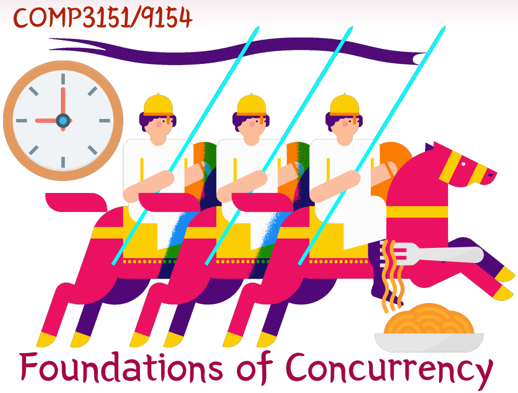



# Termination

Johannes Åman Pohjola
CSE, UNSW

Term 2 2022

## Where we are at

In W5, we introduced message passing and associated proof techniques.

This lecture, we'll be looking at proof methods for termination.

# Termination

For programs that do terminate, termination is often the most important liveness property. There are two causes of non-termination: *divergence* and 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 how many computation steps remaining computation steps from any state that the program reaches.

# Termination

| Algorithm 2.1: |
| --- |
| int x |
| p1: **while** (x > 0) **do** |
| p2: x ← x - 1 |

### Question

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

# Termination

| **Algorithm 2.2:** |
|---|
| int x |
| p1: **while** (x $>$ 0) **do** |
| p2: x $\leftarrow$ x - 1 |

### Question

This program is $(0 \leq x)$-convergent. Why?
Is it $\top$-convergent?

# Termination

| Algorithm 2.3: | |
|---|---|
| int x | |
| p1: | **while** $(x < 500)$ **do** |
| p2: | $x \leftarrow x + 1$ |

## Question

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

# Termination

| Algorithm 2.4: | |
|---|---|
| int x | |
| **while** (x $>$ 0) **do**<br>x $\leftarrow$ x - 1 | **while** (x $<$ 500) **do**<br>x $\leftarrow$ x $+$ 1 |

**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.

# WFOs

## Example (Wellfounded Orders)

$(\mathbb{N}, <)$ is wellfounded. $(\mathbb{N}, >)$ and $(\mathbb{Z}, <)$ 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 $(W_1 \times W_2, <_{\text{lex}})$ with

$$(m_1, n_1) <_{\text{lex}} (m_2, n_2) \text{ iff } (m_1 \prec_1 m_2) \vee ((m_1 = m_2) \wedge (n_1 \prec_2 n_2))$$

is 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 $(W_1 \times W_2, <_{\text{lex}})$ with

$$(m_1, n_1) <_{\text{lex}} (m_2, n_2) \text{ iff } (m_1 \prec_1 m_2) \vee ((m_1 = m_2) \wedge (n_1 \prec_2 n_2))$$

is wellfounded.

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

$$(w_1, \ldots, w_n) <_{\text{cw}} (w_1', \ldots, w_n') \text{ iff } \exists i.\ w_i \prec_i w_i' \wedge \forall k \neq i.\ w_k \preceq_k w_k'$$

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 $Q : L \to (\Sigma \to \mathbb{B})$ 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:

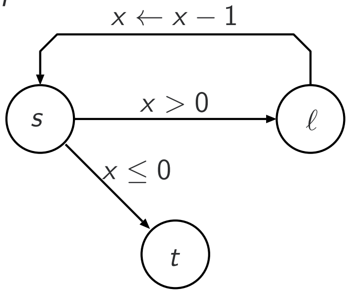
1. finding an inductive assertion network $Q : L \to (\Sigma \to \mathbb{B})$ and showing that $\models \phi \implies Q_s$;
2. choosing a wellfounded set $(W, \prec)$ and a network $(\rho_\ell)_{\ell \in L}$ of partially defined *ranking functions* from $\Sigma$ to $W$ such that:
   - $Q_\ell$ implies that $\rho_\ell$ is defined, and
   - every transition $\ell \xrightarrow{b;f} \ell' \in T$ decreases the ranking function, that is:

$$\models Q_\ell \wedge b \implies \rho_\ell \succ (\rho_{\ell'} \circ f)$$

# Example 1

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

Transition system $P$

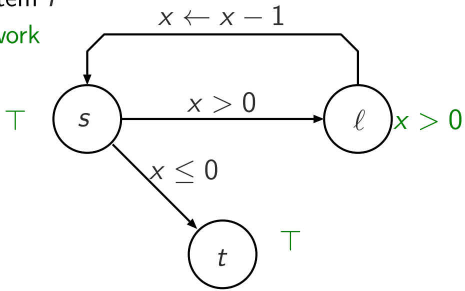


# Example 1

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

Transition system $P$

Assertion network



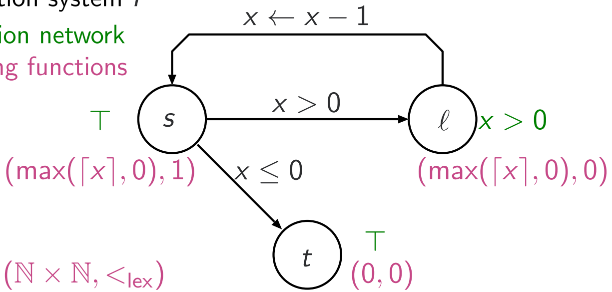
# Example 1

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

Transition system $P$

Assertion network

Ranking functions




WFO $(\mathbb{N} \times \mathbb{N}, <_{\text{lex}})$

transition $s \xrightarrow{x>0} \ell$:

$$\models \top \wedge x > 0 \implies (\max(\lceil x \rceil, 0), 1) >_{\mathsf{lex}} ((\max(\lceil x \rceil, 0), 0) \circ \mathsf{id})$$
$$\Leftarrow \models (\lceil x \rceil, 1) >_{\mathsf{lex}} (\lceil x \rceil, 0) \wedge (0, 1) >_{\mathsf{lex}} (0, 0) \qquad .\text{-}$$

transition $\ell \xrightarrow{x \leftarrow x-1} s$:

$$\models x > 0 \wedge \top \implies (\max(\lceil x \rceil, 0), 0) >_{\mathsf{lex}} ((\max(\lceil x \rceil, 0), 1) \circ [\![x \leftarrow x - 1]\!])$$
$$\Leftarrow \models x > 0 \implies \lceil x \rceil > \lceil x - 1 \rceil \geq 0 \qquad .\text{-}$$

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

$$\models \top \wedge x \leq 0 \implies (\max(\lceil x \rceil, 0), 1) >_{\mathsf{lex}} (0, 0)$$
$$\Leftarrow \models (0, 1) >_{\mathsf{lex}} (0, 0) \qquad .\text{-}$$

. . . 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 what language is used to express the assertions and ranking functions. You may call this cheating.

# 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.

# 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. This leads to the usual exponential blowup problem.

# 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. This leads to the usual exponential blowup problem.

**Answer** (better); find a method that doesn't require constructing the parallel composition explicitly, à la Owicki-Gries.

# 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. (Possibly introducing auxiliary variables.)

1. Prove that the assertions and ranking functions are *locally consistent*, i.e., that $\rho_\ell$ is defined whenever $Q_\ell$ is true.
2. Prove *local correctness* of every $P_i$, i.e., for $\ell \xrightarrow{b;f} \ell' \in T_i$:

$$\models Q_\ell \wedge b \implies Q_{\ell'} \circ f$$
$$\models Q_\ell \wedge b \implies \rho_\ell \succ_i (\rho_{\ell'} \circ f)$$

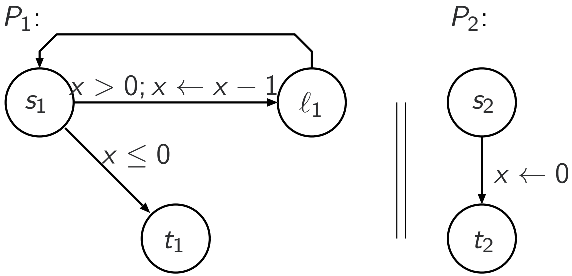
3. Prove *interference freedom* for both local networks, i.e., for $\ell \xrightarrow{b;f} \ell' \in T_i$ and $\ell'' \in L_k$, for $k \neq i$:

$$\models Q_\ell \wedge Q_{\ell''} \wedge b \implies Q_{\ell''} \circ f$$
$$\models Q_\ell \wedge Q_{\ell''} \wedge b \implies \rho_{\ell''} \succeq_k (\rho_{\ell''} \circ f)$$

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

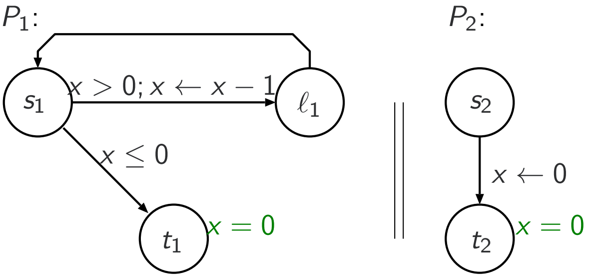
# Example 2

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



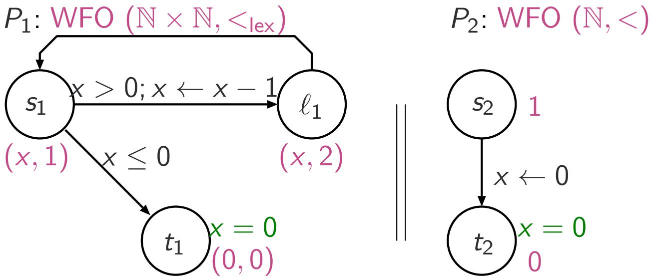
# Example 2

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



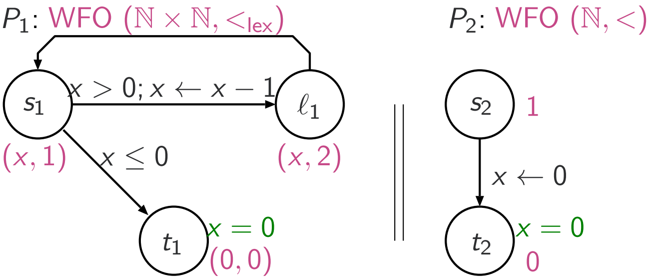
# Example 2

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



# Example 2

Let $\Sigma = [\{x\} \to \mathbb{N}]$. 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).*

# Convergence à la AFR I

To prove that a synchronous transition diagram $P = P_1 \parallel \ldots \parallel P_n$ (where the $P_i = (L_i, T_i, s_i, t_i)$ have the usual restrictions) is $\phi$-convergent, follow the AFR method[1] 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. both networks are locally consistent: for all states $\sigma$

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

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

$$\models Q_\ell \wedge b \implies \rho_\ell \succ_i (\rho_{\ell'} \circ f)$$

# Convergence à la AFR II

3. local ranking functions *cooperate*, namely, for every matching pair $\ell_1 \xrightarrow{b;C\Leftarrow e;f} \ell_2 \in L_i$ and $\ell_1' \xrightarrow{b';C\Rightarrow x;f'} \ell_2' \in L_k$, with $i \neq k$ show:

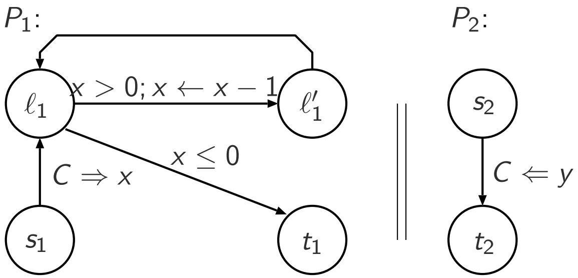
$$\models I \wedge Q_{\ell_1} \wedge Q_{\ell_1'} \wedge b \wedge b' \implies ((\rho_{\ell_1}, \rho_{\ell_1'}) >_{\mathsf{cw}} (\rho_{\ell_2} \circ g, \rho_{\ell_2'} \circ g)) \quad ,$$

where $g = f \circ f' \circ [\![x \leftarrow e]\!]$.

[1] You may ignore the step where we establish the post-condition from the exit state annotations.

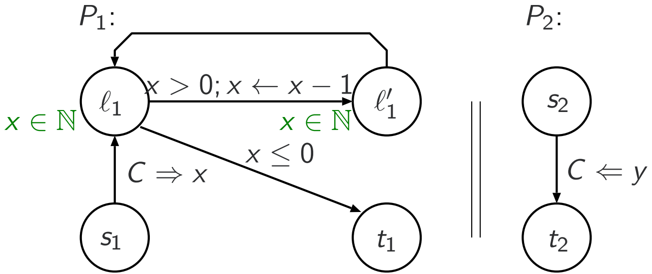
# Example 4

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



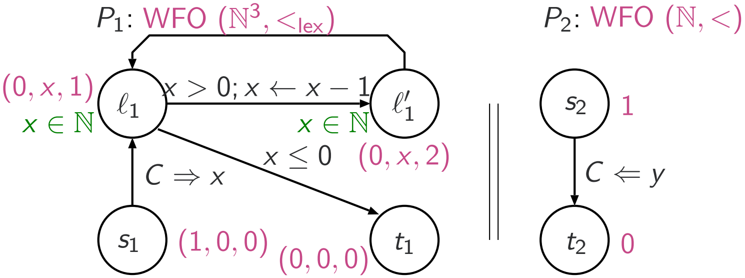
# Example 4

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



# Example 4

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



# Deadlock Classes

A program is *deadlocked* if some of its processes are not terminated, yet none of its processes can do anything. In our setting, there are two causes:

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

**Resource deadlock:** 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 operating systems and databases. (cf. dining philosophers).

# 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 Transtion Diagrams

A transition $\ell \xrightarrow{b;f} \ell'$ is *enabled* in a state $\sigma$ if $\sigma \models b$.

A process is *blocked* in state $\sigma$ at location $\ell$ if:

1. It has not terminated ($\ell \neq t$)
2. None of the transitions from $\ell$ are enabled in $\sigma$.

A concurrent program is *deadlocked* if some of its processes are blocked, and the remaining ones have terminated.

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 $(Q_\ell)_{\ell \in L_i}$ that is inductive, interference free and where the precondition $\phi$ implies the entry location annotations.

# 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 $(Q_\ell)_{\ell \in L_i}$ that is inductive, interference free and where the precondition $\phi$ implies the entry location annotations.

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 guards $b_1, \ldots, b_m$, respectively, if $\sigma \models \mathrm{CanBlock}_\ell$, where

$$\mathrm{CanBlock}_\ell = Q_\ell \wedge \neg \bigvee_{k=1}^{m} b_k \quad .$$

# Characterisation of Blocking cont'd

Consequently, using predicates

$$\mathsf{Blocked}_i = \bigvee_{\ell \in L_i \setminus \{t_i\}} \mathsf{CanBlock}_\ell$$

deadlock can only occur in a state $\sigma$ if

$$\sigma \models \bigwedge_{i=1}^{n} (Q_{t_i} \vee \mathsf{Blocked}_i) \wedge \bigvee_{i=1}^{n} \mathsf{Blocked}_i$$

holds. (Every process has terminated or blocked and at least one is blocked.)

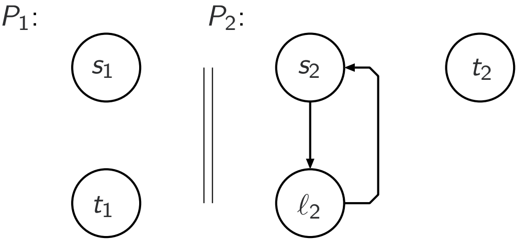
# Owicki/Gries Deadlock-Freedom Condition

$$\models \neg\left(\bigwedge_{i=1}^{n}(Q_{t_i} \vee \mathsf{Blocked}_i) \wedge \bigvee_{i=1}^{n}\mathsf{Blocked}_i\right) \qquad \text{DFC}$$

ensures that $P$ will not deadlock when started in a state satisfying $\phi$.

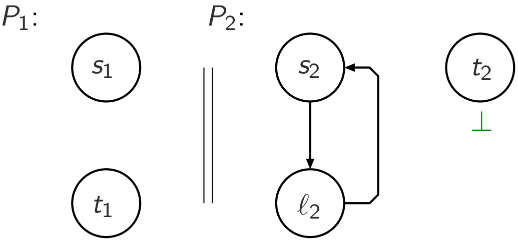
# Example 3

Prove deadlock freedom of this program:



# Example 3

Prove deadlock freedom of this program:



# 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

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

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 $\sigma \models \textbf{live}\, \ell$, 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 the synchronous transition diagram $P$ is deadlock-free, relative to precondition $\phi$:

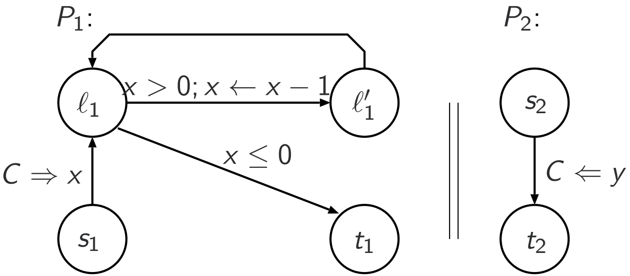
1. Follow the AFR method, but skip the point where the postcondition is established.
2. Verify the *deadlock-freedom condition* for every global label $\langle \ell_1, \ldots, \ell_n \rangle \in L_1 \times \ldots \times L_n$:

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

**Note**

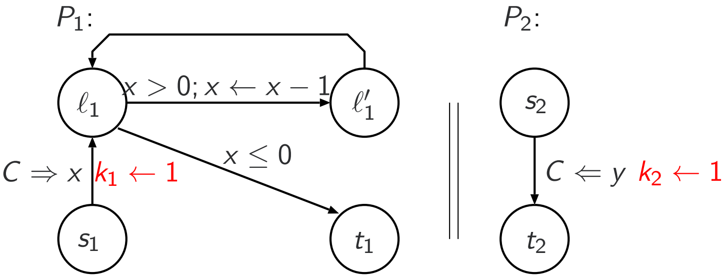
This method generates a verification condition for each global location, i.e., $|L_1 \times \ldots \times L_n| = \prod_{i=1}^{n} |L_i|$ many.

# Example 4 cont'd



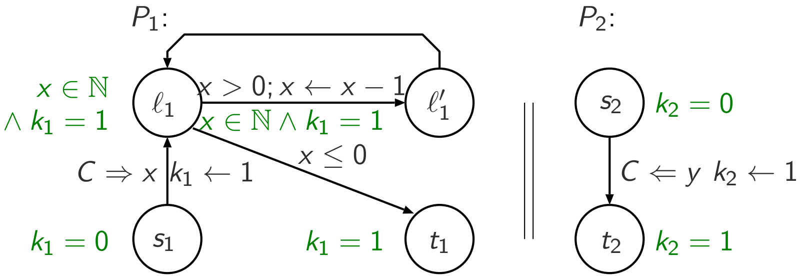

$I = (k_1 = k_2)$.

# Example 4 cont'd




$I = (k_1 = k_2)$.

# Example 4 cont'd




$I = (k_1 = k_2)$.

# 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?

We'll look at a compositional proof methods, reasoning about asynchronous communication, and, time allowing, we'll talk about process algebra.

Then, the remainder of the course is about distributed algorithms.

Assignment 1 is out! You should probably be working on it..