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

```plaintext
var x_1, ..., x_m; // shared variables
P_0;  // prelude initializing x_1, ..., x_m
[P_1 || P_2 || ... || P_n]
```

- P₁, P₂, ..., Pₙ are the **processes** modifying the shared variables and their **local registers** R, ... 

- 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:**

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

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

P0 \hspace{1cm} P1 ;

1: r[] r1 x \hspace{0.5cm} 11: r[] r2 y;

2: w[] y 1 \hspace{0.5cm} 12: w[] x 1 ;

3: \hspace{1cm} 13: \ ;

- **Specification** \( S_{inv}: \)

\[ \text{at } 3 \land \text{at } 13 \Rightarrow \neg (r1=1 \land r2=1) \]
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 ∧ 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}:$$**

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

---

Weak memory/consistency models

- Sequential consistency:

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

atomic instantaneous communications

- Weak memory models:

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

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)

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

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1: r[] \ r1 \ x$</td>
<td>$11: r[] \ r2 \ y$</td>
</tr>
<tr>
<td>$2: w[] \ y \ 1$</td>
<td>$12: w[] \ x \ 1$</td>
</tr>
<tr>
<td>$3:$</td>
<td>$13:$</td>
</tr>
</tbody>
</table>

- at $3 \land at \ 13 \land r1=1 \land r2=1$

- This erroneous behavior can be observed on TSO machines
Example: Peterson (incorrect)

- Can read the wrong flags

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

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

at 6 \land at 16: \neg R1 \land R2=1 \land \neg R3 \land R4=2 \text{ 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 \land R2 \neq 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 \land R4 \neq 2;
16: skip (* CS2 *)
17: w F2 false;
18: 

at 6 \land at 16: \neg R1 \land R2=1 \land \neg R3 \land 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:

- Invariance domain

- Domain of sets of candidate executions
  \( \gamma_c \)

- Domain of sets of histories
  \( \gamma_h \)

- Domain of sets of executions
  \( \gamma_e \)

- 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 $\rightarrow \textit{global time}$

• **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: \(\{ x = 0; \ y = 0; \} \)
  - \(\text{P0} \parallel \text{P1} \)
  - 1: \(r[] \ r1 \ x\)
  - 2: \(w[] \ y \ 1\)
  - 3: \(\)
  - \(w^0_x \quad w^0_y\)

- start \(\frac{0: \ x = 0; \ y = 0}{\quad \rightarrow \quad \langle \{ x \leftarrow w^0_x, y \leftarrow w^0_y \}, 1: \{ r1 \leftarrow 0 \}, 11: \{ r2 \leftarrow 0 \} \rangle}

- \(\frac{1:r[] \ r1 \ x}{\quad \rightarrow \quad \langle \{ x \leftarrow w^1_{x1}, y \leftarrow w^0_y \}, 2: \{ r1 \leftarrow 1 \}, 11: \{ r2 \leftarrow 0 \} \rangle}

- \(\frac{\langle \{ x \leftarrow w^1_{x1}, y \leftarrow w^2_y \}, 2: \{ r1 \leftarrow 1 \}, 12: \{ r2 \leftarrow 1 \} \rangle}{\quad \rightarrow \quad \langle \{ x \leftarrow w^1_{x1}, y \leftarrow w^2_y \}, 2: \{ r1 \leftarrow 1 \}, 13: \{ r2 \leftarrow 1 \} \rangle}

- \(\frac{2:w[] \ y \ 1}{\quad \rightarrow \quad \langle \{ x \leftarrow w^1_{x1}, y \leftarrow w^2_y \}, 3: \{ r1 \leftarrow 1 \}, 13: \{ r2 \leftarrow 1 \} \rangle}
Sets of truly parallel execution traces

- project \textbf{traces per process} \rightarrow \textit{local time on computations}

- get rid of shared memory states using a \textbf{read-from relation \textit{rf}} \rightarrow \textit{no time on communications}
  
  \[
  \langle r, w \rangle \in \textit{rf} \iff \tau = \tau_0 \langle \nu, \ldots \rangle \overset{r}{\rightarrow} \langle \nu', \ldots \rangle \tau_1 \land \nu'(X(r)) = w
  \]

- keep \textbf{local states} on process control points and values of registers

- keep computation progress information using \textbf{cuts} of parallel traces \rightarrow \textit{global time}
Example of truly parallel execution for 1b

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

0:, start
1!: r1=0
2!: r1=1
3!: r1=1

11!: r2=0
12!: r2=1
13:

w_x^0
w_y^0
w_x^1
w_y^1
w_x^2
w_y^2
w_x^11
w_y^11
w_x^12
w_y^12
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  11:r[] r2 y;  
2:w[] y 1  12:w[] x 1;  
3:  13:  

0:, start  

1:, r1=0  

11:, r2=0  

11:, r1=1  

12:, r2=1  

2:, r1=1  

21:, w[] y 1.  

3:, r1=1  

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 \( \tau_0 \prod_{i=1}^{n} \tau_i, \text{ rf} \) by
  - the set of initial writes \( \text{IW} \) in \( \tau_0 \)
  - the program order \( \text{po} \)
    \[ \langle e, e' \rangle \in \text{po} \iff \tau_i = \tau_i' \xrightarrow{e} \tau_i'' \xrightarrow{e'} \tau_i''' \]
    \( \rightarrow \) relational on events
- Get rid of states
  \( \rightarrow \) no values
Example of candidate execution for 1b

0: { x = 0; y = 0; }
P0
1: r[] r1 x 11: r[] r2 y;
P1
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

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

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

11: P1
11: r[] r2 y;
12: w[] x 1;
13:

0: x = 0; y = 0

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

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

rf
co
fr
po
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

- \[ \text{lb:} \]

  \[
  \begin{array}{c}
  \text{start} \quad \frac{w^0_x}{0: x = 0; \quad y = 0} \rightarrow <\{x \leftarrow w^0_x, y \leftarrow w^0_y\}, \; 1:\{r1 \leftarrow 0\}, \; 11:\{r2 \leftarrow 0\}> \\
  \end{array}
  \]

  \[
  \begin{array}{c}
  \frac{r^{12}_x}{1:r[] \; r1 \; x} \rightarrow <\{x \leftarrow w^{12}_x, y \leftarrow w^0_y\}, \; 2:\{r1 \leftarrow 1\}, \; 11:\{r2 \leftarrow 0\}> \quad \frac{r^{11}_y}{11:r[] \; r2 \; y} \\
  \end{array}
  \]

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

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

- **lb:**
Example: sequential consistency for 1b

- **Parallel histories**: abstract to candidate execution and check it is allowed

- **Candidate executions**: irreducible 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 ;
\text{r[]} r1 x | \text{r[]} r2 y ;
\text{w[]} y 1 | \text{w[]} x 1 ;

Example of communication specification in the cat language for LB

 irreversible (po | rf)+

Rejects only the anarchic execution:

\[\times\]

Thread 0                     Thread 1

\[a: R() \ x=1\]
\[b: W() \ y=1\]
\[c: R() \ y=1\]
\[d: W() \ x=1\]


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

- **For lb:**
  
  acyclic (po | rf) as lb

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

• In Lisa:

\[
\begin{align*}
\{ & x = 0; y = 0; \\
& P0 \quad | \quad P1 \quad ; \\
r[] & r1 \ x \quad | \quad r[] \ r2 \ y \quad ; \\
f[dep] & \quad | \quad f[1w] \quad ; \\
w[] & y \ 1 \quad | \quad w[] \ x \ 1 \quad ;
\end{align*}
\]

• Implementation with dependencies and fences in TSO:

\[
\begin{align*}
\{ & x = 0; y = 0; \\
& P0 \quad | \quad P1 \quad ; \\
r[] & r1 \ x \quad | \quad r[] \ r2 \ y \quad ; \\
r2 & = \ xor \ r1 \ r1 \ | \quad mfence \quad ; \\
r3 & = r2+1 \quad | \\
w[] & y \ r3 \quad | \quad w[] \ x \ r3 \quad ;
\end{align*}
\]
cat

- Handles one history at a time
- For each execution relies on:
  - the set \( E (:) \) of events of the execution (partitionned into initial writes \( \mathcal{IW} \), writes \( \mathcal{W} \), read \( \mathcal{R} \), fences \( \mathcal{F} \), ...)
  - the program order \( \mathcal{po} \) of events per process
  - the read-from relation \( \mathcal{rf} \) per variable
- Has predefined relations \( \mathcal{loc}, \mathcal{ext} \), ...
- Can define new relations e.g. \( *, ;, |, &, \backslash, +, \neg 1, ... \)
- 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:
  \[\Rightarrow \text{a way to mix invariant } S_{com} \text{ and } \text{cat specifications}\]
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)

\[
0: \{ \text{w F1 false; w F2 false; w T 0; } \} \\
\text{P0:} \\
1: \text{w[] F1 true} \\
2: \text{w[] T 2} \\
3: \text{repeat } \{i\} \\
4: \text{r[] R1 F2 } \{\rightsquigarrow \text{ F2}_{4}^{i}\} \\
5: \text{r[] R2 T } \{\rightsquigarrow \text{ T}_{5}^{i}\} \\
6: \text{until } \neg R1 \lor R2 = 1 \{i_{\text{end}}\} \\
7: \text{skip } (* \text{CS1} *) \\
8: \text{w[] F1 false} \\
9: \\
\text{P1:} \\
10: \text{w[] F2 true;} \\
11: \text{w[] T 1;} \\
12: \text{repeat } \{j\} \\
13: \text{r[] R3 F1; } \{\rightsquigarrow \text{ F1}_{13}^{j}\} \\
14: \text{r[] R4 T; } \{\rightsquigarrow \text{ T}_{14}^{j}\} \\
15: \text{until } \neg R3 \lor R4 = 2; \{j_{\text{end}}\} \\
16: \text{skip } (* \text{CS2} *) \\
17: \text{w[] F2 false;} \\
18: \\
\text{Stamp: label, counter} \\
\text{Pythia variables}
\]
Invariance abstraction

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

\[ \langle k_0^i, \theta_0^i, \rho_0^i, \nu_0^i \rangle \]

\[ \langle \ell, \theta_p^i, \rho_p^i, \nu_p^i \rangle \]

\[ \langle k_{n-1}^i, \theta_{n-1}^i, \rho_{n-1}^i, \nu_{n-1}^i \rangle \]

\[ \text{state} = \text{program point} \]

\[ \text{stamp} \]

\[ \text{environment} \]

\[ \text{(value of registers)} \]

\[ \text{valuation} \]

\[ \text{(value of pythia variables)} \]
Invariance abstraction

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

\[ \zeta^i = \tau_{\text{start}} \times \mathbf{r}^i, \quad i \in \Delta \]

\[ \alpha_a(\{\pi^i \mid i \in \Delta\}) \triangleq \prod_{p \in \mathbb{P}} \prod_{\ell \in \mathbb{L}(p)} \bigcup_{i \in \Delta} \{\langle \kappa_0^i, \theta_0^i, \rho_0^i, \nu_0^i \rangle \mid \forall q \in [0, n \setminus \{p\} \cdot \tau_q^i \rangle \}

\[ \nu_{p-1,k_{p-1}}, \theta_{p,k_p}, \rho_{p,k_p}, \nu_{p,k_p}, \kappa_{p+1,k_{p+1}}^i, \ldots, \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, \mathbf{r}^i \rangle \}

\[ \mathbf{s}\langle \kappa_q^i, \theta_q^i, \rho_q^i, \nu_q^i \rangle \wedge \tau_p^i \]

\[ \mathbf{s}\langle \ell, \theta_p^i, \rho_p^i, \nu_p^i \rangle \} . \]
Invariant

- An invariant $S_{inv}(p)$ at point $p$ of process $P_i$ is a statement relating

  - the **program points** $p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_m$ 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 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 \land R1=F2^i_{4} \land R2=T^i_{5})}  
   r[] R1 F2 \{\leadsto F2^i_{4}\}  
5: {R1=F2^i_{4} \land (i=0 \land R2=0) \lor (i>0 \land R2=T^i_{5-1})}  
   r[] R2 T \{\leadsto T^i_{5}\}  
6: {R1=F2^i_{4} \land R2=T^i_{5}}  
while R1 \land R2\neq1 {iend}  
7: {\neg F2^i_{4} \land T^i_{5}=1}  
skip (* CS1 *)  
8: {\neg F2^i_{4} \land T^i_{5}=1}  
w[] F1 false  
9: {\neg F2^i_{4} \land T^i_{5}=1}  
10: {R3=0 \land R4=0}  
w[] F2 true;  
11: {R3=0 \land R4=0}  
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^j_{13} \land R4=T^j_{14-1})}  
   r[] R3 F1 \{\leadsto 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; \{\leadsto T^j_{14}\}  
15: {R3=F1^j_{13} \land R4=T^j_{14}}  
while R3 \land R4\neq2 {jend} ;  
16: {\neg F1^j_{13} \land T^j_{14}=2}  
skip (* CS2 *)  
17: {\neg F1^j_{13} \land T^j_{14}=2}  
w[] F2 false;  
18: {\neg F1^j_{13} \land T^j_{14}=2}  

(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:

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

- \{ P \} \textit{fence} \{ P \} (fences are markers in the execution)

- \{ P \} \textit{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)

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

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

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

(preventing the incorrect case)
Soundness and completeness

• The invariance proof method is derived from the truly parallel semantics with cuts by calculational design ⇒ 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
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 \( S_{com} \):
    \[
    S_{com} \triangleq \neg \left[ \exists i, j. \left( rf\langle F_{24}^i, \langle 0: \rangle, false \rangle \lor rf\langle F_{24}^i, \langle 17: \rangle, false \rangle \lor rf\langle T_{5}^i, \langle 11: \rangle, 1 \rangle \right) \land \left( rf\langle F_{13}^j, \langle 0: \rangle, false \rangle \lor rf\langle F_{13}^j, \langle 8: \rangle, false \rangle \lor rf\langle T_{14}^j, \langle 2: \rangle, 2 \rangle \right) \right]\]

  • Prevent each of them by a cat specification

  • \( H_{com} \) is their conjunction
Example: Peterson

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

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 \land R2 \neq 1
7: f[p0] (* CS1 *)
8:w[] F1 false

4 \rightarrow fr \rightarrow 10 \rightarrow po \rightarrow 13 \rightarrow fr \rightarrow 1 \rightarrow po \rightarrow 4

case 1: 0:F2,0:F1

11 \rightarrow co \rightarrow 2 \rightarrow po \rightarrow 4 \rightarrow fr \rightarrow 10 \rightarrow po \rightarrow 11

case 2b: 0:F2,1:F1 (11 \rightarrow co \rightarrow 2)

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 \land R2 \neq 1
7: f[p0] (* CS1 *)
8:w[] F1 false

14 \rightarrow fr \rightarrow 11 \rightarrow po \rightarrow 14

case 2a: 0:F2,1:F1 (2 \rightarrow co \rightarrow 11)

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 \land R2 \neq 1
7: f[p0] (* CS1 *)
8:w[] F1 false

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 \land R2 \neq 1
7: f[p0] (* CS1 *)
8:w[] F1 false

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

14 \rightarrow fr \rightarrow 11 \rightarrow po \rightarrow 14

case 2a: 0:F2,1:F1 (2 \rightarrow co \rightarrow 11)
Example: Peterson

0: \{ w \text{ F1 false; w F2 false; w T 0; } \}
P0:
1: w[] \text{ F1 true}
2: w[] \text{ T 2}
3: do
4: r[] R1 F2
5: r[] R2 T
6: while R1 \land R2 \neq 1
7: f[p0] (* CS1 *)
8: w[] \text{ F1 false}

\begin{align*}
5 & \text{ fr } 2 \text{ po } 5 \\
\text{case 3b: 10:F2, 0:F1 (11 co 2)}
\end{align*}

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

\begin{align*}
14 \text{ fr } 11 \text{ po } 14 \\
\text{case 4a: 10:F2, 1:F1 (2 co 11)}
\end{align*}

0: \{ w \text{ F1 false; w F2 false; w T 0; } \}
P0:
1: w[] \text{ F1 true}
2: w[] \text{ T 2}
3: do
4: r[] R1 F2
5: r[] R2 T
6: while R1 \land R2 \neq 1
7: f[p0] (* CS1 *)
8: w[] \text{ F1 false}

\begin{align*}
5 & \text{ fr } 2 \text{ po } 5 \\
\text{case 4b: 10:F2, 1:F1 (11 co 2)}
\end{align*}

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

\begin{align*}
4 \text{ po } 8 \text{ fr } 13 \text{ po } 17 \text{ fr } 4 \\
\text{case 5: 17:F2, 8:F1}
\end{align*}
Example: Peterson

- The cut relation can be expressed in `cat` using tags on fence markers `f[p0]` and `f[p1]`

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

- \( H_{\text{com}} \) 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 \( \text{cat} \):
  
  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:
  
  let \( fr = (rf^{-1};co) \)
  let \( po\text{-}loc = po \& loc \)
  acyclic \( po\text{-}loc \mid rf \mid co \mid fr \) as \( scpv \)
  let \( ppo = po \setminus (W*R) \)
  let \( rfe = rf \& ext \)
  acyclic \( ppo \mid rfe \mid co \mid fr \) as \( tso \)

- For example the case 1,
  
  \( ⟨w_1,r_4⟩ \in fr \; ; \; po \; ; fr \; ; po \)
  is not forbidden by TSO since \( ⟨w,r⟩ \) 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 ∧ r2 ≠ 1 | 15: while r3 ∧ r4 ≠ 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: $fhw = \text{no fence}$
  - TSO: $fhw = mfence$
  - ARM: $fhw = dbm \mid dsb$
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 $\text{cat}$ from $S_{com}$

6. Prove that the machine memory model $M$ in $\text{cat}$ 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