Static Analysis of Concurrent Programs

MPRI 2–6: Abstract Interpretation, application to verification and static analysis

Antoine Miné

CNRS, École normale supérieure

course 11, 2012–2013

Concurrent programming

Idea:

Decompose a program into a set of (loosely) interacting processes.

Why concurrent programs?

- exploit parallelism in current computers
  (multi-processors, multi-cores, hyper-threading)

  "Free lunch is over"
  change in Moore's law ($\times 2$ transistors every 2 years)
- exploit several computers
  (distributed computing)
- ease of programming
  (GUI, network code, reactive programs)

Models of concurrent programs

Many models:

- process calculi: CSP, $\pi-$calculus, join calculus
- message passing
- shared memory (threads)
- transactional memory
- combination of several models

Example implementations:

- C, C++, etc. with a thread library (POSIX threads, Win32)
- C, C++, etc. with a message library (MPI, OpenMP)
- Java (native threading API)
- Erlang (based on $\pi-$calculus)
- JoCaml (OCaml + join calculus)
- processor-level (interrupts, test-and-set instructions)

Scope

In this talk: thread model

- implicit communication through shared memory
- explicit communication through synchronisation primitives
- a fixed number of threads (no dynamic creation of threads)
- numeric programs (real-valued variables)

Goal: static analysis

- to infer numeric program invariants
- to discover possible run-time errors (e.g., division by 0))
- parametrized by a choice of abstract domains

Outline

- State-based analyses
  - Sequential programs (reminders)
  - Concurrent programs
- Toward thread-modular analyses
  - Detour through proof methods (Floyd–Hoare, Owicki–Gries, Jones)
  - Rely-guarantee in abstract interpretation form
- Interference-based abstract analyses
  - A denotational-style analysis
  - Weakly consistent memory models
  - Synchronisation

Simple structured numeric language

- finite set of (toplevel) threads: $stat_1$ to $stat_n$
- finite set of numeric program variables $X \in \mathbb{V}$
- finite set of statement locations $\ell \in \mathcal{L}$
- finite set of potential error locations $\omega \in \Omega$

Language syntax

$$
\begin{array}{lcll}
prog & ::= & {}^{\ell}stat_1{}^{\ell} \;||\; \ldots \;||\; {}^{\ell}stat_n{}^{\ell} & \textit{(parallel composition)} \\
{}^{\ell}stat{}^{\ell} & ::= & {}^{\ell}X \leftarrow expr{}^{\ell} & \textit{(assignment)} \\
 & | & {}^{\ell}\textbf{if } expr \bowtie 0 \textbf{ then } {}^{\ell}stat{}^{\ell} & \textit{(conditional)} \\
 & | & {}^{\ell}\textbf{while } {}^{\ell}expr \bowtie 0 \textbf{ do } {}^{\ell}stat{}^{\ell} & \textit{(loop)} \\
 & | & {}^{\ell}stat; {}^{\ell}stat{}^{\ell} & \textit{(sequence)} \\
expr & ::= & X \mid [c_1, c_2] \mid -expr \mid expr \diamond_{\omega} expr &
\end{array}
$$

$c_1, c_2 \in \mathbb{R} \cup \{+\infty, -\infty\}$, $\diamond \in \{+, -, \times, /\}$, $\bowtie \in \{=, >, \geq, <, \leq\}$

State-based analyses

Sequential program semantics (reminders)

Transition systems (reminder)

Transition system: (Σ, τ, I)

- Σ: a set of program states
- $\tau \subseteq \Sigma \times \Sigma$: a transition relation
  we note $(\sigma, \sigma') \in \tau$ as $\sigma \rightarrow_\tau \sigma'$
- $I \subseteq \Sigma$: a set of initial states

Traces: sequences of states $\langle \sigma_0, \ldots, \sigma_n, \ldots \rangle$

- Σ^*: finite traces
- Σ^ω: infinite countable traces
- $\Sigma^\infty \stackrel{\text{def}}{=} \Sigma^* \cup \Sigma^\omega$: finite or infinite countable traces
- $\mathit{pref}(t)$ set of prefixes of t (including t)

We view program semantics and properties as sets of traces

Traces of a transition system (reminder)

Maximal trace semantics: $M \subseteq \mathcal{P}(\Sigma^\infty)$

- set of total executions $\langle \sigma_0, \ldots, \sigma_n, \ldots \rangle$
  - starting in an initial state $\sigma_0 \in I$ and either
  - ending in a blocking state in $B \stackrel{\text{def}}{=} \{ \sigma \,|\, \forall \sigma' : \sigma \not\to_\tau \sigma' \}$
  - or infinite

$$M \stackrel{\text{def}}{=} \{ \langle \sigma_0, \ldots, \sigma_n \rangle \,|\, \sigma_0 \in I \wedge \sigma_n \in B \wedge \forall i < n : \sigma_i \to_\tau \sigma_{i+1} \} \cup \{ \langle \sigma_0, \ldots, \sigma_n \ldots \rangle \,|\, \sigma_0 \in I \wedge \forall i : \sigma_i \to_\tau \sigma_{i+1} \}$$

- note: traces in M have not strict prefix in M
  $t, u \in M \wedge t \in \mathit{pref}(u) \implies t = u$
- able to express many properties of programs, e.g.:
  - safety: $M \subseteq S^\infty$ (executions stay in S)
  - ordering: $M \subseteq S_1^\infty S_2^\infty$ (S_2 can only occur after S_1)
  - termination: $M \subseteq \Sigma^*$ (executions are finite)
  - inevitability: $M \subseteq \Sigma^* S \Sigma^\infty$ (executions pass through S)

Traces of a transition system (reminder)

Finite prefix trace semantics: $T \subseteq \mathcal{P}(\Sigma^*)$

- set of finite prefixes of executions:
  $T \stackrel{\text{def}}{=} \{ \langle \sigma_0, \ldots, \sigma_n \rangle \mid \sigma_0 \in I, \forall i < n : \sigma_i \rightarrow_\tau \sigma_{i+1} \}$
- T is an abstraction of the maximal trace semantics
  $T = \alpha_p(M)$ where $\alpha_p(X) \stackrel{\text{def}}{=} \{ t \in \Sigma^* \mid \exists u \in X : t \in \mathit{pref}(u) \}$
- T can prove safety properties: $T \subseteq S^*$ (executions stay in S)

  T can prove ordering: $T \subseteq S_1^* S_2^*$
  (if S_1 and S_2 occur, S_2 occurs after S_1)

  T cannot prove termination nor inevitability
  $\alpha_p(\{ a^n b \mid n \geq 0 \}) =$ (terminates)
  $\alpha_p(\{ a^n b \mid n \geq 0 \} \cup \{ a^\omega \}) =$ (may not terminate)
  $\{ a^n, a^n b \mid n \geq 0 \}$
- fixpoint characterisation: $T = \mathrm{lfp}\, F$ where

$$F(X) = I \cup \{ \langle \sigma_0, \ldots, \sigma_{n+1} \rangle \mid \langle \sigma_0, \ldots, \sigma_n \rangle \in X \wedge \sigma_n \rightarrow_\tau \sigma_{n+1} \}$$

From sequential programs to transition systems

Simple sequential numeric program: $prog = {}^{\ell i} stat^{\ell x}$.

Program states: $\Sigma \stackrel{\text{def}}{=} (\mathcal{L} \times \mathcal{E}) \cup \Omega$

- a control state in $\mathcal{L}$
- a memory state: an environment in $\mathcal{E} \stackrel{\text{def}}{=} \mathbb{V} \to \mathbb{R}$
- an error state in Ω

Initial states:

start at the first control point ℓi, and with variables set to 0:

$I \stackrel{\text{def}}{=} \{ (\ell i, \lambda V.0) \}$

Note that $\mathcal{P}(\Sigma) \simeq (\mathcal{L} \to \mathcal{P}(\mathcal{E})) \times \mathcal{P}(\Omega)$:

- a state property in $\mathcal{P}(\mathcal{E})$ at each program point in $\mathcal{L}$,
- and a set of errors in $\mathcal{P}(\Omega)$.

From sequential programs to transition systems (cont.)

Expression semantics: $\mathrm{E}[\![expr]\!] : \mathcal{E} \to (\mathcal{P}(\mathbb{R}) \times \mathcal{P}(\Omega))$

$$
\begin{array}{lcl}
\mathrm{E}[\![X]\!]\,\rho & \stackrel{\text{def}}{=} & \langle \{ \rho(X) \}, \emptyset \rangle \\
\mathrm{E}[\![[c_1, c_2]]\!]\,\rho & \stackrel{\text{def}}{=} & \langle \{ x \in \mathbb{R} \mid c_1 \leq x \leq c_2 \}, \emptyset \rangle \\
\mathrm{E}[\![-e_1]\!]\,\rho & \stackrel{\text{def}}{=} & \text{let } \langle V_1, O_1 \rangle = \mathrm{E}[\![e_1]\!]\,\rho \text{ in} \\
 & & \langle \{ -v_1 \mid v_1 \in V_1 \}, O_1 \rangle \\
\mathrm{E}[\![e_1 \diamond_\omega e_2]\!]\,\rho & \stackrel{\text{def}}{=} & \text{let } \forall i \in \{1,2\}: \langle V_i, O_i \rangle = \mathrm{E}[\![e_i]\!]\,\rho \text{ in} \\
 & & \langle \{ v_1 \diamond v_2 \mid v_i \in V_i, \diamond \neq / \vee v_2 \neq 0 \}, \\
 & & O_1 \cup O_2 \cup \{ \omega \text{ if } \diamond = / \wedge 0 \in V_2 \} \rangle
\end{array}
$$

- defined by structural induction on the syntax
- evaluates in an environment ρ to a set of values
- also returns a set of accumulated errors (divisions by zero)

From sequential programs to transition systems (cont.)

Transitions generated by statements: $\tau[{}^{\ell}stat^{\ell'}] \subseteq \Sigma \times \Sigma$

$$\tau[{}^{\ell 1}X \leftarrow e^{\ell 2}] \stackrel{\text{def}}{=} \{ (\ell 1, \rho) \to (\ell 2, \rho[X \mapsto v]) \mid \rho \in \mathcal{E},\ v \in \mathit{fst}(\mathbb{E}[\![e]\!] \rho) \} \cup \{ (\ell 1, \rho) \to \omega \mid \rho \in \mathcal{E},\ \omega \in \mathit{snd}(\mathbb{E}[\![e]\!] \rho) \}$$

$$\tau[{}^{\ell 1}\textbf{if } e \bowtie 0 \textbf{ then } {}^{\ell 2}s^{\ell 3}] \stackrel{\text{def}}{=} \{ (\ell 1, \rho) \to (\ell 2, \rho) \mid \rho \in \mathcal{E},\ \exists v \in \mathit{fst}(\mathbb{E}[\![e]\!] \rho) \colon v \bowtie 0 \} \cup \{ (\ell 1, \rho) \to (\ell 3, \rho) \mid \rho \in \mathcal{E},\ \exists v \in \mathit{fst}(\mathbb{E}[\![e]\!] \rho) \colon v \not\bowtie 0 \} \cup \tau[{}^{\ell 2}s^{\ell 3}] \cup \{ (\ell 1, \rho) \to \omega \mid \rho \in \mathcal{E},\ \omega \in \mathit{snd}(\mathbb{E}[\![e]\!] \rho) \}$$

$$\tau[{}^{\ell 1}\textbf{while } {}^{\ell 2}e \bowtie 0 \textbf{ do } {}^{\ell 3}s^{\ell 4}] \stackrel{\text{def}}{=} \{ (\ell 1, \rho) \to (\ell 2, \rho) \mid \rho \in \mathcal{E} \} \cup \{ (\ell 2, \rho) \to (\ell 3, \rho) \mid \rho \in \mathcal{E},\ \exists v \in \mathit{fst}(\mathbb{E}[\![e]\!] \rho) \colon v \bowtie 0 \} \cup \{ (\ell 2, \rho) \to (\ell 4, \rho) \mid \rho \in \mathcal{E},\ \exists v \in \mathit{fst}(\mathbb{E}[\![e]\!] \rho) \colon v \not\bowtie 0 \} \cup \tau[{}^{\ell 3}s^{\ell 2}] \cup \{ (\ell 2, \rho) \to \omega \mid \rho \in \mathcal{E},\ \omega \in \mathit{snd}(\mathbb{E}[\![e]\!] \rho) \}$$

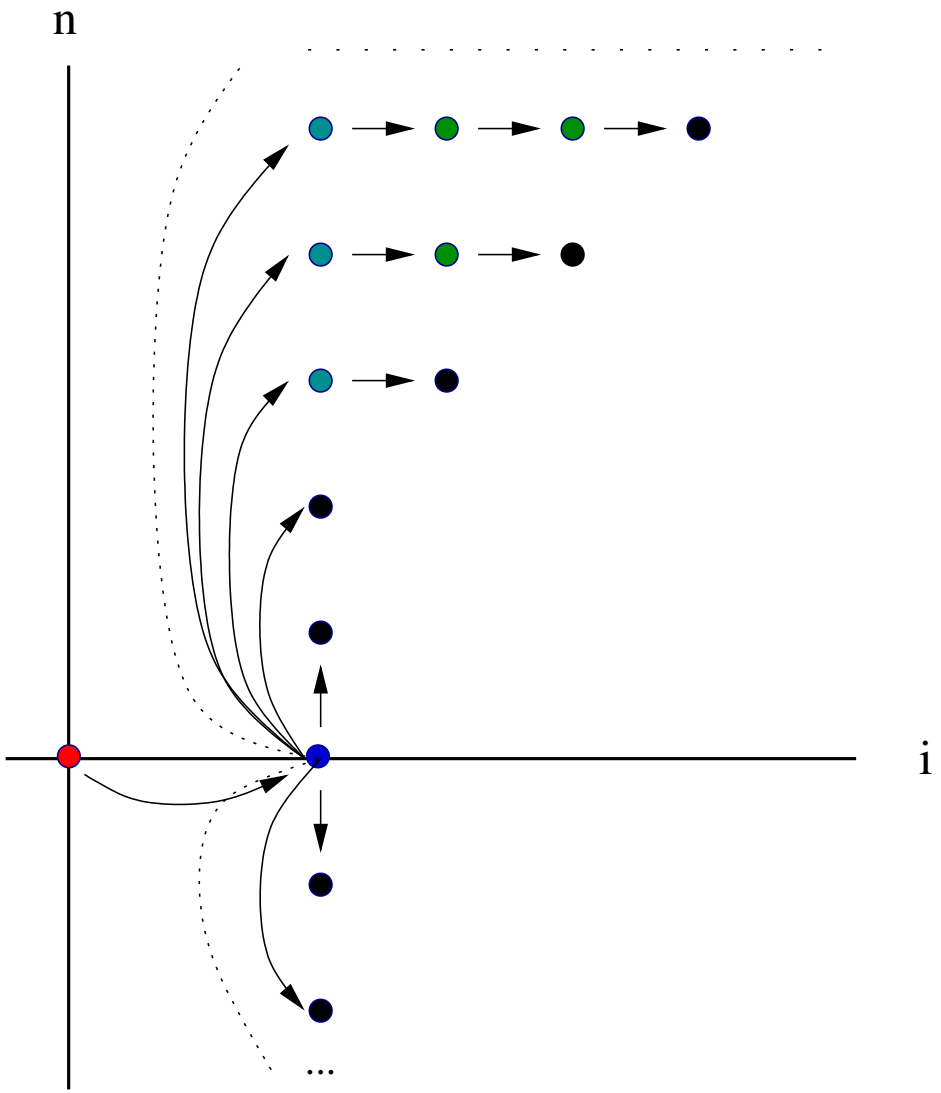
$$\tau[{}^{\ell 1}s_1;{}^{\ell 2}s_2{}^{\ell 3}] \stackrel{\text{def}}{=} \tau[{}^{\ell 1}s_1{}^{\ell 2}] \cup \tau[{}^{\ell 2}s_2{}^{\ell 3}]$$

Again, defined by structural induction on the syntax.

From sequential programs to transition systems (cont.)

Example

- $i \leftarrow 2;$
- $n \leftarrow [-\infty, +\infty];$
- **while** $i < n$ **do**
  if $[0, 1] = 0$ **then**
  $i \leftarrow i + 1$
-



State abstraction (reminder)

Reachable state semantics: $R \subseteq \mathcal{P}(\Sigma)$

- set of states reachable in any execution:
  $R \stackrel{\text{def}}{=} \{ \sigma \mid \exists \langle \sigma_0, \ldots, \sigma_n \rangle : \sigma_0 \in I, \forall i < n : \sigma_i \rightarrow_\tau \sigma_{i+1} \wedge \sigma = \sigma_n \}$
- R is an abstraction of the finite trace semantics: $R = \alpha_s(T)$
  where $\alpha_s(X) \stackrel{\text{def}}{=} \{ \sigma \mid \exists \langle \sigma_0, \ldots, \sigma_n \rangle \in X : \sigma = \sigma_n \}$
- R can prove safety properties: $R \subseteq S$ (executions stay in S)
  R cannot prove ordering, termination, inevitability
- fixpoint characterisation: $R = \text{lfp}\, G$ where
  $G(X) = I \cup \{ \sigma \mid \exists \sigma' \in X : \sigma' \rightarrow_\tau \sigma \}$

Equational form

Principle:

- see $\mathrm{lfp}\, f$ as the least solution of an equation $x = f(x)$
- partition states by control: $\mathcal{P}(\mathcal{L} \times \mathcal{E}) \simeq \mathcal{L} \to \mathcal{P}(\mathcal{E})$
  $\mathcal{X}_\ell \in \mathcal{P}(\mathcal{E})$: invariants at $\ell \in \mathcal{L}$
  $\forall \ell \in \mathcal{L}: \mathcal{X}_\ell \stackrel{\text{def}}{=} \{ m \in \mathcal{E} \mid (\ell, m) \in R \}$

$\Longrightarrow$ set of (recursive) equations on $\mathcal{X}_\ell$

Example:

$\ell 1$ $i \leftarrow 2$;
$\ell 2$ $n \leftarrow [-\infty, +\infty]$;
$\ell 3$ **while** $\ell 4$ $i < n$ **do**
  $\ell 5$ **if** $[0,1] = 0$ **then**
    $\ell 6$ $i \leftarrow i + 1$
  $\ell 7$
$\ell 8$

$$
\begin{aligned}
\mathcal{X}_1 &= I \\
\mathcal{X}_2 &= \mathrm{C}[\![i \leftarrow 2]\!]\, \mathcal{X}_1 \\
\mathcal{X}_3 &= \mathrm{C}[\![n \leftarrow [-\infty, +\infty]]\!]\, \mathcal{X}_2 \\
\mathcal{X}_4 &= \mathcal{X}_3 \cup \mathcal{X}_7 \\
\mathcal{X}_5 &= \mathrm{C}[\![i < n]\!]\, \mathcal{X}_4 \\
\mathcal{X}_6 &= \mathcal{X}_5 \\
\mathcal{X}_7 &= \mathcal{X}_5 \cup \mathrm{C}[\![i \leftarrow i + 1]\!]\, \mathcal{X}_6 \\
\mathcal{X}_8 &= \mathrm{C}[\![i \geq n]\!]\, \mathcal{X}_4
\end{aligned}
$$

Equational form (cont.)

We derive the equation system $eq({}^{\ell}stat^{\ell'})$
from the program syntax ${}^{\ell}stat^{\ell'}$ by induction:

$$eq({}^{\ell 1}X \leftarrow e^{\ell 2}) \stackrel{\text{def}}{=} \{\, \mathcal{X}_{\ell 2} = \mathsf{C}[\![X \leftarrow e]\!]\, \mathcal{X}_{\ell 1} \,\}$$

$$eq({}^{\ell 1}\textbf{if } e \bowtie 0 \textbf{ then } {}^{\ell 2}s^{\ell 3}) \stackrel{\text{def}}{=} \{\, \mathcal{X}_{\ell 2} = \mathsf{C}[\![e \bowtie 0]\!]\, \mathcal{X}_{\ell 1},\ \mathcal{X}_{\ell 3} = \mathcal{X}_{\ell 3'} \cup \mathsf{C}[\![e \not\bowtie 0]\!]\, \mathcal{X}_{\ell 1} \,\} \cup eq({}^{\ell 2}s^{\ell 3'})$$

$$eq({}^{\ell 1}\textbf{while } {}^{\ell 2}e \bowtie 0 \textbf{ do } {}^{\ell 3}s^{\ell 4}) \stackrel{\text{def}}{=} \{\, \mathcal{X}_{\ell 2} = \mathcal{X}_{\ell 1} \cup \mathcal{X}_{\ell 4'},\ \mathcal{X}_{\ell 3} = \mathsf{C}[\![e \bowtie 0]\!]\, \mathcal{X}_{\ell 2},\ \mathcal{X}_{\ell 4} = \mathsf{C}[\![e \not\bowtie 0]\!]\, \mathcal{X}_{\ell 2} \,\} \cup eq({}^{\ell 3}s^{\ell 4'})$$

$$eq({}^{\ell 1}s_1;{}^{\ell 2}s_2{}^{\ell 3}) \stackrel{\text{def}}{=} eq({}^{\ell 1}s_1{}^{\ell 2}) \cup ({}^{\ell 2}s_2{}^{\ell 3})$$

where:

- $\mathcal{X}^{\ell 3'}$, $\mathcal{X}^{\ell 4'}$ are fresh variables storing intermediate results
- $\mathsf{C}[\![X \leftarrow e]\!]\, \mathcal{X} \stackrel{\text{def}}{=} \{\, \rho[X \mapsto v] \mid \rho \in \mathcal{X},\ v \in \mathsf{E}[\![e]\!]\, \rho \,\}$
  $\mathsf{C}[\![e \bowtie 0]\!]\, \mathcal{X} \stackrel{\text{def}}{=} \{\, \rho \in \mathcal{X} \mid \exists v \in \mathsf{E}[\![\rho]\!]\, \rho \colon v \bowtie 0 \,\}$
  (for the sake of simplicity, we ignore error collecting here)

Abstract equation system

Given a numeric abstract domain:

- abstract elements $\mathcal{E}^\sharp$ abstracting $\mathcal{P}(\mathcal{E})$
  with concretization $\gamma : \mathcal{E}^\sharp \to \mathcal{P}(\mathcal{E})$
- sound abstract operators $\mathsf{C}^\sharp[\![X \leftarrow e]\!]$, $\mathsf{C}^\sharp[\![e \bowtie 0]\!]$, $\cup^\sharp$
  $f^\sharp$ is sound $\iff \forall X^\sharp \in \mathcal{E}^\sharp : f(\gamma(X^\sharp)) \subseteq \gamma(f^\sharp(X^\sharp))$
- a widening operator $\triangledown$

we can over-approximate in the abstract the solution of the system

Advantages:

- separate programming language from equation language
- various choice of solving strategies
  (chaotic iterations [Bour93])

Denotational form

Input-output function $\mathrm{C}[\![\mathit{stat}]\!]$.

$$\mathrm{C}[\![\mathit{stat}]\!] : (\mathcal{P}(\mathcal{E}) \times \mathcal{P}(\Omega)) \to (\mathcal{P}(\mathcal{E}) \times \mathcal{P}(\Omega))$$

$$\mathrm{C}[\![X \leftarrow e]\!] \langle R, O \rangle \stackrel{\text{def}}{=} \langle \emptyset, O \rangle \sqcup \bigsqcup_{\rho \in R} \langle \{ \rho[X \mapsto v] \mid v \in V_\rho \}, O_\rho \rangle$$

$$\mathrm{C}[\![e \bowtie 0?]\!] \langle R, O \rangle \stackrel{\text{def}}{=} \langle \emptyset, O \rangle \sqcup \bigsqcup_{\rho \in R} \langle \{ \rho \mid \exists v \in V_\rho : v \bowtie 0 \}, O_\rho \rangle$$

where $\langle V_\rho, O_\rho \rangle \stackrel{\text{def}}{=} \mathrm{E}[\![e]\!] \rho$

$$\mathrm{C}[\![\textbf{if } e \bowtie 0 \textbf{ then } s]\!] X \stackrel{\text{def}}{=} (\mathrm{C}[\![s]\!] \circ \mathrm{C}[\![e \bowtie 0?]\!]) X \sqcup \mathrm{C}[\![e \not\bowtie 0?]\!] X$$

$$\mathrm{C}[\![\textbf{while } e \bowtie 0 \textbf{ do } s]\!] X \stackrel{\text{def}}{=} \mathrm{C}[\![e \not\bowtie 0?]\!] (\mathrm{lfp} \lambda Y . X \sqcup (\mathrm{C}[\![s]\!] \circ \mathrm{C}[\![e \bowtie 0?]\!]) Y)$$

$$\mathrm{C}[\![s_1 ; s_2]\!] \stackrel{\text{def}}{=} \mathrm{C}[\![s_2]\!] \circ \mathrm{C}[\![s_1]\!]$$

- mutate memory states in $\mathcal{E}$, accumulate errors in Ω
  ($\sqcup$ is the element-wise $\cup$ in $\mathcal{P}(\mathcal{E}) \times \mathcal{P}(\Omega)$)
- structured: nested loops yield nested fixpoints
- big-step: forget information on intermediate locations ℓ

Abstract denotational analysis

Extend $\mathcal{E}^\sharp$ to $\mathcal{D}^\sharp \stackrel{\text{def}}{=} \mathcal{E}^\sharp \times \mathcal{P}(\Omega)$.

$\mathrm{C}^\sharp[\![\mathit{stat}]\!] : \mathcal{D}^\sharp \to \mathcal{D}^\sharp$

$\mathrm{C}^\sharp[\![X \leftarrow e]\!] \langle R^\sharp, O \rangle$ and $\mathrm{C}^\sharp[\![e \bowtie 0?]\!] \langle R^\sharp, O \rangle$ are given

$$\mathrm{C}^\sharp[\![\textbf{if } e \bowtie 0 \textbf{ then } s]\!] X^\sharp \stackrel{\text{def}}{=} (\mathrm{C}^\sharp[\![s]\!] \circ \mathrm{C}^\sharp[\![e \bowtie 0?]\!]) X^\sharp \sqcup^\sharp \mathrm{C}^\sharp[\![e \not\bowtie 0?]\!] X^\sharp$$

$$\mathrm{C}^\sharp[\![\textbf{while } e \bowtie 0 \textbf{ do } s]\!] X^\sharp \stackrel{\text{def}}{=} \mathrm{C}^\sharp[\![e \not\bowtie 0?]\!] (\mathrm{lim} \lambda Y^\sharp . Y^\sharp \triangledown (X^\sharp \sqcup (\mathrm{C}^\sharp[\![s]\!] \circ \mathrm{C}^\sharp[\![e \bowtie 0?]\!]) Y^\sharp))$$

$$\mathrm{C}^\sharp[\![s_1 ; s_2]\!] \stackrel{\text{def}}{=} \mathrm{C}^\sharp[\![s_2]\!] \circ \mathrm{C}^\sharp[\![s_1]\!]$$

- the abstract interpreter mimicks an actual interpreter
- efficient in memory (intermediate invariants are not kept)
- less flexibility on the iteration scheme

Concurrent program semantics

Labelled transition systems

Labelled transition system: $(\Sigma, \mathcal{A}, \tau, I)$

- Σ: a set of program states
- $\mathcal{A}$: a set of actions
- $\tau \subseteq \Sigma \times \mathcal{A} \times \Sigma$: a transition relation
  we note $(\sigma, a, \sigma') \in \tau$ as $\sigma \xrightarrow{a}_{\tau} \sigma'$
- $I \subseteq \Sigma$: a set of initial states

Traces: sequences of states interspersed with actions,
denoted as $\sigma_0 \xrightarrow{a_0} \sigma_1 \xrightarrow{a_1} \cdots \sigma_n \xrightarrow{a_n} \sigma_{n+1}$

From concurrent programs to labelled transition systems

Notations:

- concurrent program: $prog ::= {}^{\ell^i_1}stat_1{}^{\ell^x_1} \;||\; \cdots \;||\; {}^{\ell^i_n}stat_n{}^{\ell^x_n}$
- thread are identified by number in $\mathcal{T} \stackrel{\text{def}}{=} \{1, \ldots, n\}$

Program states: $\Sigma \stackrel{\text{def}}{=} ((\mathcal{T} \to \mathcal{L}) \times \mathcal{E}) \cup \Omega$

- a control state $\bar{\ell}(t) \in \mathcal{L}$ for each thread $t \in \mathcal{T}$
- a single shared memory state in $\rho \in \mathcal{E}$
- or an error state in $\omega \in \Omega$

Initial states:

threads start at their first control point ℓ^i_t, variables are set to 0:
$I \stackrel{\text{def}}{=} \{\lambda t.\ell^i_t, \lambda V.0\}$

Actions: thread identifiers $\mathcal{A} \stackrel{\text{def}}{=} \mathcal{T}$

From concurrent programs to labelled transition systems

Transition relation: $\tau \subseteq \Sigma \times \mathcal{A} \times \Sigma$

$$(\overline{\ell}, \rho) \xrightarrow{t}_{\tau} (\overline{\ell}', \rho') \iff_{\text{def}} (\overline{\ell}(t), \rho) \rightarrow_{\tau[stat_t]} (\overline{\ell}'(t), \rho') \wedge \forall u \neq t : \overline{\ell}(u) = \overline{\ell}'(u)$$

- based on the transition relation of individual threads $\tau[stat_t]$ seen as sequential processes $stat_t$
  - choose a thread t to run
  - update ρ and $\overline{\ell}(t)$
  - leave $\overline{\ell}(u)$ intact for $u \neq t$
- each $\sigma \rightarrow \sigma'$ in $\tau[stat_t]$ leads to many transitions in τ!

Interleaved trace semantics

Maximal and finite prefix trace semantics as before:

blocking states: $B \stackrel{\text{def}}{=} \{ \sigma \mid \forall \sigma', t: \sigma \overset{t}{\not\rightarrow}_\tau \sigma' \}$

Maximal traces: M (finite or infinite)

$$M \stackrel{\text{def}}{=} \{ \sigma_0 \xrightarrow{t_0} \cdots \xrightarrow{t_{n-1}} \sigma_n \mid \sigma_0 \in I \wedge \sigma_n \in B \wedge \forall i < n: \sigma_i \xrightarrow{t_i}_\tau \sigma_{i+1} \} \cup \{ \sigma_0 \xrightarrow{t_0} \sigma_1 \ldots \mid \sigma_0 \in I \wedge \forall i: \sigma_i \xrightarrow{t_i}_\tau \sigma_{i+1} \}$$

Finite prefix traces: T

$$T \stackrel{\text{def}}{=} \{ \sigma_0 \xrightarrow{t_0} \cdots \xrightarrow{t_{n-1}} \sigma_n \mid \sigma_0 \in I \wedge \forall i < n: \sigma_i \xrightarrow{t_i}_\tau \sigma_{i+1} \}$$

fixpoint form: $T = \mathrm{lfp}\, F$ where

$$F(X) = I \cup \{ \sigma_0 \xrightarrow{t_0} \cdots \xrightarrow{t_n} \sigma_{n+1} \mid \sigma_0 \xrightarrow{t_0} \cdots \xrightarrow{t_{n-1}} \sigma_n \in X \wedge \sigma_n \xrightarrow{t_n}_\tau \sigma_{n+1} \}$$

abstraction: $T = \alpha_p(M)$

Fairness

Fairness conditions: avoid threads being denied to run

Given $\mathit{enabled}(\sigma, t) \overset{\text{def}}{\iff} \exists \sigma' \in \Sigma : \sigma \xrightarrow{t}_\tau \sigma'$,

an infinite trace $\sigma_0 \xrightarrow{t_0} \cdots \sigma_n \xrightarrow{t_n} \cdots$ is:

- weakly fair if $\forall t \in \mathcal{T}$:
  $(\exists i : \forall j \geq i : \mathit{enabled}(\sigma_j, t)) \implies (\forall i : \exists j \geq i : a_j = t)$
  (no thread can be continuously enabled without running)
- strongly fair if $\forall t \in \mathcal{T}$:
  $(\forall i : \exists j \geq i : \mathit{enabled}(\sigma_j, t)) \implies (\forall i : \exists j \geq i : a_j = t)$
  (no thread can be infinitely often enabled without running)

Proofs under fairness conditions given:

- the maximal traces M of a program
- a property X to prove (as a set of traces)
- the set F of all (weakly or strongly) fair and of finite traces

$\implies$ prove $M \cap F \subseteq X$ instead of $M \subseteq X$

Fairness (cont.)

Example: **while** $x \geq 0$ **do** $x \leftarrow x + 1 \;||\; x \leftarrow -1$

- may not terminate without fairness
- always terminates under weak and strong fairness

Finite prefix traces

$M \cap F \subseteq X$ reduces to $\alpha_p(M \cap F) \subseteq \alpha_p(X)$

for all fairness conditions F, $\alpha_p(M \cap F) = \alpha_p(M) = T$

$\Longrightarrow$ fairness-dependent properties cannot be proved with finite prefixes

In the following, we ignore fairness conditions.
(see [Cous85])

Equational state semantics

State abstraction R: as before

- $R \stackrel{\text{def}}{=} \{\, \sigma \mid \exists \sigma_0 \xrightarrow{t_0} \cdots \sigma_n : \sigma_0 \in I \ \forall i < n : \sigma_i \xrightarrow{t_i}_\tau \sigma_{i+1} \wedge \sigma = \sigma_n \,\}$
- $R = \alpha_s(T)$ where $\alpha_s(X) \stackrel{\text{def}}{=} \{\, \sigma \mid \exists \sigma_0 \xrightarrow{t_0} \cdots \sigma_n \in X : \sigma = \sigma_n \,\}$
- $R = \text{lfp}\, G$ where $G(X) = I \cup \{\, \sigma \mid \exists \sigma' \in X, t \in \mathcal{T} : \sigma' \xrightarrow{t}_\tau \sigma \,\}$

Equational form: (without error handling)

- for each $\bar{\ell} \in \mathcal{T} \to \mathcal{L}$, a variable $\mathcal{X}_{\bar{\ell}}$ with value in $\mathcal{E}$
- equations are derived from thread equations $eq(stat_t)$ as:
  $\mathcal{X}_{\bar{\ell}_1} = \bigcup_{t \in \mathcal{T}} \{\, \mathcal{F}(\mathcal{X}_{\bar{\ell}_2}, \ldots, X_{\bar{\ell}_N}) \mid$
  $\exists \mathcal{X}_{\ell_1} = \mathcal{F}(\mathcal{X}_{\ell_2}, \ldots, \mathcal{X}_{\ell_N}) \in eq(stat_t) :$
  $\forall i \leq N : \bar{\ell}_i(t) = \ell_i,\ \forall u \neq t : \bar{\ell}_i(u) = \bar{\ell}_1(u) \,\}$
  (join with $\cup$ equations updating a single thread)

Equational state semantics (example)

Example: inferring $0 \leq x \leq y \leq 10$

t_1	t_2
while $^{\ell 1}\ 0 = 0$ **do**	**while** $^{\ell 4}\ 0 = 0$ **do**
$\quad ^{\ell 2}$ **if** $x < y$ **then**	$\quad ^{\ell 5}$ **if** $y < 10$ **then**
$\quad\quad ^{\ell 3}\ x \leftarrow x + 1$	$\quad\quad ^{\ell 6}\ y \leftarrow y + 1$

Equational state semantics (example)

Example: inferring $0 \leq x \leq y \leq 10$

t_1	t_2
while $^{\ell 1}\ 0 = 0$ **do**	**while** $^{\ell 4}\ 0 = 0$ **do**
$\quad ^{\ell 2}$ **if** $x < y$ **then**	$\quad ^{\ell 5}$ **if** $y < 10$ **then**
$\quad\quad ^{\ell 3}\ x \leftarrow x + 1$	$\quad\quad ^{\ell 6}\ y \leftarrow y + 1$

(Simplified) equation system:

$$
\begin{aligned}
\mathcal{X}_{1,4} &= I \cup \mathrm{C}[\![x \leftarrow x+1]\!]\, \mathcal{X}_{3,4} \cup \mathrm{C}[\![x \geq y]\!]\, \mathcal{X}_{2,4} \\
&\quad \cup \mathrm{C}[\![y \leftarrow y+1]\!]\, \mathcal{X}_{1,6} \cup \mathrm{C}[\![y \geq 10]\!]\, \mathcal{X}_{1,5} \\
\mathcal{X}_{2,4} &= \mathcal{X}_{1,4} \cup \mathrm{C}[\![y \leftarrow y+1]\!]\, \mathcal{X}_{2,6} \cup \mathrm{C}[\![y \geq 10]\!]\, \mathcal{X}_{2,5} \\
\mathcal{X}_{3,4} &= \mathrm{C}[\![x < y]\!]\, \mathcal{X}_{2,4} \cup \mathrm{C}[\![y \leftarrow y+1]\!]\, \mathcal{X}_{3,6} \cup \mathrm{C}[\![y \geq 10]\!]\, \mathcal{X}_{3,5} \\
\mathcal{X}_{1,5} &= \mathrm{C}[\![x \leftarrow x+1]\!]\, \mathcal{X}_{3,5} \cup \mathrm{C}[\![x \geq y]\!]\, \mathcal{X}_{2,5} \cup \mathcal{X}_{1,4} \\
\mathcal{X}_{2,5} &= \mathcal{X}_{1,5} \cup \mathcal{X}_{2,4} \\
\mathcal{X}_{3,5} &= \mathrm{C}[\![x < y]\!]\, \mathcal{X}_{2,5} \cup \mathcal{X}_{3,4} \\
\mathcal{X}_{1,6} &= \mathrm{C}[\![x \leftarrow x+1]\!]\, \mathcal{X}_{3,6} \cup \mathrm{C}[\![x \geq y]\!]\, \mathcal{X}_{2,6} \cup \mathrm{C}[\![y < 10]\!]\, \mathcal{X}_{1,5} \\
\mathcal{X}_{2,6} &= \mathcal{X}_{1,6} \cup \mathrm{C}[\![y < 10]\!]\, \mathcal{X}_{2,5} \\
\mathcal{X}_{3,6} &= \mathrm{C}[\![x < y]\!]\, \mathcal{X}_{2,6} \cup \mathrm{C}[\![y < 10]\!]\, \mathcal{X}_{3,5}
\end{aligned}
$$

Equational state semantics (example)

Example: inferring $0 \leq x \leq y \leq 10$

t_1	t_2
while $^{\ell 1}\ 0 = 0$ **do**	**while** $^{\ell 4}\ 0 = 0$ **do**
$\quad ^{\ell 2}$ **if** $x < y$ **then**	$\quad ^{\ell 5}$ **if** $y < 10$ **then**
$\quad\quad ^{\ell 3}\ x \leftarrow x + 1$	$\quad\quad ^{\ell 6}\ y \leftarrow y + 1$

pros

- easy to construct
- easy to further abstract in an abstract domain $\mathcal{E}^\sharp$

cons

- explosion of the number of variables and equations
- explosion of the size of equations
  $\Longrightarrow$ efficiency issues
- the equation system does *not* reflect the program structure
  (not defined by induction on the concurrent program)

Wish-list

We would like to:

- keep information attached to syntactic program locations
  (control points in $\mathcal{L}$, not control point tuples in $\mathcal{T} \to \mathcal{L}$)
- be able to abstract away control information
  (precision/cost trade-off control)
- avoid duplicating thread instructions
- have a computation structure based on the program syntax
  (denotational style)

Ideally:

thread-modular denotational-style semantics
(analyze each thread independently by induction on its syntax)

Detour through proof methods

Floyd–Hoare logic

Logic to prove properties about sequential programs [Hoar69].

Hoare triples: $\{P\}\, stat\, \{Q\}$

- annotate programs with logic assertions $\{P\}\, stat\, \{Q\}$
  (if P holds before $stat$, then Q holds after $stat$)
- check that $\{P\} stat \{Q\}$ is derivable with the following rules
  (the assertions are program invariants)

$$\frac{}{\{P[e/x]\}\, X \leftarrow e\, \{P\}} \qquad \frac{\{P \wedge e \bowtie 0\}\, s\, \{Q\} \quad P \wedge e \not\bowtie 0 \implies Q}{\{P\}\, \textbf{if } e \bowtie 0 \textbf{ then } s\, \{Q\}}$$

$$\frac{\{P\}\, s_1\, \{Q\} \quad \{Q\}\, s_2\, \{R\}}{\{P\}\, s_1; s_2\, \{R\}} \qquad \frac{\{P \wedge e \bowtie 0\}\, s\, \{P\}}{\{P\}\, \textbf{while } e \bowtie 0 \textbf{ do } s\, \{P \wedge e \not\bowtie 0\}}$$

$$\frac{\{P'\}\, s\, \{Q'\} \quad P \implies P' \quad Q' \implies Q}{\{P\}\, s\, \{Q\}}$$

Floyd–Hoare logic as abstract interpretation

Link with the equational state semantics:

Correspondence between ${}^{\ell}stat^{\ell'}$ and $\{P\}\, stat\, \{Q\}$:

- if P (resp. Q) models exactly the points in $\mathcal{X}_\ell$ (resp. $\mathcal{X}_{\ell'}$) then $\{P\}\, stat\, \{Q\}$ is a derivable Hoare triple
- if $\{P\}\, stat\, \{Q\}$ is derivable, then $\mathcal{X}_\ell \models P$ and $\mathcal{X}_{\ell'} \models Q$
  (all the points in $\mathcal{X}_l$ (resp. $\mathcal{X}_{\ell'}$) satisfy P (resp. Q))

$\Longrightarrow \mathcal{X}_\ell$ provide the most precise Hoare assertions in a constructive form

- $\gamma(\mathcal{X}^\sharp)$ provide (less precise) Hoare assertions in a computable form

Link with the denotational semantics:

both $\mathrm{C}[\![\, stat \,]\!]$ and the proof tree for $\{P\}\, stat\, \{Q\}$ reflect the syntactic structure of *stat*
(compositional methods)

Owicki–Gries proof method

Extension of Floyd–Hoare to concurrent programs [Owic76].

Principle: add a new rule, for $||$

$$\frac{\{P_1\}\, s_1\, \{Q_1\} \quad \{P_2\}\, s_2\, \{Q_2\}}{\{P_1 \wedge P_2\}\, s_1 \,||\, s_2\, \{Q_1 \wedge Q_2\}}$$

Owicki–Gries proof method

Extension of Floyd–Hoare to concurrent programs [Owic76].

Principle: add a new rule, for $||$

$$\frac{\{P_1\}\, s_1\, \{Q_1\} \quad \{P_2\}\, s_2\, \{Q_2\}}{\{P_1 \wedge P_2\}\, s_1 \,||\, s_2\, \{Q_1 \wedge Q_2\}}$$

This rule is not always sound!

e.g., we have $\{X = 0, Y = 0\}\, X \leftarrow 1\, \{X = 1, Y = 0\}$
and $\{X = 0, Y = 0\}$ **if** $X = 0$ **then** $Y \leftarrow 1 \{X = 0, Y = 1\}$
but not $\{X = 0, Y = 0\}\, X \leftarrow 1 \,||$ **if** $X = 0$ **then** $Y \leftarrow 1\, \{\mathit{false}\}$

$\Longrightarrow$ we need a side-condition to the rule:
$\{P_1\}\, s_1\, \{Q_1\}$ and $\{P_2\}\, s_2\, \{Q_2\}$ must not interfere

Owicki–Gries proof method (cont.)

interference freedom

given proofs Δ_1 and Δ_2 of $\{P_1\}\, s_1\, \{Q_1\}$ and $\{P_2\}\, s_2\, \{Q_2\}$

Δ_1 does not interfere with Δ_2 if:

for any Φ appearing before a statement in Δ_1
for any $\{P_2'\}\, s_2'\, \{Q_2'\}$ appearing in Δ_2
$\{\Phi \wedge P_2'\}\, s_2'\, \{\Phi\}$ holds
and moreover $\{Q_1 \wedge P_2'\}\, s_2'\, \{Q_1\}$

i.e.: the assertions used to prove $\{P_1\}\, s_1\, \{Q_1\}$ are stable by s_2

e.g., $\{X = 0, Y \in [0,1]\}\, X \leftarrow 1\, \{X = 1, Y \in [0,1]\}$
$\{X \in [0,1], Y = 0\}$ **if** $X = 0$ **then** $Y \leftarrow 1 \{X \in [0,1], Y \in [0,1]\}$
$\Longrightarrow$ $\{X = 0, Y = 0\}\, X \leftarrow 1 \,||$ **if** $X = 0$ **then** $Y \leftarrow 1\, \{X = 1, Y \in [0,1]\}$

Summary:

- pros: the invariants are local to threads
- cons: the proof is not compositional
  (proving one thread requires delving into the proof of other threads)

$\Longrightarrow$ not satisfactory

Jones' rely-guarantee proof method

Idea: explicit interferences with (more) annotations [Jone81].

Rely-guarantee "quintuples": $R, G \vdash \{P\}\, stat\, \{Q\}$

- if P is true before *stat* is executed
- and the effect of other threads is included in R (rely)
- then Q is true after *stat*
- and the effect of *stat* is included in G (guarantee)

where:

- P and Q are assertions on states (in $\mathcal{P}(\Sigma)$)
- R and G are assertions on transitions (in $\mathcal{P}(\Sigma \times \mathcal{A} \times \Sigma)$)

The parallel composition rule becomes:

$$\frac{R \cup G_2, G_1 \vdash \{P_1\}\, s_1\, \{Q_1\} \quad R \cup G_1, G_2 \vdash \{P_2\}\, s_2\, \{Q_2\}}{R, G_1 \cup G_2 \vdash \{P_1 \wedge P_2\}\, s_1 \,||\, s_2\, \{Q_1 \wedge Q_2\}}$$

Rely-guarantee example

Example: proving $0 \leq x \leq y \leq 10$

checking t_1

$\ell 1$ **while** $0 = 0$ **do**
  $\ell 2$ **if** $x < y$ **then**
    $\ell 3$ $x \leftarrow x + 1$

at $\ell 1, \ell 2 : 0 \leq x \leq y \leq 10$
at $\ell 3 : 0 \leq x < y \leq 10$

checking t_2

$\ell 4$ **while** $0 = 0$ **do**
  $\ell 5$ **if** $y < 10$ **then**
    $\ell 6$ $y \leftarrow y + 1$

at $\ell 4, \ell 5 : 0 \leq x \leq y \leq 10$
at $\ell 6 : 0 \leq x \leq y < 10$

Rely-guarantee example

Example: proving $0 \leq x \leq y \leq 10$

checking t_1

$\ell 1$ **while** $0 = 0$ **do**	x unchanged
$\ell 2$ **if** $x < y$ **then**	y incremented
$\ell 3$ $x \leftarrow x + 1$	$y \leq 10$

at $\ell 1, \ell 2 : 0 \leq x \leq y \leq 10$
at $\ell 3 : 0 \leq x < y \leq 10$

checking t_2

y unchanged	$\ell 4$ **while** $0 = 0$ **do**
	$\ell 5$ **if** $y < 10$ **then**
	$\ell 6$ $y \leftarrow y + 1$

at $\ell 4, \ell 5 : 0 \leq x \leq y \leq 10$
at $\ell 6 : 0 \leq x \leq y < 10$

In this example:

- guarantee exactly what is relied on ($R_1 = G_1$ and $R_2 = G_2$)
- rely and guarantee are global assertions

Benefits of rely-guarantee:

- invariants are still local to threads
- checking a thread does not require looking at other threads, only at an abstraction of their semantics

Auxiliary variables

Example

t_1	t_2
$\ell 1\ x \leftarrow x + 1\ \ell 2$	$\ell 3\ x \leftarrow x + 1\ \ell 4$

Goal: prove $\{x = 0\}\ t_1 \mid\mid t_2\ \{x = 2\}$.

Auxiliary variables

Example

t_1	t_2
$\ell 1\ x \leftarrow x+1\ \ell 2$	$\ell 3\ x \leftarrow x+1\ \ell 4$

Goal: prove $\{x = 0\}\ t_1 \mid\mid t_2\ \{x = 2\}$.

we must rely on and guarantee that
each thread increments x exactly once!

Solution: auxiliary variables

do not change the semantics but store extra information:

- past values of variables (history of the computation)
- program counter of other threads (pc_t)

Example: for t_1: $\{(pc_2 = \ell 3 \wedge x = 0) \vee (pc_2 = \ell 4 \wedge x = 1)\}$
$x \leftarrow x + 1$
$\{(pc_2 = \ell 3 \wedge x = 1) \vee (pc_2 = \ell 4 \wedge x = 2)\}$

Rely-guarantee as abstract interpretation

Local invariants

State projection: on a thread $t \in \mathcal{T}$

- add auxiliary variables $\mathbb{V}_t \stackrel{\text{def}}{=} \mathbb{V} \cup \{ pc_u \mid u \in \mathcal{T}, u \neq t \}$
- enriched environments for t: $\mathcal{E}_t \stackrel{\text{def}}{=} \mathbb{V}_t \to \mathbb{R}$
  (for simplicity, pc_u are numeric variables, i.e., $\mathcal{L} \subseteq \mathbb{R}$)
- local states: $\Sigma_t \stackrel{\text{def}}{=} (\mathcal{L} \times \mathcal{E}_t) \cup \Omega$
  (recall that $\Sigma \stackrel{\text{def}}{=} ((\mathcal{T} \to \mathcal{L}) \times \mathcal{E}) \cup \Omega$)
- projection: $\pi_t(\overline{\ell}, \rho) \stackrel{\text{def}}{=} (\overline{\ell}(t), \rho[\forall u \neq t : pc_u \mapsto \overline{\ell}(u)])$
  extended naturally to $\pi_t : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma_t)$

Local invariants on t: $R_t \stackrel{\text{def}}{=} \pi_t(R)$
(where R is the reachable state abstraction)

Note: π_t is a bijection, no information is lost

Interferences

Interference: caused by a thread $t \in \mathcal{T}$

$A_t \in \mathcal{P}(\Sigma \times \Sigma)$

$A_t \stackrel{\text{def}}{=} \alpha_t(T)$ where $\alpha_t(X) \stackrel{\text{def}}{=} \{ (\sigma, \sigma') \mid \exists \cdots \sigma \xrightarrow{t} \sigma' \cdots \in X \}$

subset of the transition system τ

- spawned by t and
- actually observed in some execution trace
  (recall that T is the prefix trace abstraction)

Nested fixpoint form

Nested fixpoints:

We note: $\overline{A} \stackrel{\text{def}}{=} \lambda t \in \mathcal{T}.A_t$, $\overline{R} \stackrel{\text{def}}{=} \lambda t \in \mathcal{T}.R_t$.

1. we express R_t as a function of $\overline{A}$ and thread $t \in \mathcal{T}$:

   $R_t = \text{lfp}\, G_t(\overline{A})$ where

   $$G_t : (\mathcal{T} \to \mathcal{P}(\Sigma \times \Sigma)) \to \mathcal{P}(\Sigma_t) \to \mathcal{P}(\Sigma_t)$$
   $$G_t(\overline{Y})(X) \stackrel{\text{def}}{=} \pi_t(I) \cup \{\, \pi_t(\sigma') \mid \exists \pi_t(\sigma) \in X : \sigma \stackrel{t}{\to}_\tau \sigma' \vee \exists u \neq t : (\sigma, \sigma') \in \overline{Y}(u) \,\}$$

   (a state is reachable if it is reachable by transitions from t and from the environment $\overline{A}$)

2. we express A_t as a function of $\overline{R}$ and thread $t \in \mathcal{T}$:

   $A_t = \overline{B}(\overline{R})(t)$ where

   $$\overline{B} : (\Pi t : \mathcal{T}.\mathcal{P}(\Sigma_t)) \to \mathcal{T} \to \mathcal{P}(\Sigma \times \Sigma)$$
   $$\overline{B}(\overline{Y})(t) \stackrel{\text{def}}{=} \{\, (\sigma, \sigma') \mid \pi_t(\sigma) \in \overline{Y}(t) \wedge \sigma \stackrel{t}{\to}_\tau \sigma' \,\}$$

   (collect transitions starting from reachable states)

3. ...

Nested fixpoint form (cont.)

1. $R_t = \text{lfp}\, G_t(\overline{A})$
2. $A_t = \overline{B}(\overline{R})(t)$
3. we deduce: $\overline{S} = \overline{\text{lfp}}\ \overline{H}$ where

$$\overline{H} : (\Pi t : \mathcal{T}.\mathcal{P}(\Sigma_t)) \to (\Pi t : \mathcal{T}.\mathcal{P}(\Sigma_t))$$
$$\overline{H}(\overline{X}) \stackrel{\text{def}}{=} \lambda t.\text{lfp}\, G_t(\overline{B}(\overline{X}))$$

($\Pi t : \mathcal{T}.\Sigma_t$ are functions from $t \in \mathcal{T}$ to Σ_t)
($\overline{\text{lfp}}$ is a fixpoint on vectors indexed by $\mathcal{T}$)

$\Longrightarrow$ nested fixpoints

Nested iterations:

By constructive versions of fixpoint theorems:

1. $\overline{S} = \overline{\text{lfp}}\ \overline{H} = \overline{\bigcup}_{n\in\mathbb{N}} \overline{H}^n(\emptyset)$
2. $\overline{H}(\overline{X})(t) = \text{lfp}\, G_t(\overline{B}(\overline{X})) = \bigcup_{n\in\mathbb{N}} (G_t(\overline{B}(\overline{X})))^n(\emptyset)$
   ($\simeq$ sequential semantics of each thread in isolation)

Abstract rely-guarantee

Suggested algorithm: nested iterations

once abstract domains for states and interferences are chosen

- start from $\overline{S}^\sharp_0 \stackrel{\text{def}}{=} \overline{A}^\sharp_0 \stackrel{\text{def}}{=} \bot^\sharp$
- while $\overline{A}^\sharp_n$ is not stable
  - compute $\overline{S}^\sharp_{n+1} \stackrel{\text{def}}{=} \lambda t.\text{lfp } G^\sharp_t(\overline{A}^\sharp_n)$ by iteration with widening $\triangledown$ ($\simeq$ separate analysis of each thread)
  - compute $\overline{A}^\sharp_{n+1} \stackrel{\text{def}}{=} \overline{A}^\sharp_n \triangledown \overline{B}^\sharp(\overline{S}^\sharp_{n+1})$
- when $\overline{A}^\sharp_n = \overline{A}^\sharp_{n+1}$, return $\overline{S}^\sharp_n$

$\Longrightarrow$ thread-modular analysis
parameterized by abstract domains
able to easily reuse existing sequential analyses

Abstracting states and interference

Flow-insensitive abstractions:

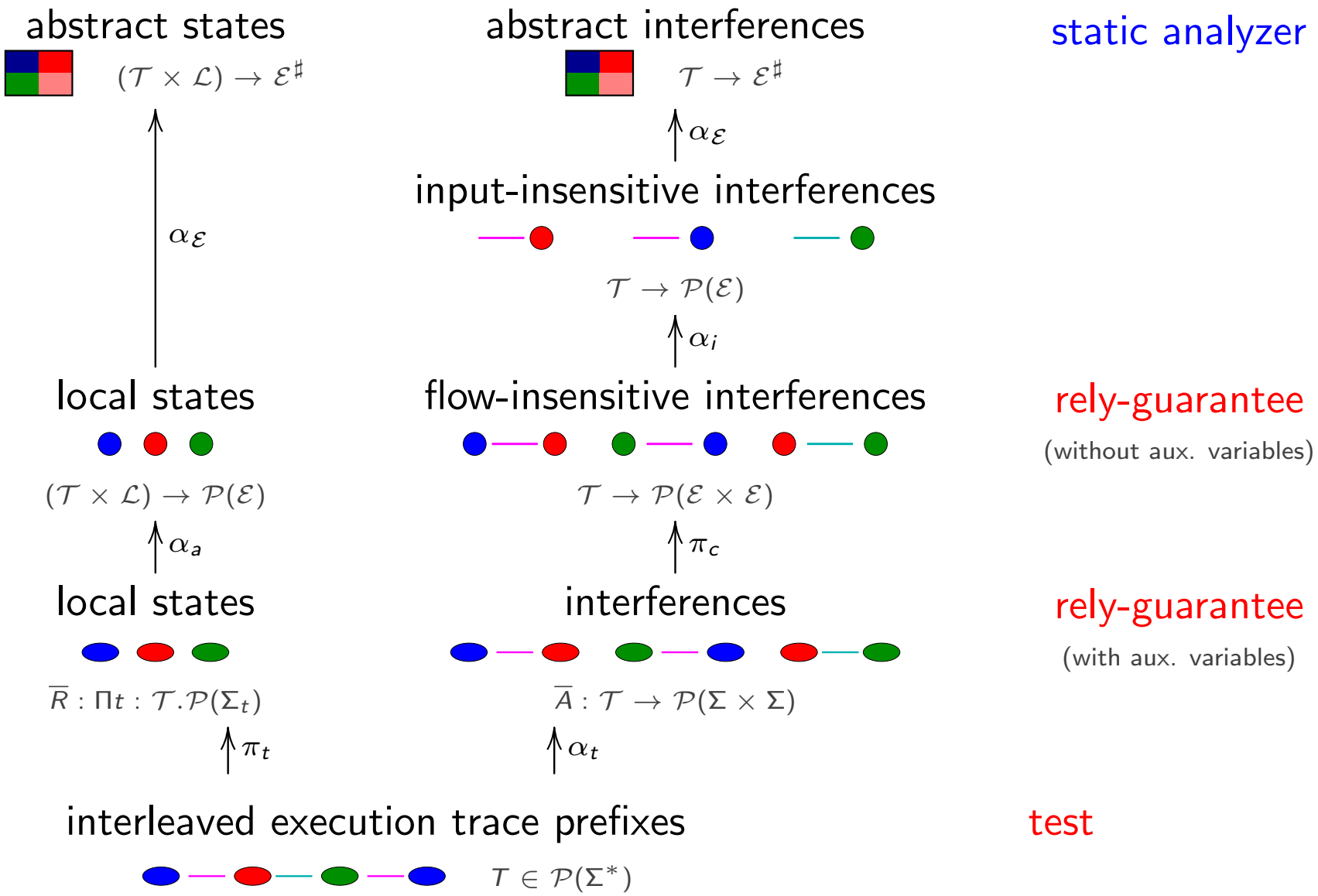
- on states: forget auxiliary variables
  $\alpha_a : \mathcal{P}(\Sigma_t) \to (\mathcal{L} \times \mathcal{P}(\mathcal{E}))$
  $\alpha_a(X) \stackrel{\text{def}}{=} \{ (\ell, \rho_{|\mathbb{V}\to\mathbb{R}}) \,|\, (\ell, \rho) \in X \}$
- on interferences: forget all control locations:
  $\alpha_c : \mathcal{P}(\Sigma \times \Sigma) \to \mathcal{P}(\mathcal{E} \times \mathcal{E})$
  $\alpha_c(X) \stackrel{\text{def}}{=} \{ (\rho, \rho') \,|\, \exists \overline{\ell}, \overline{\ell}' : ((\overline{\ell}, \rho), (\overline{\ell}', \rho')) \in X \}$

Non-relational abstractions: on interferences

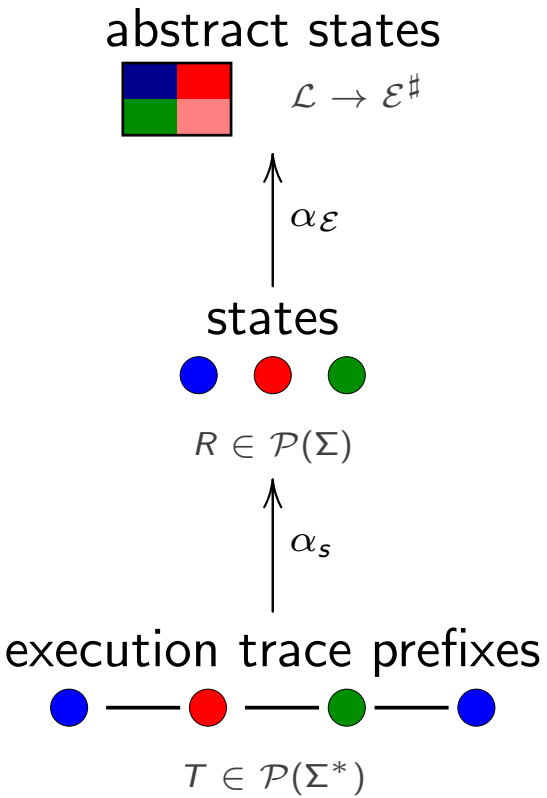
- forget the input-sensitivity:
  $\alpha_i : \mathcal{P}(\mathcal{E} \times \mathcal{E}) \to \mathcal{P}(\mathcal{E})$
  $\alpha_i(X) \stackrel{\text{def}}{=} \{ \rho' \,|\, \exists \rho : (\rho, \rho') \in X \}$
- forget the relationship between variables:
  $\alpha_n : \mathcal{P}(\mathcal{E}) \to (\mathbb{V} \to \mathcal{P}(\mathbb{R}))$
  $\alpha_n(X) \stackrel{\text{def}}{=} \lambda V \in \mathbb{V}. \{ \rho(V) \,|\, \rho \in X \}$

Further abstractions in numeric abstract domains.

From traces to thread-modular analyses



Compare with sequential analyses. . .



static analyzer

reachability

test

Construction of an interference-based analysis

Reminder: sequential analysis in denotational form

Expression semantics: $\mathsf{E}[\![expr]\!] : \mathcal{E} \to (\mathcal{P}(\mathbb{R}) \times \mathcal{P}(\Omega))$

$$\mathsf{E}[\![X]\!]\rho \stackrel{\text{def}}{=} \langle \{ \rho(X) \}, \emptyset \rangle$$
$$\mathsf{E}[\![[c_1, c_2]]\!]\rho \stackrel{\text{def}}{=} \langle \{ x \in \mathbb{R} \mid c_1 \leq x \leq c_2 \}, \emptyset \rangle$$
$$\mathsf{E}[\![-e_1]\!]\rho \stackrel{\text{def}}{=} \text{let } \langle V_1, O_1 \rangle = \mathsf{E}[\![e_1]\!]\rho \text{ in } \langle \{ -v_1 \mid v_1 \in V_1 \}, O_1 \rangle$$
$$\mathsf{E}[\![e_1 \diamond_\omega e_2]\!]\rho \stackrel{\text{def}}{=} \text{let } \forall i \in \{1,2\}: \langle V_i, O_i \rangle = \mathsf{E}[\![e_i]\!]\rho \text{ in}$$
$$\langle \{ v_1 \diamond v_2 \mid v_i \in V_i, \diamond \neq / \vee v_2 \neq 0 \}, O_1 \cup O_2 \cup \{ \omega \text{ if } \diamond = / \wedge 0 \in V_2 \} \rangle$$

Statement semantics: $\mathsf{C}[\![stat]\!] : (\mathcal{P}(\mathcal{E}) \times \mathcal{P}(\Omega)) \to (\mathcal{P}(\mathcal{E}) \times \mathcal{P}(\Omega))$

$$\mathsf{C}[\![X \leftarrow e]\!]\langle R, O \rangle \stackrel{\text{def}}{=} \langle \emptyset, O \rangle \sqcup \bigsqcup_{\rho \in R} \langle \{ \rho[X \mapsto v] \mid v \in V_\rho \}, O_\rho \rangle$$
$$\mathsf{C}[\![e \bowtie 0?]\!]\langle R, O \rangle \stackrel{\text{def}}{=} \langle \emptyset, O \rangle \sqcup \bigsqcup_{\rho \in R} \langle \{ \rho \mid \exists v \in V_\rho : v \bowtie 0 \}, O_\rho \rangle$$
$$\mathsf{C}[\![\textbf{if } e \bowtie 0 \textbf{ then } s]\!] X \stackrel{\text{def}}{=} (\mathsf{C}[\![s]\!] \circ \mathsf{C}[\![e \bowtie 0?]\!])X \sqcup \mathsf{C}[\![e \not\bowtie 0?]\!] X$$
$$\mathsf{C}[\![\textbf{while } e \bowtie 0 \textbf{ do } s]\!] X \stackrel{\text{def}}{=}$$
$$\mathsf{C}[\![e \not\bowtie 0?]\!] (\text{lfp} \lambda Y . X \sqcup (\mathsf{C}[\![s]\!] \circ \mathsf{C}[\![e \bowtie 0?]\!])Y)$$
$$\mathsf{C}[\![s_1 ; s_2]\!] \stackrel{\text{def}}{=} \mathsf{C}[\![s_2]\!] \circ \mathsf{C}[\![s_1]\!]$$

where $\langle V_\rho, O_\rho \rangle \stackrel{\text{def}}{=} \mathsf{E}[\![e]\!]\rho$

Denotational semantics with interferences

Interferences in $\mathcal{I} \stackrel{\text{def}}{=} \mathcal{T} \times \mathbb{V} \times \mathbb{R}$
$\langle\, t, X, v \,\rangle$ means: t can store the value v into the variable X

We define the analysis of a thread t
with respect to a set of interferences $I \subseteq \mathcal{I}$.

Expressions with interference: for thread t

$\mathsf{E}_t[\![\, expr \,]\!] : (\mathcal{E} \times \mathcal{P}(\mathcal{I})) \to (\mathcal{P}(\mathbb{R}) \times \mathcal{P}(\Omega))$

- Apply interferences to read variables:
  $\mathsf{E}_t[\![\, X \,]\!] \langle\, \rho, I \,\rangle \stackrel{\text{def}}{=} \langle\, \{\, \rho(X) \,\} \cup \{\, v \mid \exists u \neq t : \langle\, u, X, v \,\rangle \in I \,\}, \emptyset \,\rangle$
- Pass recursively I down to sub-expressions:
  $\mathsf{E}_t[\![\, -e_1 \,]\!] \langle\, \rho, I \,\rangle \stackrel{\text{def}}{=}$
  $\quad$ let $\langle\, V_1, O_1 \,\rangle = \mathsf{E}_t[\![\, e_1 \,]\!] \langle\, \rho, I \,\rangle$ in $\langle\, \{\, -v_1 \mid v_1 \in V_1 \,\}, O_1 \,\rangle$
  $\quad \ldots$

Denotational semantics with interferences (cont.)

Statements with interference: for thread t

$\mathsf{C}_t[\![stat]\!] : (\mathcal{P}(\mathcal{E}) \times \mathcal{P}(\Omega) \times \mathcal{P}(\mathcal{I})) \to (\mathcal{P}(\mathcal{E}) \times \mathcal{P}(\Omega) \times \mathcal{P}(\mathcal{I}))$

- pass interferences to expressions
- collect new interferences due to assignments
- accumulate interferences from inner statements

$\mathsf{C}_t[\![X \leftarrow e]\!] \langle R, O, I \rangle \stackrel{\text{def}}{=}$
$\langle \emptyset, O, I \rangle \sqcup \bigsqcup_{\rho \in R} \langle \{ \rho[X \mapsto v] \mid v \in V_\rho \}, O_\rho, \{ \langle t, X, v \rangle \mid v \in V_\rho \} \rangle$

$\mathsf{C}_t[\![s_1; s_2]\!] \stackrel{\text{def}}{=} \mathsf{C}_t[\![s_2]\!] \circ \mathsf{C}_t[\![s_1]\!]$

...

(noting $\langle V_\rho, O_\rho \rangle \stackrel{\text{def}}{=} \mathsf{E}_t[\![e]\!] \langle \rho, I \rangle$)
($\sqcup$ is now the element-wise $\cup$ in $\mathcal{P}(\mathcal{E}) \times \mathcal{P}(\Omega) \times \mathcal{P}(\mathcal{I})$)

Denotational semantics with interferences (cont.)

Program semantics: $\mathrm{P}[\![\, prog \,]\!] \subseteq \Omega$

Given $prog ::= stat_1 \;||\; \cdots \;||\; stat_n$, we compute:

$$\mathrm{P}[\![\, prog \,]\!] \stackrel{\text{def}}{=} \left[\mathrm{lfp}\, \lambda \langle\, O,\, I \,\rangle .\, \bigsqcup_{t \in \mathcal{T}} [\mathrm{C_t}[\![\, stat_t \,]\!] \langle\, \mathcal{E}_0,\, \emptyset,\, I \,\rangle]_{\Omega,\mathcal{I}} \right]_{\Omega}$$

- each thread analysis starts in an initial environment set $\mathcal{E}_0 \stackrel{\text{def}}{=} \{\, \lambda V.0 \,\}$
- $[X]_{\Omega,\mathcal{I}}$ projects $X \in \mathcal{P}(\mathcal{E}) \times \mathcal{P}(\Omega) \times \mathcal{P}(\mathcal{I})$ on $\mathcal{P}(\Omega) \times \mathcal{P}(\mathcal{I})$ and interferences and errors from all threads are joined
  (the output environments are ignored)
- $\mathrm{P}[\![\, prog \,]\!]$ only outputs the set of possible run-time errors

Example

Example

t_1	t_2
while $^{\ell 1}\ 0 = 0$ **do**	**while** $^{\ell 4}\ 0 = 0$ **do**
$^{\ell 2}$ **if** $x < y$ **then**	$^{\ell 5}$ **if** $y < 10$ **then**
$^{\ell 3}\ x \leftarrow x + 1$	$^{\ell 6}\ y \leftarrow y + 1$

Concrete interference semantics:

iteration 1

$I = \emptyset$

$\ell 1 : x = 0, y = 0$

$\ell 4 : x = 0, y \in [0, 10]$

new $I = \{ \langle t_2, y, 1 \rangle, \ldots, \langle t_2, y, 10 \rangle \}$

Example

Example

t_1	t_2
while $^{\ell 1}\ 0 = 0$ **do**	**while** $^{\ell 4}\ 0 = 0$ **do**
$\quad ^{\ell 2}$ **if** $x < y$ **then**	$\quad ^{\ell 5}$ **if** $y < 10$ **then**
$\quad\quad ^{\ell 3}\ x \leftarrow x + 1$	$\quad\quad ^{\ell 6}\ y \leftarrow y + 1$

Concrete interference semantics:

iteration 2

$I = \{ \langle t_2, y, 1 \rangle, \ldots, \langle t_2, y, 10 \rangle \}$

$\ell 1 : x \in [0, 10], y = 0$

$\ell 4 : x = 0, y \in [0, 10]$

new $I = \{ \langle t_1, x, 1 \rangle, \ldots, \langle t_1, x, 10 \rangle, \langle t_2, y, 1 \rangle, \ldots, \langle t_2, y, 10 \rangle \}$

Example

Example

t_1	t_2
while $^{\ell 1}\ 0 = 0$ **do**	**while** $^{\ell 4}\ 0 = 0$ **do**
$\quad ^{\ell 2}$ **if** $x < y$ **then**	$\quad ^{\ell 5}$ **if** $y < 10$ **then**
$\quad\quad ^{\ell 3}\ x \leftarrow x + 1$	$\quad\quad ^{\ell 6}\ y \leftarrow y + 1$

Concrete interference semantics:

iteration 3

$I = \{ \langle t_1, x, 1 \rangle, \ldots, \langle t_1, x, 10 \rangle, \langle t_2, y, 1 \rangle, \ldots, \langle t_2, y, 10 \rangle \}$

$\ell 1 : x \in [0, 10], y = 0$

$\ell 4 : x = 0, y \in [0, 10]$

new $I = \{ \langle t_1, x, 1 \rangle, \ldots, \langle t_1, x, 10 \rangle, \langle t_2, y, 1 \rangle, \ldots, \langle t_2, y, 10 \rangle \}$

Example

Example

t_1	t_2
while $^{\ell 1}\ 0 = 0$ **do**	**while** $^{\ell 4}\ 0 = 0$ **do**
$\quad ^{\ell 2}$ **if** $x < y$ **then**	$\quad ^{\ell 5}$ **if** $y < 10$ **then**
$\quad\quad ^{\ell 3}\ x \leftarrow x + 1$	$\quad\quad ^{\ell 6}\ y \leftarrow y + 1$

Concrete interference semantics:

iteration 3

$I = \{ \langle t_1, x, 1 \rangle, \ldots, \langle t_1, x, 10 \rangle, \langle t_2, y, 1 \rangle, \ldots, \langle t_2, y, 10 \rangle \}$

$\ell 1 : x \in [0, 10],\ y = 0$

$\ell 4 : x = 0,\ y \in [0, 10]$

new $I = \{ \langle t_1, x, 1 \rangle, \ldots, \langle t_1, x, 10 \rangle, \langle t_2, y, 1 \rangle, \ldots, \langle t_2, y, 10 \rangle \}$

Note: we don't get that $x \leq y$ at $\ell 1$, only that $x, y \in [0, 10]$

Interference abstraction

Abstract interferences $\mathcal{I}^\sharp$

$\mathcal{P}(\mathcal{I}) \stackrel{\text{def}}{=} \mathcal{P}(\mathcal{T} \times \mathbb{V} \times \mathbb{R})$ is abstracted as $\mathcal{I}^\sharp \stackrel{\text{def}}{=} (\mathcal{T} \times \mathbb{V}) \to \mathcal{R}^\sharp$
where $\mathcal{R}^\sharp$ abstracts $\mathcal{P}(\mathbb{R})$ (e.g. intervals)

Abstract semantics with interferences $\mathrm{C}^\sharp_{\mathrm{t}}[\![s]\!]$

derived from $\mathrm{C}^\sharp[\![s]\!]$ in a generic way:

Example: $\mathrm{C}^\sharp_{\mathrm{t}}[\![X \leftarrow e]\!] \langle R^\sharp, \Omega, I^\sharp \rangle$

- for each Y in e, get its interference $Y^\sharp_{\mathcal{R}} = \bigsqcup^\sharp_{\mathcal{R}} \{ I^\sharp \langle u, Y \rangle \mid u \neq t \}$
- if $Y^\sharp_{\mathcal{R}} \neq \bot^\sharp_{\mathcal{R}}$, replace Y in e with $get \langle Y, R^\sharp \rangle \sqcup^\sharp_{\mathcal{R}} Y^\sharp_{\mathcal{R}}$
  (where $get(Y, R^\sharp)$ extracts the abstract values in $\mathcal{R}^\sharp$ of a variable Y from $R^\sharp \in \mathcal{E}^\sharp$)
- compute $\langle R^{\sharp\prime}, O' \rangle = \mathrm{C}^\sharp[\![e]\!] \langle R^\sharp, O \rangle$
- enrich $I^\sharp \langle t, X \rangle$ with $get(X, R^{\sharp\prime})$

Static analysis with interferences

Abstract analysis

$$\mathrm{P}^\sharp[\![\, prog \,]\!] \stackrel{\text{def}}{=} \left[\lim \lambda \langle\, O,\, I^\sharp \,\rangle .\, \langle\, O,\, I^\sharp \,\rangle \triangledown \bigsqcup^\sharp_{t \in \mathcal{T}} \left[\mathrm{C}^\sharp_{\mathrm{t}}[\![\, stat_t \,]\!] \langle\, \mathcal{E}^\sharp_0,\, \emptyset,\, I^\sharp \,\rangle \right]_{\Omega, \mathcal{I}^\sharp} \right]_\Omega$$

- effective analysis by structural induction
- termination ensured by a widening
- parametrized by a choice of abstract domains $\mathcal{R}^\sharp$, $\mathcal{E}^\sharp$

- interferences are flow-insensitive and non-relational in $\mathcal{R}^\sharp$
- thread analysis remains flow-sensitive and relational in $\mathcal{E}^\sharp$

[Miné12]

Path-based semantics

Control paths

$atomic ::= X \leftarrow expr \mid expr \bowtie 0?$

Control paths

$\pi : stat \to \mathcal{P}(atomic^*)$

$\pi(X \leftarrow e) \stackrel{\text{def}}{=} \{ X \leftarrow e \}$

$\pi(\textbf{if } e \bowtie 0 \textbf{ then } s) \stackrel{\text{def}}{=} (\{ e \bowtie 0? \} \cdot \pi(s)) \cup \{ e \not\bowtie 0? \}$

$\pi(\textbf{while } e \bowtie 0 \textbf{ do } s) \stackrel{\text{def}}{=} \left(\bigcup_{i \geq 0} (\{ e \bowtie 0? \} \cdot \pi(s))^i \right) \cdot \{ e \not\bowtie 0? \}$

$\pi(s_1; s_2) \stackrel{\text{def}}{=} \pi(s_1) \cdot \pi(s_2)$

$\pi(stat)$ is a (generally infinite) set of finite control paths

Path-based concrete semantics of sequential programs

Join-over-all-path semantics

$$\underline{\Pi[\![P]\!] : (\mathcal{P}(\mathcal{E}) \times \mathcal{P}(\Omega)) \to (\mathcal{P}(\mathcal{E}) \times \mathcal{P}(\Omega))} \qquad P \subseteq \mathit{atomic}^*$$

$$\Pi[\![P]\!]\langle R, O \rangle \stackrel{\text{def}}{=} \bigsqcup_{s_1 \cdot \ldots \cdot s_n \in P} (\mathrm{C}[\![s_n]\!] \circ \cdots \circ \mathrm{C}[\![s_1]\!])\langle R, O \rangle$$

Semantic equivalence

$$\mathrm{C}[\![\mathit{stat}]\!] = \Pi[\![\pi(\mathit{stat})]\!]$$

(not true in the abstract)

Advantages:

- easily extended to concurrent programs (path interleavings)
- able to model program transformations (weak memory models)

Path-based concrete semantics of concurrent programs

Concurrent control paths

$$\begin{aligned}\pi_* &\stackrel{\text{def}}{=} \{\,\text{interleavings of } \pi(stat_t),\ t \in \mathcal{T}\,\}\\ &= \{\,p \in atomic^* \mid \forall t \in \mathcal{T},\ proj_t(p) \in \pi(stat_t)\,\}\end{aligned}$$

Interleaving program semantics

$$\mathsf{P}_*[\![\, prog \,]\!] \stackrel{\text{def}}{=} [\,\sqcap[\![\,\pi_*\,]\!]\langle\,\mathcal{E}_0, \emptyset\,\rangle\,]_\Omega$$

($proj_t(p)$ keeps only the atomic statement in p coming from thread t)

Soundness of the interference semantics

Soundness theorem

$$\mathsf{P}_*[\![\, prog \,]\!] \subseteq \mathsf{P}[\![\, prog \,]\!]$$

Proof sketch:

- define $\mathbb{\Pi}_t[\![\, P \,]\!]\, X \stackrel{\text{def}}{=} \bigsqcup \{\, \mathsf{C}_t[\![\, s_1; \ldots; s_n \,]\!]\, X \mid s_1 \cdot \ldots \cdot s_n \in P \,\}$, then $\mathbb{\Pi}_t[\![\, \pi(s) \,]\!] = \mathsf{C}_t[\![\, s \,]\!]$;
- given the interference fixpoint $I \subseteq \mathcal{I}$ from $\mathsf{P}[\![\, prog \,]\!]$, prove by recurrence on the length of $p \in \pi_*$ that:
  - $\forall t \in \mathcal{T}, \forall \rho \in [\mathbb{\Pi}[\![\, p \,]\!] \langle\, \mathcal{E}_0, \emptyset \,\rangle]_{\mathcal{E}}$, $\exists \rho' \in [\mathbb{\Pi}_t[\![\, proj_t(p) \,]\!] \langle\, \mathcal{E}_0, \emptyset, I \,\rangle]_{\mathcal{E}}$ such that $\forall X \in \mathbb{V}$, $\rho(X) = \rho'(X)$ or $\langle\, u, X, \rho(X) \,\rangle \in I$ for some $u \neq t$.
  - $[\mathbb{\Pi}[\![\, p \,]\!] \langle\, \mathcal{E}_0, \emptyset \,\rangle]_\Omega \subseteq \bigcup_{t \in \mathcal{T}} [\mathbb{\Pi}_t[\![\, proj_t(p) \,]\!] \langle\, \mathcal{E}_0, \emptyset, I \,\rangle]_\Omega$

Note: sound but not complete

Weakly consistent memories

Issues with weak consistency

program written

$F_1 \leftarrow 1;$	$F_2 \leftarrow 1;$
if $F_2 = 0$ **then**	**if** $F_1 = 0$ **then**
S_1	S_2

(simplified Dekker mutual exclusion algorithm)

S_1 and S_2 cannot execute simultaneously.

Issues with weak consistency

program written

$F_1 \leftarrow 1;$	$F_2 \leftarrow 1;$
if $F_2 = 0$ **then**	**if** $F_1 = 0$ **then**
S_1	S_2

$\longrightarrow$

program executed

if $F_2 = 0$ **then**	**if** $F_1 = 0$ **then**
$F_1 \leftarrow 1;$	$F_2 \leftarrow 1;$
S_1	S_2

(simplified Dekker mutual exclusion algorithm)

S_1 and S_2 can execute simultaneously.
Not a sequentially consistent behavior!

Caused by:

- write FIFOs, caches, distributed memory
- hardware or compiler optimizations, transformations
- ...

behavior accepted by Java [Mans05]

Out of thin air principle

original program

$R_1 \leftarrow X;$	$R_2 \leftarrow Y;$
$Y \leftarrow R_1$	$X \leftarrow R_2$

(example from causality test case #4 for Java by Pugh et al.)

We should not have $R_1 = 42$.

Out of thin air principle

original program

$$\begin{array}{l|l} R_1 \leftarrow X; & R_2 \leftarrow Y; \\ Y \leftarrow R_1 & X \leftarrow R_2 \end{array}$$

$\longrightarrow$

"optimized" program

$$\begin{array}{l|l} Y \leftarrow 42; & \\ R_1 \leftarrow X; & R_2 \leftarrow Y; \\ Y \leftarrow R_1 & X \leftarrow R_2 \end{array}$$

(example from causality test case #4 for Java by Pugh et al.)

We should not have $R_1 = 42$.

Possible if we allow speculative writes!
$\Longrightarrow$ we disallow this kind of program transformations.

(also forbidden in Java)

Atomicity and granularity

original program

$$X \leftarrow X + 1 \mid X \leftarrow X + 1$$

We assumed that assignments are atomic...

Atomicity and granularity

original program

$$X \leftarrow X + 1 \mid X \leftarrow X + 1$$

$\longrightarrow$

executed program

$$\begin{array}{l|l} r_1 \leftarrow X + 1 & r_2 \leftarrow X + 1 \\ X \leftarrow r_1 & X \leftarrow r_2 \end{array}$$

We assumed that assignments are atomic...
but that may not be the case

The second program admits more behaviors
e.g.: $X = 1$ at the end of the program

[Reyn04]

Path-based definition of weak consistency

Acceptable control path transformations: $p \rightsquigarrow q$

only reduce interferences and errors

- Reordering: $X_1 \leftarrow e_1 \cdot X_2 \leftarrow e_2 \rightsquigarrow X_2 \leftarrow e_2 \cdot X_1 \leftarrow e_1$
  (if $X_1 \notin var(e_2)$, $X_2 \notin var(e_1)$, and e_1 does not stop the program)
- Propagation: $X \leftarrow e \cdot s \rightsquigarrow X \leftarrow e \cdot s[e/X]$
  (if $X \notin var(e)$, $var(e)$ are thread-local, and e is deterministic)
- Factorization: $s_1 \cdot \ldots \cdot s_n \rightsquigarrow X \leftarrow e \cdot s_1[X/e] \cdot \ldots \cdot s_n[X/e]$
  (if X is fresh, $\forall i$, $var(e) \cap lval(s_i) = \emptyset$, and e has no error)
- Decomposition: $X \leftarrow e_1 + e_2 \rightsquigarrow T \leftarrow e_1 \cdot X \leftarrow T + e_2$
  (change of granularity)
- ...

but **NOT:**

- "out-of-thin-air" writes: $X \leftarrow e \rightsquigarrow X \leftarrow 42 \cdot X \leftarrow e$

Soundness of the interference semantics

Interleaving semantics of transformed programs $\mathsf{P}'_*[\![\, prog \,]\!]$

- $\pi'(s) \stackrel{\text{def}}{=} \{\, p \mid \exists p' \in \pi(s) : p' \rightsquigarrow^* p \,\}$
- $\pi'_* \stackrel{\text{def}}{=} \{\,$ interleavings of $\pi'(stat_t),\ t \in \mathcal{T} \,\}$
- $\mathsf{P}'_*[\![\, prog \,]\!] \stackrel{\text{def}}{=} [\sqcap[\![\, \pi'_* \,]\!]\langle\, \mathcal{E}_0, \emptyset \,\rangle]_\Omega$

Soundness theorem

$\mathsf{P}'_*[\![\, prog \,]\!] \subseteq \mathsf{P}[\![\, prog \,]\!]$

$\Longrightarrow$ the interference semantics is sound
wrt. weakly consistent memories and changes of granularity

Synchronisation

Scheduling

Synchronization primitives

$$
\begin{array}{rcl}
\mathit{stat} & ::= & \mathbf{lock}(m) \\
& | & \mathbf{unlock}(m) \\
& | & X \leftarrow \mathbf{islocked}(m) \\
& | & \mathbf{yield}
\end{array}
$$

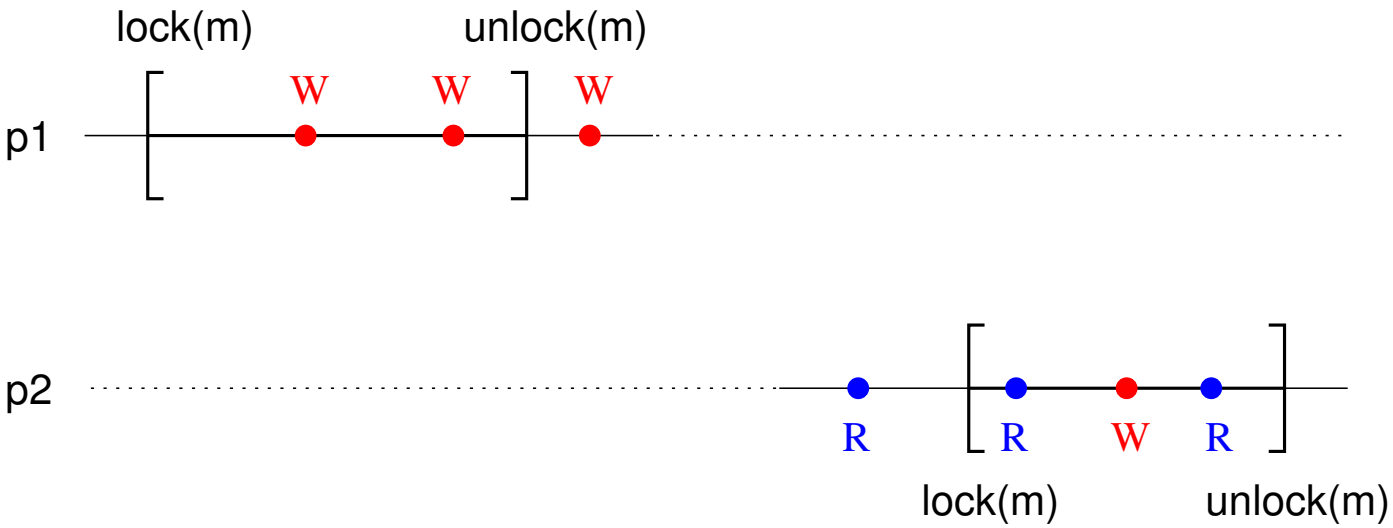
$m \in \mathbb{M}$: finite set of non-recursive mutexes

each thread has a fixed, distinct priority

Real-time scheduling

- only the highest priority unblocked thread can run
- **lock** and **yield** may block
- **yield**ing threads wake up non-deterministically (preempting lower-priority threads)
- explicit synchronisation enforces memory consistency

Mutual exclusion



Interleaving semantics $\mathrm{P}_*[\![\, prog \,]\!]$:

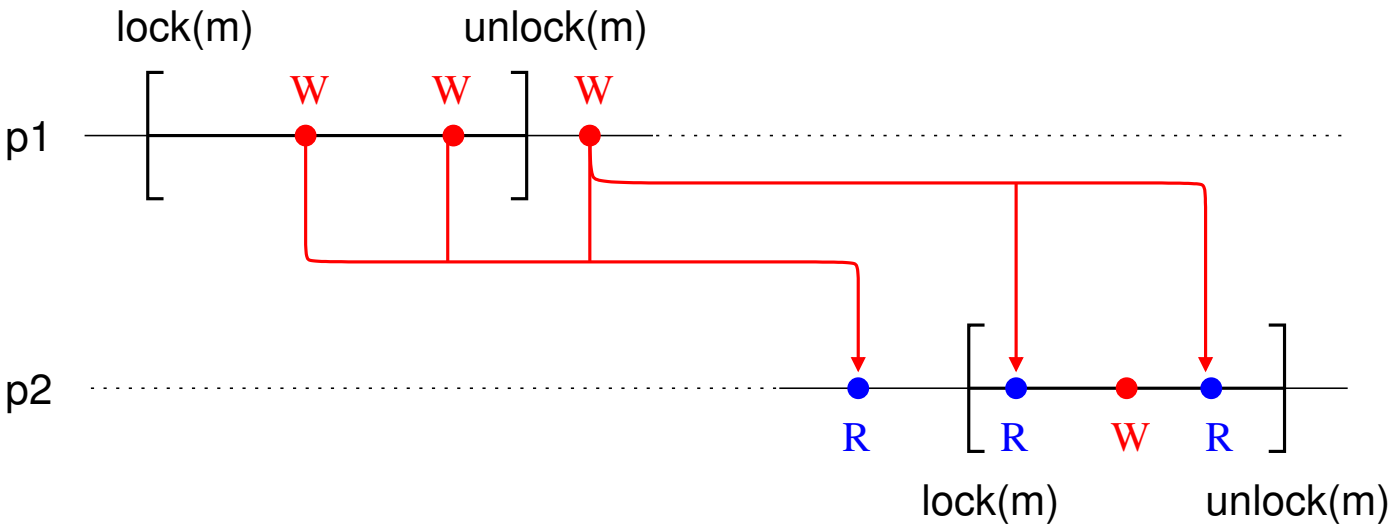
restrict interleavings of control paths

Interference semantics $\mathrm{P}[\![\, prog \,]\!]$, $\mathrm{P}^\sharp[\![\, prog \,]\!]$:

partition wrt. an abstract local view of the scheduler $\mathbb{C}$

- $\mathcal{E} \rightsquigarrow \mathcal{E} \times \mathbb{C}, \quad \mathcal{E}^\sharp \rightsquigarrow \mathbb{C} \to \mathcal{E}^\sharp$
- $\mathcal{I} \stackrel{\text{def}}{=} \mathcal{T} \times \mathbb{V} \times \mathbb{R} \rightsquigarrow \mathcal{I} \stackrel{\text{def}}{=} \mathcal{T} \times \mathbb{C} \times \mathbb{V} \times \mathbb{R}$,
  $\mathcal{I}^\sharp \stackrel{\text{def}}{=} (\mathcal{T} \times \mathbb{V}) \to \mathcal{R}^\sharp \rightsquigarrow \mathcal{I}^\sharp \stackrel{\text{def}}{=} (\mathcal{T} \times \mathbb{C} \times \mathbb{V}) \to \mathcal{R}^\sharp$

Mutual exclusion

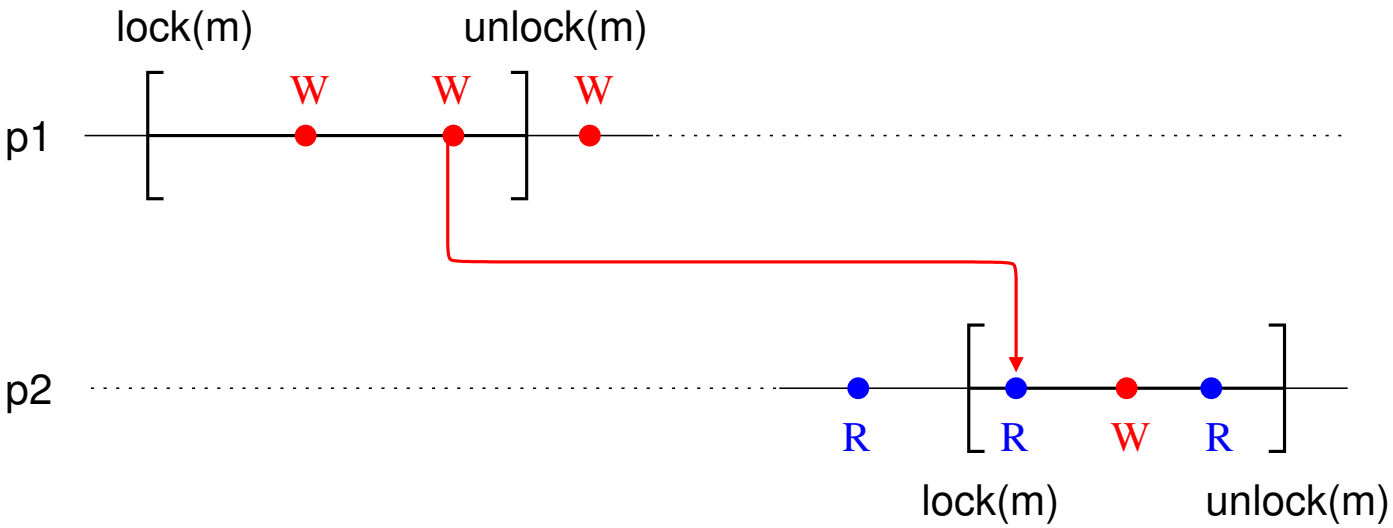


Data-race effects

Partition wrt. mutexes $M \subseteq \mathbb{M}$ held by the current thread t

- $\mathsf{C}_t[\![X \leftarrow e]\!] \langle \rho, M, I \rangle$ adds
  $\{ \langle t, M, X, v \rangle \mid v \in \mathsf{E}_t[\![X]\!] \langle \rho, M, I \rangle \}$ to I
- $\mathsf{E}_t[\![X]\!] \langle \rho, M, I \rangle =$
  $\{ \rho(X) \} \cup \{ v \mid \langle t', M', X, v \rangle \in I,\ t \neq t',\ M \cap M' = \emptyset \}$
- flow-insensitive, subject to weak memory consistency

Mutual exclusion



Well-synchronized effects

- last write before unlock affects first read after lock
- partition interferences wrt. a protecting mutex m (and M)
- $C_t[\![\textbf{unlock}(m)]\!] \langle \rho, M, I \rangle$ stores $\rho(X)$ into I
- $C_t[\![\textbf{lock}(m)]\!] \langle \rho, M, I \rangle$ imports values form I into ρ
- imprecision: non-relational, largely flow-insensitive

Example analysis

abstract consumer/producer

t_1	t_2
while 0=0 **do**	**while** 0=0 **do**
lock(m);$^{\ell 1}$	**lock**(m);
if $X > 0$ **then** $^{\ell 2}$ $X \leftarrow X - 1$;	$X \leftarrow X + 1$;
unlock(m);	**if** $X > 10$ **then** $X \leftarrow 10$;
$^{\ell 3}$ $Y \leftarrow X$	**unlock**(m)

- at $\ell 1$, the **unlock** $-$ **lock** effect from t_2 imports $\{X\} \times [1, 10]$
- at $\ell 2$, $X \in [1, 10]$, no effect from t_2: $X \leftarrow X - 1$ is safe
- at $\ell 3$, $X \in [0, 9]$, and t_2 has the effects $\{X\} \times [1, 10]$
  so, $Y \in [0, 10]$

Real-time scheduling

priority-based critical sections

high thread	low thread
$L \leftarrow$ **islocked**(m);	**lock**(m);
if $L = 0$ **then**	$Z \leftarrow Y$;
$Y \leftarrow Y + 1$;	$Y \leftarrow 0$;
yield	**unlock**(m)

Partition interferences and memory states wrt. scheduling state

- partition wrt. mutexes tested with **islocked**
- $L \leftarrow$ **islocked**(m) creates two partitions
  - P_0 where $L = 0$ and m is free
  - P_1 where $L = 1$ and m is locked
- P_0 handled as if m where locked
- blocking primitives merge P_0 and P_1 (**lock**, **yield**)

Limitations of the interference abstraction

Lack of relational lock invariants

a difficult example

$$\mathcal{E}_0 : X = Y = 5$$

while 1 **do**	**while** 1 **do**
lock(m);	**lock**(m);
if $X > 0$ **then**	**if** $X < 10$ **then**
$X \leftarrow X - 1;$	$X \leftarrow X + 1;$
$Y \leftarrow Y - 1;$	$Y \leftarrow Y + 1;$
unlock(m)	**unlock**(m)

Our analysis finds $X \in [0, 10]$, but no bound on Y.

Actually $Y \in [0, 10]$.

To prove this, we would need to infer the relational invariant $X = Y$ at lock boundaries.

Lack of inter-process flow-sensitivity

a more difficult example

while 1 **do**	**while** 1 **do**
lock(m);	**lock**(m);
$X \leftarrow X + 1$;	$X \leftarrow X + 1$;
unlock(m);	**unlock**(m);
lock(m);	**lock**(m);
$X \leftarrow X - 1$;	$X \leftarrow X - 1$;
unlock(m)	**unlock**(m)

Our analysis finds no bound on X.

Actually $X \in [-2, 2]$ at all program points.
To prove this we need to infer an invariant on the history of interleaved executions:
no more than two incrementation (resp. decrementation) can occur without a decrementation (resp. incrementation).

Summary

Conclusion

We presented a static analysis that is:

- inspired from thread-modular proof methods
- sound for all interleavings
- sound for weakly consistent memory semantics
- aware of scheduling and synchronization
- parametrized by abstract domains

Future work: leverage the connection with rely-guarantee

- relational interferences
  (especially for synchronized program parts)
- flow-sensitive interferences and invariants

Bibliography

Bibliography

[Bour93] **F. Bourdoncle**. *Efficient chaotic iteration strategies with widenings.* In Proc. FMPA'93, LNCS vol. 735, pp. 128–141, Springer, 1993.

[Carr09] **J.-L. Carré & C. Hymans**. *From single-thread to multithreaded: An efficient static analysis algorithm.* In arXiv:0910.5833v1, EADS, 2009.

[Cous84] **P. Cousot & R. Cousot**. *Invariance proof methods and analysis techniques for parallel programs.* In Automatic Program Construction Techniques, chap. 12, pp. 243–271, Macmillan, 1984.

[Cous85] **R. Cousot**. *Fondements des méthodes de preuve d'invariance et de fatalité de programmes parallèles.* In Thèse d'Etat es sc. math., INP Lorraine, Nancy, 1985.

[Hoar69] **C. A. R. Hoare**. *An axiomatic basis for computer programming.* In Com. ACM, 12(10):576–580, 1969.

Bibliography (cont.)

[Jone81] **C. B. Jones**. *Development methods for computer programs including a notion of interference.* In PhD thesis, Oxford University, 1981.

[Lamp77] **L. Lamport**. *Proving the correctness of multiprocess programs.* In IEEE Trans. on Software Engineering, 3(2):125–143, 1977.

[Lamp78] **L. Lamport**. *Time, clocks, and the ordering of events in a distributed system.* In Comm. ACM, 21(7):558–565, 1978.

[Mans05] **J. Manson, B. Pugh & S. V. Adve**. *The Java memory model.* In Proc. POPL'05, pp. 378–391, ACM, 2005.

[Miné12] **A. Miné**. *Static analysis of run-time errors in embedded real-time parallel C programs.* In LMCS 8(1:26), 63 p., arXiv, 2012.

[Owic76] **S. Owicki & D. Gries**. *An axiomatic proof technique for parallel programs I.* In Acta Informatica, 6(4):319–340, 1976.

Bibliography (cont.)

[Reyn04] **J. C. Reynolds**. *Toward a grainless semantics for shared-variable concurrency.* In Proc. FSTTCS'04, LNCS vol. 3328, pp. 35–48, Springer, 2004.

[Sara07] **V. A. Saraswat, R. Jagadeesan, M. M. Michael & C. von Praun**. *A theory of memory models.* In Proc. PPoPP'07, pp. 161–172, ACM, 2007.