Static Analysis of Concurrent Programs

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

Antoine Miné

year 2013–2014

course 08-A
15 November 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 (×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
- fixed number of threads (no dynamic creation of threads)
- numeric programs (real-valued variables)

Goal: static analysis

- infer numeric program invariants
- 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
  - 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

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

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

State-based analyses

Sequential program semantics (reminders)

Transition systems

Transition system: $(\Sigma, \tau, \mathcal{I})$

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

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

- Σ^*: finite traces
- Σ^ω: infinite countable traces
- $\Sigma^\infty \stackrel{\text{def}}{=} \Sigma^* \cup \Sigma^\omega$: finite or infinite countable traces
- $u \preceq t$: u is a prefix of t

We view program semantics and properties as sets of traces.

Traces of a transition system

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

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

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

- able to express many properties of programs, e.g.:
  - safety: $\mathcal{M}_\infty \subseteq S^\infty$ (executions stay in S)
  - ordering: $\mathcal{M}_\infty \subseteq S_1^\infty \cdot S_2^\infty$ (S_2 can only occur after S_1)
  - termination: $\mathcal{M}_\infty \subseteq \Sigma^*$ (executions are finite)
  - inevitability: $\mathcal{M}_\infty \subseteq \Sigma^* \cdot S \cdot \Sigma^\infty$ (executions pass through S)

Traces of a transition system

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

- set of finite prefixes of executions:
  $\mathcal{T}_p \stackrel{\text{def}}{=} \{\, \sigma_0, \ldots, \sigma_n \,|\, \sigma_0 \in \mathcal{I}, \forall i < n\colon \sigma_i \to_\tau \sigma_{i+1} \,\}$
- $\mathcal{T}_p$ is an abstraction of the maximal trace semantics
  $\mathcal{T}_p = \alpha_{*\preceq}(\mathcal{M}_\infty)$ where $\alpha_{*\preceq}(X) \stackrel{\text{def}}{=} \{\, t \in \Sigma^* \,|\, \exists u \in X\colon t \preceq u \,\}$
- $\mathcal{T}_p$ can prove safety properties: $\mathcal{T}_p \subseteq S^*$ (executions stay in S)
  $\mathcal{T}_p$ can prove ordering properties: $\mathcal{T}_p \subseteq S_1^* \cdot S_2^*$
  (if S_1 and S_2 occur, S_2 occurs after S_1)
- $\mathcal{T}_p$ cannot prove termination nor inevitability properties
- fixpoint characterisation: $\mathcal{T}_p = \text{lfp}\, F_p$ where
  $F_p(X) = \mathcal{I} \cup \{\, \sigma_0, \ldots, \sigma_{n+1} \,|\, \sigma_0, \ldots, \sigma_n \in X \wedge \sigma_n \to_\tau \sigma_{n+1} \,\}$

State abstraction

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

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

States of a sequential program

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:

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

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 $\mathcal{P}(\Omega)$

Expression semantics with errors

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

$$\mathsf{E}[\![\, X \,]\!]\,\rho \stackrel{\text{def}}{=} \langle\, \{\, \rho(X) \,\},\ \emptyset \,\rangle$$

$$\mathsf{E}[\![\, [c_1, c_2] \,]\!]\,\rho \stackrel{\text{def}}{=} \langle\, \{\, x \in \mathbb{R} \mid c_1 \leq x \leq c_2 \,\},\ \emptyset \,\rangle$$

$$\mathsf{E}[\![\, -e_1 \,]\!]\,\rho \stackrel{\text{def}}{=} \text{let } \langle\, V_1,\ O_1 \,\rangle = \mathsf{E}[\![\, e_1 \,]\!]\,\rho \text{ in } \langle\, \{\, -v_1 \mid v_1 \in V_1 \,\},\ O_1 \,\rangle$$

$$\mathsf{E}[\![\, e_1 \diamond_\omega e_2 \,]\!]\,\rho \stackrel{\text{def}}{=} \text{let } \forall i \in \{\, 1, 2 \,\} \colon \langle\, V_i,\ O_i \,\rangle = \mathsf{E}[\![\, e_i \,]\!]\,\rho \text{ in } \langle\, \{\, v_1 \diamond v_2 \mid v_i \in V_i,\ \diamond \neq / \vee v_2 \neq 0 \,\},\ O_1 \cup O_2 \cup \{\, \omega \text{ if } \diamond = / \wedge 0 \in V_2 \,\} \,\rangle$$

- 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)

Reminders: semantics in equational form

Principle: (without handling errors in Ω)

- see 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 $\ell \in \mathcal{L}$

  $\forall \ell \in \mathcal{L}$: $\mathcal{X}_\ell \stackrel{\text{def}}{=} \{ m \in \mathcal{E} \mid (\ell, m) \in \mathcal{R} \}$

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

Example:

```
ℓ1 i ← 2;
ℓ2 n ← [−∞, +∞];
ℓ3 while ℓ4 i < n do
     ℓ5 if [0, 1] = 0 then
          ℓ6 i ← i + 1
     ℓ7
ℓ8
```

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

Semantics in denotational form

Input-output function $C[\![\, stat \,]\!]$

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

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

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

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

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

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

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

- 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 semantics in denotational form

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

$\mathsf{C}^\sharp[\![\, stat \,]\!] : \mathcal{D}^\sharp \to \mathcal{D}^\sharp$

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

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

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

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

- the abstract interpreter mimicks an actual interpreter
- efficient in memory (intermediate invariants are not kept)
- less flexibility in the iteration scheme
  (iteration order, widening points, etc.)

Concurrent program semantics

Labelled transition systems

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

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

Labelled 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: $prog ::= {}^{\ell^i_1} stat_1{}^{\ell^x_1} \mid\mid \cdots \mid\mid {}^{\ell^i_n} stat_n{}^{\ell^x_n}$
- thread are identified by number in $\mathbb{T} \stackrel{\text{def}}{=} \{1, \ldots, n\}$

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

- control state $L(t) \in \mathcal{L}$ for each thread $t \in \mathbb{T}$, and
- single shared memory state $\rho \in \mathcal{E}$
- or error state in $\omega \in \Omega$

Initial states:

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

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

From concurrent programs to labelled transition systems

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

$$(L,\rho) \xrightarrow{t}_{\tau} (L',\rho') \overset{\text{def}}{\iff} (L(t),\rho) \to_{\tau[stat_t]} (L'(t),\rho') \wedge \forall u \neq t\colon L(u) = L'(u)$$

$$(L,\rho) \xrightarrow{t}_{\tau} \omega \overset{\text{def}}{\iff} (L(t),\rho) \to_{\tau[stat_t]} \omega$$

- based on the transition relation of individual threads seen as sequential processes $stat_t$:[1]
  $\tau[stat] \subseteq (\mathcal{L} \times \mathcal{E}) \times ((\mathcal{L} \times \mathcal{E}) \cup \Omega)$
  - choose a thread t to run
  - update ρ and $L(t)$
  - leave $L(u)$ intact for $u \neq t$
- each $\sigma \to \sigma'$ in $\tau[stat_t]$ leads to many transitions in τ!

[1] See lesson 02-B for the full definition of $\tau[stat]$.

Interleaved trace semantics

Maximal and finite prefix trace semantics are as before:

Blocking states: $\mathcal{B} \stackrel{\text{def}}{=} \{\, \sigma \mid \forall \sigma', t: \sigma \stackrel{t}{\not\to}_\tau \sigma' \,\}$

Maximal traces: $\mathcal{M}_\infty$ (finite or infinite)

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

Finite prefix traces: $\mathcal{T}_p$

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

Fixpoint form: $\mathcal{T}_p = \text{lfp}\, F_p$ where

$$F_p(X) = \mathcal{I} \cup \{\, \sigma_0 \stackrel{t_0}{\to} \cdots \stackrel{t_n}{\to} \sigma_{n+1} \mid \sigma_0 \stackrel{t_0}{\to} \cdots \stackrel{t_{n-1}}{\to} \sigma_n \in X \wedge \sigma_n \stackrel{t_n}{\to}_\tau \sigma_{n+1} \,\}$$

Abstraction: $\mathcal{T}_p = \alpha_{*\preceq}(\mathcal{M}_\infty)$

Fairness

Fairness conditions: avoid threads being denied to run

Given *enabled*$(\sigma, t) \stackrel{\text{def}}{\iff} \exists \sigma' \in \Sigma: \sigma \xrightarrow{t}_\tau \sigma'$,
an infinite trace $\sigma_0 \stackrel{t_0}{\to} \cdots \sigma_n \stackrel{t_n}{\to} \cdots$ is:

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

Proofs under fairness conditions given:

- the maximal traces $\mathcal{M}_\infty$ 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 $\mathcal{M}_\infty \cap F \subseteq X$ instead of $\mathcal{M}_\infty \subseteq X$

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

$\mathcal{M}_\infty \cap F \subseteq X$ reduces to $\alpha_{*\preceq}(\mathcal{M}_\infty \cap F) \subseteq \alpha_{*\preceq}(X)$

for all fairness conditions F, $\alpha_{*\preceq}(\mathcal{M}_\infty \cap F) = \alpha_{*\preceq}(\mathcal{M}_\infty) = \mathcal{T}_p$

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

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

Equational state semantics

State abstraction $\mathcal{R}$: as before

- $\mathcal{R} \stackrel{\text{def}}{=} \{\, \sigma \mid \exists \sigma_0 \stackrel{t_0}{\rightarrow} \cdots \sigma_n \colon \sigma_0 \in \mathcal{I}\ \forall i < n \colon \sigma_i \stackrel{t_i}{\rightarrow}_\tau \sigma_{i+1} \wedge \sigma = \sigma_n \,\}$
- $\mathcal{R} = \alpha_p(\mathcal{T}_p)$ where $\alpha_p(X) \stackrel{\text{def}}{=} \{\, \sigma \mid \exists \sigma_0 \stackrel{t_0}{\rightarrow} \cdots \sigma_n \in X \colon \sigma = \sigma_n \,\}$
- $\mathcal{R} = \mathrm{lfp}\, F_\mathcal{R}$ where $F_\mathcal{R}(X) = \mathcal{I} \cup \{\, \sigma \mid \exists \sigma' \in X, t \in \mathbb{T} \colon \sigma' \stackrel{t}{\rightarrow}_\tau \sigma \,\}$

Equational form: (without handling errors in Ω)

- for each $L \in \mathbb{T} \rightarrow \mathcal{L}$, a variable $\mathcal{X}_L$ with value in $\mathcal{E}$
- equations are derived from thread equations $eq(stat_t)$ as:[2]

$$\mathcal{X}_{L_1} = \bigcup_{t \in \mathbb{T}} \{\, F(\mathcal{X}_{L_2}, \ldots, X_{L_N}) \mid \exists (\mathcal{X}_{\ell_1} = F(\mathcal{X}_{\ell_2}, \ldots, \mathcal{X}_{\ell_N})) \in eq(stat_t) \colon \forall i \leq N \colon L_i(t) = \ell_i,\ \forall u \neq t \colon L_i(u) = L_1(u) \,\}$$

(join with $\cup$ equations updating a single thread)

[2] See lesson 02-B for the definition of $eq(stat)$.

Equational state semantics (example)

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

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**
$^{\ell 3}\ x \leftarrow x + 1$	$^{\ell 6}\ y \leftarrow y + 1$

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**
$^{\ell 2}$ **if** $x < y$ **then**	$^{\ell 5}$ **if** $y < 10$ **then**
$^{\ell 3}$ $x \leftarrow x + 1$	$^{\ell 6}$ $y \leftarrow y + 1$

(Simplified) equation system:

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

Equational state semantics (example)

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

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**
$^{\ell 3}$ $x \leftarrow x + 1$	$^{\ell 6}$ $y \leftarrow y + 1$

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 $\mathbb{T} \to \mathcal{L}$)
- 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)

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

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

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

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 $\mathcal{X}_\ell$ (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 $\mathsf{C}[\![\,stat\,]\!]$ 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 $\{X = 0, Y = 0\}\; \textbf{if } X = 0 \textbf{ then } Y \leftarrow 1 \{X = 0, Y = 1\}$

but not $\{X = 0, Y = 0\}\; X \leftarrow 1 \,||\, \textbf{if } X = 0 \textbf{ then } Y \leftarrow 1\; \{\mathit{false}\}$

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

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": $R, G \vdash \{P\}\ \mathit{stat}\ \{Q\}$

- 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:

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

Rely-guarantee example

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

checking t_1

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

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

checking t_2

$x \le y$

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

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

Rely-guarantee example

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

checking t_1

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

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

checking t_2

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

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

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 $t \in \mathbb{T}$

- add auxiliary variables $\mathbb{V}_t \stackrel{\text{def}}{=} \mathbb{V} \cup \{ pc_u \mid u \in \mathbb{T}, u \neq t \}$
- 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 $\Sigma \stackrel{\text{def}}{=} ((\mathbb{T} \to \mathcal{L}) \times \mathcal{E}) \cup \Omega$)
- projection: $\pi_t(L, \rho) \stackrel{\text{def}}{=} (L(t), \rho[\forall u \neq t: pc_u \mapsto L(u)])$
  extended naturally to $\pi_t : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma_t)$

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

Note: π_t is a bijection, no information is lost

Interferences

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

$A \in \mathbb{T} \to \mathcal{P}(\Sigma \times \Sigma)$

$A(t) \stackrel{\text{def}}{=} \alpha^{itf}(\mathcal{T}_p)(t)$

where $\alpha^{itf}(X)(t) \stackrel{\text{def}}{=} \{\, (\sigma, \sigma') \,|\, \exists \cdots \sigma \stackrel{t}{\to} \sigma' \cdots \in X \,\}$

subset of the transition system τ

- spawned by t and
- actually observed in some execution trace
  (recall that $\mathcal{T}_p$ is the prefix trace abstraction)

Fixpoint form

Local state fixpoint:

- we express $\mathcal{RI}(t)$ as a function of A and thread $t \in \mathbb{T}$:

$$\mathcal{RI}(t) = \text{lfp}\, R_t(A) \text{ where}$$

$$R_t : (\mathbb{T} \to \mathcal{P}(\Sigma \times \Sigma)) \to \mathcal{P}(\Sigma_t) \to \mathcal{P}(\Sigma_t)$$

$$R_t(Y)(X) \stackrel{\text{def}}{=} \pi_t(\mathcal{I}) \cup$$
$$\{\, \pi_t(\sigma') \mid \exists \pi_t(\sigma) \in X \colon \sigma \xrightarrow{t}_\tau \sigma' \lor \exists u \neq t \colon (\sigma, \sigma') \in Y(u) \,\}$$

A state is reachable if it is initial,
or reachable by transitions from t or from the environment A.

R_t only looks into the syntax of thread t.
R_t is parameterized by the interferences from other threads Y.

Fixpoint form (cont.)

Interferences:

- we express $A(t)$ as a function of $\mathcal{R}I$ and thread $t \in \mathbb{T}$:

  $A(t) = B(\mathcal{R}I)(t)$ where

  $B : (\prod_{t \in \mathbb{T}} \{t\} \to \mathcal{P}(\Sigma_t)) \to \mathbb{T} \to \mathcal{P}(\Sigma \times \Sigma)$

  $B(Z)(t) \stackrel{\text{def}}{=} \{\, (\sigma, \sigma') \mid \pi_t(\sigma) \in Z(t) \wedge \sigma \xrightarrow{t}_\tau \sigma' \,\}$

  Collect transitions starting from reachable states.

  No fixpoint needed.

Fixpoint form (cont.)

Nested fixpoint characterization:

1. $\mathcal{RI}(t) = \text{lfp}\ R_t(A)$
2. $A(t) = B(\mathcal{RI})(t)$
3. mutual dependency between $\mathcal{RI}$ and A

Fixpoint form (cont.)

Nested fixpoint characterization:

1. $\mathcal{RI}(t) = \text{lfp}\, R_t(A)$
2. $A(t) = B(\mathcal{RI})(t)$
3. mutual dependency between $\mathcal{RI}$ and A
   $\Longrightarrow$ solved using a fixpoint:

   $\mathcal{RI} = \text{lfp}\, H$ where

   $H : (\prod_{t \in \mathbb{T}} \{t\} \to \mathcal{P}(\Sigma_t)) \to (\prod_{t \in \mathbb{T}} \{t\} \to \mathcal{P}(\Sigma_t))$

   $H(Z)(t) \stackrel{\text{def}}{=} \text{lfp}\, R_t(B(Z))$

Fixpoint form (cont.)

Constructive fixpoint form:

Use Kleene's iteration to construct fixpoints:

- $\mathcal{R}I = \text{lfp } H = \bigsqcup_{n \in \mathbb{N}} H^n(\lambda t.\emptyset)$
  in the pointwise powerset lattice $\prod_{t \in \mathbb{T}} \{t\} \to \mathcal{P}(\Sigma_t)$
- $H(Z)(t) = \text{lfp } R_t(B(Z)) = \bigcup_{n \in \mathbb{N}} (R_t(B(Z)))^n(\emptyset)$
  in the powerset lattice $\mathcal{P}(\Sigma_t)$
  (similar to the sequential semantics of thread t in isolation)

$\Longrightarrow$ nested iterations

Abstract rely-guarantee

Suggested algorithm: nested iterations with acceleration

once abstract domains for states and interferences are chosen

- start from $\mathcal{R}I_0^\sharp \stackrel{\text{def}}{=} A_0^\sharp \stackrel{\text{def}}{=} \lambda t.\bot^\sharp$
- while $A_n^\sharp$ is not stable
  - compute $\forall t \in \mathbb{T}$: $\mathcal{R}I_{n+1}^\sharp(t) \stackrel{\text{def}}{=} \text{lfp } R_t^\sharp(A_n^\sharp)$
    by iteration with widening $\triangledown$

    ($\simeq$ separate analysis of each thread)
  - compute $A_{n+1}^\sharp \stackrel{\text{def}}{=} A_n^\sharp \triangledown B^\sharp(\mathcal{R}I_{n+1}^\sharp)$
- when $A_n^\sharp = A_{n+1}^\sharp$, return $\mathcal{R}I_n^\sharp$

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

Flow-insensitive abstraction

Idea:

- reduce as much control information as possible
- but keep flow-sensitivity on each thread's control location

Local state abstraction: remove auxiliary variables

$$\alpha^{nf}_{\mathcal{R}} : \mathcal{P}(\Sigma_t) \to \mathcal{P}((\mathcal{L} \times \mathcal{E}) \cup \Omega)$$
$$\alpha^{nf}_{\mathcal{R}}(X) \stackrel{\text{def}}{=} \{ (\ell, \rho_{|\mathbb{V}}) \mid (\ell, \rho) \in X \} \cup (X \cap \Omega)$$

Interference abstraction: remove all control state

$$\alpha^{nf}_{A} : \mathcal{P}(\Sigma \times \Sigma) \to \mathcal{P}(\mathcal{E} \times \mathcal{E})$$
$$\alpha^{nf}_{A}(Y) \stackrel{\text{def}}{=} \{ (\rho, \rho') \mid \exists L, L' \in \mathbb{T} \to \mathcal{L}: ((L, \rho), (L', \rho')) \in Y \}$$

Flow-insensitive abstraction (cont.)

Flow-insensitive fixpoint semantics: (omitting errors Ω)

We apply $\alpha_{\mathcal{R}}^{nf}$ and α_A^{nf} to the nested fixpoint semantics.

$\mathcal{RI}^{nf} \stackrel{\text{def}}{=} \text{lfp}\, \lambda Z.\lambda t.\, \text{lfp}\, {R^{nf}}_t(B^{nf}(Z))$, where

$B^{nf}(Z)(t) \stackrel{\text{def}}{=} \{\, (\rho, \rho') \mid \exists \ell, \ell' \in \mathcal{L}\colon (\ell, \rho) \in Z(t) \wedge (\ell, \rho) \to_t (\ell', \rho') \,\}$

$R_t^{nf}(Y)(X) \stackrel{\text{def}}{=} R_t^{loc}(X) \cup A_t^{nf}(Y)(X)$

$R_t^{loc}(X) \stackrel{\text{def}}{=} \{(\ell_t^i, \lambda V.0)\} \cup \{\, (\ell', \rho') \mid \exists (\ell, \rho) \in X\colon (\ell, \rho) \to_t (\ell', \rho') \,\}$

$A_t^{nf}(Y)(X) \stackrel{\text{def}}{=} \{\, (\ell, \rho') \mid \exists \rho,\, u \neq t\colon (\ell, \rho) \in X \wedge (\rho, \rho') \in Y(u) \,\}$

where $\to_t$ is the transition relation for thread t alone: $\tau[stat_t]$

Cost/precision trade-off:

- less variables

  $\Longrightarrow$ subsequent numeric abstractions are more efficient
- sufficient to analyze our first example (p. 34)
- insufficient to analyze $x \leftarrow x + 1 \mid\mid x \leftarrow x + 1$

Non-relational interference abstraction

Idea: simplify further flow-insensitive interferences

- numeric relations are more costly than numeric sets
  $\Longrightarrow$ remove input sensitivity
- relational domains are more costly than non-relational
  $\Longrightarrow$ abstract the interference on each variable separately

Non-relational interference abstraction:

$\alpha_A^{nr} : \mathcal{P}(\mathcal{E} \times \mathcal{E}) \to (\mathbb{V} \to \mathcal{P}(\mathbb{R}))$

$\alpha_A^{nr}(Y) \stackrel{\text{def}}{=} \lambda V.\{\, x \in \mathbb{V} \,|\, \exists (\rho, \rho') \in Y\colon \rho(V) \neq x \wedge \rho'(V) = x \,\}$

(remember which variables are modified and their new values)

To apply interferences, we get, in the nested fixpoint form:

$A_t^{nr}(Y)(X) \stackrel{\text{def}}{=}$
$\{\, (\ell, \rho[V \mapsto v]) \,|\, (\ell, \rho) \in X, V \in \mathbb{V}, \exists u \neq t\colon v \in Y(u)(V) \,\}$

A note on unbounded threads

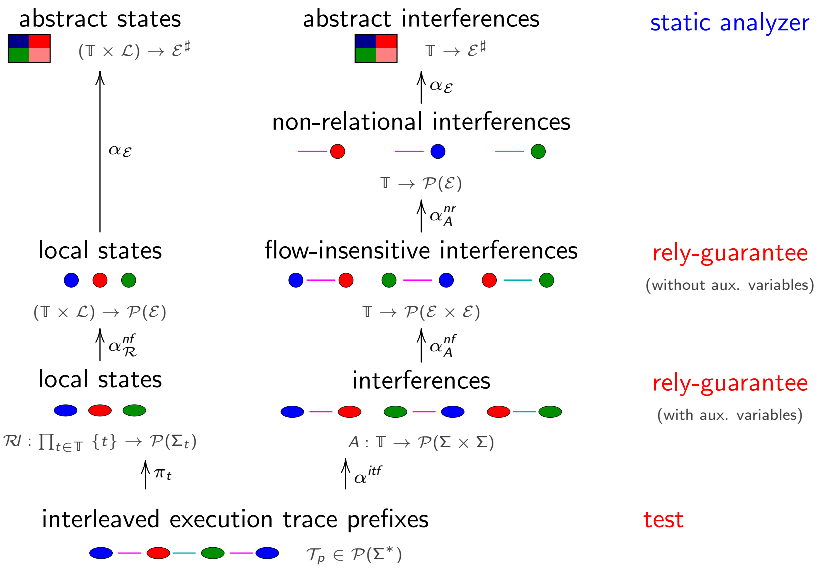
Extension: relax the finiteness constraint on $\mathbb{T}$

- there is still a finite syntactic set of threads $\mathbb{T}_s$
- some threads $\mathbb{T}_\infty \subseteq \mathbb{T}_s$ can have several instances
  (possibly an unbounded number)

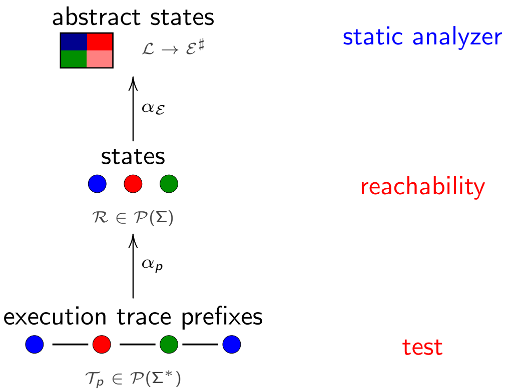
Flow-insensitive analysis:

- local state and interference domains have finite dimensions
  ($\mathcal{E}_t$ and $(\mathcal{L} \times \mathcal{E}) \times (\mathcal{L} \times \mathcal{E})$, as opposed to $\mathcal{E}$ and $\mathcal{E} \times \mathcal{E}$)
- all instances of a thread $t \in \mathbb{T}_s$ are isomorphic
  $\Longrightarrow$ iterate the analysis on the finite set $\mathbb{T}_s$ (instead of $\mathbb{T}$)
- we must handle self-interferences for threads in $\mathbb{T}_\infty$:
  $A_t^{nf}(Y)(X) \stackrel{\text{def}}{=}$
  $\{ (\ell, \rho') \mid \exists \rho,\ u{:}\ (u \neq t \vee t \in \mathbb{T}_\infty) \wedge (\ell, \rho) \in X \wedge (\rho, \rho') \in Y(u) \}$

From traces to thread-modular analyses



Compare with sequential analyses...



Construction of an interference-based analysis

Reminder: sequential analysis in denotational form

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

$$\mathsf{E}[\![\, X \,]\!]\,\rho \stackrel{\text{def}}{=} \langle\, \{\, \rho(X) \,\},\, \emptyset \,\rangle$$

$$\mathsf{E}[\![\, [c_1, c_2] \,]\!]\,\rho \stackrel{\text{def}}{=} \langle\, \{\, x \in \mathbb{R} \mid c_1 \leq x \leq c_2 \,\},\, \emptyset \,\rangle$$

$$\mathsf{E}[\![\, -e_1 \,]\!]\,\rho \stackrel{\text{def}}{=} \text{let } \langle\, V_1,\, O_1 \,\rangle = \mathsf{E}[\![\, e_1 \,]\!]\,\rho \text{ in } \langle\, \{\, -v_1 \mid v_1 \in V_1 \,\},\, O_1 \,\rangle$$

$$\mathsf{E}[\![\, e_1 \diamond_\omega e_2 \,]\!]\,\rho \stackrel{\text{def}}{=} \text{let } \forall i \in \{\, 1, 2 \,\}\text{: } \langle\, V_i,\, O_i \,\rangle = \mathsf{E}[\![\, e_i \,]\!]\,\rho \text{ in}$$
$$\langle\, \{\, v_1 \diamond v_2 \mid v_i \in V_i,\, \diamond \neq / \vee v_2 \neq 0 \,\},\, O_1 \cup O_2 \cup \{\, \omega \text{ if } \diamond = / \wedge 0 \in V_2 \,\} \,\rangle$$

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

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

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

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

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

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

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

Denotational semantics with interferences

Interferences in $\mathbb{I} \stackrel{\text{def}}{=} \mathbb{T} \times \mathbb{V} \times \mathbb{R}$

$\langle\, t,\, X,\, v \,\rangle$ 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 $I \subseteq \mathbb{I}$.

Expressions with interference: for thread t

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

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

  ...

Denotational semantics with interferences (cont.)

Statements with interference: for thread t

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

- 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 $\mathcal{P}(\mathcal{E}) \times \mathcal{P}(\Omega) \times \mathcal{P}(\mathbb{I})$)

Denotational semantics with interferences (cont.)

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

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

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

- each thread analysis starts in an initial environment set $\mathcal{E}_0 \stackrel{\text{def}}{=} \{\, \lambda V.0 \,\}$
- $[X]_{\Omega,\mathbb{I}}$ projects $X \in \mathcal{P}(\mathcal{E}) \times \mathcal{P}(\Omega) \times \mathcal{P}(\mathbb{I})$ on $\mathcal{P}(\Omega) \times \mathcal{P}(\mathbb{I})$ and interferences and errors from all threads are joined (the output environments are ignored)
- $\mathsf{P}[\![\, prog \,]\!]$ 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**
$^{\ell 2}$ **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 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**
$^{\ell 2}$ **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 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**
$^{\ell 2}$ **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 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 $\mathbb{I}^\sharp$

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

Abstract semantics with interferences $\mathsf{C}^\sharp_t[\![s]\!]$

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

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

- 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 $\langle R^{\sharp\prime}, O' \rangle = \mathsf{C}^\sharp[\![e]\!] \langle R^\sharp, O \rangle$
- enrich $I^\sharp\langle t, X \rangle$ with $get(X, R^{\sharp\prime})$

Static analysis with interferences

Abstract analysis

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

- 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

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

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

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

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

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

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

Path-based concrete semantics of sequential programs

Join-over-all-path semantics

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

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

Semantic equivalence

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

(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

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

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

Note: sound but not complete

Weakly consistent memories

Issues with weak consistency

program written

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

(simplified Dekker mutual exclusion algorithm)

S_1 and S_2 cannot execute simultaneously.

Issues with weak consistency

program written

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

$\longrightarrow$

program executed

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

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

$\longrightarrow$

"optimized" program	
$Y \leftarrow 42;$	
$R_1 \leftarrow X;$	$R_2 \leftarrow Y;$
$Y \leftarrow R_1$	$X \leftarrow R_2$

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

We should not have $R_1 = 42$.

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

$r_1 \leftarrow X+1$	$r_2 \leftarrow X+1$
$X \leftarrow r_1$	$X \leftarrow r_2$

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

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

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

$$
\begin{array}{lll}
\mathit{stat} & ::= & \mathbf{lock}(m) \\
 & | & \mathbf{unlock}(m)
\end{array}
$$

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

Scheduling

- mutexes ensure mutual exclusion

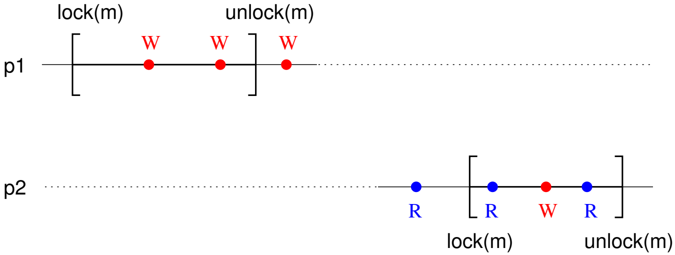
  a each time, each mutex can be locked by a single thread

- mutexes enforce memory consistency and atomicity

  no optimization across **lock** and **unlock** instructions

  memory caches and buffer are flushed

Mutual exclusion



Interleaving semantics $P_*[\![\textit{prog}]\!]$:

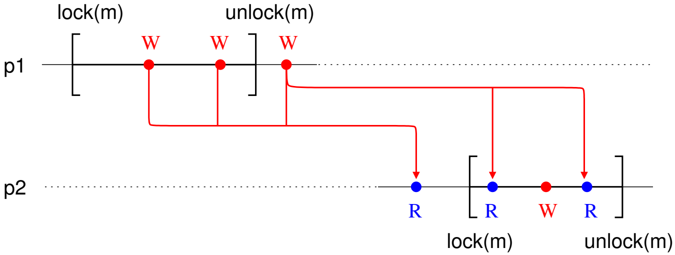
restrict interleavings of control paths

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

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

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

Mutual exclusion

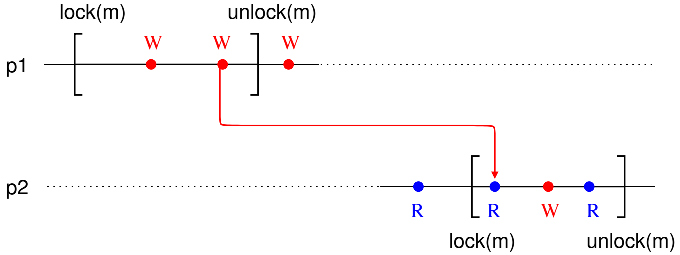


Data-race effects

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

- $C_t[\![X \leftarrow e]\!]\langle \rho, M, I \rangle$ adds $\{ \langle t, M, X, v \rangle \mid v \in E_t[\![X]\!]\langle \rho, M, I \rangle \}$ to I
- $E_t[\![X]\!]\langle \rho, M, I \rangle =$ $\{ \rho(X) \} \cup \{ v \mid \langle t', M', X, v \rangle \in I,\ t \neq t',\ M \cap M' = \emptyset \}$
- 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)
- $C_t[\![\,\mathbf{unlock}(m)\,]\!]\langle\,\rho,\ M,\ I\,\rangle$ stores $\rho(X)$ into I
- $C_t[\![\,\mathbf{lock}(m)\,]\!]\langle\,\rho,\ M,\ I\,\rangle$ 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, $Y \in [0, 10]$

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