrow} s_i \stackrel{+}{\rightarrow} s_j\) can be extended by simulating \(s_i \stackrel{+}{\rightarrow} s_j\) from above: \(s_j \stackrel{+}{\rightarrow} s_{2j-i}\), then \(s_{2j-i} \stackrel{+}{\rightarrow} s_{3j-2i}\), etc.

**Coro.** One can decide Termination for a WSTS by enumerating all finite runs from \(s_{\text{init}}\) 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 \(s_{\text{init}}\).

**Lem.** [Finite Witnesses for Infinite Runs]

A WSTS \((S, \rightarrow, \leq)\) has an infinite run from \(s_{\text{init}}\) iff it has a finite run from \(s_{\text{init}}\) that is a good sequence.

**Recall:** \(s_0, s_1, s_2, \ldots, s_n\) is good \(\overset{\text{def}}{\iff}\) there exist \(i < j\) s.t. \(s_i \leq s_j\)

**Proof.** \(\Rightarrow\): the infinite run contains an increasing pair

\(\Leftarrow\): good finite run \(s_0 \xrightarrow{*} s_i \xrightarrow{\dagger} s_j\) can be extended by simulating \(s_i \xrightarrow{\dagger} s_j\) from above: \(s_j \xrightarrow{\dagger} s_{2j-i}\), then \(s_{2j-i} \xrightarrow{\dagger} s_{3j-2i}\), 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 \(s_{\text{init}}\) 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 $s_{\text{init}}$.

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

**Recall:** $s_0, s_1, s_2, \ldots, s_n$ is good $\iff$ there exist $i < j$ s.t. $s_i \leq s_j$.

**Coro.** One can decide Termination for a WSTS by enumerating all finite runs from $s_{\text{init}}$ 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 \((S, \to, \ldots)\), a state \(s_{\text{init}}\) and a target state \(t\), whether there is a run \(s_{\text{init}} \to s_1 \to s_2 \cdots \to s_n\) with \(s_n \geq t\).

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

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 \(s_0 \to \cdots \to s_{n'} = t\) from some \(s_0 \leq s_{\text{init}}\) 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 \(\text{def}\) for all \(0 \leq i < n'\), \(s_i\) 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 is the question, given \((S, \to, \ldots)\), a state \(s_{\text{init}}\) and a target state \(t\), whether there is a run \(s_{\text{init}} \to s_1 \to s_2 \cdots \to s_n\) with \(s_n \geq t\).

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

**Picture**

\[
\begin{align*}
\text{1st pseudo-step} & \quad s_0 \to s'_1 \geq s_1 \to s'_2 \geq s_2 \to \cdots \geq \cdots \\
\text{2nd pseudo-step} & \\
\text{last pseudo-step} & \quad \cdots \to s_{n-1} \to s'_n \geq s_n \geq t
\end{align*}
\]

**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 \(s_0 \to \geq \cdots \to \geq s_n' = t\) from some \(s_0 \leq s_{\text{init}}\) 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 \(\defeq\) for all \(0 \leq i < n'\), \(s_i\) 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 \((S, \rightarrow, \ldots)\), a state \(s_{\text{init}}\) and a target state \(t\), whether there is a run \(s_{\text{init}} \rightarrow s_1 \rightarrow s_2 \cdots \rightarrow s_n\) with \(s_n \geq t\).

This is equivalent to having a pseudo-run \(s_{\text{init}}, s_1, \ldots, s_n\) with \(s_n \geq t\), where a pseudo-run is a sequence of pseudo-steps \(s_{i-1} \rightarrow s_i' \geq s_i\).

\[
\text{Picture:} \quad s_0 \rightarrow s_1' \geq s_1 \rightarrow s_2' \geq s_2 \rightarrow \cdots \geq \cdots s_{n-1} \rightarrow s_n' \geq s_n \geq t
\]

**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 \(s_0 \rightarrow \geq \cdots \rightarrow \geq s_{n'} = t\) from some \(s_0 \leq s_{\text{init}}\) 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 \(\overset{\text{def}}{\iff} \) for all \(0 \leq i < n'\), \(s_i\) 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 \((S, \rightarrow, \ldots)\), a state \(s_{\text{init}}\) and a target state \(t\), whether there is a run \(s_{\text{init}} \rightarrow s_1 \rightarrow s_2 \cdots \rightarrow s_n\) with \(s_n \geq t\).

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

Picture

\[
\begin{align*}
1\text{st pseudo-step} & \quad s_0 \rightarrow s'_1 \geq s_1 \rightarrow s'_2 \geq s_2 & \rightarrow \cdots \geq \cdots s_{n-1} \rightarrow s'_n \geq s_n \geq t \\
2\text{nd pseudo-step} &
\end{align*}
\]

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 \(s_0 \rightarrow \geq \cdots \rightarrow \geq s_n, = t\) from some \(s_0 \leq s_{\text{init}}\) 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 \(\overset{\text{def}}{=}\) for all \(0 \leq i < n', s_i\) 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 \( \ell \)— as a function of the input \((S, \to, \leq), s_{\text{init}}, t, ..\)

This is the topic for next course …
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 $\text{CO}NTIME(f(n))$ or $\text{CO}NSPACE(f(n))$ if we can bound the size of the sequence —in practice: bound its length $\ell$— as a function of the input $(S, \rightarrow, \leq), s_{\text{init}}, t, \ldots$

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 $\ell$— as a function of the input $(S, \rightarrow, \leq), s_{\text{init}}, t, \ldots$

This is the topic for next course . . .