Ogre et Pythia: An invariance proof method for weak consistency models

Jade Alglave (MSR-Cambridge, UCL, UK)
Patrick Cousot (NYU, Emer. ENS, PSL)

POPL 2017 18 January 2017

Objective

Example (Peterson)

```
0:{ w F1 false; w F2 false; w T 0; }
                                   21:w[] F2 true;
22:w[] T 1;
1: w[] F1 true
2: w[] T 2
                                   23:do
3: do
                                   25: r[] R3 F1;
26: r[] R4 T;
5: r[] R1 F2
  r[] R2 T
                                 \parallel 27:while R3 \wedge R4 \neq 2;
7: while R1 \wedge R2 \neq 1
                                   28:¬at 8
8: ¬at 28
                                   29:w[] F2 false;
9: w[] F1 false
10:
```

critical section

An invariance proof method for WCMs

- Extend Lamport's invariance proof method for parallel programs from sequentially consistent to weak consistency models so that
 - The weak consistency model is a parameter of the proof
 - We don't have to redo the whole proof when changing the consistency model

Note: Owicki & Gries is Lamport with auxiliary variables instead of programs counters

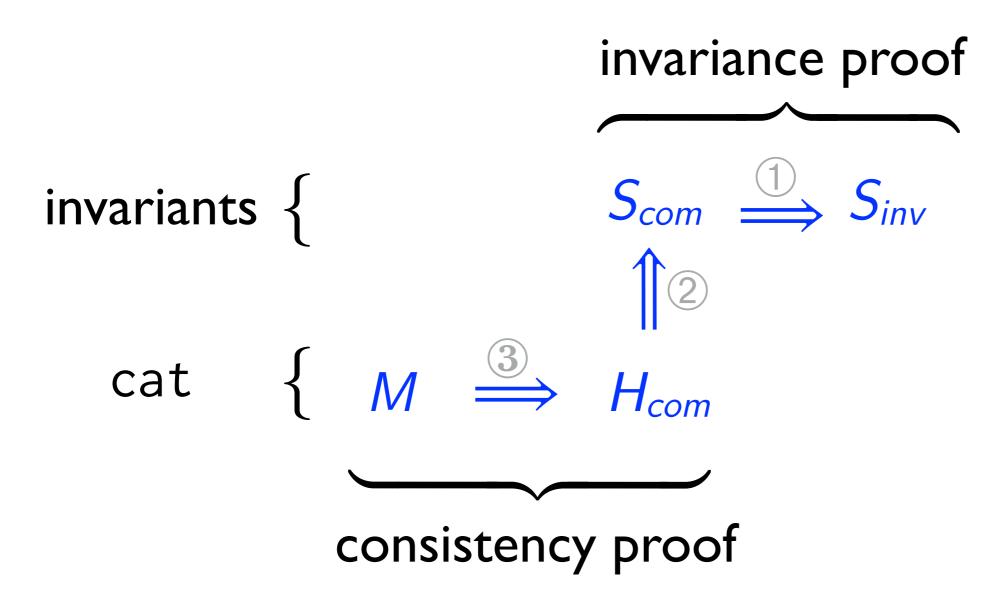
Separating invariance from WCM

- The invariance proof (that a specification S_{inv} is invariant for a program):
 - Done for a program consistency hypothesis S_{com} :
 - Sufficient for the program to be correct
 - Or better, also necessary for correctness (weakest consistency model)
 - This program consistency hypothesis S_{com} is expressed as an invariant
 - sound and (relatively) complete

Separating invariance from WCM

- Consistency proof:
 - a. The program consistency hypothesis S_{com} is strengthen into H_{com} written in a consistency specification language (e.g. cat)
 - b. A cat architecture consistency model M is shown to imply the cat program consistency model H_{com}
- only b. to be redone when changing the architecture
- sound but possibly incomplete

Methodology

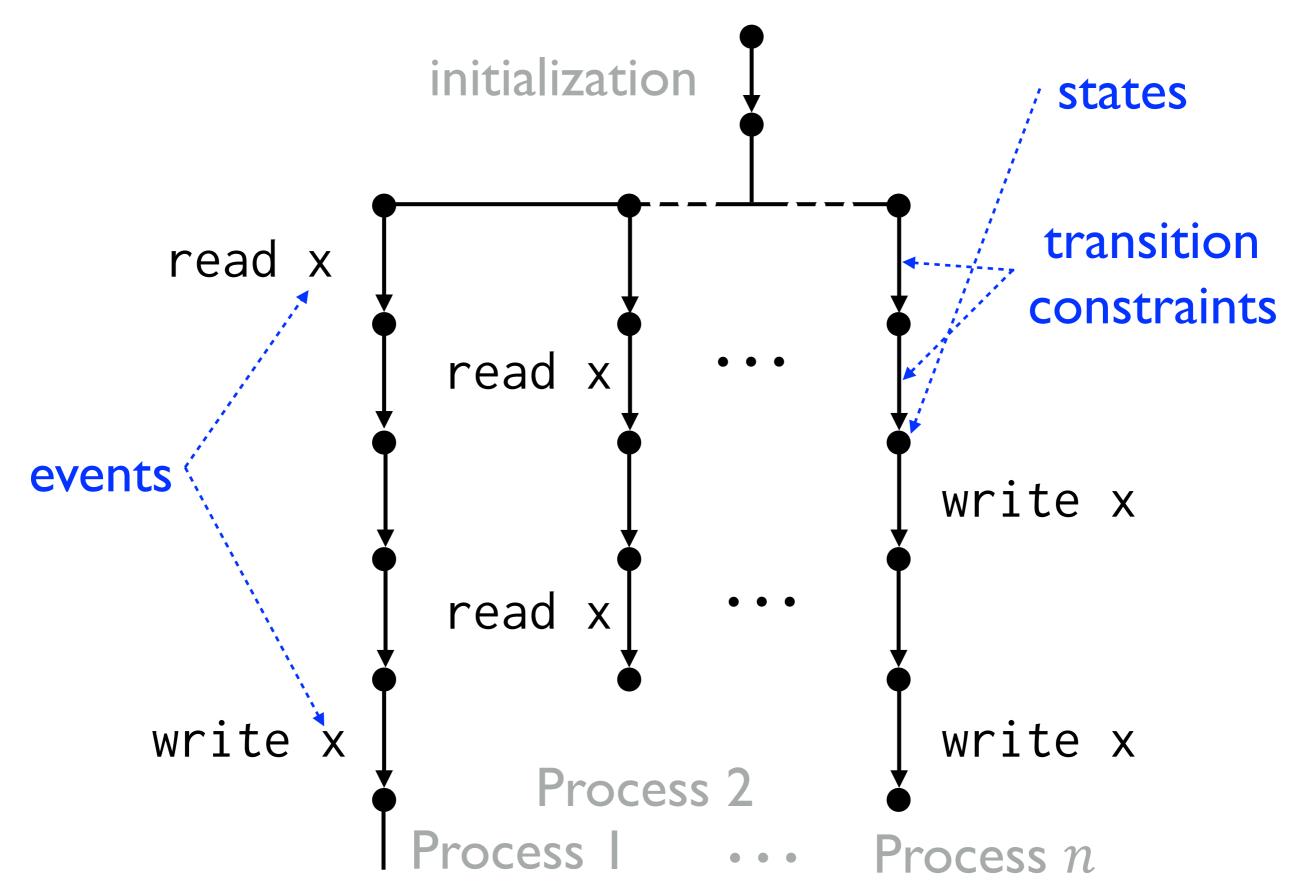


The invariance proof method is designed by abstract interpretation of an *analytic* semantics

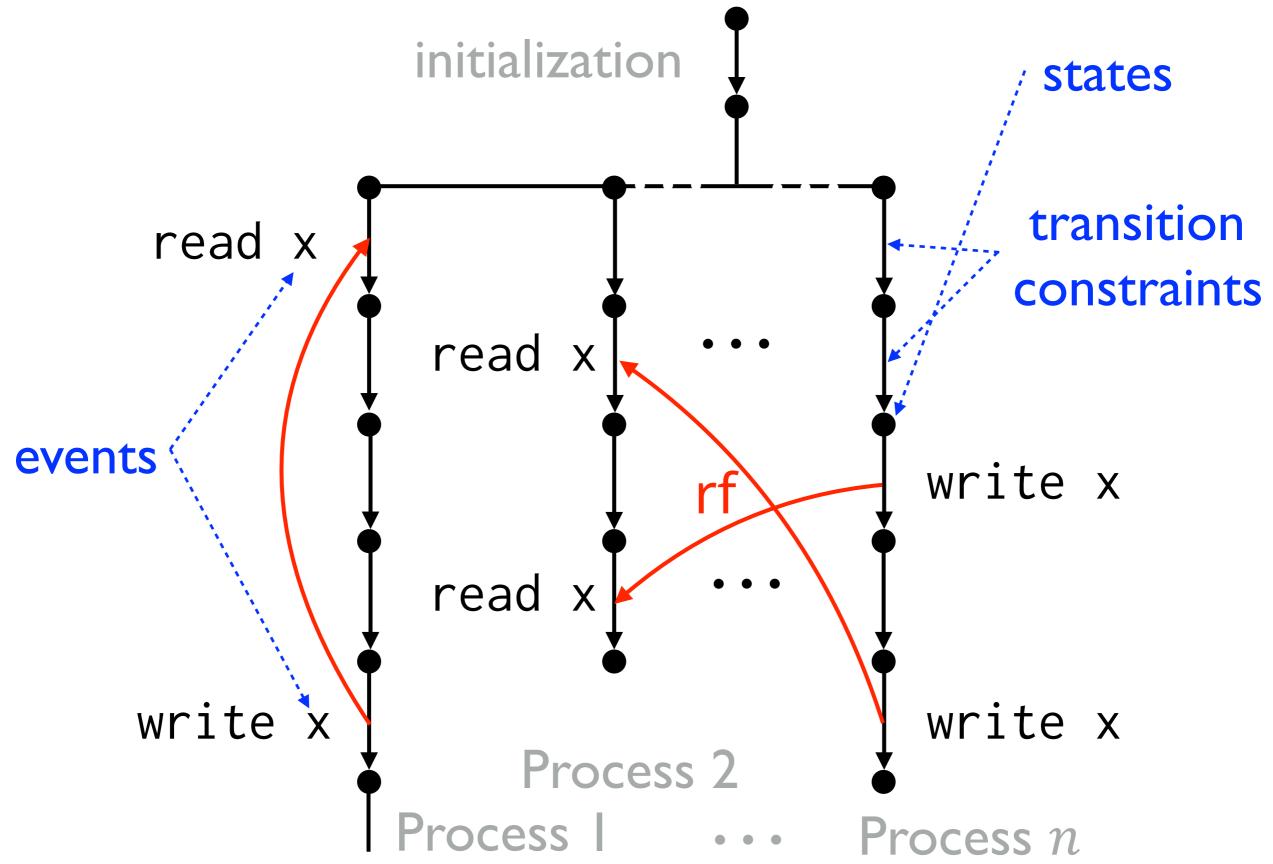
Analytic semantics Anarchic semantics Weak consistency model

The anarchic semantics

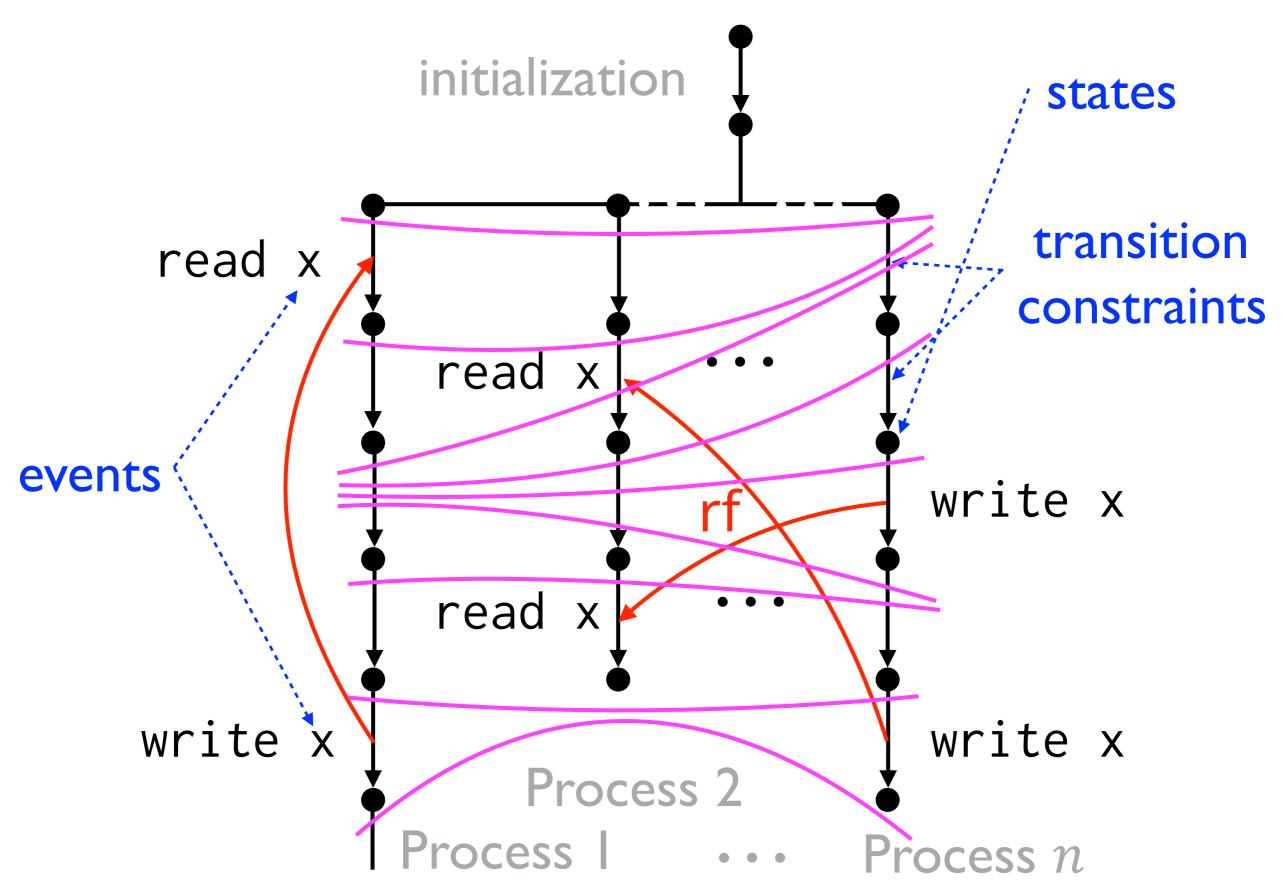
The anarchic semantics



The read-from relation rf



Cuts



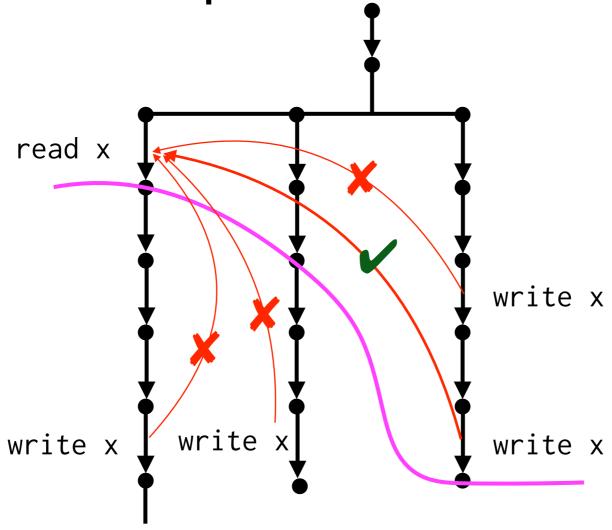
Anarchic semantics of fences

- The anarchic semantics of (localized) fences is skip (the state is unmodified)
- Fences are static marker events used by the WCM in cat to restrict the read-from relation rf

The weak consistency model

Weak consistency models

- Put restrictions on the read-from relation rf
- e.g. sequential consistency: a read at a cut reads from that last write in a process before that cut



Difficulties

Naming entities

- Invariants are logical formulæ
- can only describe entities that they name
- L/O-G use the name of shared variables to designate their current value in invariants

Naming entities

- Invariants are logical formulæ
- can only describe entities that they name
- L/O-G use the name of shared variables to designate their current value in invariants

Difficulty

 Meaningless with WCMs since there is no notion of "the current value of a shared variable"

What is known on communications?

- Each process only knows the value of the shared variables from its last read
- Need to be named → Pythia Variables

What we know on communications?

- Each process only knows the value of the shared variables from its last read
- Need to be named → Pythia Variables

Difficulty

- Its dynamic, not static!
- A program read action can read from a different write each time it is executed → Stamps (abstraction of local time)

Back to the anarchic semantics

State

- Per process:
 - A stamp (local time, no global time)
 - A program counter
 - The value of the local variables (registers) of the process
 - The stamped pythia variables (uniquely identifying all reads along a trace)
 - The value of the pythia variables (what was read)
- The read-from relation (rf)

Example (Peterson)

```
0:{ w F1 false; w F2 false; w T 0; }
 P0:
                                          |10:w[] F2 true;
 1:w[] F1 true
                                          11:w[] T 1;
 2:w[] T 2
 3:do \{i\}
                                          | 12:do \{j\}
                                         | 13: \dot{r}[] R3 F1; \{ \leadsto \text{F1}_{13}^{j} \}
| 14: \dot{r}[] R4 T; \{ \leadsto \text{T}_{14}^{j} \}
 4: r[] R1 F2 \{ \rightsquigarrow F2_4^i \}
 5: r[] R2 T \{ \rightsquigarrow T_5^i \}
                                          | 15: while R3 \wedge R4 \neq 2; \{j_{	ext{end}}\}
 6: while R1 \wedge R2 \neq 1 \{i_{\text{end}}\}
 7:skip (* CS1 *)
                                           16:skip (* CS2 *)
                                           17:w[] F2 false;
 8:w[] F1 false
Stamps (loop counters)
```

Stamps (on loop exit)

© J. Alglave & P. Cousot

Example (Peterson)

```
0:{ w F1 false; w F2 false; w T 0; }
 P0:
                                         10:w[] F2 true;
 1:w[] F1 true
                                         11:w[] T 1;
 2:w[] T 2
 3:do \{i\}
                                          12:do \{j\}
                                        13: \mathbf{r}[] R3 F1; \{ \leadsto \mathbf{F1}_{13}^{j} \}
14: \mathbf{r}[] R4 T; \{ \leadsto \mathbf{T}_{14}^{j} \}
 4: r[] R1 F2 \{ \rightsquigarrow F2_4^i \}
 5: r[] R2 T \{ \rightsquigarrow T_5^i \}
 6: while R1 \wedge R2 \neq 1 \{i_{\rm end}\} | 15: while R3 \wedge R4 \neq 2; \{j_{\rm end}\}
                                          16:skip (* CS2 *)
 7:skip (* CS1 *)
                                           17:w[] F2 false;
 8:w[] F1 false
                                                    Pythia variables
Stamps (loop counters)
```

Stamps (on loop exit)

The abstraction

For each process

- For each process
 - For each program point of that process

- For each process
 - For each program point of that process
 - For each execution of the program

- For each process
 - For each program point of that process
 - For each execution of the program
 - For each cut of that execution going through the program point of that process

- For each process
 - For each program point of that process
 - For each execution of the program
 - For each cut of that execution going through the program point of that process

collect:

- For each process
 - For each program point of that process
 - For each execution of the program
 - For each cut of that execution going through the program point of that process

collect:

The states of all processes, and

- For each process
 - For each program point of that process
 - For each execution of the program
 - For each cut of that execution going through the program point of that process

collect:

- The states of all processes, and
- The read-from relation rf

Example: Peter

```
0: { w F1 false; w F2 false; w T 0; }
\{F1=false \land F2=false \land T=0\} \}
1: \{R1=0 \land R2=0\}
     w[] F1 true
2: \{R1=0 \land R2=0\}
     w[] T 2
3: \{R1=0 \land R2=0\}
     do \{i\}
4: \{(i=0 \land R1=0 \land R2=0) \lor
        (i>0 \wedge R1=F2_{\Delta}^{i-1} \wedge R2=T_{\Xi}^{i-1})
            r[] R1 F2 \{ \rightsquigarrow F2_4^i \}
5: \{R1=F2_4^i \land (i=0 \land R2=0) \lor
                      (i>0 \wedge R2=T_{5}^{i-1})
            r[] R2 T \{ \rightsquigarrow \mathsf{T}_5^i \}
6: \{R1=F2_4^i \land R2=T_5^i\}
     while R1 \wedge R2\neq1 {i<sub>end</sub>}
7: \{\neg F2_4^{i \text{end}} \lor T_5^{i \text{end}} = 1\}
     skip (* CS1 *)
8: \{\neg F2^{i_{end}} \lor T^{i_{end}} = 1\}
     w[] F1 false
9: \{\neg F2_4^{i_{\text{end}}} \lor T_5^{i_{\text{end}}} = 1\}
```

```
10: \{R3=0 \land R4=0\}
w[] F2 true; before/after which actions of any (other) process. In absence at 11: {R3=0 \( \) R4=0 } \( \) R4=0 for one process. In absence of loops events are
      w[] T 1;
12: \{R3=0 \land R4=0\}
      do \{j\}
13: \{(j=0 \land R3=0 \land R4=0) \lor
         (j>0 \land R3=F1_{13}^{j-1} \land R4=T_{14}^{j-1})
             r[] R3 F1 \{ \sim \text{F1}_{13}^{j} \};
                                                        3.4.6 • Ab
14: \{R3=F1_{13}^{j} \land (j=0 \land R4=0)\} \lor 
                          (j>0 \land R4=T_{14}^{j}
                                                        The abstra
             r[] R4 T; \{ \rightsquigarrow T_{14}^{\jmath} \}
15: \{R3=F1_{13}^{j} \land R4=T_{14}^{j}\}
      while R3 \wedge R4\neq2 {j<sub>end</sub>};
                                                                The abs
16: \{\neg F1_{13}^{j \text{ end}} \lor T_{14}^{j \text{ end}} = 2\}
      skip (* CS2 *)
                                                        3.40(H)a
17: \{\neg F1_{13}^{j \text{ end}} \lor T_{14}^{j \text{ end}} = 2\}
      w[] F2 false;
                                                        The candid
18: \{\neg F1_{13}^{j \text{ end}} \lor T_{14}^{j \text{ end}} = 2\}
```

Example: Peter

```
0: { w F1 false; w F2 false; w T 0; }
\{F1=false \land F2=false \land T=0\} \}
1: \{R1=0 \land R2=0\}
    w[] F1 true
2: \{R1=0 \land R2=0\}
    w[] T 2
```

```
10: {R3=0 ∧ R4=0
     w[] F2 true; before/after which
11: {R3=0 \ R4=0} in figure ??. After execution of the prelug
                       of one process. In absence of loops events ar
    w[] T 1;
```

```
3:
            \{(i=0)\}
                                           \wedge R2=0) \vee
                        \wedge R1=0
                (i>0 \wedge R1=F2_{4}^{i-1}
                                                      \wedge R2=T<sub>5</sub><sup>1</sup>
```

```
5: \{R1=F2^{i}_{4} \land (i=0 \land R2=0) \lor \}
                        (i>0 \wedge R2=T_{\kappa}^{i-1})}
             r[] R2 T \{ \rightsquigarrow \mathsf{T}_5^i \}
6: \{R1=F2_4^i \land R2=T_5^i\}
      while R1 \wedge R2\neq1 {i<sub>end</sub>}
7: \{\neg F2^{i}_{A} \lor T^{i}_{5} = 1\}
      skip (* CS1 *)
8: \{\neg F2^{i}_{A} \lor T^{i}_{5} = 1\}
      w[] F1 false
9: \{\neg F2^{i_{\text{end}}} \lor T^{i_{\text{end}}} = 1\}
```

```
14: \{R3=F1_{13}^{j} \land (j=0 \land R4=0)\}
                            (j>0 \land R4=T_{14}^{j}
                                                            The abstra
              r[] R4 T; \{ \rightsquigarrow T_{14}^j \}
15: \{R3=F1_{13}^{j} \land R4=T_{14}^{j}\}
       while R3 \wedge R4\neq2 {j<sub>end</sub>};
16: \{\neg F1_{13}^{j \text{ end}} \lor T_{14}^{j \text{ end}} = 2\}
       skip (* CS2 *)
17: \{\neg F1_{13}^{j \text{end}} \lor T_{14}^{j \text{end}} = 2\}
       w[] F2 false;
18: \{\neg F1_{13}^{j \text{ end}} \lor T_{14}^{j \text{ end}} = 2\}
```

The calculational design of the verification conditions by abstract interpretation

The induction principle

- Given an invariance specification S_{inv} find a stronger inductive invariant S_{ind}
- Prove that S_{ind} satisfy verification conditions
 - Holds after initialization
 - Remains true after a computation step
 - Remains true after a communication
- Assuming S_{com} / H_{com}

The induction principle

- Given an ir inductive ir
- Prove that
 - Holds aft
 - Remains
 - Remains
- Assuming

Verification conditions = abstraction of the concrete transformer for one computation step

nd a stronger

itions

P

e and P. Cousot, Ogre and Pythia

verification conditions

```
\alpha_{\mathsf{inv}}(\alpha_{\mathsf{ana}} \llbracket H_{\mathsf{com}} \rrbracket (S^{\mathsf{a}} \llbracket \mathsf{P} \rrbracket)) \subseteq S_{\mathsf{inv}}
   \Leftrightarrow \alpha_{\mathsf{inv}}(\{\xi \in S^{\mathsf{a}}[\![\mathsf{P}]\!] \mid S[\![H_{com}]\!]\xi = \mathsf{allowed}\}) \overset{.}{\subset} S_{\mathsf{inv}} \ \ \mathsf{def.} \ \alpha_{\mathsf{ana}}[\![H_{com}]\!] \ \mathsf{for} \ \mathsf{for} \ \ \ \mathsf{for} \ \ \ \mathsf{for} \ \
   \Leftrightarrow \alpha_{\mathsf{inv}}(S^{\mathsf{a}}\llbracket \mathtt{P} \rrbracket \cap \{\xi \in S^{\mathsf{a}}\llbracket \mathtt{P} \rrbracket \mid S\llbracket H_{com} \rrbracket \xi = \mathsf{allowed} \}) \overset{.}{\subset} S_{\mathsf{inv}} \quad \mathsf{def.} \cap \mathsf{s}
   \Leftrightarrow \alpha_{\mathsf{inv}}(S^{\mathsf{a}}\llbracket \mathtt{P} \rrbracket) \cap \alpha_{\mathsf{inv}}(\{\xi \in \Xi \mid S \llbracket H_{com} \rrbracket \xi = \mathtt{allowed} \}) \overset{.}{\subset} S_{\mathsf{inv}}
                                                                                                                                                                                                                                                         \langle \text{since } \alpha_{\text{inv}} \text{ preserves intersections} \rangle
   \Leftrightarrow \alpha_{\mathsf{inv}}(S^{\mathsf{a}}\llbracket P \rrbracket) \dot{\cap} \alpha_{\mathsf{inv}}(\alpha_{\mathsf{ana}}\llbracket H_{\mathsf{com}} \rrbracket (S^{\mathsf{a}}\llbracket P \rrbracket)) \dot{\subseteq} S_{\mathsf{inv}} \qquad \mathsf{def.} \ \alpha_{\mathsf{ana}}\llbracket H_{\mathsf{com}} \rrbracket \mathsf{s}
   \Leftrightarrow \exists S_{com} : \alpha_{inv}(S^{a} \llbracket P \rrbracket) \dot{\cap} S_{com} \dot{\subseteq} S_{inv} \wedge \alpha_{inv}(\alpha_{ana} \llbracket H_{com} \rrbracket (S^{a} \llbracket P \rrbracket)) \dot{\subseteq} S_{com}
                       (\Leftarrow) For soundness, we have \alpha_{\mathsf{inv}}(S^{\mathsf{a}}\llbracket \mathsf{P} \rrbracket) \dot{\cap} \alpha_{\mathsf{inv}}(\alpha_{\mathsf{ana}}\llbracket H_{\mathsf{com}} \rrbracket (S^{\mathsf{a}}\llbracket \mathsf{P} \rrbracket))
                       \dot{\subseteq} \alpha_{\mathsf{inv}}(S^{\mathsf{a}}\llbracket \mathsf{P} \rrbracket) \dot{\cap} S_{\mathsf{com}} \dot{\subseteq} S_{\mathsf{inv}};
                    (\Rightarrow) For completeness, we choose to describe exactly the communica-
                   tions that is S_{com} = \alpha_{inv}(\alpha_{ana}[H_{com}](S^a[P])).
\Leftrightarrow \exists S_{com} . (S_{com} \Rightarrow S_{inv}) \land (H_{com} \Rightarrow S_{com})
                   by defining the conditional invariance proof S_{com} \Rightarrow S_{inv} to be
                    \alpha_{\text{inv}}(S^{a}[P]) \cap S_{com} \subseteq S_{inv} and the inclusion proof H_{com} \Rightarrow S_{com} to
                    be \alpha_{\mathsf{inv}}(\alpha_{\mathsf{ana}} \llbracket H_{\mathsf{com}} \rrbracket (S^{\mathsf{a}} \llbracket \mathsf{P} \rrbracket)) \subseteq S_{\mathsf{com}}.
```

 $\alpha_{\mathsf{inv}}(P) \triangleq \prod_{p \in \mathbb{P}^n_{\ell} \ell \in \mathbb{P}} \bigcap_{\substack{(S) \in P \\ \ell \in \mathbb{P}}} \bigcap_{\substack{(S) \in P \\ \ell \in P \\ \ell \in P}} \bigcap_{\substack{(S) \in P \\ \ell \in$

e and P. Cousot, Ogre and Pythia

verification conditions

```
\alpha_{\mathsf{inv}}(\alpha_{\mathsf{ana}}\llbracket H_{\mathsf{com}}\rrbracket(S^{\mathsf{a}}\llbracket P\rrbracket)) \subseteq S_{\mathsf{inv}}
\Leftrightarrow \alpha_{\mathsf{inv}}(\{\xi \in S^{\mathsf{a}}\llbracket P\rrbracket \mid S\llbracket H_{\mathsf{com}}\rrbracket \xi = \mathsf{allowed} \}) \subseteq S_{\mathsf{inv}} \ (\mathsf{def.} \ \alpha_{\mathsf{ana}}\llbracket H_{\mathsf{com}}\rrbracket)
\Leftrightarrow \alpha_{\mathsf{inv}}(S^{\mathsf{a}}\llbracket P\rrbracket \cap \{\xi \in S^{\mathsf{a}}\llbracket P\rrbracket \mid S\llbracket H_{\mathsf{com}}\rrbracket \xi = \mathsf{allowed} \}) \subseteq S_{\mathsf{inv}} \ (\mathsf{def.} \ \cap)
\Leftrightarrow \alpha_{\mathsf{inv}}(S^{\mathsf{a}}\llbracket P\rrbracket) \cap \alpha_{\mathsf{inv}}(\{\xi \in \Xi \mid S\llbracket H_{\mathsf{com}}\rrbracket \xi = \mathsf{allowed} \}) \subseteq S_{\mathsf{inv}}
```

$$\Leftrightarrow \exists S_{com} \ . \ (S_{com} \Rightarrow S_{inv}) \land (H_{com} \Rightarrow S_{com})$$

```
(\Rightarrow) For completeness, we choose to describe exactly the communications that is S_{com} = \alpha_{inv}(\alpha_{ana}[H_{com}](S^a[P])). S_{com} : (S_{com} \Rightarrow S_{inv}) \land (H_{com} \Rightarrow S_{com}) by defining the conditional invariance proof S_{com} \Rightarrow S_{inv} to be \alpha_{inv}(S^a[P]) \cap S_{com} \subseteq S_{inv} and the inclusion proof H_{com} \Rightarrow S_{com} to be \alpha_{inv}(\alpha_{ana}[H_{com}](S^a[P])) \subseteq S_{com}.
```

Verification conditions

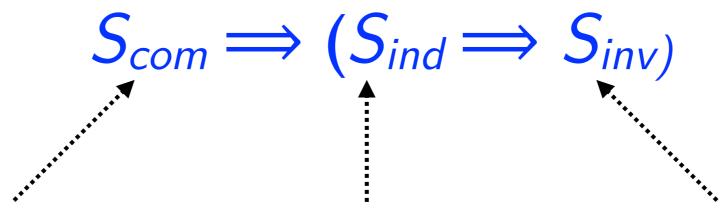
- Sequential proof
- Non-interference proof (like L/O-G but for different kind of invariants)
- Communication proof
 - a read event reading from a write event must be in rf
 - the value read for a variable is the one written
 - reading is fair in rf (cannot be delayed indefinitely)
 - ...

(useless in L/O-G since rf is fixed)

The program consistency hypothesis S_{com}

Communication hypothesis S_{com}

 A sufficient communication hypothesis can be discovered by calculational design:



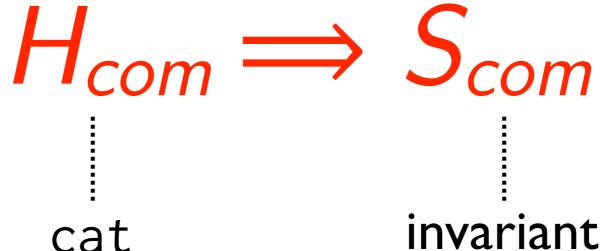
Communication hypothesis

anarchic invariant

Program inductive Program invariance specification

- i.e. (Sind $\land \neg Sinv$) $\Longrightarrow \neg Scom$
- Necessary: by counter examples

Proving Consistency



Proof method

Obtained by calculational design:

$$\begin{split} &\alpha_{\mathsf{inv}}(\alpha_{\mathsf{ana}}\llbracket H_{\mathsf{com}}\rrbracket(S^{\mathsf{a}}\llbracket P\rrbracket)) \stackrel{.}{\subseteq} S_{\mathsf{com}} \\ &\Leftrightarrow \alpha_{\mathsf{inv}}(S^{\mathsf{ana}}\llbracket H_{\mathsf{com}}\rrbracket P) \stackrel{.}{\subseteq} S_{\mathsf{com}} \\ &\Leftrightarrow \forall \xi \in S^{\mathsf{ana}}\llbracket H_{\mathsf{com}}\rrbracket P \cdot \alpha_{\mathsf{inv}}(\{\xi\}) \stackrel{.}{\subseteq} S_{\mathsf{com}} \\ &\Leftrightarrow \forall \xi \in S^{\mathsf{ana}}\llbracket H_{\mathsf{com}}\rrbracket P \cdot \bigcap_{n} \bigcap_{i} \{\alpha_{\mathsf{inv}}(\xi')_p(\mathsf{L}) \mid \xi' \in \{\xi\}\} \stackrel{.}{\subseteq} S_{\mathsf{com}} \\ &\Leftrightarrow \forall \xi \in S^{\mathsf{ana}}\llbracket H_{\mathsf{com}}\rrbracket P \cdot \bigcap_{p=1} \{\alpha_{\mathsf{inv}}(\xi')_p(\mathsf{L}) \mid \xi' \in \{\xi\}\} \stackrel{.}{\subseteq} S_{\mathsf{com}} \\ &\Leftrightarrow \forall (\tau_{\mathsf{start}} \times \prod_{p=0}^{n-1} \tau_p \times \pi \times \mathsf{rf}) \in S^{\mathsf{ana}}\llbracket H_{\mathsf{com}}\rrbracket P \cdot \forall p \in [1,n] \cdot \forall \mathsf{L} \in \mathsf{P}_p \cdot \\ &\alpha_{\mathsf{inv}}(\tau_{\mathsf{start}} \times \prod_{p=0}^{n-1} \tau_p \times \pi \times \mathsf{rf})_p(\mathsf{L}) \subseteq S_{\mathsf{com}_p}(\mathsf{L}) \\ &\lozenge \mathsf{def.} \in , \bigcup, \subseteq, \text{ and } S^{\mathsf{ana}}\llbracket H_{\mathsf{com}}\rrbracket P \text{ so that } \xi \text{ has the form } \xi = \\ &\tau_{\mathsf{start}} \times \prod_{p=0}^{n-1} \tau_p \times \pi \times \mathsf{rf} \cdot \mathsf{By def.} \ (19) \text{ of } \alpha_{\mathsf{inv}} \text{ and } \subseteq, \text{ we get} \\ &\Leftrightarrow \forall (\tau_{\mathsf{start}} \times \prod_{p=0}^{n-1} \tau_p \times \pi \times \mathsf{rf}) \in S^{\mathsf{ana}}\llbracket H_{\mathsf{com}}\rrbracket P \cdot \forall i \in (20) \\ &[1,n] \cdot \forall \mathsf{L} \in \mathsf{P}_p \cdot \forall q \in [0,n[\cdot,\forall k_q < |\tau_q|\cdot (\tau_{q}_{\mathsf{l}_q} = s\langle \kappa_{q,k_q},\theta_{q,k_q},\rho_{q,k_q},\rho_{q,k_q}\rangle \wedge \kappa_{p,k_p} = \mathsf{L}) \Rightarrow \\ &\langle \kappa_{0,k_0},\theta_{0,k_0},\rho_{0,k_0},\rho_{0,k_0},\dots,\nu_{p-1,k_{p-1}},\theta_{p,k_p},\rho_{p,k_p},\nu_{p,k_p},\kappa_{p+1,k_{p+1}},\dots,\kappa_{n-1,k_{n-1}},\theta_{n-1,k_{n-1}},\rho_{n-1,k_{n-1}},\nu_{n-1,k_{n-1}},\mathsf{rf}\rangle \in S_{\mathsf{com}_i}(\mathsf{L}) \end{aligned}$$

Proof method

- The anarchic invariants can be used to calculate all communication scenarios violating S_{com}
- These scenarios must be forbidden by the cat specification H_{com}

(no need to reason at the level of traces of the anarchic semantics)

pythia variable. More precisely a read ℓ : r reading the variable x and placing its result decorated with the pythia variable $\{ \sim \mathbf{x}_{\ell}^{n} \}$, v the read in the program, x is the variable that and n is an instance of the iteration counter (record all iteration counters of all surrounding

pythia variable. More precisely a read ℓ : r[] R x at line ℓ : reading the variable x and placing its result into register R, is decorated with the pythia variable $\{ \rightsquigarrow \mathbf{x}_{\ell}^{n} \}$, where ℓ is the line of the read in the program, x is the variable that the read is reading and n is an instance of the iteration counter (for nested loops we record all iteration counters of all surrounding loops).

Invariant_specification S_{inv}

2.1.2 Invariant specification S_{inv}

Fig. 6 gives an invariant specification In the the second an invariant specification of our implementation of simultaneously in their critical sections. simultaneously in the

1: {trueSCenarios 7: $\{\neg at\{16\}\}\$ 16: $\{\neg at\{7\}\}\$

Figure 6: Invariant specification for $\frac{1}{2}$ for $\frac{1}$

Fig. 6 gives an invariant specification of our

2.2 Communication specific the son 2.2

The next step in our specification process consists in stating an step in our specification process consists in stating an step in our specification process consists in stating an step in our specification process consists in stating an step in our specification process consists in stating an step in our specification process consists in stating an step in our specification process consists in stating an step in our specification process consists in stating an step in our specification process consists in stating an step in our specification process consists in stating an step in our specification process consists in stating and stati

tion S_{inv} given above in Fig. 6, that both processes cannot be simply about the solution of the solution ultaneously in their critical section). ultaneously in their critical segments contains one of the values 0.1 or 2 indifferently only states as the remains of the values of the values of the remains of the remai To see this, consider Fig. 7a. The plain red arrows are an informal consider Fig. 7a. The plain red arrows are an informal consider Fig. 7a.

representation of a communication scenario where: representation of a colling the sense of F1 or from 8: so that R3 contains false. The reads from 8: so that R3 contains false. The reads from 0:, 11:, or 1: so that R4 contains 0, F1; the sense of F1 or from 0:, 11:, or 1: so that R4 contains 0, F1; the false of F1 or from 0:, 11:, or 1: so that R4 contains 0, F1; the false of F1 or from 0:, 11:, or 1: so that R4 contains 0, F1; the false of F1 or from 8: so that R3 contains 0, F1; the reads from 0:, 11:, or 1: so that R4 contains 0, F1; the false of F1 or from 8: so that R3 contains 0, F1; the reads from 0:, 11:, or 1: so that R4 contains 0; the reads from 0:, 11:, or 1: so that R4 contains 0; the reads from 0:, 11:, or 1: so that R4 contains 0; the reads from 0:, 11:, or 1: so that R4 contains 0; the reads from 0:, 11:, or 1: so that R4 contains 0; the reads from 0:, 11:, or 1: so that R4 contains 0; the reads from 0:, 11:, or 1: so that R4 • on the first process (see the left-hand side), the read at line 14: read noting the first process (see the left-hand side), the read at line 15: process (see the left-hand side), the read at line 15: process (see the left-hand side), the read at line 15: so that R1 that powering the process (see the left-hand side) to at line 10: so that R1 that powering the process (see the left-hand side). reads the value that F2 was initialised to, at line 0:, reads the value that F2 was initialised to, at line 0:, reads the value that F2 was initialised to, at line 0:, reads the value that F2 was initialised to, at line 0:, reads the value that F2 was initialised to, at line 0:, reads the value that F2 was initialised to, at line 0:, reads the value that F2 was initialised to, at line 0:, reads the value that F2 was initialised to, at line 0:, reads the value that F2 was initialised to, at line 0:, reads the value that F2 was initialised to, at line 0:, reads the value that F2 was initialised to the value that F2 was initialised to

contains false. And, the read at line 5: reads from any write of alse. And the T, so that R2 contains one of the values 0, 1, or 2, indifferently, R2 contains one of

J. Alglave and P. Cousot, Ogre and Pythia

line 14: reads from 0:, 11:, or 1: J. Alglave and P. Cousot, Ogre and Principality.

In this e sinuation so which and finipossible under SC), both loop exits e sinuation so the prospect of the sinuation of the prospect of the p

8:w[] F1 false 17:w[] F2 false; (b) Incorrect turns

Figure 7: Incorrections of a Peterson algorithm with

F2 and Fhirs variable reMe But effects & Portlads Tin R1xfathing this situation impossible under SC) the turns are wrong, the read in the program, x is the variable that the read is read both processes can be simultaneously in their (rottlesses from invalidating the expecification of solutions of solutions and solution invalidations that the expecification of solutions are solutions and solutions are solutions and solutions are solutions.

2.2.2.1.2 Invariant specification S_{com}

Fig. 6 gives an invariant specification of our implementation. Let us express the communication scenarios depicted in Fig. 6 gives an invariant specification states that both processes cannot be specification states.

the readerth much this per crossius in again Peterson's algorithm readerth much this per crossius in and the angular peterson's algorithm readerth much this per crossius in a contraction of the contracti Communication specification specification as follows we are file to the communication as follows we are filed to the communication as follows as follows are the communication as follows as follows are the communication as follows as follows are the communication are the communication as follows are the communication as follows are the communication a

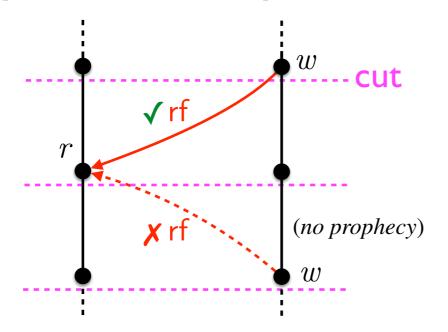
The next step in our specification process consists in specification S_{com} , expressing which interpreted in our specification S_{com} , expressing which interpreted in our specification S_{com} , expressing which interpreted in the process communications are allowed for the algorithm f_{com} and f_{com} and f_{com} are allowed for the algorithm f_{com} and f_{com} and f_{com} are allowed for the algorithm f_{com} and f_{com} are algorithm f_{com} and f_{com} are algorithm f_{com} and f_{com} are al

indifferentlyF180740000755886620000

conditions can be true so that both processes can be simultaneously tions can be figure with the state of the sound of the in their critical section, thus invalidating the specification in their critical receipts in the specification in Another erroneous behaviour is illustrated in Fig. 7b. The value of Another erroneous behaviour is illustrated in Fig. 7b. The value of Another erroneous behaviour is illustrated in Fig. 7b. The value of Another erroneous behaviour is illustrated in Fig. 7b. F2 and F1 is indifferent. But process P0 reads T in R1 from the write F2 and F1 is indifferent. But process P0 reads T in R1 from the write F2 and F1 is indifferent. But process P0 reads T in R1 from the write F2 and F1 is indifferent. 11: so R1=1 while P1 reads T in R4 from the write 2: so R4=12: In R1=1 while P1 reads T in R4 from the write 2: so R4=12: In R1=1 while P1 reads T in R4 from the write 2: from R4=12: In R1=1 while P1 reads T in R4 from the write 2: so R4=12: In R1=1 while P1 reads T in R4 from the write 2: so R4=12: In R1=1 while P1 reads T in R4 from the write 2: so R4=12: In R1=1 while P1 reads T in R4 from the write 2: so R4=12: In R1=1 while P1 reads T in R4 from the write 2: so R4=12: In R1=1 while P1 reads T in R4 from the write 2: so R4=12: In R1=1 while P1 reads T in R4 from the write 2: so R4=12: In R1=1 while P1 reads T in R4 from the write 2: so R4=12: In R1=1 while P1 reads T in R4 from the write 2: so R4=12: In R1=1 while P1 reads T in R4 from the write 2: so R4=12: In R1=1 while P1 reads T in R4 from the write 2: so R4=12: In R1=1 while P1 reads T in R4 from the write 2: so R4=12: In R1=1 while P1 reads T in R4 from the write 2: so R4=12: In R1=1 while P1 reads T in R4 from the write 2: so R4=12: In R1=1 while P1 reads T in R4 from the write 2: so R4=12: In R1=1 while P1 reads T in R4 from the write 2: so R4=12: In R1=1 while P1 reads T in R4 from the write 2: so R4=12: In R4 from the write 2: so R4 from t this situation (impossible under SC) the turns are wrong, so that this situation (impossible under SC) the turns are wrong, so that this situation (impossible under SC) the abstract domains to the abstract domains are wrong. both processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in their critical section, again processes can be simultaneously in the section of the se

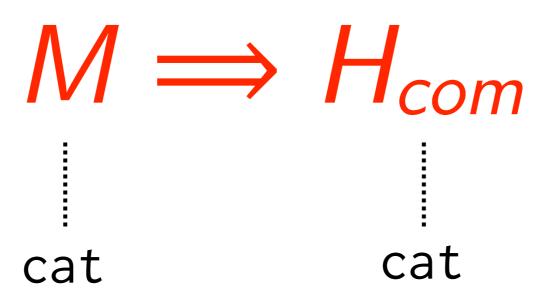
Incompleteness

- In general you have to add fences for H_{com} (do not change the invariants, S_{inv} , S_{ind} , and S_{com} remain valid)
- S_{com} can refer to communicated values not H_{com} in cat (redesign your algorithm without assuming that the hardware does know about communicated values)
- cat may not be expressive enough:



No read beyond cut

Proving Architectural Consistency



$H_{com} \Longrightarrow M$ in cat

- sound and complete proof method
- unpublished paper of JA and PC with Luc Maranget

Beyond L/O-G: non-starvation

Reasoning on one execution only

- A particular execution can be uniquely characterized by its read-from relation rf
- We can reason on one execution only (Scom for this execution + Sind)
- Not directly possible with L/O-G
- Can be used to prove non-starvation

Non-starvation (e.g. PostgrSQL)

- Consider all traces that may starve (for an appropriate S'_{com} for each trace)
- Prove each of them to be infeasible:
 - the inductive invariant S_{ind} under the program communication hypothesis S_{com} is unsatisfied
 - or, by strengthening the program communications S_{com} (maybe implemented by adding fences in H_{com})
 - or, by a fairness hypothesis.

Communication fairness hypothesis®

- All writes eventually hit the memory:
 - If, at a cut of the execution, all the processes infinitely often write the same value υ to a shared variable x and only that value υ
 - and from a later cut point of that execution, a
 process infinitely often repeats reads to that variable
 - ullet then the reads will end up reading that value υ

^(*) The SPARC Architecture Manual, Version 8, Section K2, p. 283: ``if one processor does an S, and another processor repeatedly does L 's to the same location, then there is an L that will be after the S".

Conclusion

Conclusion

- To design a correct parallel algorithm, specify:
 - the algorithm
 - the invariance specification S_{inv}
 - the required program consistency model S_{com}
- Find an anarchic inductive invariant S_{ind} satisfying the verification conditions such that $(S_{com} \land S_{inv}) \Longrightarrow S_{inv}$

Conclusion

- To implement a parallel algorithm correctly:
 - Implement the program consistency model on an architecture consistency model M (possibly adding fences)
 - Prove $M \Longrightarrow S_{com}$
- Or better
 - Find a minimal/weakest H_{com} such that $H_{com} \Longrightarrow S_{com}$
 - \bullet $M \Longrightarrow H_{com}$

More work needed

- Specification of parallel/distributed <u>program</u> consistency models (more refined than <u>architecture</u> consistency models, e.g. cuts needed)
- Liveness (beyond non-starvation)
- Collection of certified algorithms for WCM (e.g. transactional memory, databases, etc)
- Static analysis (by abstract interpretation of the analytic semantics parameterized by a WCM)

The End, Thank You