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

1: w[] \text{ F1 true}

2: w[] \text{ T 2}

3: do

5: r[] \text{ R1 F2}

6: r[] \text{ R2 T}

7: while R1 \land R2 \neq 1

8: ¬at 28

9: w[] \text{ F1 false}

10:

21: w[] \text{ F2 true;}

22: w[] \text{ T 1;}

23: do

25: r[] \text{ R3 F1;}

26: r[] \text{ R4 T;}

27: while R3 \land R4 \neq 2;

28: ¬at 8

29: w[] \text{ F2 false;}

39:

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

invariance proof

\[ S_{com} \implies S_{inv} \]

consistency proof

\[ M \implies H_{com} \]
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

Initialization

States

Transition constraints

Events

Processes

1. Read x
2. Write x

1. Read x
2. Write x

...
The read-from relation $rf$

- **initialization**
- **states**
- **transition constraints**
- **events**

**Read events:**
- $read\ x$
- $read\ x$
- $read\ x$
- $read\ x$
- $read\ x$
- $read\ x$

**Write events:**
- $write\ x$
- $write\ x$
- $write\ x$
- $write\ x$
- $write\ x$
- $write\ x$

**Processes:**
- Process 1
- Process 2
- \(\cdots\)
- Process \(n\)
Cuts

initialization

states

transition constraints

events

read x

write x

read x

read x

write x

write x

Process 1

... Process n
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 \(\rightarrow\) Pythia Variables

**Difficulty**

- Its *dynamic*, not static!
- A program read action can read from a different write each time it is executed \(\rightarrow\) 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:
1:w[] F1 true
2:w[] T 2
3:do {i}
4: r[] R1 F2 \(\leadsto F2^i_4\)
5: r[] R2 T \(\leadsto T^i_5\)
6:while R1 ∧ R2 ≠ 1 {i_end}
7:skip (* CS1 *)
8:w[] F1 false

P1:
10:w[] F2 true;
11:w[] T 1;
12:do {j}
13: r[] R3 F1; \(\leadsto F1^j_3\)
14: r[] R4 T; \(\leadsto T^j_14\)
15:while R3 ∧ R4 ≠ 2; {j_end}
16:skip (* CS2 *)
17:w[] F2 false;

Stamps (loop counters)

Stamps (on loop exit)
Example (Peterson)

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 {→ F2i4}
5: r[] R2 T {→ Ti5}
6: while R1 ∧ R2 ≠ 1 { i end }
7: skip (* CS1 *)
8: w[] F1 false

P1:
10: w[] F2 true;
11: w[] T 1;
12: do { j }
13: r[] R3 F1; {→ F1j13}
14: r[] R4 T; {→ Tj14}
15: while R3 ∧ R4 ≠ 2; { j end }
16: skip (* CS2 *)
17: w[] F2 false;
The abstraction
The invariance abstraction

• For each process
The invariance abstraction

• For each process

• For each program point of that process
The invariance abstraction

- For each process
  - For each program point of that process
    - For each execution of the program
The invariance abstraction

- 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
The invariance abstraction

• 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 invariance abstraction

• 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 invariance abstraction

- 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: Peterson

0: { w F1 false; w F2 false; w T 0; }
{F1=false ∧ F2=false ∧ T=0} }
1: {R1=0 ∧ R2=0}
   w[] F1 true
2: {R1=0 ∧ R2=0}
   w[] T 2
3: {R1=0 ∧ R2=0}
do {i}
4: {((i=0 ∧ R1=0 ∧ R2=0) ∨
   (i>0 ∧ R1=F2_4^{i-1} ∧ R2=T_5^{i-1}))}
   r[] R1 F2 {→ F2_4}
5: {R1=F2_4^i ∧ (i=0 ∧ R2=0) ∨
   (i>0 ∧ R2=T_5^{i-1})}
   r[] R2 T {→ T_5^i}
6: {R1=F2_4^i ∧ R2=T_5^i}
   while R1 ∧ R2≠1 {i_end}
7: {¬F2_4^i ∧ T_5^i_end=1}
   skip (* CS1 *)
8: {¬F2_4^i ∧ T_5^i_end=1}
   w[] F1 false
9: {¬F2_4^i ∧ T_5^i_end=1}
10: {R3=0 ∧ R4=0}
    w[] F2 true;
11: {R3=0 ∧ R4=0}
    w[] T 1;
12: {R3=0 ∧ R4=0}
do {j}
13: {((j=0 ∧ R3=0 ∧ R4=0) ∨
    (j>0 ∧ R3=F1_1^{j-1} ∧ R4=T_14^{j-1}))}
    r[] R3 F1 {→ F1_1^j}
14: {R3=F1_1^{j} ∧ (j=0 ∧ R4=0) ∨
    (j>0 ∧ R4=T_14^{j-1})}
    r[] R4 T; {→ T_14^j}
15: {R3=F1_1^{j} ∧ R4=T_14^j}
    while R3 ∧ R4≠2 {j_end} ;
16: {¬F1_1^{j_end} ∧ T_14^{j_end}=2}
    skip (* CS2 *)
17: {¬F1_1^{j_end} ∧ T_14^{j_end}=2}
    w[] F1 false;
18: {¬F1_1^{j_end} ∧ T_14^{j_end}=2}
Example: Peterson

0: { w F1 false; w F2 false; w T 0; }
{F1=false  \&  F2=false  \&  T=0} }
1: {R1=0  \&  R2=0}
w[] F1 true
2: {R1=0  \&  R2=0}
w[] T 2
3:...
4: (i=0  \&  R1=0  \&  R2=0) \lor 
   (i>0  \&  R1=F2^{i-1}  \&  R2=T^{i-1}_5 )}
5: {R1=F2^i  \&  (i=0  \&  R2=0) \lor 
   (i>0  \&  R2=T^{i-1}_5 )}
   r[] R2 T {\sim T^i_5 }
6: {R1=F2^i  \&  R2=T^i_5 }
   while R1  \&  R2\neq 1 {i_end}
7: {\neg F2^i  \lor T^i_end=1}
   skip (* CS1 *)
8: {\neg F2^i  \lor T^i_end=1}
   w[] F1 false
9: {\neg F2^i  \lor T^i_end=1}
10: {R3=0  \&  R4=0}
w[] F2 true;
11: {R3=0  \&  R4=0}
w[] T 1;

14: {R3=F1^j  \&  (j=0  \&  R4=0) \lor 
   (j>0  \&  R4=T^{j-1}_4 )}
   r[] R4 T {\sim T^j_4 }
15: {R3=F1^j  \&  R4=T^{j}_4 }
   while R3  \&  R4\neq 2 {j_end} ;
16: {\neg F1^j_end  \lor T^{j_end}=2}
   skip (* CS2 *)
17: {\neg F1^j_end  \lor T^{j_end}=2}
   w[] F1 false;
18: {\neg F1^j_end  \lor T^{j_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 $S_{com} \land H_{com}$
The induction principle

- Given an invariance specification \( S_{inv} \), find a stronger inductive invariant \( S_{ind} \).
- Prove that \( S_{ind} \) satisfies verification conditions:
  - Holds after initialization
  - Remains true after a computation step
  - Remains true after a communication
- Assuming \( S_{com} \), abstraction of the concrete transformer for one computation step

**Verification conditions = abstraction of the concrete transformer for one computation step**
Calculational design of the verification conditions

\[ \alpha_{\text{inv}}(\alpha_{\text{ana}}[H_{\text{com}}](S^a[P])) \subseteq S_{\text{inv}} \]

\[ \Leftrightarrow \alpha_{\text{inv}}(\{\xi \in S^a[P] \mid S[H_{\text{com}}]\xi = \text{allowed}\}) \subsetneq S_{\text{inv}} \quad \text{def. } \alpha_{\text{ana}}[H_{\text{com}}] \]

\[ \Leftrightarrow \alpha_{\text{inv}}(S^a[P] \cap \{\xi \in S^a[P] \mid S[H_{\text{com}}]\xi = \text{allowed}\}) \subsetneq S_{\text{inv}} \quad \text{def. } \cap \]

\[ \Leftrightarrow \alpha_{\text{inv}}(S^a[P]) \cap \alpha_{\text{inv}}(\{\xi \in \Xi \mid S[H_{\text{com}}]\xi = \text{allowed}\}) \subsetneq S_{\text{inv}} \]

\[ \quad \text{since } \alpha_{\text{inv}} \text{ preserves intersections} \]

\[ \Leftrightarrow \alpha_{\text{inv}}(S^a[P]) \cap \alpha_{\text{inv}}(\alpha_{\text{ana}}[H_{\text{com}}](S^a[P])) \subsetneq S_{\text{inv}} \quad \text{def. } \alpha_{\text{ana}}[H_{\text{com}}] \]

\[ \Leftrightarrow \exists S_{\text{com}} \cdot \alpha_{\text{inv}}(S^a[P]) \cap S_{\text{com}} \subsetneq S_{\text{inv}} \land \alpha_{\text{inv}}(\alpha_{\text{ana}}[H_{\text{com}}](S^a[P])) \subsetneq S_{\text{com}} \]

\[ \text{for soundness, we have } \alpha_{\text{inv}}(S^a[P]) \cap \alpha_{\text{inv}}(\alpha_{\text{ana}}[H_{\text{com}}](S^a[P])) \subsetneq \alpha_{\text{inv}}(S^a[P]) \cap S_{\text{com}} \subsetneq S_{\text{inv}} ; \]

\[ \Rightarrow \forall S_{\text{com}} \cdot (S_{\text{com}} \Rightarrow S_{\text{inv}}) \land (H_{\text{com}} \Rightarrow S_{\text{com}}) \]

by defining the conditional invariance proof \( S_{\text{com}} \Rightarrow S_{\text{inv}} \) to be \( \alpha_{\text{inv}}(S^a[P]) \cap S_{\text{com}} \subsetneq S_{\text{inv}} \) and the inclusion proof \( H_{\text{com}} \Rightarrow S_{\text{com}} \) to be \( \alpha_{\text{inv}}(\alpha_{\text{ana}}[H_{\text{com}}](S^a[P])) \subsetneq S_{\text{com}} \).

\[ \text{...} \]

\[ \text{...} \]
Calculational design of the verification conditions

\[ \Rightarrow \exists S_{com} . (S_{com} \Rightarrow S_{inv}) \land (H_{com} \Rightarrow S_{com}) \]

(\Rightarrow) For completeness, we choose to describe exactly the communications that is \( S_{com} = \alpha_{\text{inv}}(\alpha_{\text{ana}}[H_{com}](S^a[P])) \).

\[ \Leftarrow \exists S_{com} . (S_{com} \Rightarrow S_{inv}) \land (H_{com} \Rightarrow S_{com}) \]

by defining the conditional invariance proof \( S_{com} \Rightarrow S_{inv} \) to be \( \alpha_{\text{inv}}(S^a[P]) \land S_{com} \Rightarrow S_{inv} \) and the inclusion proof \( H_{com} \Rightarrow S_{com} \) to be \( \alpha_{\text{inv}}(\alpha_{\text{ana}}[H_{com}](S^a[P])) \Rightarrow S_{com} \).

\[ \vdots \]

\[ \vdots \]

\[ \vdots \]
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:

  $$S_{com} \implies (S_{ind} \implies S_{inv})$$

- Communication hypothesis
- Program inductive anarchic invariant
- Program invariance specification

- i.e. $(S_{ind} \land \neg S_{inv}) \implies \neg S_{com}$

- **Necessary**: by counter examples
Proving Consistency

\[ H_{com} \implies S_{com} \]

- cat
- invariant
Proof method

• Obtained by calculational design:

\[
\alpha_{inv}(\alpha_{ana}[H_{com}](S^a[p])) \subseteq S_{com}
\]

\[\Leftrightarrow \alpha_{inv}(S^{ana}[H_{com}][p]) \subseteq S_{com} \quad \text{(def. } S^{ana}[H_{com}][p])\]

\[\Leftrightarrow \forall \xi \in S^{ana}[H_{com}][p], \alpha_{inv}(\{\xi\}) \subseteq S_{com} \quad \text{(}\alpha_{inv} \text{ preserves } \cup\text{)}\]

\[\Leftrightarrow \forall \xi \in S^{ana}[H_{com}][p], \bigcup_{p=1}^{n-1} \bigcup_{L \in P_p} \{\alpha_{inv}(\xi')_p(L) | \xi' \in \{\xi\}\} \subseteq S_{com} \quad \text{(def. (19) of } \alpha_{inv})\]

\[\Leftrightarrow \forall (\tau_{start} \times \prod_{p=0}^{n-1} \tau_p \times \pi \times \text{rf}) \in S^{ana}[H_{com}][p], \forall p \in [1, n], \forall L \in P_p . \alpha_{inv}(\tau_{start} \times \prod_{p=0}^{n-1} \tau_p \times \pi \times \text{rf})_p(L) \subseteq S_{com}(L) \quad \text{(def. } \in, \subseteq, \text{ and } S^{ana}[H_{com}][p] \text{ so that } \xi \text{ has the form } \xi = \tau_{start} \times \prod_{p=0}^{n-1} \tau_p \times \pi \times \text{rf}. \text{ By def. (19) of } \alpha_{inv} \text{ and } \subseteq, \text{ we get})\]

\[\Leftrightarrow \forall (\tau_{start} \times \prod_{p=0}^{n-1} \tau_p \times \pi \times \text{rf}) \in S^{ana}[H_{com}][p], \forall i \in [1, n], \forall L \in P_p, \forall q \in [0, n], \forall k_q < |\tau_q| . \quad \text{(}\tau_{q,k_q} = \mathcal{S}(\kappa_q, q, \kappa_q, \theta_q, q, \rho_q, q, \nu_q, q) \wedge \kappa_p, k_p = L) \Rightarrow \]

\[\langle \kappa_0, 0, 0, 0, 0, \rho_0, k_0, \nu_0, k_0, \cdots, \nu_{p-1}, k_{p-1}, \theta_p, k_p, \rho_p, k_p, \nu_p, k_p, k_{p+1}, \cdots, k_{n-1}, k_{n-1}, \theta_{n-1}, k_{n-1}, k_{n-1}, \nu_{n-1}, k_{n-1}, \text{rf} \rangle \in S_{com_i}(L)\]
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)
Example (Peterson)

Communication scenarios violating $S_{com}$ for Peterson

$$S_{com} \triangleq \neg [\exists i, j. [rf(F2_4^i, 0, false) \lor rf(F2_4^i, 17, false) \lor rf(T_5^i, 11, 1) \land rf(F1_{13}^j, 0, false) \lor rf(F1_{13}^j, 8, false) \lor rf(T_{14}^j, 2, 2)]]$$
Incompleteness

• In general you have to add fences for $H_{\text{com}}$ (do not change the invariants, $S_{\text{inv}}$, $S_{\text{ind}}$, and $S_{\text{com}}$ remain valid)

• $S_{\text{com}}$ can refer to communicated values not $H_{\text{com}}$ in cat (redesign your algorithm without assuming that the hardware does know about communicated values)

• cat may not be expressive enough:

![Diagram showing the incompleteness of cat](image-url)
Proving Architectural Consistency

\[ M \Rightarrow H_{com} \]

\[
\begin{array}{c|c|c|c|c}
 & \text{cat} & \text{cat} \\
\end{array}
\]
\[ H_{com} \Rightarrow M \text{ in cat} \]

- 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. PostgreSQL)

- 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 $\nu$ to a shared variable $x$ and only that value $\nu$
  - and from a later cut point of that execution, a process infinitely often repeats reads to that variable $x$
  - then the reads will end up reading that value $\nu$

 (*) 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 $(S_{com} \land S_{inv}) \Rightarrow S_{inv}$
Conclusion

• To implement a parallel algorithm correctly:
  • Implement the program consistency model on an architecture consistency model $M$ (possibly adding fences)
  • Prove $M \Rightarrow S_{com}$

• Or better
  • Find a minimal/weakest $H_{com}$ such that $H_{com} \Rightarrow S_{com}$
  • $M \Rightarrow H_{com}$
More work needed

• Specification of parallel/distributed program consistency models (more refined than architecture 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