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

0: \{ w F1 false; w F2 false; w T 0; \}
1: w[] F1 true
2: w[] T 2
3: do
5: r[] R1 F2
6: r[] R2 T
7: while R1 \land R2 \neq 1
8: \neg \text{at 28}
9: w[] F1 false
10:
21: w[] F2 true;
22: w[] T 1;
23: do
25: r[] R3 F1;
26: r[] R4 T;
27: while R3 \land R4 \neq 2;
28: \neg \text{at 8}
29: w[] F2 false;
39:
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 strengthened 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

\begin{align*}
\text{invariants} \quad \{ & \quad S_{\text{com}} \xrightarrow{1} S_{\text{inv}} \\
\text{cat} \quad \{ & \quad M \xrightarrow{3} H_{\text{com}}
\end{align*}

\text{invariance proof} \quad \text{consistency proof}
The invariance proof method is designed by abstract interpretation of an analytic semantics
Analytic semantics

= 

Anarchic semantics

Γ

Weak consistency model
The anarchic semantics
The anarchic semantics

initialization

read x

write x

read x

states

transition constraints

events

write x

read x

write x

read x

write x

Process 1

Process 2

...  

Process n
The read-from relation $rf$

initialization

states

transition constraints

events

write $x$

write $x$

read $x$

read $x$

read $x$

Process 1

Process 2

... 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 \(\rightarrow\) Pythia Variables
What we know on communications?

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

Difficulty

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

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

0: \{ w F1 false; w F2 false; w T 0; \}
P0:
1: w[] F1 true
2: w[] T 2
3: do \{i\}
4: r[] R1 F2 \{\sim F2^i_4\}
5: r[] R2 T \{\sim T^i_5\}
6: while R1 \land R2 \neq 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; \{\sim F1^j_{13}\}
14: r[] R4 T; \{\sim T^j_{14}\}
15: while R3 \land R4 \neq 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 {→ F2^i_4}
5: r[] R2 T {→ 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; {→ F1^j_13}
14: r[] R4 T; {→ T^j_14}
15: while R3 ∧ R4 ≠ 2; {j_end}
16: skip (* CS2 *)
17: w[] F2 false;

Stamps (loop counters)
Stamps (on loop exit)

Pythia variables
The abstraction
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 $r_f$
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_{i-1} ∧ R2=T^i_5)}
r[] R1 F2 {∼ F2_{i-1}^i}

5: {R1=F2_{i} ∧ (i=0 ∧ R2=0) ∨
  (i>0 ∧ R2=T^i_5)}
r[] R2 T {∼ T^i_5}

6: {R1=F2_{i} ∧ R2=T^i_5}
while R1 ∧ R2≠1 {i_end}

7: {¬F2_{i} ∧ T^i_{end}=1}
  skip (* CS1 *)

8: {¬F2_{i} ∧ T^i_{end}=1}
w[] F1 false

9: {¬F2_{i} ∧ T^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^j_{i+1} ∧ R4=T^j_{14})}
r[] R3 F1 {∼ F1^j_{i+1}^j}

14: {R3=F1^j_{i+1} ∧ (j=0 ∧ R4=0) ∨
  (j>0 ∧ R4=T^j_{14})}
r[] R4 T; {∼ T^j_{14}}

15: {R3=F1^j_{i+1} ∧ R4=T^j_{14}}
while R3 ∧ R4≠2 {j_end} ;

16: {¬F1^j_{i+1} ∧ T^j_{end}=2}
  skip (* CS2 *)

17: {¬F1^j_{i+1} ∧ T^j_{end}=2}
w[] F2 false;

18: {¬F1^j_{i+1} ∧ T^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:
5: { R1=F24 ∧ (i=0 ∧ R2=0) ∨ (i>0 ∧ R2=T5i−1) }
r[] R2 T { i T5 }
6: { R1=F24 ∧ R2=T5 }
while R1 ∧ R2≠1 {i_end}
7: { ¬F24 ∧ T5i_end=1 }
skip (* CS1 *)
8: { ¬F24 ∧ T5i_end=1 }
w[] F1 false
9: { ¬F24 ∧ T5i_end=1 }
10: { R3=0 ∧ R4=0 }
w[] F2 true;
11: { R3=0 ∧ R4=0 }
w[] T 1;
12:
13:
14: { R3=F1j13 ∧ (j=0 ∧ R4=Tj14−1) } 
r[] R4 T; { i T5j }
15: { R3=F1j13 ∧ R4=Tj14−1 }
while R3 ∧ R4≠2 {j_end} ;
16: { ¬F1j13 ∧ Tj14=2 }
skip (* CS2 *)
17: { ¬F1j13 ∧ Tj14=2 }
w[] F2 false;
18: { ¬F1j13 ∧ Tj14=2 }

The calculational design of the verification conditions by abstract interpretation
The induction principle

• Given an invariance specification $S_{inv}$ find a stronger inductive invariant $S_{ind}$

• Prove that $S_{ind}$ satisfy verification conditions
  • Holds after initialization
  • Remains true after a computation step
  • Remains true after a communication

• Assuming $S_{com} / H_{com}$
The induction principle

• Given an invariance specification $S_{inv}$ find a stronger inductive invariant $S_{ind}$

• Prove that $S_{ind}$ satisfy verification conditions
  • Holds after initialization
  • Remains true after a computation step
  • Remains true after a communication

• Assuming $S_{com} / H_{com}$

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

\[ \alpha_{\text{inv}}(\alpha_{\text{ana}}[H_{\text{com}}](S^a[P])) \subseteq S_{\text{inv}} \]
\[ \Leftrightarrow \alpha_{\text{inv}}(\{\xi \in S^a[P] \mid S[H_{\text{com}}]\xi = \text{allowed}\}) \subseteq S_{\text{inv}} \quad \{\text{def. } \alpha_{\text{ana}}[H_{\text{com}}]\} \]
\[ \Leftrightarrow \alpha_{\text{inv}}(S^a[P] \cap \{\xi \in S^a[P] \mid S[H_{\text{com}}]\xi = \text{allowed}\}) \subseteq S_{\text{inv}} \quad \{\text{def. } \cap\} \]
\[ \Leftrightarrow \alpha_{\text{inv}}(S^a[P]) \cap \alpha_{\text{inv}}(\{\xi \in S[H_{\text{com}}]\xi = \text{allowed}\}) \subseteq S_{\text{inv}} \]
\[ \quad \{\text{since } \alpha_{\text{inv}} \text{ preserves intersections}\} \]
\[ \Leftrightarrow \alpha_{\text{inv}}(S^a[P]) \cap \alpha_{\text{inv}}(\alpha_{\text{ana}}[H_{\text{com}}](S^a[P])) \subseteq S_{\text{inv}} \quad \{\text{def. } \alpha_{\text{ana}}[H_{\text{com}}]\} \]
\[ \Leftrightarrow \exists S_{\text{com}} : \alpha_{\text{inv}}(S^a[P]) \cap S_{\text{com}} \subseteq S_{\text{inv}} \land \alpha_{\text{inv}}(\alpha_{\text{ana}}[H_{\text{com}}](S^a[P])) \subseteq S_{\text{com}} \]
\[ \quad (\Leftrightarrow) \text{ For soundness, we have } \alpha_{\text{inv}}(S^a[P]) \cap \alpha_{\text{inv}}(\alpha_{\text{ana}}[H_{\text{com}}](S^a[P])) \subseteq \alpha_{\text{inv}}(S^a[P]) \cap S_{\text{com}} \subseteq S_{\text{inv}}; \]
\[ \Rightarrow \exists S_{\text{com}} : (S_{\text{com}} \Rightarrow S_{\text{inv}}) \land (H_{\text{com}} \Rightarrow S_{\text{com}}) \]
\[ \quad \text{by defining the conditional invariance proof } S_{\text{com}} \Rightarrow S_{\text{inv}} \text{ to be } \alpha_{\text{inv}}(S^a[P]) \cap S_{\text{com}} \subseteq S_{\text{inv}} \text{ and the inclusion proof } H_{\text{com}} \Rightarrow S_{\text{com}} \text{ to be } \alpha_{\text{inv}}(\alpha_{\text{ana}}[H_{\text{com}}](S^a[P])) \subseteq S_{\text{com}}. \]
\[ \cdots \]
\[ \cdots \]
Calcutational design of the verification conditions

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

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

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

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

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

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

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

by defining the conditional invariance proof \(S_{\text{com}} \Rightarrow S_{\text{inv}}\) to be
\[
\alpha_{\text{inv}}(S^a[P]) \cap S_{\text{com}} \subseteq S_{\text{inv}}
\]

and the inclusion proof \(H_{\text{com}} \Rightarrow S_{\text{com}}\) to be
\[
\alpha_{\text{inv}}(\alpha_{\text{ana}}[H_{\text{com}}](S^a[P])) \subseteq S_{\text{com}}
\]

\[
\vdots
\]

\[
\vdots
\]

\[
\vdots
\]
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

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

\[ \text{cat} \]

\[ \text{invariant} \]
Proof method

- Obtained by calculational design:

\[
\alpha_{\text{inv}}(\alpha_{\text{ana}}[H_{\text{com}}](S^a[P])) \subseteq S_{\text{com}} \\
\Leftrightarrow \alpha_{\text{inv}}(S^a_{\text{ana}}[H_{\text{com}}]P) \subseteq S_{\text{com}} \quad \text{(\text{def. } S^a_{\text{ana}}[H_{\text{com}}]P)} \\
\Leftrightarrow \forall \xi \in S^a_{\text{ana}}[H_{\text{com}}]P : \alpha_{\text{inv}}(\{\xi\}) \subseteq S_{\text{com}} \quad \text{(\text{def. } \alpha_{\text{inv}} \text{ preserves } \cup)} \\
\Leftrightarrow \forall \xi \in S^a_{\text{ana}}[H_{\text{com}}]P : \bigcup_{p=1}^{n-1} \bigcup_{L \in P_p} \{\alpha_{\text{inv}}(\xi')_p(L) | \xi' \in \{\xi\}\} \subseteq S_{\text{com}} \quad \text{(\text{def. } (19) \text{ of } \alpha_{\text{inv}})} \\
\Leftrightarrow \forall (\tau_{\text{start}} \times \prod_{p=0}^{n-1} \tau_p \times \pi \times \text{rf}) \in S^a_{\text{ana}}[H_{\text{com}}]P : \forall p \in [1, n] : \forall L \in P_p . \\
\alpha_{\text{inv}}(\tau_{\text{start}} \times \prod_{p=0}^{n-1} \tau_p \times \pi \times \text{rf})_p(L) \subseteq S_{\text{com}}_p(L) \quad \text{(\text{def. } \in, \bigcup, \subseteq, \text{ and } S^a_{\text{ana}}[H_{\text{com}}]P \text{ so that } \xi \text{ has the form } \xi = \tau_{\text{start}} \times \prod_{p=0}^{n-1} \tau_p \times \pi \times \text{rf}. \text{ By def. (19) of } \alpha_{\text{inv}} \text{ and } \subseteq, \text{ we get})} \\
\Leftrightarrow \forall (\tau_{\text{start}} \times \prod_{p=0}^{n-1} \tau_p \times \pi \times \text{rf}) \in S^a_{\text{ana}}[H_{\text{com}}]P : \forall i \in [1, n] : \forall L \in P_p : \forall q \in [0, n] : \forall k_q < |\tau_q| . \\
(\tau_{L,k_q} = s(\kappa_{0,k_0}, \theta_{0,k_0}, \rho_{0,k_0}, \nu_0,k_0) \wedge \kappa_p,k_p = L) \Rightarrow \\
(\kappa_{p+1}, \kappa_{p+1}, \cdots, \kappa_{n-1}, \kappa_{n-1}, \theta_{p-1,k_{p-1}}, \theta_{p,k_p}, \rho_{p,k_p}, \nu_{p,k_p}, \text{rf}) \in S_{\text{com}}_i(L) \\
\]
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

\[ S_{com} \triangleq \neg \exists i, j. \left[ rf(F2^i_4, 0, false) \lor rf(F2^i_4, 17, false) \right. \\
\left. \lor rf(T^i_5, 11, 1) \land [ rf(F1^j_3, 0, false) \lor rf(F1^j_3, 8, false) \lor rf(T^j_4, 2, 2) \right] \]
Incompleteness

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

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

- \( cat \) may not be expressive enough:
Proving Architectural Consistency

\[ M \implies H_{\text{com}} \]

cat \hspace{1cm} cat
$M \Rightarrow H_{com}$ in \text{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 ($S_{com}$ for this execution + $S_{ind}$)

- 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.
• All writes eventually hit the memory:
  • If, at a cut of the execution, all the processes infinitely often write the same value $\upsilon$ to a shared variable $x$ and only that value $\upsilon$
  • 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 $\upsilon$

(*) 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 program-specific consistency model $S_{com}$
- Find an anarchic inductive invariant $S_{ind}$ satisfying the verification conditions such that $(S_{com} \land S_{ind}) \Rightarrow S_{inv}$
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