

Semantics and invariance proof method for weakly consistent parallelism

Jade Alglave (MSR-Cambridge, UCL, UK)

Patrick Cousot (NYU, Emer. ENS, PSL)

IFIP WG 2.3

Château de Villebrumier, France

October 4th, 2016

Weakly consistent parallel programs

Weakly consistent parallel programs

```
var  $x_1, \dots, x_m$ ; // shared variables  
 $P_0$ ; // prelude initializing  $x_1, \dots, x_m$   
[ $P_1 \parallel P_2 \parallel \dots \parallel P_n$ ]
```

- P_1, P_2, \dots, P_n are the **processes** modifying the shared variables and their **local registers** R, \dots
- The execution of a **write** $x := E$ to a shared variable and the **read** $R := x$ of a shared variable is not instantaneous (as in sequential consistency)

Example (1b, load buffer)

- Algorithm A:

0: { x = 0; y = 0; }	
P0	P1
1: r[] r1 x	11: r[] r2 y;
2: w[] y 1	12: w[] x 1;
3:	13:

- Specification S_{inv} :

$$\text{at } 3 \wedge \text{at } 13 \Rightarrow \neg(r1=1 \wedge r2=1)$$

Example (Peterson)

- Algorithm A:

```
0: { w F1 false; w F2 false; w T 0; }
```

```
P0:
```

```
1: w[] F1 true
```

```
2: w[] T 2
```

```
3: do {i}
```

```
4:   r[] R1 F2
```

```
5:   r[] R2 T
```

```
6: while R1  $\wedge$  R2  $\neq$  1
```

```
7: skip (* CS1 *)
```

```
8: w[] F1 false
```

```
9:
```

```
P1:
```

```
10: w[] F2 true;
```

```
11: w[] T 1;
```

```
12: do {j}
```

```
13:   r[] R3 F1;
```

```
14:   r[] R4 T;
```

```
15: while R3  $\wedge$  R4  $\neq$  2;
```

```
16: skip (* CS2 *)
```

```
17: w[] F2 false;
```

```
18:
```

- Specification S_{inv} :

```
1: {true}
```

```
10: {true}
```

```
...
```

```
...
```

```
7: { $\neg$ at{16}}
```

```
16: { $\neg$ at{7}}
```

```
...
```

```
...
```

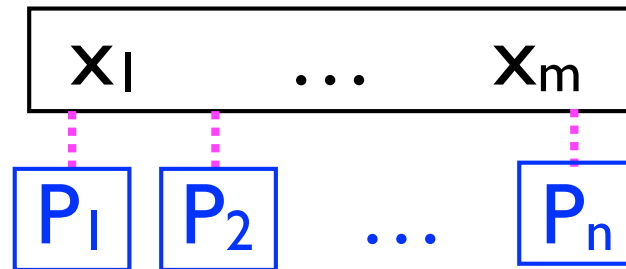
```
9: {true}
```

```
18: {true}
```

G. L. Peterson. Myths about the mutual exclusion problem. *Inf. Process. Lett.*, 12(3):115–116, 1981. doi: 10.1016/0020-0190(81)90106-X. URL [http://dx.doi.org/10.1016/0020-0190\(81\)90106-X](http://dx.doi.org/10.1016/0020-0190(81)90106-X).

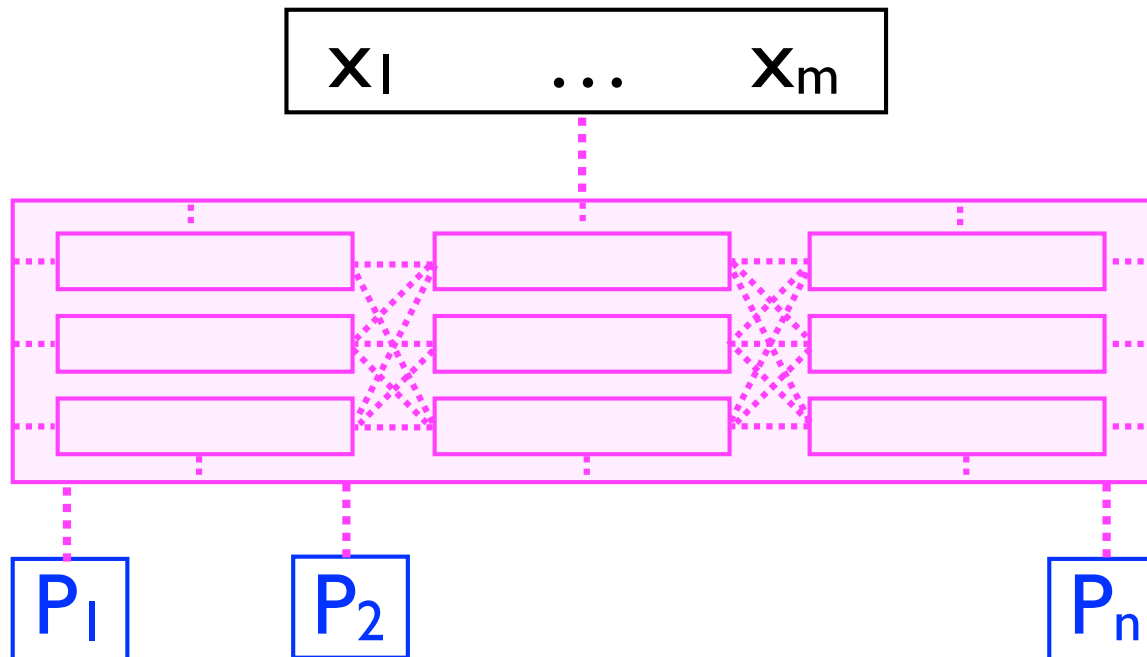
Weak memory/consistency models

- Sequential consistency:



atomic
instantaneous
communications

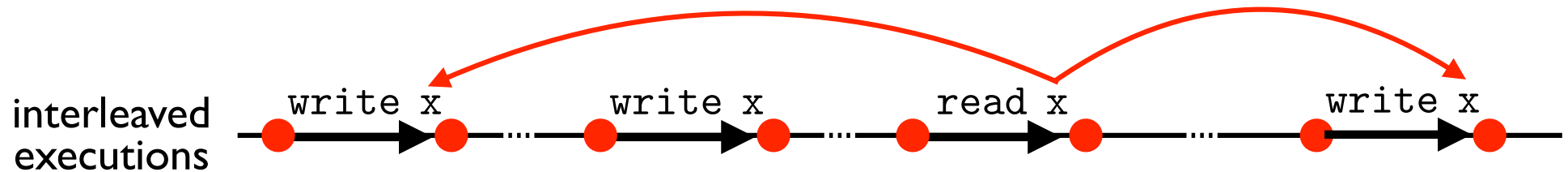
- Weak memory models:



communication
network
(anticipations,
delays, shuffles...)

Read/write matching

- In the worst case a read x can read from any past or future write x of any process (including for the reading process)



Example (1b, incorrect)

- $0:\{ x = 0; y = 0; \}$

P0		P1	
1:r[] r1 x		11:r[] r2 y;	
2:w[] y 1		12:w[] x 1 ;	
3:		13:	

- $\text{at } 3 \wedge \text{at } 13 \wedge r1=1 \wedge r2=1$
- This erroneous behavior can be observed on TSO machines

Example: Peterson (incorrect)

- Can read the wrong flags

0: { w F1 false; w F2 false; w T 0; }	
P0:	P1:
1: w[] F1 true	10: w[] F2 true;
2: w[] T 2	11: w[] T 1;
3: do	12: do
4: r[] R1 F2	13: r[] R3 F1;
5: r[] R2 T	14: r[] R4 T;
6: while R1 \wedge R2 \neq 1	15: while R3 \wedge R4 \neq 2;
7: skip (* CS1 *)	16: skip (* CS2 *)
8: w[] F1 false	17: w[] F2 false;
9:	18:

at 6 \wedge at 16: $\neg R1 \wedge R2=1 \wedge \neg R3 \wedge R4=2$ holds

\Rightarrow both processes simultaneously enter their critical section

Example: Peterson (incorrect)

- Can read the wrong turns

0: { w F1 false; w F2 false; w T 0; }	
P0:	P1:
1: w[] F1 true	10: w[] F2 true;
2: w[] T 2	11: w[] T 1;
3: do	12: do
4: r[] R1 F2	13: r[] R3 F1;
5: r[] R2 T	14: r[] R4 T;
6: while R1 \wedge R2 \neq 1	15: while R3 \wedge R4 \neq 2;
7: skip (* CS1 *)	16: skip (* CS2 *)
8: w[] F1 false	17: w[] F2 false;
9:	18:

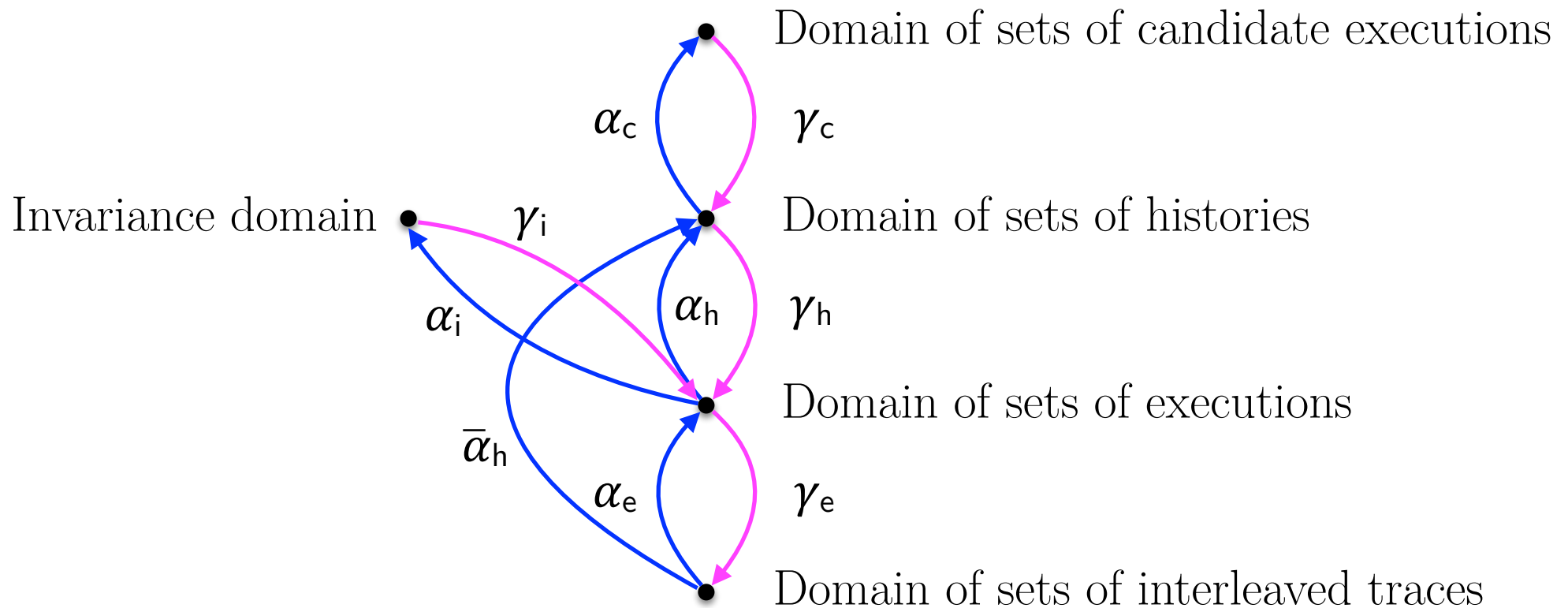
at 6 \wedge at 16: $\neg R1 \wedge R2=1 \wedge \neg R3 \wedge R4=2$ holds

\Rightarrow both processes simultaneously enter their critical section

A hierarchy of semantics of weakly consistent parallelism

Hierarchy of semantics

- Hierarchy of semantic domains:



- Induces a hierarchy of semantics

Sets of interleaved traces

- **Traces**: maximal finite or infinite sequence of states separated by events generated by computation and communication steps \rightarrow *global time*
- **States**: shared memory assigning values to global variables, store buffers, ... program point of each process, assignment to local registers
- **Events** e : $P(e)$ process executed, $A(e)$: labelled action executed, $X(e)$: shared variable involved, $V(e)$: value involved, ...
- No restriction on who can read which write on the same shared variable!

Example of interleaved trace for 1b

- 0: { x = 0; y = 0; }

P0		P1	
1:r[] r1 x		11:r[] r2 y;	
2:w[] y 1		12:w[] x 1 ;	
3:		13:	

- $$\text{start} \xrightarrow{0: \overbrace{x = 0}^{w_x^0}; \overbrace{y = 0}^{w_y^0}} \langle \{x \leftarrow w_x^0, y \leftarrow w_y^0\}, 1: \{r1 \leftarrow 0\}, 11: \{r2 \leftarrow 0\} \rangle$$

$$\xrightarrow{0: \overbrace{1:r[] \ r1 \ x}^{r_x^1}} \langle \{x \leftarrow w_x^{12}, y \leftarrow w_y^0\}, 2: \{r1 \leftarrow 1\}, 11: \{r2 \leftarrow 0\} \rangle$$

$$\xrightarrow{11: \overbrace{r[] \ r2 \ y}^{r_y^{11}}} \langle \{x \leftarrow w_x^{12}, y \leftarrow w_y^2\}, 2: \{r1 \leftarrow 1\}, 12: \{r2 \leftarrow 1\} \rangle$$

$$\xrightarrow{12: \overbrace{w[] \ x \ 1}^{w_x^{12}}} \langle \{x \leftarrow w_x^{12}, y \leftarrow w_y^2\}, 2: \{r1 \leftarrow 1\}, 13: \{r2 \leftarrow 1\} \rangle$$

$$\xrightarrow{13: \overbrace{2:w[] \ y \ 1}^{w_y^2}} \langle \{x \leftarrow w_x^{12}, y \leftarrow w_y^2\}, 3: \{r1 \leftarrow 1\}, 13: \{r2 \leftarrow 1\} \rangle$$

Sets of truly parallel execution traces

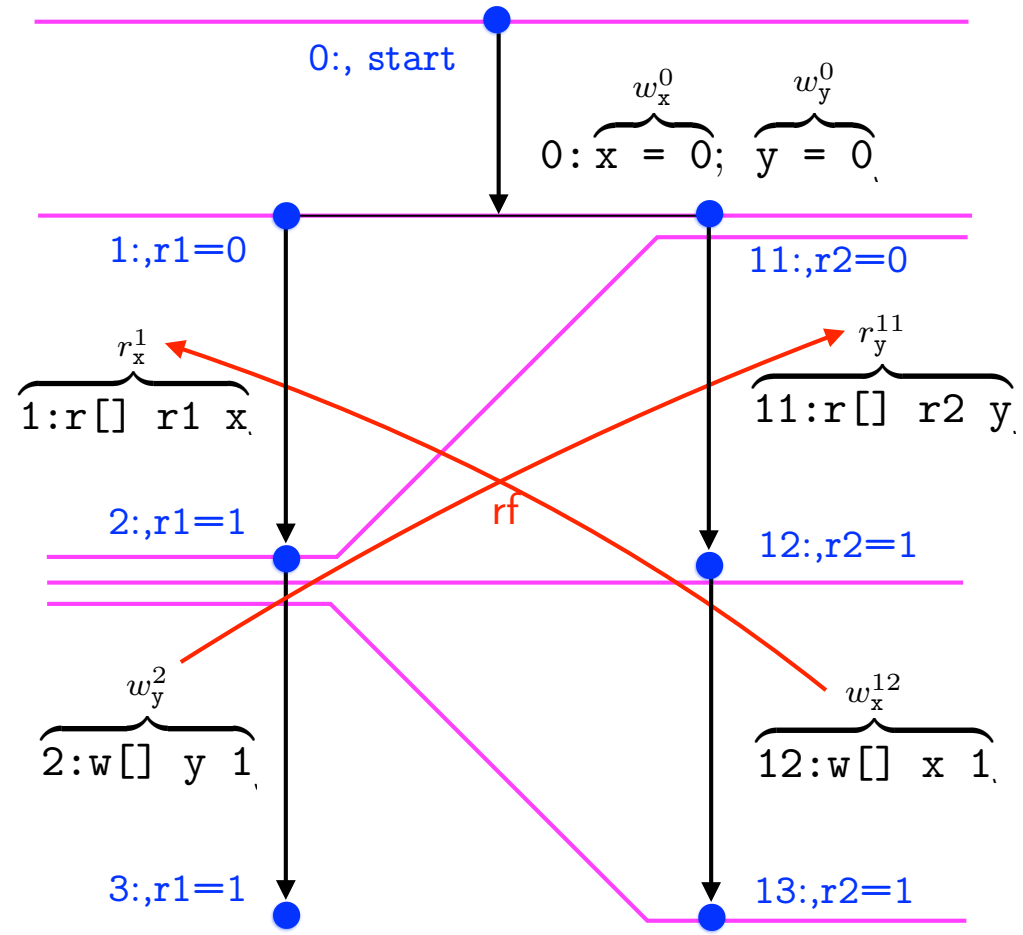
- project **traces per process** \rightarrow *local time on computations*
- get rid of shared memory states using a **read-from relation** rf \rightarrow *no time on communications*

$$\langle r, w \rangle \in \text{rf} \iff \tau = \tau_0 \langle \nu, \dots \rangle \xrightarrow{r} \langle \nu', \dots \rangle \tau_1 \wedge \nu'(X(r)) = w$$

- keep **local states** on process control points and values of registers
- keep computation progress information using **cuts** of parallel traces \rightarrow *global time*

Example of truly parallel execution for lb

0: { x = 0; y = 0; }	
P0	P1 ;
1:r[] r1 x	11:r[] r2 y;
2:w[] y 1	12:w[] x 1 ;
3:	13: ;

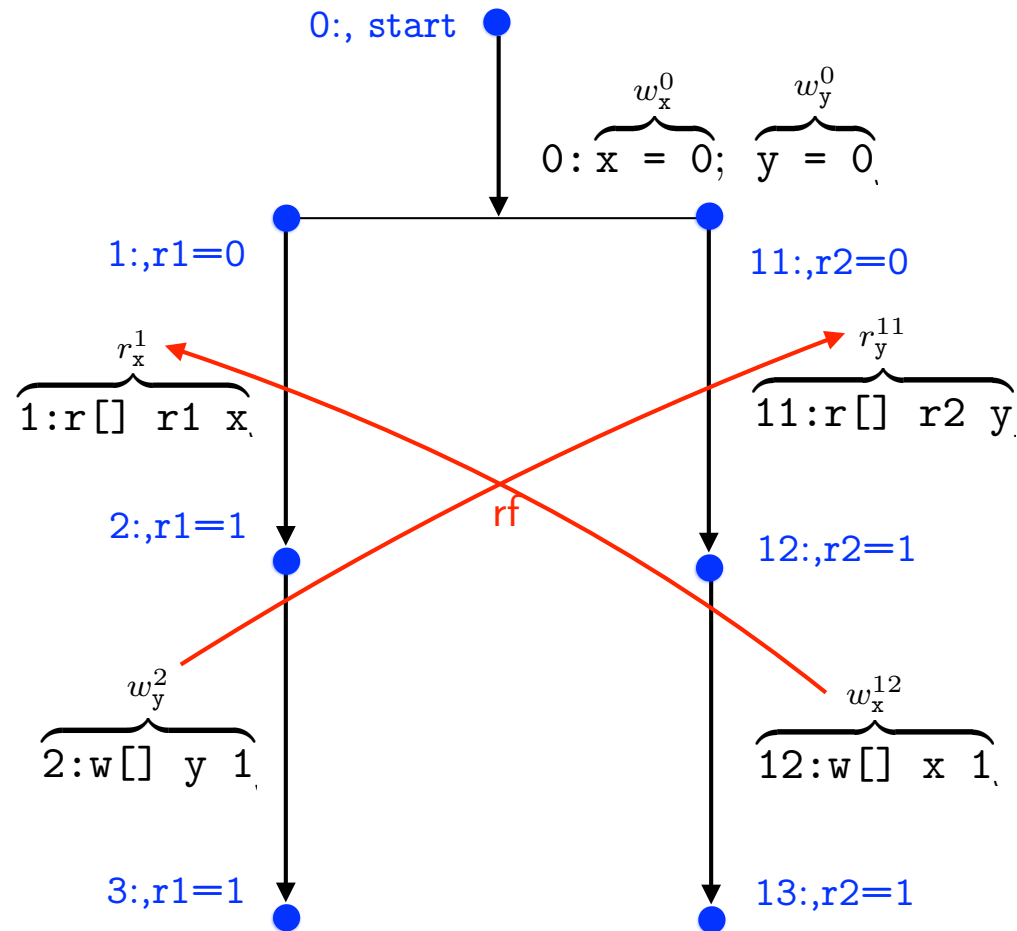


Sets of histories

- Get rid of cuts \longrightarrow *no global time*
- A processor cannot know where the others parallel processors are in their computations

Example of history for 1b

0: { x = 0; y = 0; }	
P0	P1
1:r[] r1 x	11:r[] r2 y;
2:w[] y 1	12:w[] x 1 ;
3:	13:



Sets of candidate executions

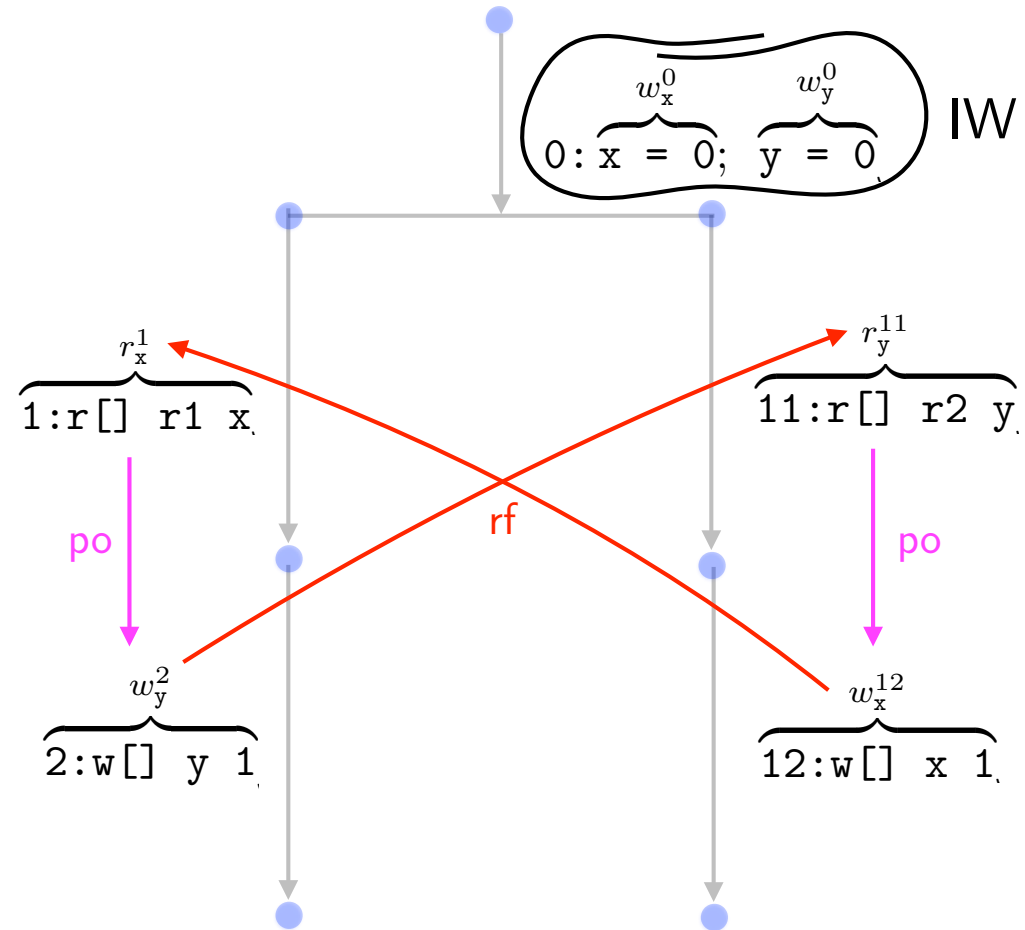
- Keep the **set of events**
- Keep the **read-from relation rf**
- Represent process traces $\tau_0 \prod_{i=1}^n \tau_i$, **rf** by
 - the set of **initial writes IW** in τ_0
 - the **program order po**
$$\langle e, e' \rangle \in \text{po} \iff \tau_i = \tau_i' \xrightarrow{e} \tau_i'' \xrightarrow{e'} \tau_i'''$$

 $\longrightarrow \text{relational on events}$
- Get rid of states
 $\longrightarrow \text{no values}$

Example of candidate execution for 1b

0: { x = 0; y = 0; }

P0	P1	
1:r[] r1 x	11:r[] r2 y;	
2:w[] y 1	12:w[] x 1 ;	
3:	13:	



Auxiliary relations

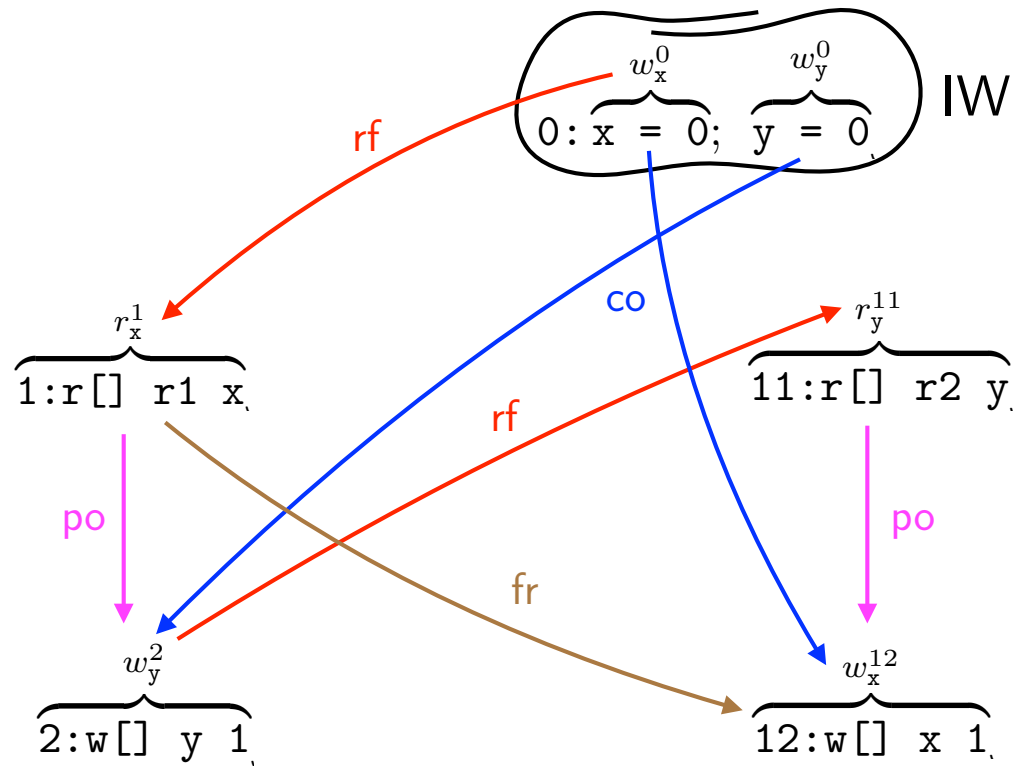
- **loc**: between events on the same shared variable
- **ext**: between events on different processes
- coherence order **co**: between a write and the later ones on the same shared variable
- from-read **fr**: between a read reading from a write and the later writes to the same shared variable

$$\text{fr} = \text{rf}^{-1} ; \text{co}$$

Auxiliary relations

0: { x = 0; y = 0; }

P0	P1		
1:r[] r1 x	11:r[] r2 y;		
2:w[] y 1	12:w[] x 1 ;		
3:	13:		



co in cat

"co.cat"

```
let fold f =  
  let rec fold_rec (es,y) = match es with  
    || {} -> y  
    || e ++ es -> fold_rec (es, f(e,y))  
  end in  
  fold_rec  
  
let map f = fun es -> fold (fun (e,y) -> f e ++ y) (es,{})  
  
let rec cross S = match S with  
  || {} -> { 0 }  
  || S1 ++ S ->  
    let yss = cross S in  
    fold  
      (fun (e1,r) -> map (fun t -> e1 | t) yss | r)  
(S1,{}) end  
  
let co0 = loc & (IW * (W\IW))  
let makeCo(s) = linearisations(s,co0)  
let same-loc-writes = loc & (W*W)  
let allCoL = map makeCo (classes (same-loc-writes))  
let allCo = cross allCoL  
  
with co from allCo
```

Example of specification of weakly consistent parallelism in the semantic hierarchy: sequential consistency

Sequential consistency

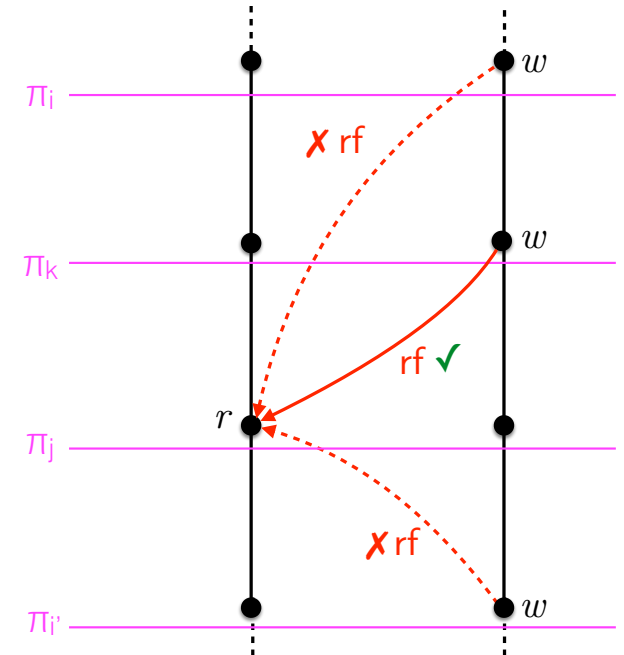
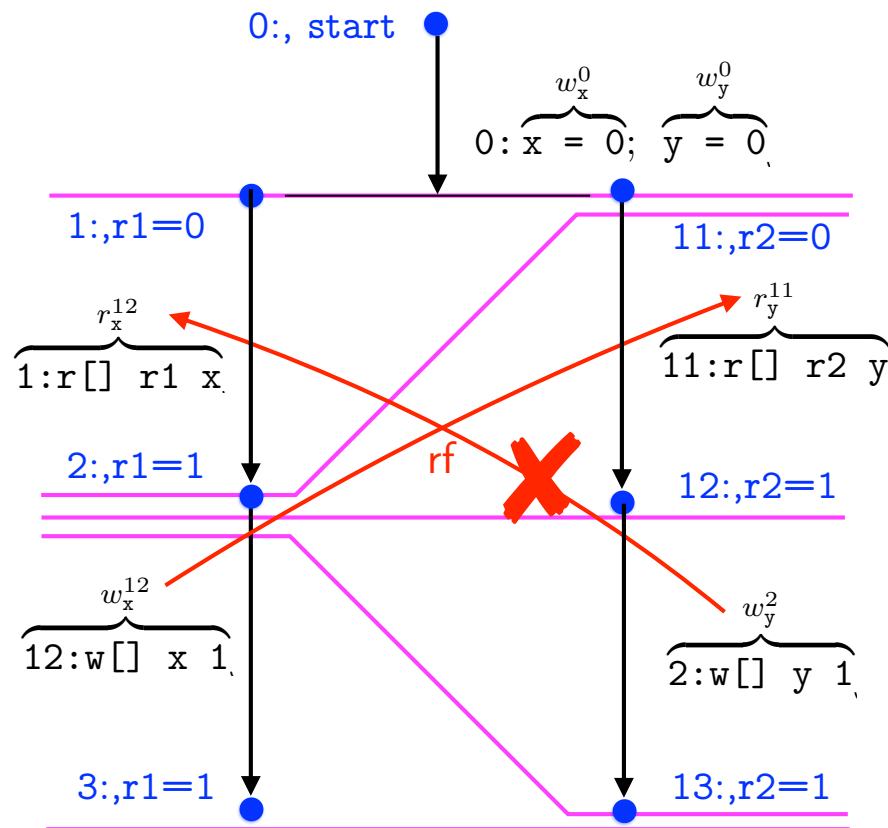
- **Interleaved semantics:** a read can only read from the last past write

• lb:

$$\begin{aligned}
 &\text{start} \quad \frac{0: \overbrace{x = 0}^{w_x^0}; \overbrace{y = 0}^{w_y^0}}{\rightarrow} \langle \{x \leftarrow w_x^0, y \leftarrow w_y^0\}, 1: \{r1 \leftarrow 0\}, 11: \{r2 \leftarrow 0\} \rangle \\
 &0\} \rangle \xrightarrow{\overbrace{1: r[] \ r1 \ x}^{r_x^{12}}} \langle \{x \leftarrow \cancel{w_x^{12}}, y \leftarrow w_y^0\}, 2: \{r1 \leftarrow 1\}, 11: \{r2 \leftarrow 0\} \rangle \xrightarrow{\overbrace{11: r[] \ r2 \ y}^{r_y^{11}}} \\
 &\langle \{x \leftarrow w_x^{12}, y \leftarrow \cancel{w_y^2}\}, 2: \{r1 \leftarrow 1\}, 12: \{r2 \leftarrow 1\} \rangle \xrightarrow{\overbrace{12: w[] \ x \ 1}^{w_x^{12}}} \langle \{x \leftarrow w_x^{12}, y \leftarrow \\
 &w_y^2\}, 2: \{r1 \leftarrow 1\}, 13: \{r2 \leftarrow 1\} \rangle \xrightarrow{\overbrace{2: w[] \ y \ 1}^{w_y^2}} \langle \{x \leftarrow w_x^{12}, y \leftarrow w_y^2\}, 3: \{r1 \leftarrow 1\}, \\
 &11: \{r2 \leftarrow 1\} \rangle
 \end{aligned}$$

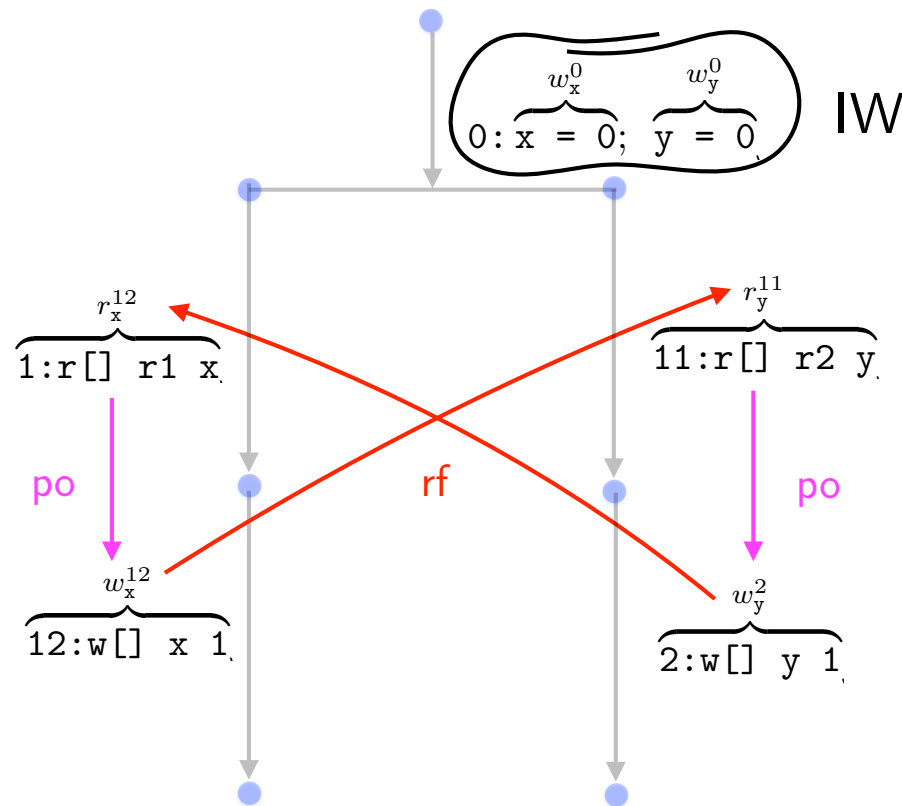
Example: sequential consistency for lb

- Parallel executions with cuts: a read can read only the last write before its cut
- lb:



Example: sequential consistency for lb

- **Parallel histories:** abstract to candidate execution and check it is allowed
- **Candidate executions:** irreflexive po ; rf ; po; rf



Analytic semantics of weakly consistent parallelism

Analytic semantics

- **Anarchic semantics**: all possible executions with cuts/histories with no restriction on rf (any read can read any value from any write to the same shared variable)
- **Communication consistency**: requirements on rf specified on an abstraction to a candidate execution
- **Analytic semantics**: all executions with cuts/histories which rf satisfies the consistency requirements

Example of anarchic semantics: LB

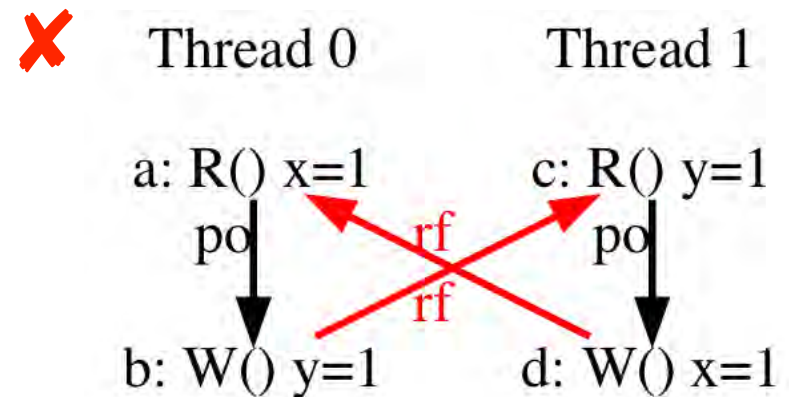
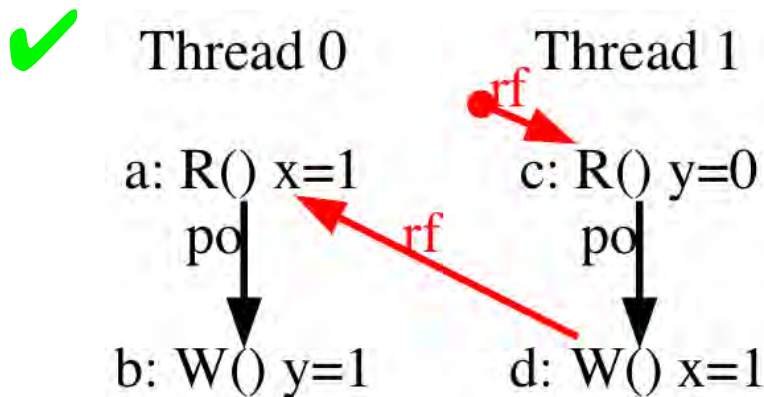
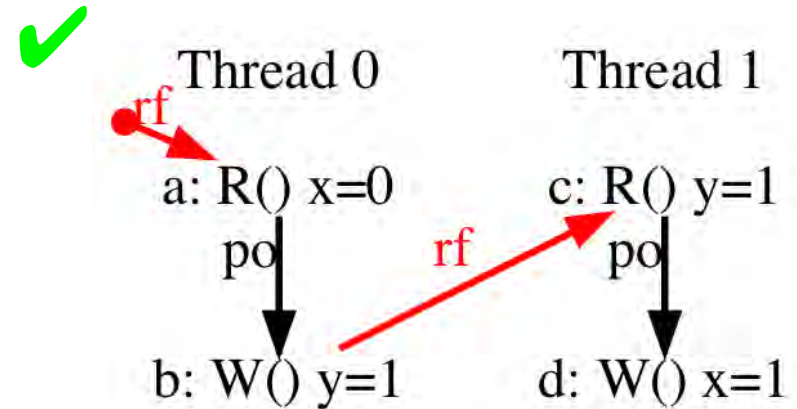
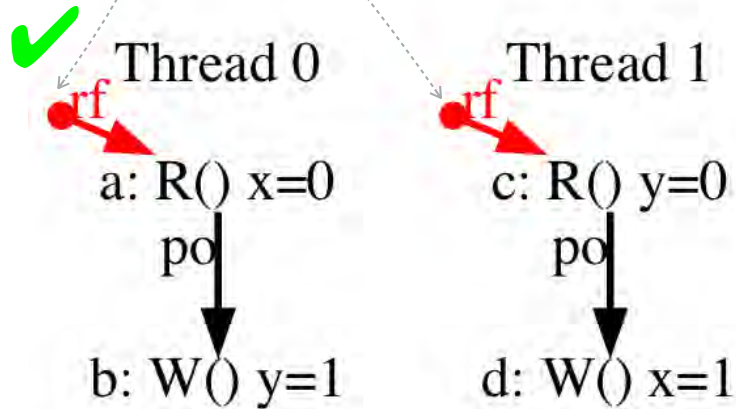
```
{ x = 0; y = 0; }
```

```
P0          | P1          ;
```

```
r[]  r1  x  | r[]  r2  y  ;
```

```
w[]  y  1   | w[]  x  1   ;
```

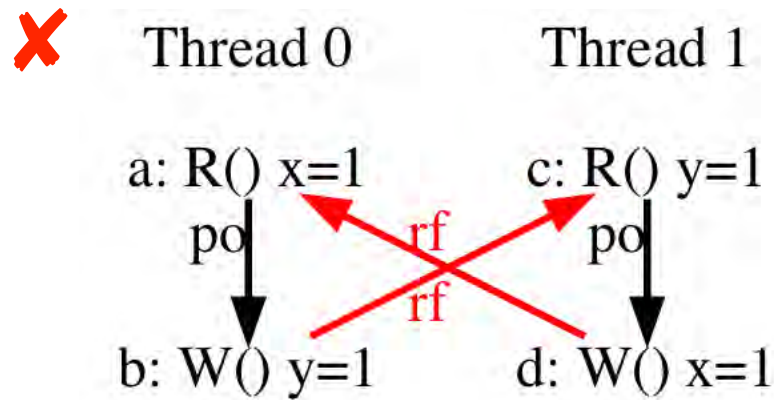
read from
initial write



Example of communication specification in the cat language for LB

irreflexive (po | rf)+

Rejects only the anarchic execution:



J. Alglave. *A Shared Memory Poetics*. PhD thesis, Université Paris 7, 2010.

J. Alglave, P. Cousot, and L. Maranget. Syntax and semantics of the cat language. *HSA Foundation*, Version 1.1:38 p., 16 Oct 2015b. URL <http://www.hsafoundation.com/?ddownload=5382>.

Examples of architecture specification

- SC (sequential consistency):

```
let co = (IW*W) & loc
let fr = (rf^-1;co)
acyclic po | rf | co | fr as sc
```

- TSO:

```
let co = (IW*W) & loc
let fr = (rf^-1;co)
let po-loc = po & loc
acyclic po-loc | rf | co | fr as scpv
let ppo = po \ (W*R)
let rfe = rf & ext
acyclic ppo | rfe | co | fr as tso
```

- For lb:

```
acyclic (po | rf) as lb
```

$sc \Rightarrow lb$, $tso \not\Rightarrow lb$

Fence specification:

- In Lisa:

```
{ x = 0; y = 0; }  
P0      | P1      ;  
r[] r1 x | r[] r2 y ;  
f[dep]   | f[lw]   ;  
w[] y 1  | w[] x 1  ;
```
- Implementation with dependencies and fences in TSO:

data
dependency

```
{ x = 0; y = 0; }  
P0      | P1      ;  
r[] r1 x | r[] r2 y ;  
r2 = xor r1 r1 | mfence ;  
r3 = r2+1      |  
w[] y r3       | w[] x r3  ;
```

cat

- Handles one history at a time
- For each execution relies on:
 - the set $E(_)$ of events of the execution (partitionned into initial writes IW , writes W , read R , fences F , ...)
 - the program order po of events per process
 - the read-from relation rf per variable
- Has predefined relations loc , ext , ...
- Can define new relations e.g. $*$, $;$, $|$, $\&$, \backslash , $+$, $\hat{-1}$, ...
- Accepts/eliminates the execution by defining relations r and checking $irreflexive\ r$, $acyclic\ r$, $empty\ r$, $not\ empty\ r$

ARM in cat

```
let fr = rf-1;co
acyclic po-loc | rf | co | fr as scpv

let deps = addr | data
let rdw = po-loc & (fre;rfe)
let detour = po-loc & (coe ; rfe)

let ii0 = deps | rfi | rdw
let ic0 = 0
let ci0 = ctrlcfence(ISB) | detour
let cc0 = deps | ctrl | (addr;po)

let rec ii = ii0 | ci | (ic;ci) | (ii;ii)
and ic = ic0 | ii | cc | (ic;cc) | (ii;ic)
and ci = ci0 | (ci;ii) | (cc;ci)
and cc = cc0 | ci | (ci;ic) | (cc;cc)

let ppo = ii & R*R | ic & R*W

let dmb = fencerel(DMB)
let dsb = fencerel(DSB)
let fences = dmb|dsb
let A-cumul = rfe;fences

let hb = ppo | fences | rfe
acyclic hb as no-thin-air

let prop-base = (fences | A-cumul);hb*
let prop = (prop-base & W*W) | (com*; prop-base*; fences; hb*)

irreflexive fre;prop;hb* as observation
acyclic co | prop as propagation
```

Invariance proof method for weakly consistent parallelism

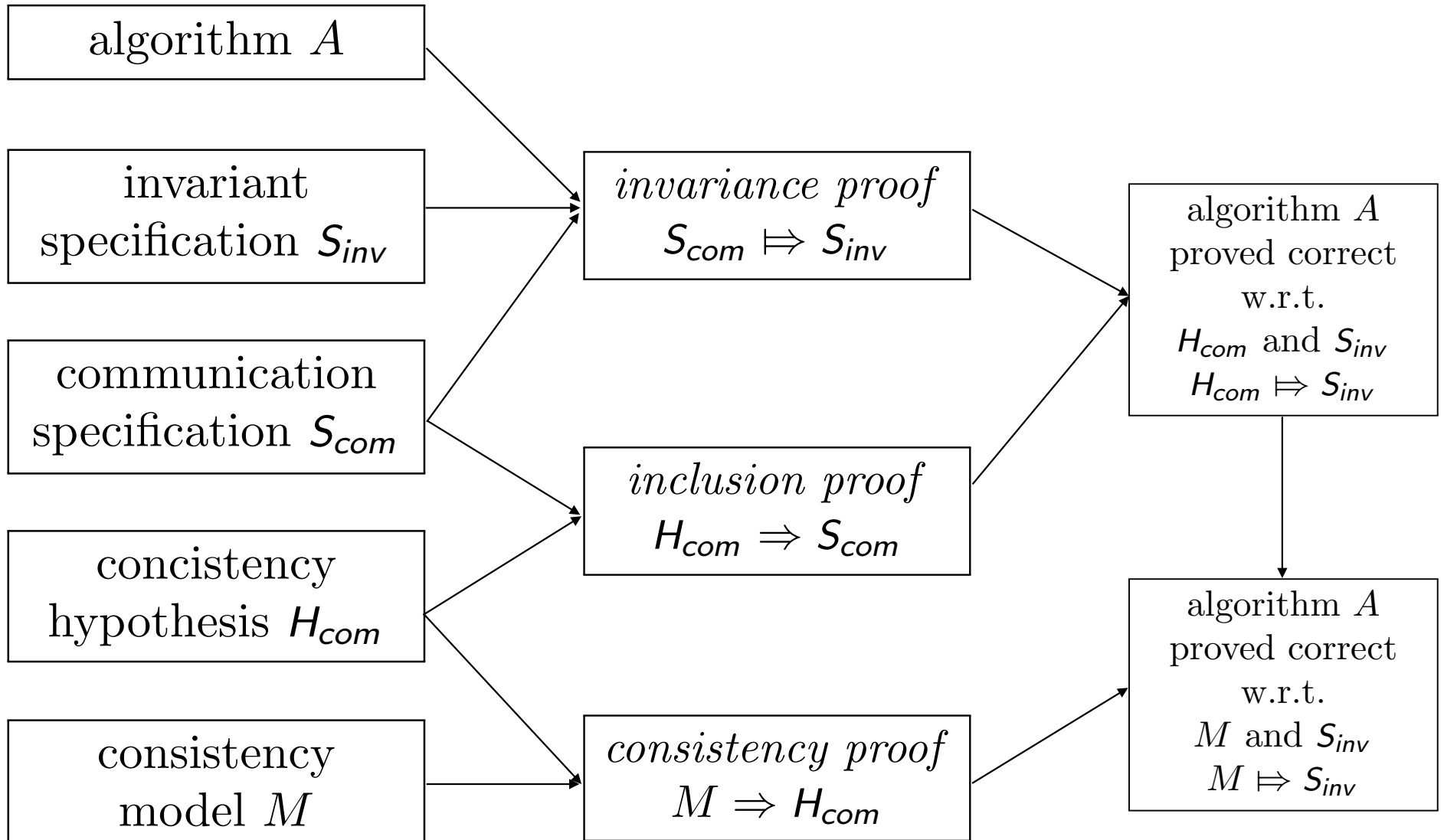
Difficulties

- There is no longer a notion of instantaneous value of the shared variables:
 - ⇒ **pythia variables** (denoting values of variables when read)
 - ⇒ **communications rf** (keeping track of which writes events the pythia variables take there values from)
 - ⇒ **stamps** (keeping track of events to distinguish different instruction executions)

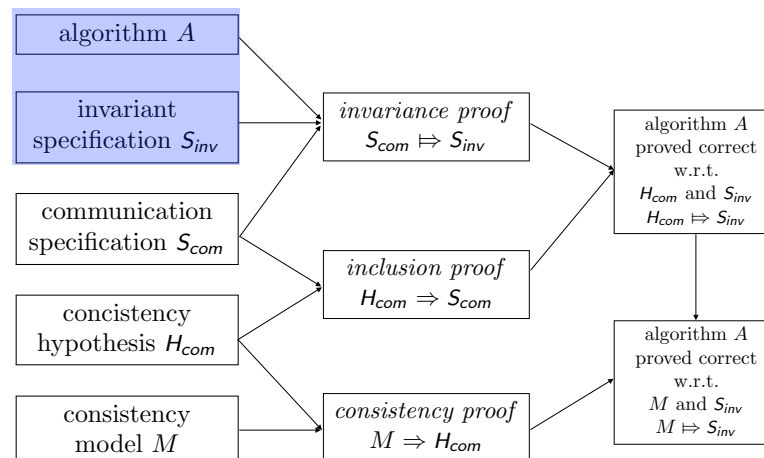
Difficulties

- We have to make hypotheses on how communications do happen:
⇒ communication specification S_{com}
- We have to show that the communication specification is correctly implemented on an architecture:
⇒ a way to mix invariant S_{com} and cat specifications

Methodology



Invariant



Pythia variables

- Unique name given to communicated values during execution (using stamps)

0: { w F1 false; w F2 false; w T 0; }

P0:

1: w[] F1 true

2: w[] T 2

3: repeat { i }

4: r[] R1 F2 { \rightsquigarrow $F2_4^i$ }

5: r[] R2 T { \rightsquigarrow T_5^i }

6: until $\neg R1 \vee R2 = 1$ { i_{end} }

7: skip (* CS1 *)

8: w[] F1 false

9:

P1:

10: w[] F2 true;

11: w[] T 1;

12: repeat { j }

13: r[] R3 F1; { \rightsquigarrow $F1_{13}^j$ }

14: r[] R4 T; { \rightsquigarrow T_{14}^j }

15: until $\neg R3 \vee R4 = 2$; { j_{end} }

16: skip (* CS2 *)

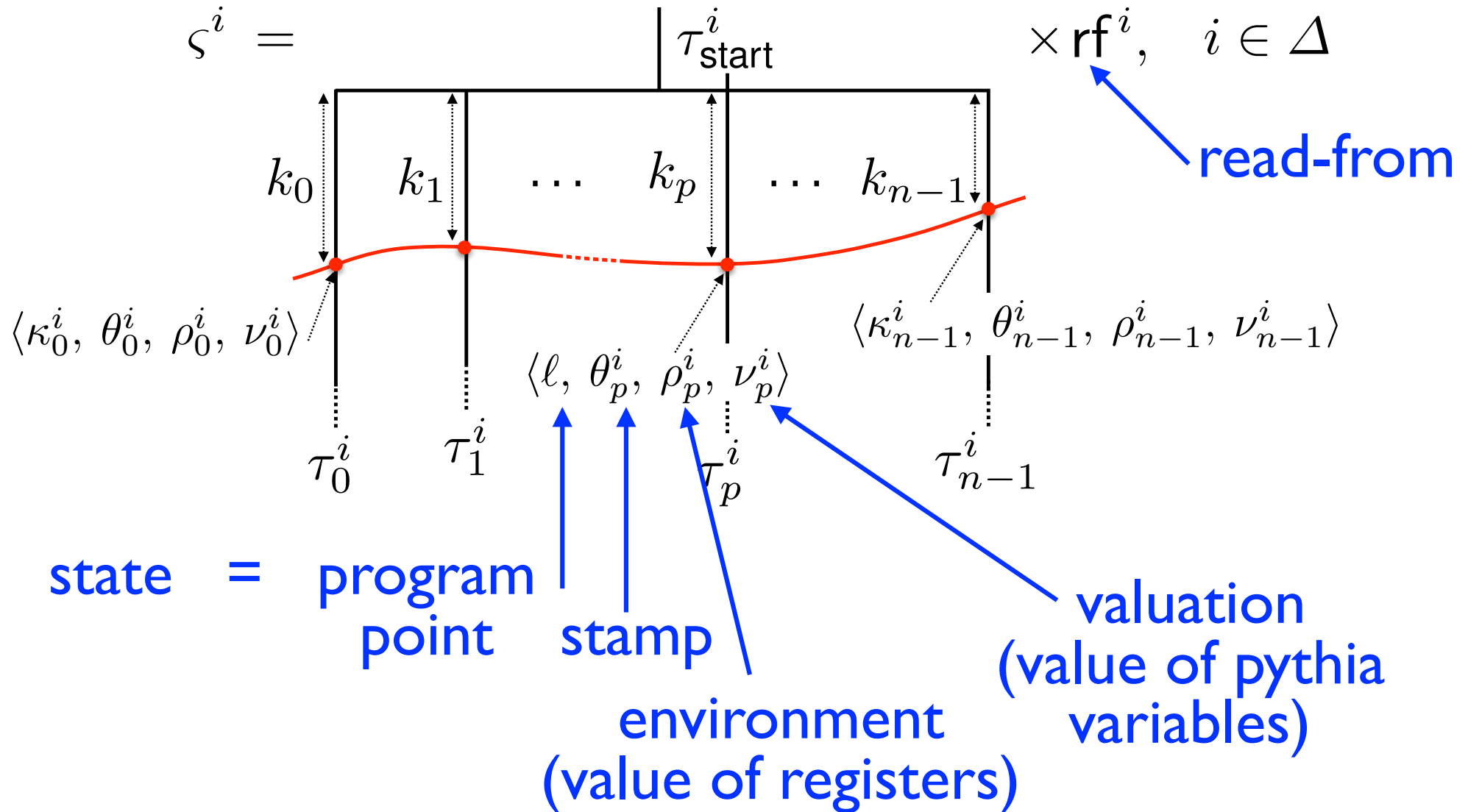
17: w[] F2 false;

18:

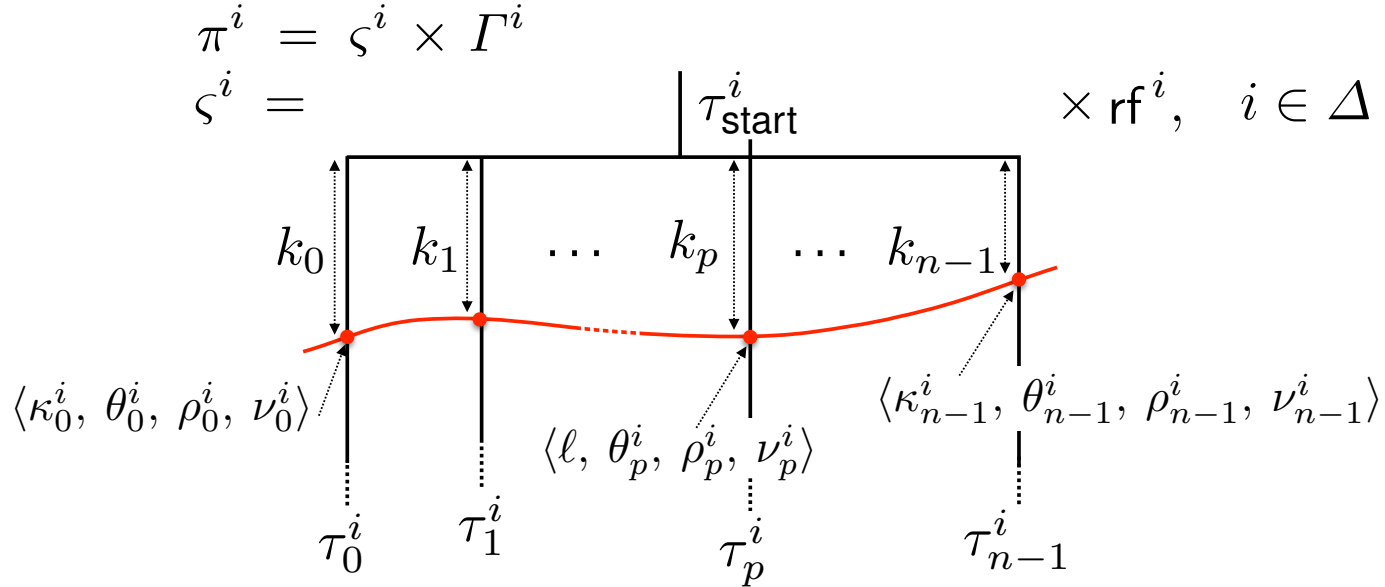
Stamp: label, counter

Pythia variables

Invariance abstraction



Invariance abstraction



$$\alpha_a(\{\pi^i \mid i \in \Delta\}) \triangleq \prod_{p \in \mathbb{P}^i} \prod_{\ell \in \mathbb{L}(p)} \bigcup_{i \in \Delta} \{ \langle \kappa_{0,k_0}^i, \theta_{0,k_0}^i, \rho_{0,k_0}^i, \nu_{0,k_0}^i, \dots, \nu_{p-1,k_{p-1}}^i, \theta_{p,k_p}^i, \rho_{p,k_p}^i, \nu_{p,k_p}^i, \kappa_{p+1,k_{p+1}}^i, \dots, \kappa_{n-1,k_{n-1}}^i, \theta_{n-1,k_{n-1}}^i, \rho_{n-1,k_{n-1}}^i, \nu_{n-1,k_{n-1}}^i, \text{rf}^i \rangle \mid \forall q \in [0, n[\setminus \{p\} . \underline{\tau}_q^i = \mathfrak{s} \langle \kappa_{q,k_q}^i, \theta_{q,k_q}^i, \rho_{q,k_q}^i, \nu_{q,k_q}^i \rangle \wedge \underline{\tau}_p^i = \mathfrak{s} \langle \ell, \theta_{p,k_p}^i, \rho_{p,k_p}^i, \nu_{p,k_p}^i \rangle \}.$$

Invariant

- An **invariant** $S_{inv}(p)$ at point p of process P_i is a statement relating
 - the **program points** $p_1, \dots, p_{i-1}, p_{i+1}, \dots, p_m$ of the other processes
 - the **pythia variables** (forbidden to mention of shared variables)
 - the **local registers** of all processes
 - the **communications** (rf)

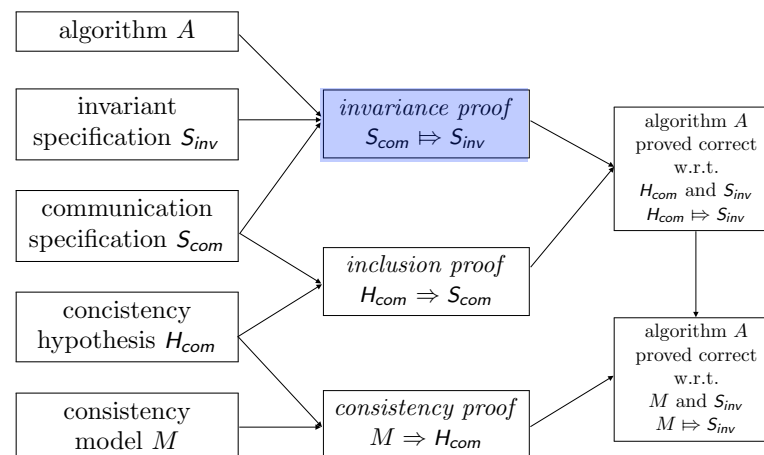
which always holds when at the cut where execution reaches point p of process P_i and the other processes are at $p_1, \dots, p_{i-1}, p_{i+1}, \dots, p_m$

Example (Peterson)

<pre> 0: { w F1 false; w F2 false; w T 0; } {F1=false ∧ F2=false ∧ T=0} } 1: {R1=0 ∧ R2=0} w[] F1 true 2: {R1=0 ∧ R2=0} w[] T 2 3: {R1=0 ∧ R2=0} do {i} 4: {(i=0 ∧ R1=0 ∧ R2=0) ∨ (i>0 ∧ R1=F2₄ⁱ⁻¹ ∧ R2=T₅ⁱ⁻¹)} r[] R1 F2 {↗ F2₄ⁱ} 5: {R1=F2₄ⁱ ∧ (i=0 ∧ R2=0) ∨ (i>0 ∧ R2=T₅ⁱ⁻¹)} r[] R2 T {↗ T₅ⁱ} 6: {R1=F2₄ⁱ ∧ R2=T₅ⁱ} while R1 ∧ R2≠1 {i_{end}} 7: {¬F2₄^{i_{end}} ∨ T₅^{i_{end}}=1} skip (* CS1 *) 8: {¬F2₄^{i_{end}} ∨ T₅^{i_{end}}=1} w[] F1 false 9: {¬F2₄^{i_{end}} ∨ T₅^{i_{end}}=1} </pre>	<pre> 10: {R3=0 ∧ R4=0} w[] F2 true; 11: {R3=0 ∧ R4=0} w[] T 1; 12: {R3=0 ∧ R4=0} do {j} 13: {(j=0 ∧ R3=0 ∧ R4=0) ∨ (j>0 ∧ R3=F1₁₃^{j-1} ∧ R4=T₁₄^{j-1})} r[] R3 F1 {↗ F1₁₃^j}; 14: {R3=F1₁₃^j ∧ (j=0 ∧ R4=0) ∨ (j>0 ∧ R4=T₁₄^{j-1})} r[] R4 T; {↗ T₁₄^j} 15: {R3=F1₁₃^j ∧ R4=T₁₄^j} while R3 ∧ R4≠2 {j_{end}} ; 16: {¬F1₁₃^{j_{end}} ∨ T₁₄^{j_{end}}=2} skip (* CS2 *) 17: {¬F1₁₃^{j_{end}} ∨ T₁₄^{j_{end}}=2} w[] F2 false; 18: {¬F1₁₃^{j_{end}} ∨ T₁₄^{j_{end}}=2} </pre>
---	--

(these invariants are for the anarchic semantics, so all communications are possible, no constraints on rf)

Invariance proof



Verification conditions

- Sequential proof
- Absence of interference proof
- Communication proof

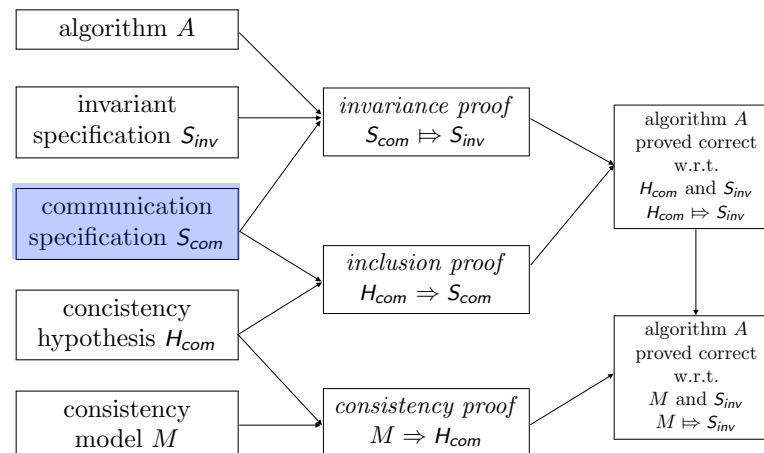
Examples:

- $\{ P(R, \dots, rf) \wedge \langle w(x, v), r(\theta, x) \rangle \in rf \}$
 $\text{read } x \text{ R } \{ \leadsto x_\theta \}$ *communication* \rightarrow
 $\{ P[R \leftarrow x_\theta, x_\theta \leftarrow v, \dots, rf] \}$
- $\{ P \}$ fence $\{ P \}$ (fences are markers in the execution)
- $\{ P \}$ write $R \ x \ \{ P \}$ (a write has no local effect)

Communication proof

- The communications rf must be checked to be well-formed (none allowed by H_{cm} should miss, see later)
- If $\langle w(P, p, \theta, x, v), r(P', p', \theta', x, x_{\theta'}) \rangle \in rf$ then:
 - The read instruction of at point p' process P' must read from an *initial* or a *reachable* write
 - A read event (for a given stamp θ') must read from a *unique write event* with the same variable x
 - The *value assigned* to the read pythia variable $x_{\theta'}$ must be that of v the matching write

Communication specification S_{com}



Communication specification

- The algorithm A is often incorrect for the anarchic semantics
- The allowable communications are specified by a communication specification S_{com} (i.e. an invariant constraining the allowed communications rf)
- This communication specification can often be calculated from the anarchic invariant and the inductive invariant S_{ind}

Example (Peterson)

$$\begin{aligned}
 & \text{at } 7 \wedge \text{at } 16 \\
 \Rightarrow & (\neg F2_4^{i_{\text{end}}} \vee T_5^{i_{\text{end}}} = 1) \wedge (\neg F1_{13}^{j_{\text{end}}} \vee T_{14}^{j_{\text{end}}} = 2) \} \\
 & \quad \{ \text{i.e. the invariants at lines 7: and 16: hold} \} \\
 \Rightarrow & \neg S_{\text{com}} \quad \{ \text{since by taking } i = i_{\text{end}} \text{ and } j = j_{\text{end}}, \text{ we have} \\
 & \quad (F2_4^i = \text{false} \vee T_5^i = 1) \wedge (F1_{13}^j = \text{false} \vee T_{14}^j = 2) \}
 \end{aligned}$$

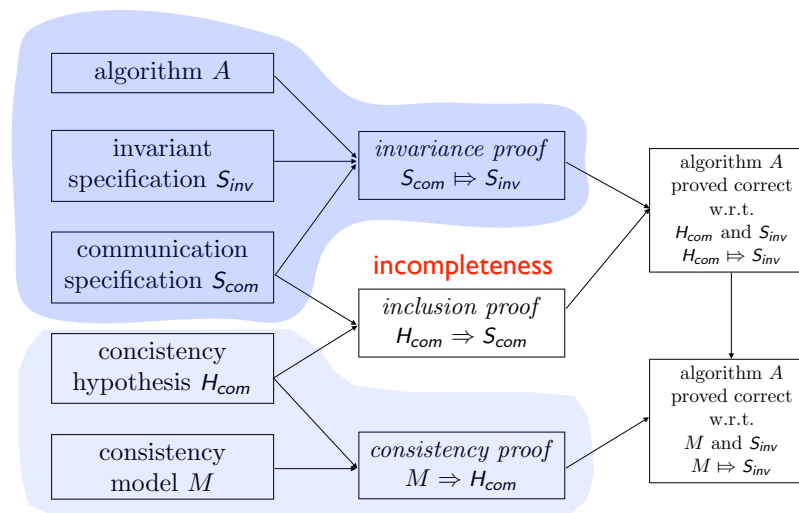
so that Peterson has been proved correct under the hypothesis that the communication specification S_{com} holds:

$$\begin{aligned}
 S_{\text{com}} \triangleq & \neg[\exists i, j. [\text{rf}\langle F2_4^i, \langle 0:, \text{false} \rangle \rangle \vee \text{rf}\langle F2_4^i, \langle 17:, \text{false} \rangle \rangle \\
 & \vee \text{rf}\langle T_5^i, \langle 11:, 1 \rangle \rangle] \wedge [\text{rf}\langle F1_{13}^j, \langle 0:, \text{false} \rangle \rangle \\
 & \vee \text{rf}\langle F1_{13}^j, \langle 8:, \text{false} \rangle \rangle \vee \text{rf}\langle T_{14}^j, \langle 2:, 2 \rangle \rangle]]
 \end{aligned}$$

(preventing the incorrect case)

Soundness and completeness

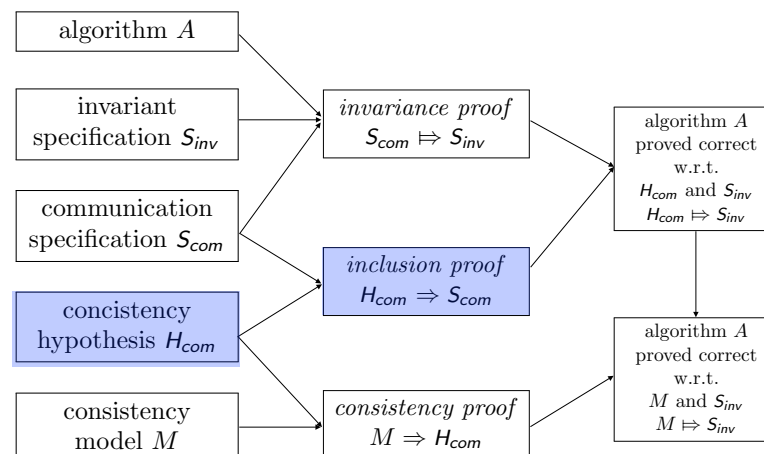
- The invariance proof method is derived from the truly parallel semantics with cuts by **calculational design**
 \Rightarrow **soundness** and (relative) **completeness**



- A consistency specification H_{com} may be less expressive than $S_{com} \Rightarrow incompleteness$ (*)

(*) e.g. hardware cannot restrict a read to input from writes writing odd numbers.

Consistency hypothesis and inclusion proof



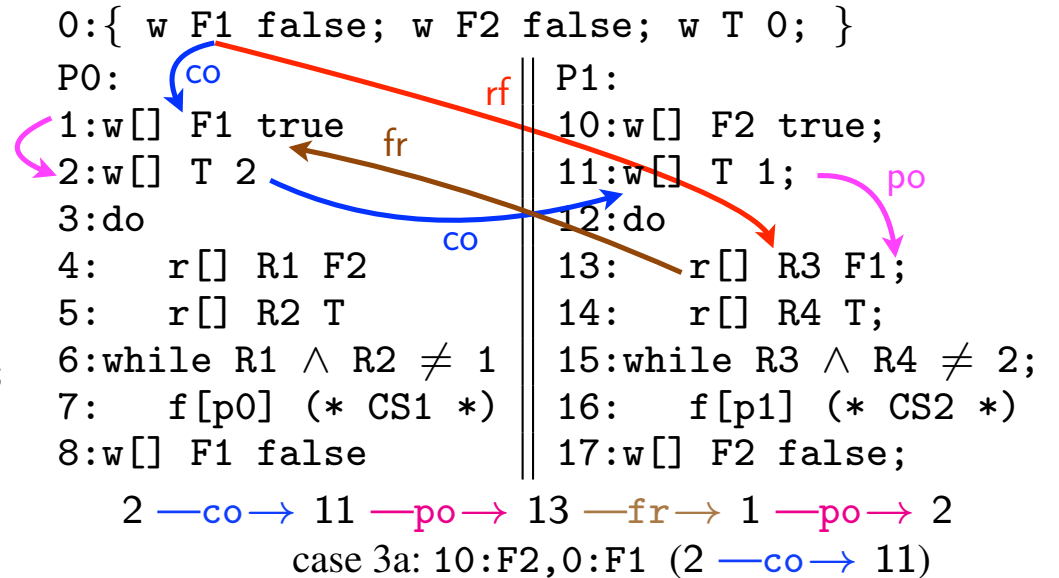
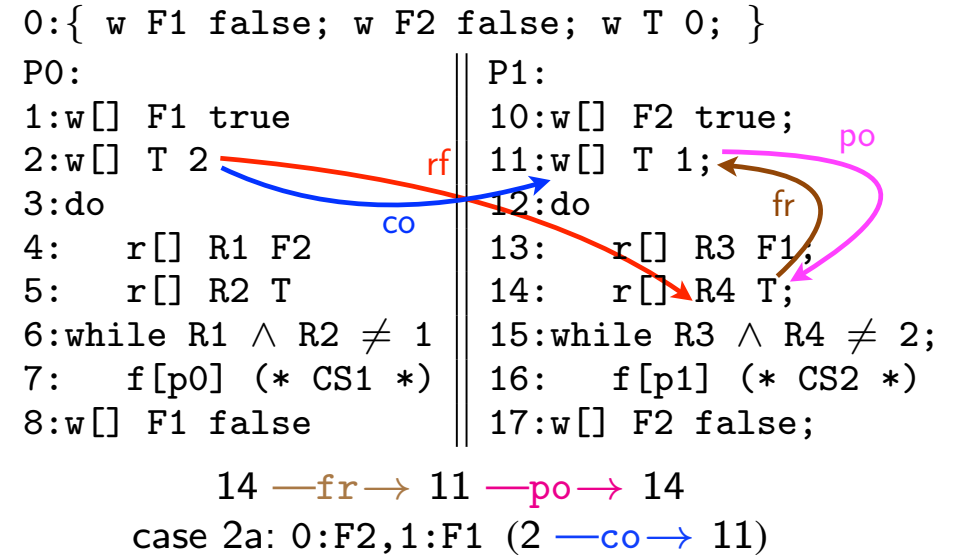
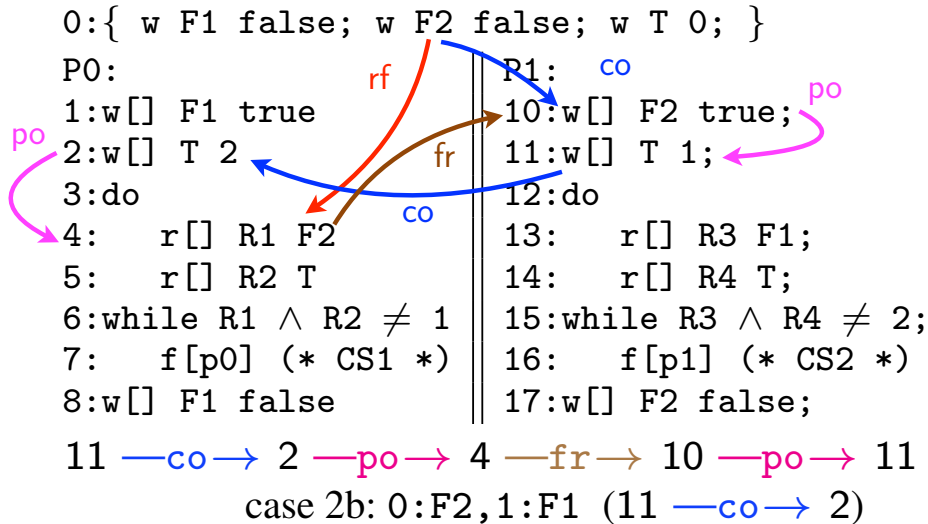
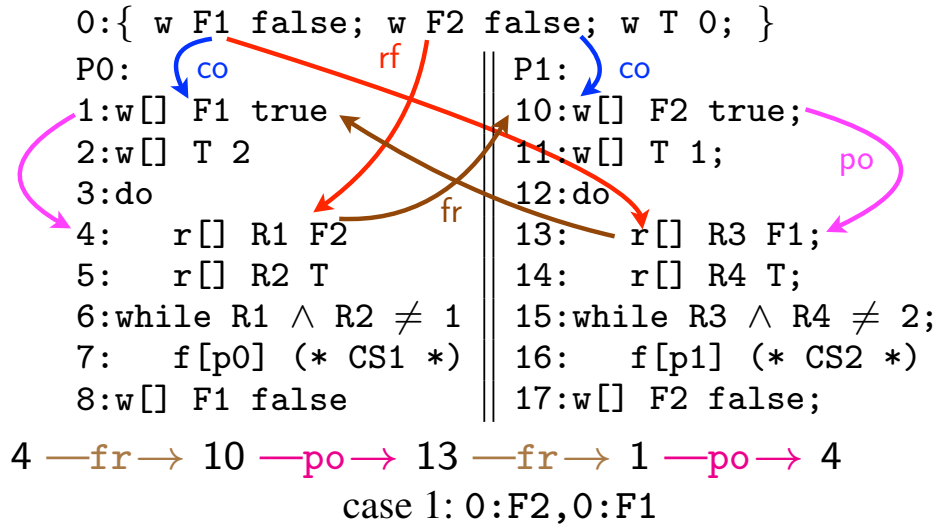
Consistency hypothesis

- The communication specification S_{com} is useful to reason on invariance, but not on machine architecture
- We express S_{com} as a consistency hypothesis H_{com} expressed in the cat language
- H_{com} is derived from S_{com} by calculations design while doing the inclusion proof

Inclusion proof

- Inclusion proof: $\neg S_{com} \Rightarrow \neg H_{com}$
- Calculational design of H_{com} :
 - Calculate all possible execution scenarios violating S_{com}
$$S_{com} \triangleq \neg[\exists i, j. [\text{rf}\langle F2_4^i, \langle 0:, \text{false} \rangle \rangle \vee \text{rf}\langle F2_4^i, \langle 17:, \text{false} \rangle \rangle \vee \text{rf}\langle T_5^i, \langle 11:, 1 \rangle \rangle] \wedge [\text{rf}\langle F1_{13}^j, \langle 0:, \text{false} \rangle \rangle \vee \text{rf}\langle F1_{13}^j, \langle 8:, \text{false} \rangle \rangle \vee \text{rf}\langle T_{14}^j, \langle 2:, 2 \rangle \rangle]]$$
 - Prevent each of them by a cat specification
 - H_{com} is their conjunction

Example: Peterson



Example: Peterson

<pre> 0:{ w F1 false; w F2 false; w T 0; } P0: 1:w[] F1 true 2:w[] T 2 3:do 4: r[] R1 F2 5: r[] R2 T 6:while R1 ∧ R2 ≠ 1 7: f[p0] (* CS1 *) 8:w[] F1 false </pre>	<pre> P1: 10:w[] F2 true; 11:w[] T 1; 12:do 13: r[] R3 F1; 14: r[] R4 T; 15:while R3 ∧ R4 ≠ 2; 16: f[p1] (* CS2 *) 17:w[] F2 false; </pre>
--	---

5 —fr→ 2 —po→ 5

case 3b: 10:F2,0:F1 (11 —co→ 2)

<pre> 0:{ w F1 false; w F2 false; w T 0; } P0: 1:w[] F1 true 2:w[] T 2 3:do 4: r[] R1 F2 5: r[] R2 T 6:while R1 ∧ R2 ≠ 1 7: f[p0] (* CS1 *) 8:w[] F1 false </pre>	<pre> P1: 10:w[] F2 true; 11:w[] T 1; 12:do 13: r[] R3 F1; 14: r[] R4 T; 15:while R3 ∧ R4 ≠ 2; 16: f[p1] (* CS2 *) 17:w[] F2 false; </pre>
--	---

5 —fr→ 2 —po→ 5

case 4b: 10:F2,1:F1 (11 —co→ 2)

<pre> 0:{ w F1 false; w F2 false; w T 0; } P0: 1:w[] F1 true 2:w[] T 2 3:do 4: r[] R1 F2 5: r[] R2 T 6:while R1 ∧ R2 ≠ 1 7: f[p0] (* CS1 *) 8:w[] F1 false </pre>	<pre> P1: 10:w[] F2 true; 11:w[] T 1; 12:do 13: r[] R3 F1; 14: r[] R4 T; 15:while R3 ∧ R4 ≠ 2; 16: f[p1] (* CS2 *) 17:w[] F2 false; </pre>
--	---

14 —fr→ 11 —po→ 14

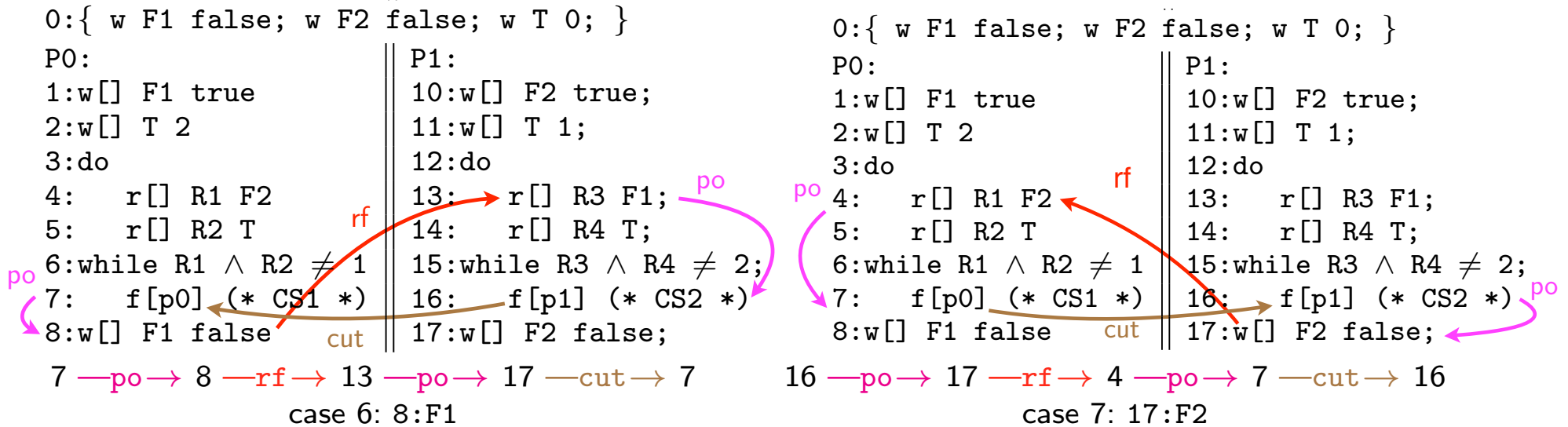
case 4a: 10:F2,1:F1 (2 —co→ 11)

<pre> 0:{ w F1 false; w F2 false; w T 0; } P0: 1:w[] F1 true 2:w[] T 2 3:do 4: r[] R1 F2 5: r[] R2 T 6:while R1 ∧ R2 ≠ 1 7: f[p0] (* CS1 *) 8:w[] F1 false </pre>	<pre> P1: 10:w[] F2 true; 11:w[] T 1; 12:do 13: r[] R3 F1; 14: r[] R4 T; 15:while R3 ∧ R4 ≠ 2; 16: f[p1] (* CS2 *) 17:w[] F2 false; </pre>
--	---

4 —po→ 8 —rf→ 13 —po→ 17 —rf→ 4

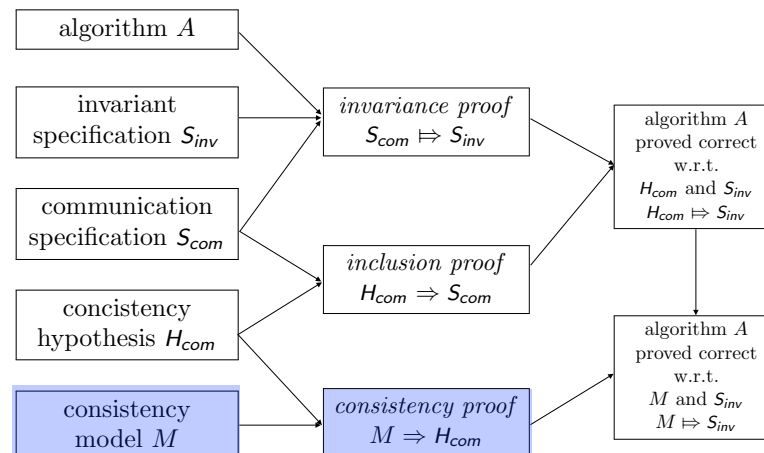
case 5: 17:F2,8:F1

Example: Peterson



- the cut relation can be expressed in cat using tags on fence markers $f[p0]$ and $f[p1]$
- H_{com} is
 - irreflexive $\text{fr}; \text{po}; \text{fr}; \text{po}$
 - irreflexive $\text{fr}; \text{po}$
 - irreflexive $\text{co}; \text{po}; \text{fr}; \text{po}$
 - irreflexive $\text{po}; \text{rf}; \text{po}; \text{rf}$
 - irreflexive $\text{po}; \text{rf}; \text{po}; \text{cut}$

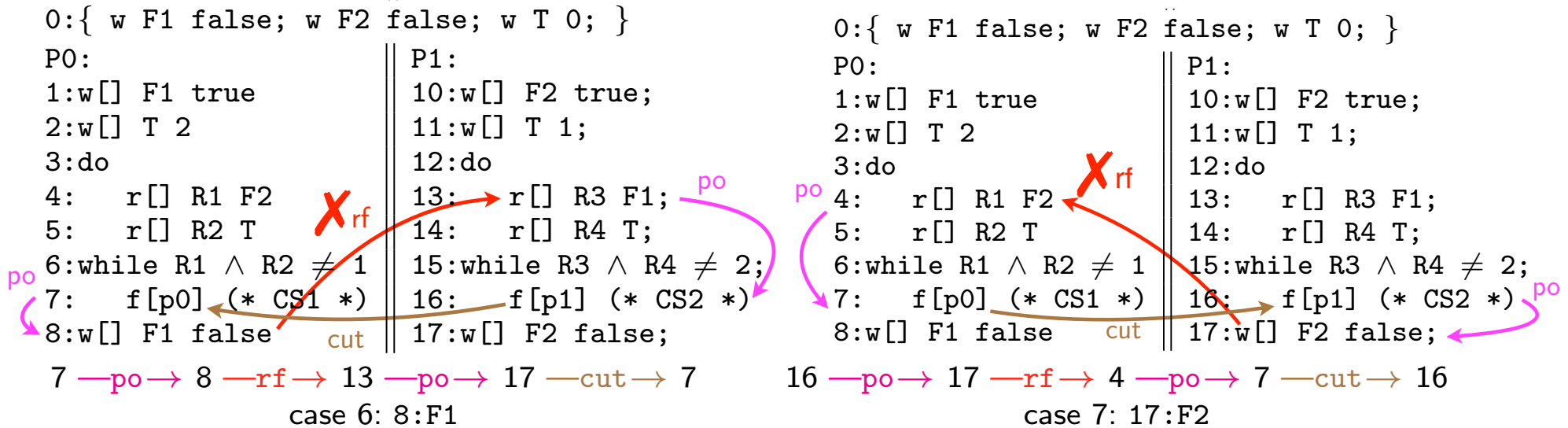
Consistency model and proof



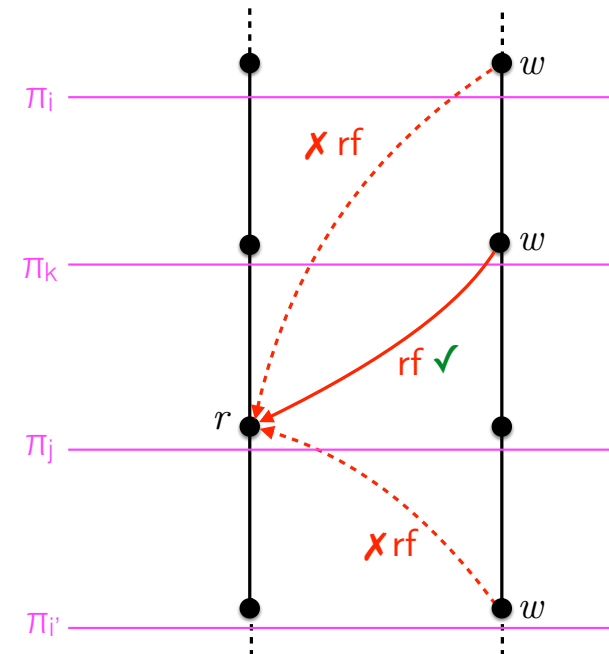
Example: Peterson in SC

- H_{com} is
 - irreflexive fr;po;fr;po
 - irreflexive fr;po
 - irreflexive co;po;fr;po
 - irreflexive po;rf;po;rf
 - irreflexive po;rf;po;cut*
- Sequential consistency in cat:
 - let fr = (rf⁻¹ ; co)
 - acyclic po | rf | co | fr as sc
- Forbid all first 4 cases

Example: Peterson in SC



- The last case follows from the truly parallel execution trace semantics with cuts for sequential consistency



Example: Peterson in TSO

- H_{com} is not forbidden by TSO:

```
let fr = (rf-1;co)
let po-loc = po & loc
acyclic po-loc | rf | co | fr as scpv
let ppo = po \ (W*R)
let rfe = rf & ext
acyclic ppo | rfe | co | fr as tso
```

- For example the case I,

$\langle w_1, r_4 \rangle \in \text{fr} ; \text{po} ; \text{fr} ; \text{po}$

is not forbidden by TSO since $\langle w, r \rangle$ pairs on different variables are excluded from ppo.

Implementation with (weak) cat fences

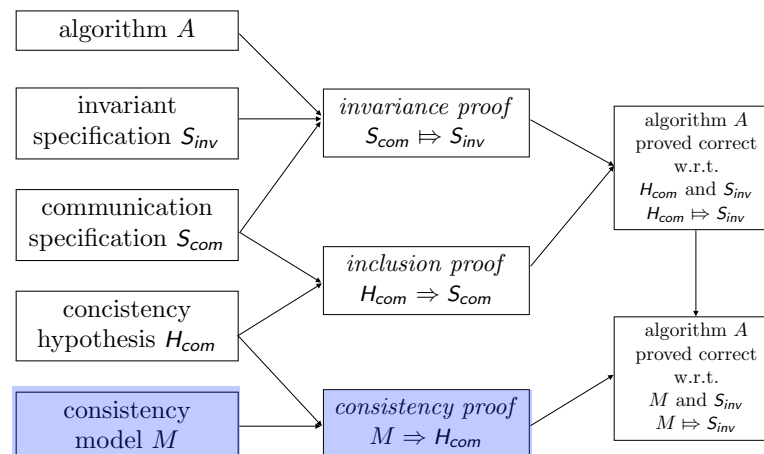
Implementation with fences

```
0:{ F1 = 0; F2 = 0; T = 0; }
1:  w[] F1 1          | 10:  w[] F2 1          ;
2:  w[] T 2           | 11:  w[] T 1           ;
3:  do                | 12:  do                ;
      f[fhw]          |      f[fhw]          ;
4:    r[] r1 F2        | 13: r[] r3 F1        ;
5:    r[] r2 T         | 14: r[] r4 T         ;
6:  while r1  $\wedge$  r2  $\neq$  1 | 15: while r3  $\wedge$  r4  $\neq$  1 ;
7:  (* CS1 *)          | 16: (* CS2 *)          ;
      f[fhw]          |      f[fhw]          ;
8:  w[] F1 0           | L17: w[] F2 0         ;
```

```
let fhw = (po & (__ * F)) ; po
let fre = (rf-1;co) & ext
irreflexive fhw;fre; fhw;fre
...
```

- Invariance proof unchanged (fence = skip)
- Proved to imply the previous fenceless cat specification
- so S_{com} unchanged

consistency proof



Example: Peterson

- The proof is valid for the **virtual machine** defined by the cat specification Peterson
- Porting the algorithm to a **different machine M'** just need refencing (and redoing the proof $M' \Rightarrow H_{cm}$)
- On machine architecture stronger fences have to be used:
 - SC: $\text{fhw} = \text{no fence}$
 - TSO: $\text{fhw} = \text{mfence}$
 - ARM: $\text{fhw} = \text{dbm} \mid \text{dsb}$

Conclusion

Algorithm design methodology

1. Design the algorithm A and its specification S in the sequential consistency model of parallelism
2. Consider the anarchic semantics of algorithm A
3. Add communication specifications S_{com} to restrict anarchic communications and ensure the correctness of A with respect to specification S
4. Do the invariance proof under WCM with S_{com}
5. Infer H_{cm} in cat from S_{com}
6. Prove that the machine memory model M in cat implies H_{cm}

Conclusion

- Modern machines have **complex memory models**
 - ⇒ **portability** has a price (refencing)
 - ⇒ **debugging** is very hard/quasi-impossible
 - ⇒ **proofs** are much harder than with sequential consistency (but still feasible?, mechanically?)
 - ⇒ **static analysis** parameterized by a WCM will be a challenge

The End, Thank You