Termination

Johannes Åman Pohjola
CSIRO’s Data61 (and UNSW)
Term 2 2021
In the last lecture, we introduced message passing and discuss simple non-compositional proof techniques for synchronous message passing. This lecture, we’ll be looking at proof methods for termination (convergence and deadlock freedom) in sequential, shared-variable concurrent, and message-passing concurrent settings.
For programs that do terminate, termination is often the most important liveness property. In addition to the typical cause of non-termination for sequential programs, namely *divergence*, concurrent programs can also deadlock.

termination = convergence + deadlock-freedom

**Definition**

A program is $\phi$-convergent if it cannot diverge (run forever) when started in an initial state satisfying $\phi$. Instead, it must terminate, or become deadlocked.

To prove convergence, we prove that there is a *bound* on the remaining computation steps from any state that the program reaches.
Termination

For programs that do terminate, termination is often the most important liveness property. In addition to the typical cause of non-termination for sequential programs, namely *divergence*, concurrent programs can also deadlock.

**termination = convergence + deadlock-freedom**

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>A program is <em>φ-convergent</em> if it cannot diverge (run forever) when started in an initial state satisfying <em>φ</em>. Instead, it must terminate, or become deadlocked.</td>
</tr>
</tbody>
</table>

To prove convergence, we prove that there is a *bound* on the remaining computation steps from any state that the program reaches.

[Is this yet another excuse for maths?]
## Termination

<table>
<thead>
<tr>
<th>Algorithm 2.1:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int x</code></td>
</tr>
</tbody>
</table>

p1: `while (x > 0) do`
p2: `x ← x - 1`

### Question

This program is $(0 \leq x)$-convergent. Why?
## Termination

### Algorithm 2.2:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p1:</td>
<td><strong>while</strong> ((x &gt; 0)) <strong>do</strong></td>
</tr>
<tr>
<td>p2:</td>
<td>(x \leftarrow x - 1)</td>
</tr>
</tbody>
</table>

### Question

This program is \((0 \leq x)\)-convergent. Why?

Is it \(T\)-convergent?
Termination

Algorithm 2.3:

```
int x

p1: while (x < 500) do
p2: x ← x + 1
```

Question

Is this program $\phi$-convergent? If so, why and for which $\phi$?
## Termination

### Algorithm 2.4:

<table>
<thead>
<tr>
<th></th>
<th>int $x$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>while</strong> ($x &gt; 0$) <strong>do</strong></td>
<td>$x \leftarrow x - 1$</td>
</tr>
<tr>
<td><strong>while</strong> ($x &lt; 500$) <strong>do</strong></td>
<td>$x \leftarrow x + 1$</td>
</tr>
</tbody>
</table>

### Question

Is *this* program $\phi$-convergent? If so, why and for which $\phi$?
Ordered and Wellfounded Sets

The bound condition is formalised by the concept of a wellfounded set. Recall that, on a set $W$, the binary relation $\prec \subseteq W^2$ is a (strict) partial order, if it is

- irreflexive ($a \not\prec a$),
- asymmetric ($a \prec b \implies b \not\prec a$), and
- transitive ($a \prec b \land b \prec c \implies a \prec c$).

**Definition**

Partially ordered set $(W, \prec)$ is wellfounded if every descending sequence $\langle w_0 \succ w_1 \succ \ldots \rangle$ in $(W, \prec)$ is finite.

**Note**

Realise that infinite ascending sequences are not ruled out.
WFOs

Example (Wellfounded Orders)

\((\mathbb{N}, <)\) is wellfounded.
Example (Wellfounded Orders)

\((\mathbb{N}, \lt)\) is wellfounded. \((\mathbb{N}, \gt)\) and \((\mathbb{Z}, \lt)\) are not wellfounded.
WFOs

Example (Wellfounded Orders)

\((\mathbb{N}, <)\) is wellfounded. \((\mathbb{N}, >)\) and \((\mathbb{Z}, <)\) are not wellfounded.

**Lexicographical order:** Given two wellfounded sets, \((W_1, \prec_1)\) and \((W_2, \prec_2)\), also \((W_1 \times W_2, \prec_{lex})\) with

\[(m_1, n_1) \prec_{lex} (m_2, n_2) \iff (m_1 \prec_1 m_2) \lor ((m_1 = m_2) \land (n_1 \prec_2 n_2))\]

is wellfounded.
Example (Wellfounded Orders)

\((\mathbb{N}, <)\) is wellfounded. \((\mathbb{N}, >)\) and \((\mathbb{Z}, <)\) are not wellfounded.

**Lexicographical order:** Given two wellfounded sets, \((W_1, \prec_1)\) and \((W_2, \prec_2)\), also \((W_1 \times W_2, \prec_{\text{lex}})\) with

\[(m_1, n_1) \prec_{\text{lex}} (m_2, n_2) \text{ iff } (m_1 \prec_1 m_2) \lor ((m_1 = m_2) \land (n_1 \prec_2 n_2))\]

is wellfounded.

**Componentwise order:** Given a family \((W_i, \prec_i)_{1 \leq i \leq n}\) of wellfounded sets, \((W_1 \times \ldots \times W_n, \prec_{\text{cw}})\) with

\[(w_1, \ldots, w_n) \prec_{\text{cw}} (w'_1, \ldots, w'_n) \text{ iff } \exists i. \ w_i \prec_i w'_i \land \forall k \neq i. \ w_k \preceq_k w'_k\]

is wellfounded.
Floyd’s Wellfoundedness Method

Given a transition diagram $P = (L, T, s, t)$ and a precondition $\phi$, we can prove $\phi$-convergence of $P$ by:

1. finding an inductive assertion network $Q : L \rightarrow (\Sigma \rightarrow \mathbb{B})$ and showing that $\models \phi \implies Q_s$;
Floyd’s Wellfoundedness Method

Given a transition diagram $P = (L, T, s, t)$ and a precondition $\phi$, we can prove $\phi$-convergence of $P$ by:

1. finding an inductive assertion network $Q : L \rightarrow (\Sigma \rightarrow \mathbb{B})$ and showing that $\models \phi \implies Q_s$;

2. choosing a wellfounded set $(W, \prec)$ and a network $(\rho_\ell)_{\ell \in L}$ of partially defined ranking functions from $\Sigma$ to $W$ such that:
   - $Q_\ell$ implies that $\rho_\ell$ is defined, and
   - every transition $\ell \xrightarrow{b;f} \ell' \in T$ decreases the ranking function, that is:
     $$\models Q_\ell \land b \implies \rho_\ell > (\rho_{\ell'} \circ f)$$
Example 1

Let $\Sigma = \{x\} \rightarrow \mathbb{R}$. Observe that $(\mathbb{R}, <)$ is not wellfounded.

Transition system $P$

```
x ← x - 1
```

```
x \geq 0
```

```
x \leq 0
```

---

16
Example 1

Let $\Sigma = \{x\} \rightarrow \mathbb{R}$. Observe that $(\mathbb{R}, <)$ is not wellfounded.

Transition system $P$

Assertion network
Example 1

Let $\Sigma = \{x\} \rightarrow \mathbb{R}$. Observe that $(\mathbb{R}, <)$ is not wellfounded.

Transition system $P$

Assertion network

Ranking functions

WFO $(\mathbb{N} \times \mathbb{N}, \prec_{\text{lex}})$
transition $s \xrightarrow{x>0} \ell$:

$$
\models \top \land x > 0 \implies (\max([x],0),1) >_{\text{lex}} ((\max([x],0),0) \circ \text{id})
$$

$$
\models (\lceil x \rceil,1) >_{\text{lex}} ([x],0) \land (0,1) >_{\text{lex}} (0,0)
$$

transition $\ell \xleftarrow{x-1} s$:

$$
\models x > 0 \land \top \implies (\max([x],0),0) >_{\text{lex}} ((\max([x],0),1) \circ \lceil x \leftarrow x-1 \rceil)
$$

$$
\models x > 0 \implies \lceil x \rceil > \lceil x - 1 \rceil \geq 0
$$

transition $s \xrightarrow{x\leq 0} t$:

$$
\models \top \land x \leq 0 \implies (\max([x],0),1) >_{\text{lex}} (0,0)
$$

$$
\models (0,1) >_{\text{lex}} (0,0)
$$

... shows that $P$ is $\top$-convergent.
Soundness & Completeness

Theorem

Floyd’s method is sound, that is, it indeed establishes $\phi$-convergence.
Theorem

Floyd’s method is semantically complete, that is, if \( P \) is \( \phi \)-convergent, then there exist assertion and ranking function networks satisfying the verification conditions for proving convergence.

Note

Recall that one might have to add auxiliary variables to the transition system to be able to express assertions. Without them, the method is not complete!

“semantically” means that we do not care about in what language to express the assertions and ranking functions. You may call this cheating.
Simplifying the Method

We can base convergence proofs on ranking functions only. Although this results in a superficially simpler method, applying it is by no means simpler than Floyd’s.

Given a transition diagram $P = (L, T, s, t)$ and a precondition $\phi$, we can prove $\phi$-convergence of $P$ by choosing a wellfounded set $(W, \prec)$ and a network $(\rho_{\ell})_{\ell \in L}$ of partially defined ranking functions from $\Sigma$ to $W$ such that:

1. For all $\sigma \in \Sigma$, if $\sigma \models \phi$, then $\rho_s$ is defined, and

2. every transition $\ell \xrightarrow{b;f} \ell' \in T$ decreases the ranking function, that is, if $\sigma \models b$ and $\rho_{\ell}$ is defined, then $\rho_{\ell'}(f(\sigma))$ is defined and $\rho_{\ell}(\sigma) \succ \rho_{\ell'}(f(\sigma))$. 
Example 1 again

Transition system

- s
- t
- l

x ← x - 1

x ≤ 0

x > 0

0 ≤ x ≤ 0

(0, 0) → (max(⌈x⌉, 0), 1)

Ranking functions only defined for x > 0!!

23
Example 1 again

Transition system

Ranking functions

Termination

Deadlock
Example 1 again

Transition system

Ranking functions

Termination

Deadlock
Shared Variables

Question

How can we extend Floyd’s method for proving $\phi$-convergence to shared-variable concurrent programs $P = P_1 \parallel \ldots \parallel P_n$?
Shared Variables

Question
How can we extend Floyd’s method for proving $\phi$-convergence to shared-variable concurrent programs $P = P_1 \parallel \ldots \parallel P_n$?

Answer (simplistic): Construct product transition system, use Floyd’s method on that.
Question
How can we extend Floyd’s method for proving $\phi$-convergence to shared-variable concurrent programs $P = P_1 \parallel \ldots \parallel P_n$?

Answer (simplistic): Construct product transition system, use Floyd’s method on that. This leads to the usual problem with exponentially growing numbers of locations, ranking functions, and thus verification conditions.
Question
How can we extend Floyd’s method for proving $\phi$-convergence to shared-variable concurrent programs $P = P_1 \parallel \ldots \parallel P_n$?

Answer (simplistic): Construct product transition system, use Floyd’s method on that. This leads to the usual problem with exponentially growing numbers of locations, ranking functions, and thus verification conditions.

Answer (better): Find a proof principle relating to Floyd’s method as the Owicki/Gries method relates to the inductive assertion method applied to the product transition system (parallel composition as defined in lecture 4).
Local Method for Proving $\phi$-Convergence

Suppose that for each $P_i = (L_i, T_i, s_i, t_i)$ we’ve found a local assertion network $(Q_\ell)_{\ell \in L_i}$, a wellfounded set $(W_i, \prec_i)$, and a network $(\rho_\ell)_{\ell \in L_i}$ of partial ranking functions. (As usual, we assume that the state transformations have been augmented with assignments to auxiliary variables if that is needed.)
1. Prove that the assertions and ranking functions are \textit{locally consistent}, i.e., that $\rho_{\ell}$ is defined whenever $Q_{\ell}$ is true.

2. Prove \textit{local correctness} of every $P_i$, i.e., for $\ell \xrightarrow{b;f} \ell' \in T_i$:

\[
\begin{align*}
    \models Q_{\ell} \land b & \implies Q_{\ell'} \circ f \\
    \models Q_{\ell} \land b & \implies \rho_{\ell} \succ_i (\rho_{\ell'} \circ f)
\end{align*}
\]

3. Prove \textit{interference freedom} for both local networks, i.e., for $\ell \xrightarrow{b;f} \ell' \in T_i$ and $\ell'' \in L_k$, for $k \neq i$:

\[
\begin{align*}
    \models Q_{\ell} \land Q_{\ell''} \land b & \implies Q_{\ell''} \circ f \\
    \models Q_{\ell} \land Q_{\ell''} \land b & \implies \rho_{\ell''} \succeq_k (\rho_{\ell''} \circ f)
\end{align*}
\]

4. Prove $\models \phi \implies \bigwedge_i Q_{s_i}$. 
Example 2

Let $\Sigma = \{x\} \rightarrow \mathbb{N}$. Again, show $\top$-convergence.
Example 2

Let $\Sigma = \{x\} \rightarrow \mathbb{N}$. Again, show $\top$-convergence.

$P_1$: $s_1 \xrightarrow{x > 0; x \leftarrow x - 1} \ell_1 \xrightarrow{x \leq 0} t_1 \xrightarrow{x = 0} \ell_1$

$P_2$: $s_2 \xrightarrow{x \leftarrow 0} t_2 \xrightarrow{x = 0}$

The resulting 8 + 9 proof obligations are easily checked.
Example 2

Let $\Sigma = \{x\} \rightarrow \mathbb{N}$. Again, show $\top$-convergence.

Let $\Sigma = \{x\} \rightarrow \mathbb{N}$. Again, show $\top$-convergence.
Example 2

Let $\Sigma = [\{x\} \rightarrow \mathbb{N}]$. Again, show $\top$-convergence.

The resulting $8 + 9$ proof obligations are easily checked.
Soundness & Completeness

Theorem

The local method is again sound and semantically complete (with auxiliary variables).

Again, we could “simplify” the method by omitting the assertion network. This requires to carefully define the respective domains of the ranking functions — in fact, one is typically forced to establish that the domains of the ranking functions form an inductive assertion network.
Soundness & Completeness

Theorem

The local method is again sound and semantically complete (with auxiliary variables).

Again, we could “simplify” the method by omitting the assertion network. This requires to carefully define the respective domains of the ranking functions — in fact, one is typically forced to establish that the domains of the ranking functions form an inductive assertion network.
So, why bother?
Convergence à la AFR I

To prove that a synchronous transition diagram \( P = P_1 \parallel \ldots \parallel P_n \) (where the \( P_i = (L_i, T_i, s_i, t_i) \) with the usual restrictions) is \( \phi \)-convergent, omit the last point from the AFR method and then choose WFO’s \((W_i, \prec_i)\) and networks \((\rho_\ell)_{\ell \in L_i}\) of local ranking functions only involving \( P_i \)'s variables and prove that\(^1\)

1. both networks are locally consistent: for all states \( \sigma \)

\[
\sigma \models Q_\ell \implies \rho_\ell(\sigma) \in W_i.
\]

2. for all internal \( \ell \xrightarrow{b;f} \ell' \in T_i \):

\[
\models Q_\ell \land b \implies \rho_\ell \succ_i (\rho_{\ell'} \circ f)
\]
Convergence à la AFR II

3 local ranking functions cooperate, namely, for every matching pair

\[ \ell_1 \xrightarrow{b;C \leftarrow e;f} \ell_2 \in L_i \text{ and } \ell'_1 \xrightarrow{b';C \Rightarrow x;f'} \ell'_2 \in L_k, \text{ with } i \neq k \]

show:

\[ I \land Q_{\ell_1} \land Q_{\ell'_1} \land b \land b' \implies ((\rho_{\ell_1}, \rho_{\ell'_1}) >_{cw} (\rho_{\ell_2} \circ g, \rho_{\ell'_2} \circ g)), \]

where \( g = f \circ f' \circ [x \leftarrow e] \).

\(^1\)In fact, the first two are the same as for Owicki/Gries.
Example 4

Let $\Sigma = \{x, y\} \rightarrow \mathbb{R}$. Precondition: $y \in \mathbb{N}$.

$P_1$: $x > 0; x \leftarrow x - 1$

$s_1 \xrightarrow{C \Rightarrow x} \ell_1 \xrightarrow{x \leq 0} t_1

P_2$: $C \leftarrow y$

$s_2 \xrightarrow{C \leftarrow y} t_2
Example 4

Let \( \Sigma = \{x, y\} \rightarrow \mathbb{R} \). Precondition: \( y \in \mathbb{N} \).
Example 4

Let $\Sigma = \{x, y\} \rightarrow \mathbb{R}$. Precondition: $y \in \mathbb{N}$.
A non-terminated process is *deadlocked* if it cannot move anymore. In our setting of transition diagrams, there are two distinct causes for deadlock:

**Message deadlock:** The process blocks on a receive (or synchronous send), but no communication partner will ever come around.

**Resource deadlock:** The process blocks is in a state where all outgoing transitions are guarded, but none of the guards will ever become true.
Deadlock-Avoidance by Order

A simple resource acquisition policy can be formulated that precludes resource deadlocks by avoiding cycles in *wait-for-graphs*.

*From [wikipedia]*

[... ] assign a precedence to each resource and force processes to request resources in order of increasing precedence.

This is a common solution in OS and DB.
Deadlock-Avoidance by Resource-Scheduling

Around 1964 Dijkstra described a *Banker’s Algorithm* to overcome a problem he called *deadly embrace*. It requires both the number of processes and their resource needs to be static. It boils down to granting resources only if all resources a process needs can be granted at that time to avoid entering unsafe states in which more than one process holds partial sets of resources.
Deadlock for Transition Diagrams

A transition $\ell \xrightarrow{b;f} \ell'$ is *enabled* in a state $\sigma$ if its boolean condition $b$ is satisfied in $\sigma$. A process is *blocked* at a location $\ell$ if it has not terminated ($\ell \neq t$) and none of its transitions are enabled there. A concurrent program is *deadlocked* if some of its processes are blocked and the remaining ones have terminated. Clearly, deadlock is an undesirable situation. How can we prove *deadlock-freedom*?
Characterisation of Blocking

Let $P = P_1 \parallel \ldots \parallel P_n$, its precondition $\phi$, and assume that for each process $P_i = (L_i, T_i, s_i, t_i)$ of $P$ there is a local assertion network $(Q_\ell)_{\ell \in L_i}$ satisfying all but the last condition ($\models \bigwedge_i Q_{t_i} \Rightarrow \psi$) of the Owicki/Gries method for proving $\{\phi\} P \{\psi\}$. 
Characterisation of Blocking

Let $P = P_1 \parallel \ldots \parallel P_n$, its precondition $\phi$, and assume that for each process $P_i = (L_i, T_i, s_i, t_i)$ of $P$ there is a local assertion network $(Q_\ell)_{\ell \in L_i}$ satisfying all but the last condition ($\models \bigwedge_i Q_{t_i} \implies \psi$) of the Owicki/Gries method for proving $\{\phi\} P \{\psi\}$.

Process $P_i$ can only be blocked in state $\sigma$ at non-final location $\ell \in L_i \setminus \{t_i\}$ from which there are $m$ transitions with boolean conditions $b_1, \ldots, b_m$, respectively, if $\sigma \models \text{CanBlock}_\ell$, where

$$\text{CanBlock}_\ell = Q_\ell \land \neg \bigvee_{k=1}^m b_k.$$
Characterisation of Blocking cont’d

Consequently, using predicates

\[ \text{Blocked}_i = \bigvee_{\ell \in L_i \setminus \{t_i\}} \text{CanBlock}_\ell \]

deadlock can only occur in a state \( \sigma \) if

\[ \sigma \models \bigwedge_{i=1}^{n} (Q_{t_i} \lor \text{Blocked}_i) \land \bigvee_{i=1}^{n} \text{Blocked}_i \]

holds. (Every process has terminated or blocked and at least one is blocked.)
Owicki/Gries Deadlock-Freedom Condition

\[ \models \neg (\bigwedge_{i=1}^{n} (Q_{t_i} \lor \text{Blocked}_i) \land \bigvee_{i=1}^{n} \text{Blocked}_i) \]  

ensures that \( P \) will not deadlock when started in a state satisfying \( \phi \).
Example 3

Prove deadlock freedom of this program:

\[ P_1: \]
\[ s_1 \]
\[ t_1 \]
\[ P_2: \]
\[ s_2 \]
\[ \ell_2 \]
\[ t_2 \]
Example 3

Prove deadlock freedom of this program:

\[ P_1: \quad s_1, t_1 \]
\[ P_2: \quad s_2, \ell_2, t_2 \]
Soundness & Completeness

**Theorem**

The Owicki/Gries method with the last condition replaced by the deadlock-freedom condition is sound and semantically complete for proving deadlock-freedom relative to some precondition $\phi$. 
Deadlock-Freedom for Synchronous Message Passing

An I/O transition can occur iff the guards of both (matching) transitions involved hold. For a global configuration\(^2\) \(\langle \ell; \sigma \rangle\) define

\[\sigma \models \text{live} \ell \iff \begin{cases} \top, & \text{if all local locations are terminal} \\ \text{a transition is enabled in } \langle \ell; \sigma \rangle, & \text{otherwise.} \end{cases}\]

If we can show that every configuration \(\langle \ell; \sigma \rangle\) reachable from an initial global state (satisfying \(\phi\) if we use a precondition) satisfies \(\sigma \models \text{live} \ell\), then we have verified deadlock freedom.

\(^2\)A global configuration is a pair consisting of a state giving values to all variables and a tuple of local locations, one for each diagram.
Deadlock-Freedom à la AFR

For $n \in \{1 \ldots n\}$ let $P_i = (L_i, T_i, s_i, t_i)$ such that the $L_i$ are pairwise disjoint and the processes’ variable sets are pairwise disjoint.

To prove that a synchronous transition diagram $P = P_1 \parallel \ldots \parallel P_n$ is deadlock-free relative to precondition $\phi$:

1. Omit the last point from the AFR method.
2. Verify the **deadlock-freedom condition** for every global label $\langle \ell_1, \ldots, \ell_n \rangle \in L_1 \times \ldots \times L_n$:

   $$\models I \land \bigwedge_i Q_{\ell_i} \implies \text{live}\langle \ell_1, \ldots, \ell_n \rangle.$$ 

Note: This method generates a verification condition for each global location, i.e.,

$$|L_1 \times \ldots \times L_n| = \prod_{i=1}^{n} |L_i|$$ many.
Example 4 cont’d

\[ I = (k_1 = k_2). \]
Example 4 cont’d

\[ I = (k_1 = k_2). \]
Example 4 cont’d

\begin{center}
\begin{tikzpicture}

\node[state] (l1) at (0,0) {$\ell_1$};
\node[state] (l1p) at (2,0) {$\ell_1'$};
\node[state] (s1) at (-2,-2) {$s_1$};
\node[state] (t1) at (2,-2) {$t_1$};
\node[state] (s2) at (0,-4) {$s_2$};
\node[state] (t2) at (2,-4) {$t_2$};

\draw[->] (l1) edge node[above] {$x \in \mathbb{N}$} node[below] {$\land k_1 = 1$} (s1);
\draw[->] (s1) edge node[above] {$C \Rightarrow x \land k_1 \gets 1$} node[below] {$x \leq 0$} (t1);
\draw[->] (t1) edge node[above] {$x > 0; x \gets x - 1$} (l1p);
\draw[->] (l1p) edge node[above] {$x \in \mathbb{N} \land k_1 = 1$} (s2);
\draw[->] (s2) edge node[above] {$C \Leftarrow y$} node[below] {$k_2 \gets 1$} (t2);
\draw[->] (t2) edge node[above] {$k_2 = 0$} (l1);

\end{tikzpicture}
\end{center}

$I = (k_1 = k_2)$. 
Soundness & Completeness

**Theorem**

The methods are once again sound and semantically complete (with auxiliary variables).
Soundness & Completeness

Theorem

The methods are once again sound and semantically complete (with auxiliary variables).

—END—
What Now?

Next week, we have a break!
After the break, we’ll be looking at a compositional proof method for verification, proving properties for asynchronous communication, and, if time on Thursday, we’ll talk about process algebra.
Assignment 1 is out! Read the spec ASAP!