Proof of mutual-exclusion and non-starvation of a program: PostgreSQL

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

Dagstuhl Seminar 16471
http://www.dagstuhl.de/16471
Concurrency with Weak Memory Models: Semantics, Languages, Compilation, Verification, Static Analysis, and Synthesis
November 22, 2016

Methodology

<table>
<thead>
<tr>
<th>Algorithm A</th>
<th>Invariant specification of A $S_{inv}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditional invariance proof $S_{com} \Rightarrow S_{inv}$</td>
<td></td>
</tr>
<tr>
<td>Communication specification of A $S_{com}$</td>
<td></td>
</tr>
<tr>
<td>Consistency hypothesis of A $H_{com}$</td>
<td></td>
</tr>
<tr>
<td>Consistency model $M$</td>
<td></td>
</tr>
</tbody>
</table>

Methodology

<table>
<thead>
<tr>
<th>Algorithm A</th>
<th>Invariant specification of A $S_{inv}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditional invariance proof $S_{com} \Rightarrow S_{inv}$</td>
<td></td>
</tr>
<tr>
<td>Communication specification of A $S_{com}$</td>
<td></td>
</tr>
<tr>
<td>Consistency hypothesis of A $H_{com}$</td>
<td></td>
</tr>
<tr>
<td>Consistency model $M$</td>
<td></td>
</tr>
</tbody>
</table>

Invariance

<table>
<thead>
<tr>
<th>Algorithm A</th>
<th>Invariant specification of A $S_{inv}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditional invariance proof $S_{com} \Rightarrow S_{inv}$</td>
<td></td>
</tr>
<tr>
<td>Communication specification of A $S_{com}$</td>
<td></td>
</tr>
<tr>
<td>Consistency hypothesis of A $H_{com}$</td>
<td></td>
</tr>
<tr>
<td>Consistency model $M$</td>
<td></td>
</tr>
</tbody>
</table>


```plaintext
{0: latch0 = 0; flag0 = 0; latch1 = 1; flag1 = 1; }
1: do
2:   do
3:     r[] Rl0 latch0
4:     while (Rl0=0)
5:     w[] latch0 0
6:     r[] Rf0 flag0
7:     if (Rf0$\neq$0) then
8:       (* critical section *)
9:       w[] flag0 0
10:      w[] flag1 1
11: fi
12: while true
13: fi
14: while true
15: fi
```

PostgreSQL
Methodology

WCM

Algorithm A

Invariant specification of A $S_{inv}$

Communication specification of A $S_{com}$

Consistency hypothesis of A $H_{com}$

Consistency model M

Conditional invariance proof $S_{com} \Rightarrow S_{inv}$

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

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

Inference proof $M \Rightarrow H_{com}$

Consistency proof $M \Rightarrow S_{inv}$

Methodology

Invariance + WCM

Algorithm A

Invariant specification of A $S_{inv}$

Communication specification of A $S_{com}$

Consistency hypothesis of A $H_{com}$

Consistency model M

Conditional invariance proof $S_{com} \Rightarrow S_{inv}$

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

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

Consistency proof $M \Rightarrow H_{com}$

Consistency proof $M \Rightarrow S_{inv}$

Methodology

Incompleteness:

Algorithm A

Invariant specification of A $S_{inv}$

Communication specification of A $S_{com}$

Consistency hypothesis of A $H_{com}$

Consistency model M

Conditional invariance proof $S_{com} \Rightarrow S_{inv}$

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

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

Consistency proof $M \Rightarrow H_{com}$

Consistency proof $M \Rightarrow S_{inv}$

Dynamic conditions on executions of one program

Static conditions on all executions of all programs in one architecture
Conditional invariance proof: Mutual exclusion

Algorithm

PostgreSQL

{0: latch0 = 0; flag0 = 0; latch1 = 1; flag1 = 1; }
1: do {i} 21: do {m}
2: do {j}
3: r[] Rl0 latch0 {→ L0,j} 23: r[] Rl1 latch1 {→ L1,m}
4: while (Rl0=0) {k} 24: while (Rl1=0) {m}
5: w[] latch0 0 25: w[] latch1 0
6: r[] Rf0 flag0 {→ F0} 26: r[] Rf1 flag1 {→ F1}
7: if (Rf0=0) then 27: if (Rf1=0) then
8: (* critical section *) 28: (* critical section *)
   w[] flag0 0 29: w[] flag1 0
9: w[] flag1 1 30: w[] latch0 1
10: w[] latch1 1 31: fi
11: fi
12: while true 32: while true
13:

Stamps

{0: latch0 = 0; flag0 = 0; latch1 = 1; flag1 = 1; }
1: do {i} 21: do {l}
2: do {j}
3: r[] Rl0 latch0 {→ L0,j} 23: r[] Rl1 latch1 {→ L1,m}
4: while (Rl0=0) {k} 24: while (Rl1=0) {m}
5: w[] latch0 0 25: w[] latch1 0
6: r[] Rf0 flag0 {→ F0} 26: r[] Rf1 flag1 {→ F1}
7: if (Rf0=0) then 27: if (Rf1=0) then
8: (* critical section *) 28: (* critical section *)
   w[] flag0 0 29: w[] flag1 0
9: w[] flag1 1 30: w[] latch0 1
10: w[] latch1 1 31: fi
11: fi
12: while true 32: while true
13:

Ensure that events are unique (your choice)
Variables in Hoare logic & L/O-G

- **Program variables**: `int x;
- In predicates you need to name the value of variable `x` to express properties of this value of `x`:
  - `valueof(x)`
  - `x`
- **WCM**: no notion of "the" value of a shared variable `x`
- The only way to know something about "the" value of a shared variable `x` is to read it
- **Pythia variable**: name given to the read value
- Not necessary in the semantics, only in assertions (but we put them in the semantics)

---

**Pythia variables**

```plaintext
{0: latch0 = 0; flag0 = 0; latch1 = 1; flag1 = 1; }
1: do {i}
2: do {j}
3: r R0 latch0 `{ ← L0`,i}`
4: while (R0=0) `{k_i}`
5: w latch0 0
6: r R0 flag0 `{ ← F0`,i}`
7: if (R0`̸=0) then
8: (* critical section *)
9: w flag0 0
10: w flag1 1
11: fi
12:while true
13:
21:do {f}
22: do {m_f}
23: r R1 latch1 `{ ← L1`,m_f}`
24: while (R1=0) `{n_f}`
25: w latch1 0
26: r R1 flag1 `{ ← F1`,f}`
27: if (R1`̸=0) then
28: (* critical section *)
29: w flag1 0
30: w flag0 1
31: fi
32: while true
33:
```

---

**Invariant specification** $S_{inv}$

![Diagram showing invariant specification $S_{inv}$]

In this context, the invariant specification $S_{inv}$ is used to ensure that the program maintains certain properties throughout its execution. This is crucial for proving the correctness of the program, especially in concurrent systems where multiple processes interact with shared resources.

---

**Mutual exclusion**

```plaintext
{0: latch0 = 0; flag0 = 0; latch1 = 1; flag1 = 1; }
1: do {i}
2: do {j}
3: r R0 latch0 `{ ← L0`,i}`
4: while (R0=0) `{k_i}`
5: w latch0 0
6: r R0 flag0 `{ ← F0`,i}`
7: if (R0`̸=0) then
8: (* critical section *)
9: w flag0 0
10: w flag1 1
11: fi
12:while true
13:
21:do {f}
22: do {m_f}
23: r R1 latch1 `{ ← L1`,m_f}`
24: while (R1=0) `{n_f}`
25: w latch1 0
26: r R1 flag1 `{ ← F1`,f}`
27: if (R1`̸=0) then
28: (* critical section *)
29: w flag1 0
30: w flag0 1
31: fi
32: while true
33:
```

(invariant $S_{inv}$ is elsewhere true)
Analytic semantics = Anarchic semantics + communication constraints

Local invariants

Analytics semantics with cuts

Local invariant

- Attached to each program point \( \ell \) of each process \( p \)
- Depends on
  - Program points of all other processes \( \kappa \)
  - Stamps \( \theta \) of all processes
  - Local registers of all processes \( \rho \)
  - Pythia variables \( \nu \)
  - Communications \( (rf) \)
**Communication relation rf**

- rf: relation between write and read events
- Each rf is encoded by \( \Gamma \), a set of pairs

\[
\tau f\langle x, \langle \ell, \theta, v \rangle \rangle
\]

<table>
<thead>
<tr>
<th>Pythia variable of the read event</th>
<th>Program label of the write action</th>
<th>Stamp of the write event</th>
<th>Value write</th>
</tr>
</thead>
</table>

- \( \Gamma \in \Gamma \) (the set of all possible communications rf)

**Anarchic communications**

- Any read can read from any write on the same shared variable (location)

\[
RL0_i^r \triangleq \{ \tau f(L0_i^r, (0:..., 0)), \tau f(L0_i^r, (5:..., 0)), \tau f(L0_i^r, (30:..., 1)) \mid i_5 \in N \land i_3 \in N \}
\]

1. do \{ i \}
2. do \{ m_i \}
3. r[] Rl0 latch0 \( \langle \sim L0_i^r \rangle \)
4. while (Rl0=0) \{ k_i \}
5. w[] latch0 0
6. r[] RF0 flag0 \( \langle \sim RF^r \rangle \)
7. if (RF0=0) then
8. \( \langle \text{critical section} \rangle \)
9. w[] flag0 0
10. w[] latch1 1
11. fi
12: while true
13: 

14: do \{ i \}
15: do \{ m_i \}
16: r[] Rl1 latch1 \( \langle \sim L1_i^m \rangle \)
17: while (Rl1=0) \{ n_i \}
18: w[] latch 0
19: r[] RF1 flag1 \( \langle \sim RF^l \rangle \)
20: if (RF1=0) then
21. \( \langle \text{critical section} \rangle \)
22. w[] flag1 0
23. r[] Rl1 latch1 \( \langle \sim L1_i^m \rangle \)
24. while (Rl1=0) \{ n_i \}
25. w[] latch 0
26. r[] RF1 flag1 \( \langle \sim RF^l \rangle \)
27. if (RF1=0) then
28. \( \langle \text{critical section} \rangle \)
29. w[] flag1 0
30. w[] latch1 1
31. fi
32: while true
33: }

- Possible communications for each read at each stamp (point in the execution):

\[
RL0_i^r \triangleq \{ \tau f(L0_i^r, (0:..., 0)), \tau f(L0_i^r, (5:..., 0)), \tau f(L0_i^r, (30:..., 1)) \mid i_5 \in N \land i_3 \in N \}
\]

- Anarchic communications:

\[
\Gamma = \{ \{ r0_i^r, r0_i^r, r1_i^m, r1_i^m \} \mid i \in N \land j, i \in [0, k_i] \land l \in N \land j \in [0, n_i] \mid \forall i \in N \land j, i \in [0, k_i] \land 0 \in RL0_i^r \land 0 \in RF^r \land 0 \in RF^l \land 0 \in RL1_i^m \land 0 \in RL1_i^m \land 0 \in RF^l \land 0 \in RF^l \land 0 \in RF^l \land 0 \in RF^l \}
\]

- Anarchic semantics: \( \Gamma \in \bar{\Gamma} \)
- WCM semantics: \( \Gamma \in \bar{\Gamma}, \Gamma \subseteq \bar{\Gamma} \)
Inductive invariant $S_{ind}$

- $S_{ind}$ is inductive under hypothesis $S_{com}$ iff, assuming $S_{com}$ we have:
  - $S_{ind}$ is true at the beginning of an execution
  - If $S_{ind}$ is true during execution is remains true after one more computation or communication step

- $S_{inv}$ holds under hypothesis $S_{com}$
  $$S_{ind} \Rightarrow S_{inv}$$
  $$S_{com} \Rightarrow S_{inv}$$

Possible communications
Inductive invariant

[0: latch0 = 0; flag0 = 0; latch1 = 1; flag1 = 1; ]

1: \( P \in F \) 
   do [r]\n
2: \( P \in F \) 
   do [l]\n
3: \( r() \) Rl0 latch0 \( \sim \) \( L0'_0 \) 
   while (Rl0 = 0) \{ \} \n
4: \( P \in F \land Rl0 \land \{ r(0)Rl0\}_F \lor \{ r(0)Rl0\}_F \} \) 
   while (Rl0 = 0) \{ \} \n
5: \( P \in F \land Rl0 \land \{ r(0)Rl0\}_F \) 
   w latch0 0 \n
6: \( P \in F \land \{ r(0)Rl0\}_F \) 
   r() Rf0 flag0 \( \sim \) \( F' \) 
   while \( \{ r(0)Rl0\}_F \) \n
7: \( P \in F \land r(0)Rl0 \land \{ r(0)Rl0\}_F \) 
   \( P \in F \land r(0)Rl0 \land \{ r(0)Rl0\}_F \) 
   if (Rl0 = 0) then \( \{ r(0)Rl0\}_F \) 

8: \( P \in F \land \{ r(0)Rl0\}_F \) 
   r() Rf0 flag0 \( \sim \) \( F' \) 
   while \( \{ r(0)Rl0\}_F \) \n
9: \( P \in F \land \{ r(0)Rl0\}_F \) 
   \( P \in F \land \{ r(0)Rl0\}_F \) 
   r() Rf0 flag0 \( \sim \) \( F' \) 
   while \( \{ r(0)Rl0\}_F \) \n
10: \( P \in F \land \{ r(0)Rl0\}_F \) 
    \( P \in F \land \{ r(0)Rl0\}_F \) 
    \( P \in F \land \{ r(0)Rl0\}_F \) 
    \( P \in F \land \{ r(0)Rl0\}_F \) 
    if (Rl0 = 0) then \( \{ r(0)Rl0\}_F \) 

11: \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
    \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
    \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
    \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
    if (Rl0 = 0) then \( \{ r(0)Rl0\}_F \) 

12: \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
   \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
   \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
   \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
   if (Rl0 = 0) then \( \{ r(0)Rl0\}_F \) 

13: \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
   \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
   \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
   \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
   if (Rl0 = 0) then \( \{ r(0)Rl0\}_F \) 

Inductive invariant

[0: latch0 = 0; flag0 = 0; latch1 = 1; flag1 = 1; ]

1: \( P \in F \) 
   do [r]\n
2: \( P \in F \) 
   do [l]\n
3: \( r() \) Rl0 latch0 \( \sim \) \( L0'_0 \) 
   while (Rl0 = 0) \{ \} \n
4: \( P \in F \land Rl0 \land \{ r(0)Rl0\}_F \lor \{ r(0)Rl0\}_F \} \) 
   while (Rl0 = 0) \{ \} \n
5: \( P \in F \land Rl0 \land \{ r(0)Rl0\}_F \) 
   w latch0 0 \n
6: \( P \in F \land \{ r(0)Rl0\}_F \) 
   r() Rf0 flag0 \( \sim \) \( F' \) 
   while \( \{ r(0)Rl0\}_F \) \n
7: \( P \in F \land r(0)Rl0 \land \{ r(0)Rl0\}_F \) 
   \( P \in F \land r(0)Rl0 \land \{ r(0)Rl0\}_F \) 
   if (Rl0 = 0) then \( \{ r(0)Rl0\}_F \) 

8: \( P \in F \land \{ r(0)Rl0\}_F \) 
   r() Rf0 flag0 \( \sim \) \( F' \) 
   while \( \{ r(0)Rl0\}_F \) \n
9: \( P \in F \land \{ r(0)Rl0\}_F \) 
   \( P \in F \land \{ r(0)Rl0\}_F \) 
   r() Rf0 flag0 \( \sim \) \( F' \) 
   while \( \{ r(0)Rl0\}_F \) \n
10: \( P \in F \land \{ r(0)Rl0\}_F \) 
    \( P \in F \land \{ r(0)Rl0\}_F \) 
    \( P \in F \land \{ r(0)Rl0\}_F \) 
    \( P \in F \land \{ r(0)Rl0\}_F \) 
    if (Rl0 = 0) then \( \{ r(0)Rl0\}_F \) 

11: \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
    \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
    \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
    \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
    if (Rl0 = 0) then \( \{ r(0)Rl0\}_F \) 

12: \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
   \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
   \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
   \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
   if (Rl0 = 0) then \( \{ r(0)Rl0\}_F \) 

13: \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
   \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
   \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
   \( P \in F \land \{ r(0)Rl0\}_F \land \{ r(0)Rl0\}_F \) 
   if (Rl0 = 0) then \( \{ r(0)Rl0\}_F \)
Calculus design of the communication specification

$$\neg S_{\text{com}}(\Gamma, \Gamma) \land S_{\text{ind}}(\Gamma, \Gamma)$$

(\text{def. invariance specification } S_{\text{com}})

$$\Rightarrow \neg S_{\text{com}}(\Gamma, \Gamma)$$

$$S_{\text{com}}(\Gamma, \Gamma) \equiv (S_{\text{com}}(\Gamma, \Gamma) \land S_{\text{com}}(\Gamma, \Gamma) \land S_{\text{com}}(\Gamma, \Gamma) \land S_{\text{com}}(\Gamma, \Gamma))$$

This proves $S_{\text{com}}$ sufficient for correctness

Counter-examples prove $S_{\text{com}}$ necessary $\Rightarrow S_{\text{com}}$ is the weakest WCM requirement for correctness
Conditional invariance proof

Sequential proof $\ell = \kappa$ and $p = q$

For a test instruction $\kappa : b[x]$ operation $I_i \kappa'$:

$$s = P_{p,r}^{\ell,\kappa}(\theta_0, \theta_1, \theta_2, \theta_3)$$

$\kappa$: specification of communication consistency algorithm $A$

Invariant specification of $A$

Communication specification of $A$

Consistency hypotheses of $A$

Consistency model $M$

algorithm $A$

algorithm $A$ proved correct w.r.t. $M_a$, and $S_a := M_a$

algorithm $A$ proved correct w.r.t. $M$, and $S := M$

Inclusion proof $M_a \Rightarrow M$

Inclusion proof $S_a \Rightarrow S$

Sequential proof $\ell = \kappa$ and $p = q$

For a read instruction $\kappa : x[r] X \kappa'$:

$$r \in \mathcal{R}_{p,r}^{\ell,\kappa'}$$

$\ell$: operation $l$

$\kappa$: consistent w.r.t. $M$

$\kappa'$: consistent w.r.t. $M$

$\ell$: local variable

$\kappa$: local variable

For local side-effect free marker instructions $\kappa : instr \kappa'$

where $ instr = f[x] [i_1 \cdots i_n] \{f_{i_1} \cdots f_{i_n}\}$, $x[r] X$ r-value, beginnrw $[x]$ x. endnrw $[x]$ x:

$$s = P_{p,r}^{\ell,\kappa'}$$

$\kappa$: specification of communication consistency algorithm $A$

Invariant specification of $A$

Communication specification of $A$

Consistency hypotheses of $A$

Consistency model $M$

algorithm $A$

algorithm $A$ proved correct w.r.t. $M_a$, and $S_a := M_a$

algorithm $A$ proved correct w.r.t. $M$, and $S := M$

Inclusion proof $M_a \Rightarrow M$

Inclusion proof $S_a \Rightarrow S$
Communication proof

The local invariants of process $p$ depend only on $\Gamma$ and local registers or Pythia variables unchanged by a step in the other process.

Communication condition

COM$_p^G$[rf] $\Delta$ $\Sigma_{\text{read}}(\ell)[rf] \land \Sigma_{\text{write}}(\ell)[rf]$

- A read event can read from only one write event.
  $\Sigma_{\text{write}}(\ell)[rf] \land \forall r. (w, f, r, \ell) \in \Gamma \land \forall s. (w, f, r, \ell) \in \Gamma \implies (w, f, r, \ell) \in \Gamma$

Communication proof

- All process read instructions $\ell : x[n] R x[k]$ must read either from an initial or a reachable program write, allowed by the communication hypothesis $\exists P[X_1, \ldots, X_n]$ means that all free variables in
  $\text{COM}_p^G[\theta_p, rf] \land rf = \exists r \forall m((q, p, x, x, x, x, x) \in r) \land (q = \text{start} \land v = 0)$. 

Communication proof

- The values $v$ allowed to be read by the communication hypothesis must originate from reachable program write instructions
  $\ell : x[n] R x[k]$. 

Proof of mutual exclusion and non-interference of a program $\text{PermapSQL}$, Datalogic Seminar 1447, 20-21 November 2014

© J. Alglave & F. Causse
Inclusion proof

Method

- The communication specification is
  \[ S_{\text{com}}(\Gamma, \overline{\Gamma}) \triangleq (\text{at}\{8\} \land \text{at}\{28\}) \implies (S_{\text{com}1}(\Gamma, \overline{\Gamma}) \land S_{\text{com}2}(\Gamma, \overline{\Gamma}) \land S_{\text{com}3}(\Gamma, \overline{\Gamma}) \land S_{\text{com}4}(\Gamma, \overline{\Gamma})) \]

- The consistency specification must satisfy
  \[ H_{\text{com}}(\Gamma, \overline{\Gamma}) \Rightarrow S_{\text{com}}(\Gamma, \overline{\Gamma}) \quad \text{i.e.} \quad \neg S_{\text{com}}(\Gamma, \overline{\Gamma}) \Rightarrow \neg H_{\text{com}}(\Gamma, \overline{\Gamma}) \]

- So the design of \( H_{\text{com}}(\Gamma, \overline{\Gamma}) \) must forbid the erroneous communications specified by the communication specification
  \[ (\text{at}\{8\} \land \text{at}\{28\} \land \bigvee_{i=1}^{4} \neg S_{\text{com}i}(\Gamma, \overline{\Gamma})) \implies \bigvee_{i=1}^{4} \neg H_{\text{com}i}(\Gamma, \overline{\Gamma}) \]

\[
\begin{align*}
S_{\text{com}1} & \triangleq \neg(\exists i, k_i, \ell, n_{\ell}, \ell_{30}, \ell_{29} \in \mathbb{N} \land \ell_{30} > 0 \land n_{\ell} \in \mathbb{N} \land \Gamma \in \Gamma \land \text{tf}(L0_{k_i}^{\ell}, \langle 30: \ell_{30}, 1 \rangle) \in \Gamma \land \text{tf}(F0_{\ell}^{\ell}, \langle 29: \ell_{29}, 1 \rangle) \in \Gamma \land \text{tf}(L1_{n_{\ell}}^{\ell}, \langle 0: \ell, 1 \rangle) \in \Gamma) \\
S_{\text{com}2} & \triangleq \neg(\exists i, k_i, \ell, n_{\ell}, \ell_{30}, \ell_{29}, i_0 \in \mathbb{N} \land \Gamma \in \Gamma \land \text{tf}(L0_{k_i}^{\ell}, \langle 30: \ell_{30}, 1 \rangle) \in \Gamma \land \text{tf}(F0_{\ell}^{\ell}, \langle 29: \ell_{29}, 1 \rangle) \in \Gamma \land \text{tf}(L1_{n_{\ell}}^{\ell}, \langle 0: \ell, 1 \rangle) \in \Gamma)
\end{align*}
\]
\[
S_{\text{com}_3} \triangleq -\exists \langle i, k_i, \ell, n_\ell, \ell_{30}, \ell_{29}, i_{10}, i_9 \rangle \in \mathbb{N} . \ \Gamma \in \Gamma \land \text{tf}(L_0^i_{k_i}, (30:, \ell_{30}, 1)) \in \Gamma \land \text{tf}(F_0^i, (29:, \ell_{29}, 1)) \in \Gamma \land \text{tf}(L_1^\ell_{n_\ell}, (10:, i_{10}, 1)) \in \Gamma \land \text{tf}(F_1^\ell, (9:, i_9, 1)) \in \Gamma
\]

\[
\{0: \text{latch}0 = 0; \text{flag}0 = 0; \text{latch}1 = 1; \text{flag}1 = 1; \}
1: \text{do } \{i\}
2: \text{do } \{j_i\}
3: \text{r}[] \text{latch}0 \quad \text{r}[] \text{latch}1 \quad \Rightarrow \quad L_0^i_{k_i} \quad \Rightarrow \quad L_1^\ell_{n_\ell}
4: \text{while } (\text{Rl}0 = 0) \ \{k_i\}
5: \text{w}[] \text{latch}0 0
6: \text{r}[] \text{flag}0 \quad \text{r}[] \text{flag}1 \quad \Rightarrow \quad F_0^i \quad \Rightarrow \quad F_1^\ell
7: \text{if } (\text{Rf}0 \neq 0) \ \text{then}
8: \text{w}[] \text{flag}0 0
9: \text{w}[] \text{flag}1 1
10: \text{w}[] \text{latch}1 1
11: \text{fi}
12: \text{while } \text{true}
13: \text{no prophecy beyond cut during execution}
\]

\[
S_{\text{com}_4} \triangleq -\exists \langle i, k_i, \ell, n_\ell, \ell_{30}, \ell_{29}, i_{10}, i_9 \rangle \in \mathbb{N} . \ \Gamma \in \Gamma \land \text{tf}(L_0^i_{k_i}, (30:, \ell_{30}, 1)) \in \Gamma \land \text{tf}(F_0^i, (29:, \ell_{29}, 1)) \in \Gamma \land \text{tf}(L_1^\ell_{n_\ell}, (10:, i_{10}, 1)) \in \Gamma \land \text{tf}(F_1^\ell, (9:, i_9, 1)) \in \Gamma
\]

\[
\{0: \text{latch}0 = 0; \text{flag}0 = 0; \text{latch}1 = 1; \text{flag}1 = 1; \}
1: \text{do } \{i\}
2: \text{do } \{j_i\}
3: \text{r}[] \text{latch}0 \quad \text{r}[] \text{latch}1 \quad \Rightarrow \quad L_0^i_{k_i} \quad \Rightarrow \quad L_1^\ell_{n_\ell}
4: \text{while } (\text{Rl}0 = 0) \ \{k_i\}
5: \text{w}[] \text{latch}0 0
6: \text{r}[] \text{flag}0 \quad \text{r}[] \text{flag}1 \quad \Rightarrow \quad F_0^i \quad \Rightarrow \quad F_1^\ell
7: \text{if } (\text{Rf}0 \neq 0) \ \text{then}
8: \text{w}[] \text{flag}0 0
9: \text{w}[] \text{flag}1 1
10: \text{w}[] \text{latch}1 1
11: \text{fi}
12: \text{while } \text{true}
13: \text{no prophecy beyond cut during execution}
\]

Conclusion on mutual exclusion

- PostgreSQL is correct on architectures satisfying the "no prophecy beyond cut during execution" property

Intuition on necessity: when waiting for a spinlock, you should look at its current value, not at later ones!

A static condition to impose a dynamic condition:

\[
\{0: \text{latch}0 = 0; \text{flag}0 = 0; \text{latch}1 = 1; \text{flag}1 = 1; \}
1: \text{do } \{i\}
2: \text{do } \{j_i\}
3: \text{r}[] \text{latch}0 \quad \text{r}[] \text{latch}1 \quad \Rightarrow \quad L_0^i_{k_i} \quad \Rightarrow \quad L_1^\ell_{n_\ell}
4: \text{while } (\text{Rl}0 = 0) \ \{k_i\}
5: \text{w}[] \text{latch}0 0
6: \text{r}[] \text{flag}0 \quad \text{r}[] \text{flag}1 \quad \Rightarrow \quad F_0^i \quad \Rightarrow \quad F_1^\ell
7: \text{if } (\text{Rf}0 \neq 0) \ \text{then}
8: \text{f}(\text{cut})
\]

(no prophecy)

\[
\text{enum fences} = \text{'cut instructions } F[\text{('cut)]]

\text{let cut} = (\text{tag2events('cut) } \ast \text{tag2events('cut) }) \& \text{ext irreflexive rf; po; cut; po}
\]
Prevents valid executions

```plaintext
{0: latch0 = 0; flag0 = 0; latch1 = 1; flag1 = 1; }
1: do {i
2: j
3: k
4: while (Rl0 = 0)
5: w[latch0 0]
6: r[Rf0 flag0]
7: if (Rf0 ≠ 0) then
8: f[cut]
9: w[flag0 0]
10: w[flag1 0]
11: fi
12: while true
13:}
```

Invalid

```plaintext
{0: latch0 = 0; flag0 = 0; latch1 = 1; flag1 = 1; }
1: do {i
2: j
3: k
4: while (Rl0 = 0)
5: w[latch0 0]
6: r[Rf0 flag0]
7: if (Rf0 ≠ 0) then
8: f[cut]
9: w[flag0 0]
10: w[flag1 0]
11: fi
12: while true
13:}
```

Valid

```plaintext
irreflexive rf; po; cut; po
```

Proof of mutual exclusion and non-starvation of a program: PosnGenQ. Dig. Seminars 1471, 29-30 November 2016

Difference with Lamport/Owicki-Gries

- The communications in L/O-G are fixed in the semantics (SC) for all executions:

  (a) No prophecy beyond cuts
  (b) Read from last write

⇒ entangled with the verification conditions
⇒ impossible to reason on one execution trace only

Reasoning on only one execution

- An execution is entirely determined by its read-from relation rf
- The verification conditions depend on a set \( \Gamma \) of verification conditions
- By choosing \( \Gamma = \{ rf \} \), we can reason on this execution
- This execution satisfies the inductive invariant \( S_{ind}(rf) \)
- To prove that this execution is impossible it is sufficient to prove that \( S_{ind}(rf) \) cannot hold (according to the verification conditions)
- Since the method is sound, if the verification conditions are not satisfied, the execution is excluded by the semantics
(1) Both processes starve in spin loops

(2) Both processes never enter their critical section

(1) Both processes be the communication for such a trace (encoded in \( \Gamma_a \))

• let \( rf \) be the communication for such a trace (encoded in \( \Gamma_a \))

• invariant false after both spin loops

• so \( latch_1 \) is in 22: can only be read from initialization

• so \( latch_1 \) is 1 not 0, a contradiction

(2) Both processes never enter their critical section

• let \( rf \) be the communication for such a trace (encoded in \( \Gamma_a \))

• the invariant inside critical sections must be false

Proof of mutual exclusion and non-starvation of a program: PosneggQL. Digital Object Identifier: 10.1109/ISG.2014.105
(2) Both processes never enter their critical section

(2) Both processes never enter their critical section

(3) Process P1 stuck in spin loop (no hypothesis on P0)
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 v to a shared variable x and only that value v
  - and from a later cut point of that execution, a process infinitely often reads to that variable x
  - then the reads will end up reading that value v

---

Proof of mutual exclusion and non-interception of a program. 
R. Alglave & F. Cousseau. 
(5) Process P0 leaves spin loop but always fails entering its CS

(6) Both processes eventually starve in spin loop

(7) Eventually, P0 starves in spin loop, P1 never enters its CS

(8) Eventually, P1 starves in spin loop, P0 never enters its CS
(9) P0 and P1 always leave spin loop and never enter their CS

- P0 and P1 eventually never starve and never enter their critical sections
- They both have a last entrance in their critical sections
- This last write of 1 to the latches will, by communication fairness, eventually reach the memory
- Then we only have infinitely many writes of 0 to the latches
- So the read of the latches in the spin loops will eventually always read 0
- So from then on, by communication fairness, all the reads will be from 0, in reads of the latch will be zero
- In contradiction with the fact that the spin loop is always exited
- The barrier prevents infinitely postponing the write 0 actions

Conclusion

- The proof method is parameterized by consistency hypotheses, expressed in
  - Invariance form: $S_{\text{com}}$
  - Consistency form: $H_{\text{com}}$ (e.g. in cat)
- Program not logic/architecture/consistency model dependent (hence the proof is portable)
- Can reason on arbitrary subsets of anarchic executions (hence flexible e.g. non-starvation)

Proposed design methodology

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

- 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
  ⇒ but we can start with $S_{com}$

Thanks

- Patrick Cousot thanks Luc Maranget for his precious help at Dagstuhl on the non-starvation part.

The End, Thank You