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 Villebrunier, France
October 4th, 2016
Weakly consistent parallel programs
Weakly consistent parallel programs

\[
\text{var } x_1, \ldots, x_m; \quad // \text{shared variables} \\
P_0; \quad // \text{prelude initializing } x_1, \ldots, x_m \\
[P_1 || P_2 || \ldots || P_n]
\]

- \(P_1, P_2, \ldots, P_n\) are the \text{processes} modifying the shared variables and their \text{local registers} \(R, \ldots\)

- The execution of a \text{write} \(x := E\) to a shared variable and the \text{read} \(R := x\) of a shared variable is \text{not} \text{ instantaneous} (as in sequential consistency)
Example (1b, load buffer)

- **Algorithm A:**

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

  P0

  1: $r[[] \ r1 \ x$

  2: $w[[] \ y \ 1$

  3: 

  P1

  11: $r[[] \ r2 \ y$

  12: $w[[] \ x \ 1$

  13: 

- **Specification $S_{inv}$:**

  \[ \text{at } 3 \land \text{at } 13 \Rightarrow \neg (r1=1 \land r2=1) \]
Example (Peterson)

- **Algorithm A:**

```plaintext
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 ∧ R2 ̸= 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 ∧ R4 ̸= 2;
  16: skip (* CS2 *)
  17: w[] F2 false;
  18: 
```

- **Specification $S_{inv}:**

```plaintext
1: {true} 10: {true}
... 16: {¬at{7}}
7: {¬at{16}} 16: {¬at{7}}
... 9: {true} 18: {true}
```

---

Weak memory/consistency models

- Sequential consistency:

\[
\begin{array}{cccc}
X_1 & \ldots & X_m \\
P_1 & P_2 & \ldots & P_n \\
\end{array}
\]

atomic instantaneous communications

- Weak memory models:

\[
\begin{array}{cccc}
X_1 & \ldots & X_m \\
P_1 & P_2 & \ldots & P_n \\
\end{array}
\]

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)

- \( 0: \{ x = 0; y = 0; \} \)
  
  \[ \begin{align*}
  P0 & \quad | \quad P1 \\
  1:r[] & \quad r1 \quad x \quad | \quad 11:r[] & \quad r2 \quad y; \\
  2:w[] & \quad y \quad 1 \quad | \quad 12:w[] & \quad x \quad 1; \\
  3: & \quad | \quad 13: \\
\end{align*} \]

- \( \text{at } 3 \land \text{at } 13 \land r1=1 \land r2=1 \)

- 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:
  1: w[] F1 true
  2: w[] T 2
  3: do
    4: r[] R1 F2
    5: r[] R2 T
  6: while R1 ∧ R2 ≠ 1
  7: skip (* CS1 *)
  8: w[] F1 false
  9: 
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: skip (* CS2 *)
  17: w[] F2 false;
  18: 
```

at 6 ∧ at 16: ¬R1 ∧ R2=1 ∧ ¬R3 ∧ R4=2 holds
⇒ 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:
  1:w[] F1 true
  2:w[] T 2
  3:do
  4: r[] R1 F2
  5: r[] R2 T
  6:while R1 ∧ R2 ≠ 1
  7:skip (* CS1 *)
  8:w[] F1 false
  9:

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:skip (* CS2 *)
  17:w[] F2 false;
  18:
```

at 6 ∧ at 16: ¬R1 ∧ R2=1 ∧ ¬R3 ∧ R4=2 holds

⇒ both processes simultaneously enter their critical section
A hierarchy of semantics of weakly consistent parallelism
Hierarchy of semantics

- **Hierarchy of semantic domains:**
  - Domain of sets of candidate executions
  - Domain of sets of histories
  - Domain of sets of executions
  - Domain of sets of interleaved traces

- **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

\begin{itemize}
\item 0: \{ x = 0; y = 0; \}
\begin{align*}
P0 & \quad |\quad P1 \\
1: & r[] \ r1 \ x \\
2: & w[] \ y \ 1 \\
3: & 12: w[] \ x \ 1 ; \\
\end{align*}
\end{itemize}

\begin{itemize}
\item start \quad \begin{array}{c}
0: x = 0; \quad y = 0 \\
\end{array}
\begin{align*}
\xrightarrow{r_x^1} \quad 1: & r[] \ r1 \ x \\
\xrightarrow{r_y^{11}} \quad 11: & r[] \ r2 \ y \\
\end{align*}
\begin{align*}
\begin{array}{c}
\langle \{ x \leftarrow w_x^0, y \leftarrow w_y^0 \}, \ 1: \{ r1 \leftarrow 0 \}, \ 11: \{ r2 \leftarrow \ \\
\langle \{ x \leftarrow w_x^{12}, y \leftarrow w_y^0 \}, \ 2: \{ r1 \leftarrow 1 \}, \ 11: \{ r2 \leftarrow 0 \} \rangle \\
\langle \{ x \leftarrow w_x^{12}, y \leftarrow w_y^2 \}, \ 2: \{ r1 \leftarrow 1 \}, \ 12: \{ r2 \leftarrow 1 \} \rangle \\
\langle \{ x \leftarrow w_x^{12}, y \leftarrow w_y^2 \}, \ 2: \{ r1 \leftarrow 1 \}, \ 13: \{ r2 \leftarrow 1 \} \rangle \\
\langle \{ x \leftarrow w_x^{12}, y \leftarrow w_y^2 \}, \ 3: \{ r1 \leftarrow 1 \}, \ 13: \{ r2 \leftarrow 1 \} \rangle \\
\end{array}
\end{align*}
\end{itemize}
Sets of truly parallel execution traces

• project traces per process $\longrightarrow$ local time on computations

• get rid of shared memory states using a read-from relation $\text{rf} \longrightarrow$ no time on communications

\[ \langle r, w \rangle \in \text{rf} \iff \tau = \tau_0 \langle \nu, \ldots \rangle \xrightarrow{r} \langle \nu', \ldots \rangle \tau_1 \land \nu'(X(r)) = w \]

• keep local states on process control points and values of registers

• keep computation progress information using cuts of parallel traces $\longrightarrow$ global time
Example of truly parallel execution for 1b

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

P0 \hspace{1cm} P1

1:r[] r1 x \hspace{1cm} 11:r[] r2 y;
2:w[] y 1 \hspace{1cm} 12:w[] x 1 ;
3: \hspace{1cm} 13: ;
Sets of histories

• Get rid of cuts  \( \rightarrow \) 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
1: r[] r1 x
2: w[] y 1
3: r1 = 1

1: start

0: x = 0; y = 0.

1: r[] r1 x
2: w[] y 1
3: r1 = 1

11: r[] r2 y
12: w[] x 1
13: r2 = 1
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 $\tau_0$
  - the program order po
    $$\langle e, e' \rangle \in po \iff \tau_i = \tau_i' \xrightarrow{e} \tau_i'' \quad e' \xrightarrow{\tau_i}$$
    $\rightarrow$ relational on events
- Get rid of states
  $\rightarrow$ no values
Example of candidate execution for 1b

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

11: r[] r2 y;
12: w[] x 1;
12: w[] x 1;

0: x = 0; y = 0;
0: x = 0; y = 0;

wi
wi

rf
rf

po
po

wx
wy

w12
w11

11: r[] r2 y.
11: r[] r2 y.

2: w[] y 1.
2: w[] y 1.
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

\[ fr = rf^{-1} ; co \]
Auxiliary relations

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

  \[
  \text{start} \quad \begin{array}{c}
  0: x = 0; \quad y = 0 \\
  \end{array} \quad \Rightarrow \quad \langle \{ x \leftarrow w_x^0, y \leftarrow w_y^0 \}, \; 1: \{r1 \leftarrow 0\}, \; 11: \{r2 \leftarrow 0\} \rangle \\
  \begin{array}{c}
  r_x^{12} \\
  1: r[] \quad r1 \quad x \\
  \end{array} \quad \Rightarrow \quad \langle \{ x \leftarrow w_x^{12}, y \leftarrow w_y^0 \}, \; 2: \{r1 \leftarrow 1\}, \; 11: \{r2 \leftarrow 0\} \rangle \quad \begin{array}{c}
  r_y^{11} \\
  11: r[] \quad r2 \quad y \\
  \end{array} \quad \Rightarrow \quad \langle \{ x \leftarrow w_x^{12}, y \leftarrow w_y^0 \} \rangle
  \]

  \[
  \langle \{ x \leftarrow w_x^{12}, y \leftarrow w_y^0 \}, \; 2: \{r1 \leftarrow 1\}, \; 12: \{r2 \leftarrow 1\} \rangle \quad \begin{array}{c}
  w_x^{12} \\
  12: w[] \quad x \quad 1 \\
  \end{array} \quad \Rightarrow \quad \langle \{ x \leftarrow w_x^{12}, y \leftarrow w_y^0 \} \rangle \quad \begin{array}{c}
  w_y^2 \\
  2: w[] \quad y \quad 1 \\
  \end{array} \quad \Rightarrow \quad \langle \{ x \leftarrow w_x^{12}, y \leftarrow w_y^2 \}, \; 3: \{r1 \leftarrow 1\}, \; 11: \{r2 \leftarrow 1\} \rangle
  \]
Example: sequential consistency for 1b

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

- 1b:

```plaintext
0: start
1: r1 = 0
1: r[] r1 x.
2: r1 = 1
12: w[] x 1.
3: r1 = 1
12: w[] x 1.

11: r2 = 0
11: r[] r2 y.
2: r1 = 1
12: r2 = 1
2: w[] y 1.

13: r2 = 1
```

Example: sequential consistency for 1b

- **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; }
PO       | P1       ;
r[] r1 x | r[] r2 y ;
w[] y 1  | w[] x 1  ;

Example of communication specification in the cat language for LB

irreflexive (po | rf)+

Rejects only the anarchic execution:

Thread 0

- a: R() x=1
- b: W() y=1

Thread 1

- c: R() y=1
- d: W() x=1


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 scp
  let ppo = po \ (W*R)
  let rfe = rf & ext
  acyclic ppo | rfe | co | fr as tso
  ```

- **For lb:**

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

  \[ \text{sc} \Rightarrow \text{lb}, \quad \text{tso} \not\Rightarrow \text{lb} \]
Fence specification:

- In Lisa:
  \[
  \{ x = 0; y = 0; \} \\
  P0 \quad | \quad P1 \quad ; \\
  r[] \ r1 \ x \ | \ r[] \ r2 \ y \ ; \\
  f[dep] \quad | \quad f[lw] \quad ; \\
  w[] \ y \ 1 \ | \ w[] \ x \ 1 \ ;
  \]

- Implementation with dependencies and fences in TSO:
  \[
  \{ x = 0; y = 0; \} \\
  P0 \quad | \quad P1 \quad ; \\
  r[] \ r1 \ x \ | \ r[] \ r2 \ y \ ; \\
  r2 = \text{xor} \ r1 \ r1 \ | \ mfence \ ; \\
  r3 = r2 + 1 \quad | \ \\
  w[] \ y \ r3 \quad | \ w[] \ x \ r3 \ ;
  \]
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. $*$, $;$, $|$, $&$, $\backslash$, $+$, $^\sim-1$, …
- Accepts/eliminates the execution by defining relations $r$ and checking irreflexive $r$, acyclic $r$, empty $r$, not empty $r
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:
  \[ \Rightarrow \text{communication specification } S_{com} \]

• We have to show that the communication specification is correctly implemented on an architecture:
  \[ \Rightarrow \text{a way to mix invariant } S_{com} \text{ and cat specifications} \]
Methodology

algorithm $A$

invariant specification $S_{inv}$

communication specification $S_{com}$

consistency hypothesis $H_{com}$

consistency model $M$

invariance proof

$S_{com} \Leftrightarrow S_{inv}$

inclusion proof

$H_{com} \Rightarrow S_{com}$

consistency proof

$M \Rightarrow H_{com}$

algorithm $A$ proved correct w.r.t. $H_{com}$ and $S_{inv}$

$H_{com} \Rightarrow S_{inv}$

algorithm $A$ proved correct w.r.t. $M$ and $S_{inv}$

$M \Rightarrow S_{inv}$
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 \{⇒ F2_{i4} \}
5: r[] R2 T \{⇒ T_{i5} \}
6: until ¬R1 ∨ 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; \{⇒ F1_{j_{13}} \}
14: r[] R4 T; \{⇒ T_{j_{14}} \}
15: until ¬R3 ∨ R4 = 2; \{j_{end}\}
16: skip (* CS2 *)
17: w[] F2 false;
18: 

Stamp: label,counter

Pythia variables
\[ s^i = \times \text{rf}^i, \quad i \in \Delta \]

**State**: program point, stamp, environment (value of registers), valuation (value of pythia variables)

**Diagram**:
- \( s^i \)
- \( \langle \kappa^i_0, \theta^i_0, \rho^i_0, \nu^i_0 \rangle \)
- \( \langle \ell, \theta^i_p, \rho^i_p, \nu^i_p \rangle \)
- \( \langle \kappa^i_{n-1}, \theta^i_{n-1}, \rho^i_{n-1}, \nu^i_{n-1} \rangle \)
- \( k_0 \)
- \( k_1 \)
- \( \cdots \)
- \( k_p \)
- \( \cdots k_{n-1} \)
- \( \tau^i_0 \)
- \( \tau^i_1 \)
- \( \tau^i_p \)
- \( \tau^i_{n-1} \)
- \( \tau_{\text{start}} \)

**Read-From**: \( \times \text{rf}^i, \quad i \in \Delta \)
Invariance abstraction

\[ \pi^i = \varsigma^i \times \Gamma^i \]

\[ \varsigma^i = \tau^i_{\text{start}} \times \text{rf}^i, \quad i \in \Delta \]

\[
\alpha_a (\{\pi^i \mid i \in \Delta\}) \triangleq \prod_{p \in \Pi} \prod_{\ell \in \mathbb{L}(p)} \bigcup \{ \langle k_0, \theta_0, \kappa_0, \nu_0 \rangle, \ldots, \langle k_{n-1}, \theta_{n-1}, \kappa_{n-1}, \nu_{n-1} \rangle \mid \forall q \in [0, n[ \setminus \{p\}. \tau^i_{q,kq} = \\
\langle \kappa_q, \theta_q, \rho_q, \nu_q \rangle \land \tau^i_{p,kp} = \langle \ell, \theta_p, \rho_p, \nu_p \rangle \}
\]
Invariant

- An invariant $S_{inv}(p)$ at point $p$ of process $P_i$ is a statement relating
  
  - the **program points** $p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_m$ 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 $p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_m$
Example (Peterson)

0: \{ \ w\ F1\ \text{false}; \ w\ F2\ \text{false}; \ w\ T\ 0; \ \}
\{F1=\text{false} \land F2=\text{false} \land T=0\} \}
1: \{ R1=0 \land R2=0 \}
\{ \ w\ F1\ \text{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 \land R1=F2^i-1 \land R2=T^i-1) \}
\{ \ r[]\ R1\ F2\ \{\sim\ F2^i\} \}
5: \{ R1=F2^i \land (i=0 \land R2=0) \lor
\ (i>0 \land R2=T^i-1) \}
\{ \ r[]\ R2\ T\ \{\sim\ T^i_5\} \}
6: \{ R1=F2^i_4 \land R2=T^i_5 \}
\{ \ \text{while}\ R1 \land R2 \neq 1 \ \{\text{i_end}\} \}
7: \{ \neg F2^i_4\ \text{end} \lor T^i_5\ \text{end}=1 \}
\{ \ \text{skip} \ (*\ \text{CS1} \ *) \}
8: \{ \neg F2^i_4\ \text{end} \lor T^i_5\ \text{end}=1 \}
\{ \ w[]\ F1\ \text{false} \}
9: \{ \neg F2^i_4\ \text{end} \lor T^i_5\ \text{end}=1 \}
10: \{ R3=0 \land R4=0 \}
\{ \ w[]\ F2\ \text{true} \}
11: \{ R3=0 \land R4=0 \}
\{ \ w[]\ T\ 1 \}
12: \{ R3=0 \land R4=0 \}
\{ \ do\ \{\text{i}\} \}
13: \{ (j=0 \land R3=0 \land R4=0) \lor
\ (j>0 \land R3=F1^j_13 \land R4=T^j_14-1) \}
\{ \ r[]\ R3\ F1\ \{\sim\ F1^j_13\} \}
14: \{ R3=F1^j_13 \land (j=0 \land R4=0) \lor
\ (j>0 \land R4=T^j_14-1) \}
\{ \ r[]\ R4\ T\ \{\sim\ T^j_14\} \}
15: \{ R3=F1^j_13 \land R4=T^j_14 \}
\{ \ \text{while}\ R3 \land R4 \neq 2 \ \{\text{j_end}\} \}
16: \{ \neg F1^j_13\ \text{end} \lor T^j_14\ \text{end}=2 \}
\{ \ \text{skip} \ (*\ \text{CS2} \ *) \}
17: \{ \neg F1^j_13\ \text{end} \lor T^j_14\ \text{end}=2 \}
\{ \ w[]\ F2\ \text{false} \}
18: \{ \neg F1^j_13\ \text{end} \lor T^j_14\ \text{end}=2 \}

(\text{these invariants are for the anarchic semantics, so all communications are possible, no constraints on rf)
Invariance proof

Algorithm $A$

- Invariant specification $S_{inv}$
- Communication specification $S_{com}$
- Consistency hypothesis $H_{com}$
- Consistency model $M$

Invariance proof

$S_{com} \Rightarrow S_{inv}$

Inclusion proof

$H_{com} \Rightarrow S_{com}$

Consistency proof

$M \Rightarrow H_{com}$

Algorithm $A$ proved correct w.r.t.

- $H_{com}$ and $S_{inv}$
- $H_{com} \Rightarrow S_{inv}$

Algorithm $A$ proved correct w.r.t.

- $M$ and $S_{inv}$
- $M \Rightarrow S_{inv}$
Verification conditions

- **Sequential** proof
- **Absence of interference** proof
- **Communication** proof

Examples:

- \( \{ P(R, \ldots, rf) \land \langle w(x, v), r(\theta, x) \rangle \in rf \} \)
  \( \xrightarrow*{\text{communication}} \)
  \( \{ \text{read } x \ R \{ \sim \ x_\theta \} \} \)
  \( \{ P[R \leftarrow x_\theta, x_\theta \leftarrow v, \ldots, rf] \} \)

- \( \{ P \} \) 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 $\theta'$) 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 $S_{\text{com}}$
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)

\[ \text{at } 7 \land \text{at } 16 \]
\[ \Rightarrow (\neg F_{2i}^{\text{end}} \lor T_{5i}^{\text{end}} = 1) \land (\neg F_{1j}^{\text{end}} \lor T_{14j}^{\text{end}} = 2) \]  
\[ \text{i.e. the invariants at lines 7: and 16: hold} \]
\[ \Rightarrow \neg S_{\text{com}} \text{ since by taking } i = i_{\text{end}} \text{ and } j = j_{\text{end}}, \text{ we have} \]
\[ (F_{2i}^i = \text{false} \lor T_{5i}^i = 1) \land (F_{1j}^j = \text{false} \lor T_{14j}^j = 2) \]

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

\[ S_{\text{com}} \triangleq \neg [\exists i, j. [\text{rf}\langle F_{2i}^i, \langle 0 :, \text{false} \rangle \rangle \lor \text{rf}\langle F_{2i}^i, \langle 17 :, \text{false} \rangle \rangle \]
\[ \lor \text{rf}\langle T_{5i}^i, \langle 11 :, 1 \rangle \rangle ] \land [\text{rf}\langle F_{1j}^j, \langle 0 :, \text{false} \rangle \rangle \]
\[ \lor \text{rf}\langle F_{1j}^j, \langle 8 :, \text{false} \rangle \rangle \lor \text{rf}\langle T_{14j}^j, \langle 2 :, 2 \rangle \rangle ]] \]

(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 \( S_{com} \Rightarrow \text{incompleteness} \) (*)

(*) e.g. hardware cannot restrict a read to input from writes writing odd numbers.
Consistency hypothesis and inclusion proof

algorithm $A$

- invariant specification $S_{inv}$
- communication specification $S_{com}$
- consistency hypothesis $H_{com}$
- consistency model $M$

**Invariance proof**

$S_{com} \Rightarrow S_{inv}$

**Inclusion proof**

$H_{com} \Rightarrow S_{com}$

**Consistency proof**

$M \Rightarrow H_{com}$

algorithm $A$ proved correct w.r.t.

- $H_{com}$ and $S_{inv}$
- $H_{com} \Rightarrow S_{inv}$

algorithm $A$ proved correct w.r.t.

- $M$ and $S_{inv}$
- $M \Rightarrow S_{inv}$
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. (rf\langle F2^i_4, \langle 0:, false \rangle \rangle \lor rf\langle F2^i_4, \langle 17:, false \rangle \rangle \lor rf\langle T^i_5, \langle 11:, 1 \rangle \rangle] \land [rf\langle F1^j_{13}, \langle 0:, false \rangle \rangle \lor rf\langle F1^j_{13}, \langle 8:, false \rangle \rangle \lor rf\langle T^j_{14}, \langle 2:, 2 \rangle \rangle]$ 

- Prevent each of them by a cat specification

- $H_{com}$ is their conjunction
Example: Peterson

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

P0:
1: w[] F1 true
1: w[] F1 true
2: w[] T 2
2: w[] T 2
3: do
3: do
4: r[] R1 F2
4: r[] R1 F2
5: r[] R2 T
5: r[] R2 T
6: while R1 ∧ R2 ≠ 1
6: while R1 ∧ R2 ≠ 1
7: f[p0] (* CS1 *)
7: f[p0] (* CS1 *)
8: w[] F1 false
8: w[] F1 false

11 → co → 2 → po → 4 → fr → 10 → po → 11
11 → co → 2 → po → 4 → fr → 10 → po → 11

---

case 1: 0:F2,0:F1

---

case 2a: 0:F2,1:F1 (2 → co → 11)

---

case 2b: 0:F2,1:F1 (11 → co → 2)

---

case 3a: 10:F2,0:F1 (2 → co → 11)
Example: Peterson

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 \neq 1
  7: f[p0] (* CS1 *)
  8: w[] F1 false

  5 ----fr→ 2 ----po→ 5
  case 3b: 10:F2, 0:F1 (11 → co → 2)

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

P1:
  10: w[] F2 true;
  11: w[] T 1;
  12: do
  13: r[] R3 F1;
  14: r[] R4 T;
  15: while R3 \& R4 \neq 2;
  16: f[p1] (* CS2 *)
  17: w[] F2 false;

  14 ----fr→ 11 ----po→ 14
  case 4a: 10:F2, 1:F1 (2 → co → 11)

P0:
  1: w[] F1 true
  2: w[] T 2
  3: do
  4: r[] R1 F2
  5: r[] R2 T
  6: while R1 \& R2 \neq 1
  7: f[p0] (* CS1 *)
  8: w[] F1 false

  5 ----fr→ 2 ----po→ 5
  case 4b: 10:F2, 1:F1 (11 → co → 2)

P1:
  10: w[] F2 true;
  11: w[] T 1;
  12: do
  13: r[] R3 F1;
  14: r[] R4 T;
  15: while R3 \& R4 \neq 2;
  16: f[p1] (* CS2 *)
  17: w[] F2 false;
Example: Peterson

\[
0: \{ \text{w F1 false; w F2 false; w T 0; } \}
\]

- the cut relation can be expressed in cat using tags on fence markers \(f[p0]\) and \(f[p1]\)

- \(H_{\text{com}}\) is
  - \text{irreflexive fr};po;fr;po
  - \text{irreflexive fr};po
  - \text{irreflexive co};po;fr;po
  - \text{irreflexive po};rf;po;rf
  - \text{irreflexive po};rf;po;cut
Consistency model and proof
Example: Peterson in SC

- \( H_{\text{com}} \) is irreleflexive fr;po;fr;po
- irreleflexive fr;po
- irreleflexive co;po;fr;po
- irreleflexive po;rf;po;rf
- irreleflexive po;rf;po;cut

- Sequential consistency in \( \text{cat} \):
  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:

  \[
  \begin{align*}
  & \text{let } \text{fr} = (\text{rf}^{-1}; \text{co}) \\
  & \text{let } \text{po-loc} = \text{po} \& \text{loc} \\
  & \text{acyclic } \text{po-loc} \mid \text{rf} \mid \text{co} \mid \text{fr} \text{ as scpv} \\
  & \text{let } \text{ppo} = \text{po} \setminus (W*R) \\
  & \text{let } \text{rfe} = \text{rf} \& \text{ext} \\
  & \text{acyclic } \text{ppo} \mid \text{rfe} \mid \text{co} \mid \text{fr} \text{ as tso}
  \end{align*}
  \]

- For example the case 1,

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

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

Implementation with (weak) cat fences
Implementation with fences

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

let \text{fhw} = (po \ & \ (_ * F)) \ ; \ po
let \text{fre} = (rf^{-1};co) \ & \ ext
irreflexive \text{fhw};\text{fre}; \text{fhw};\text{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: $fhw = \text{no fence}$
  - TSO: $fhw = \text{mfence}$
  - ARM: $fhw = \text{dbm} | \text{dsb}$
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 \texttt{cat} from $S_{com}$

6. Prove that the machine memory model M in \texttt{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