Relational interprocedural analysis for concurrent programs

Bertrand Jeannet

INRIA Rhône-Alpes

December 5, 2008

Outline

Analysis of concurrent and recursive programs

Why ?

- Verification of SystemC/TLM model
- Viewed as programs with
  - concurrency (static multithreading)
  - interacting threads specified with procedures

Analysis of concurrent and recursive programs

Why ?

- Verification of SystemC/TLM model
- Viewed as programs with
  - concurrency (static multithreading)
  - interacting threads specified with procedures

Why not ?

- Strong undecidability result

Sequential versus concurrent program analysis

Analysis of concurrent programs without recursion

- Exact analysis computable for Boolean programs

Interprocedural analysis of sequential programs

- Well-understood [CC77, SP81, KS92]
- Exact analysis computable for Boolean programs
  - Not only w.r.t. properties/invariants
  - But also w.r.t. full call-stacks

Interprocedural analysis of concurrent programs ?

- Active research, several and incomparable solutions
- Exact analysis **not computable** even for Boolean programs ([Ram00]: reduction to the Post problem)

Challenge: combination of recursion and concurrency I

Sequential recursive programs

1. Computing the denotational semantics of procedures
   - block (procedure) $P_i \leftrightarrow$ predicate transformer $\phi_i : \mathcal{P} \rightarrow \mathcal{P}$
   - The ϕ_i's are expressed in function of each other
   - Fixpoint equation in the domain $\mathcal{P} \rightarrow \mathcal{P}$
2. Propagating reachability information
   - Propagation of input predicate using the ϕ_i's for procedure calls (in the domain $\mathcal{P}$)

The two steps can be combined in one single step (predicate transformers or summaries specialized on reachable input)

Concurrent single-procedure programs

- Reduces to a sequential program by interleaved product of control-flow graphs.

Challenge: combination of recursion and concurrency II

Requirement for the combination

1. Model accurately procedure call and return semantics in each thread
2. Take into account modifications of global variables made by the other threads.

Evaluation criteria

- static or dynamic thread creation
- communications between threads ?
  - with `parbegin` construct, no communication *during* thread execution
  - with shared global variables, any synchronisation mechanism
- local variables in procedures ?

  less easier than global variables, because of their temporary lifetime

Thread-modular model-checking [FQ03]

Principle

- To each thread t:
  - $R(t, g, l)$: true if global and local store is reachable;
  - $G(t, g, g')$: true if a step of t can go from g to g'.
- Inference rules

$$\frac{R(t,g,l) \quad G(e,g,g') \quad t \neq e}{R(t,g',l)} \qquad \frac{R(t,g,l) \quad \mathcal{T}(t,g,l,g',l')}{R(t,g',l') \quad G(t,g,g')}$$

Generalization to recursive programs: explicit stack for each thread

Pros/Cons

+ Efficiency (the initial goal)
− Termination non-guaranteed, unless further abstraction
− Precision: abstracts the local stores, ignores the order of steps performed by the environment

Other approaches I

Identifying transactions [QRR04]

- Notion of transaction and transaction boundaries
- Boundaries may not match procedure calls and returns
- Subclass for which termination is guaranteed

Analysis under a context bound

- Considers only executions with a bounded number of context switches
  - unbounded number of steps possible between switches
- Allows reduction to sequential analysis
- More reminiscent of symbolic execution
  - discovers bugs, does not prove properties

Other approaches II

Alternative representations of the state-space

- Regular model-checking techniques (Bouajjani, Esparza, Touili)
  - Concurrent programs: set of PDA
  - Various abstractions applied to their stacks
- Rewriting techniques: SPADE tool (Sighireanu, Touili)
  - Represents states with terms and uses rewriting techniques (Timbuk tree automata library of Thomas Genet)
  - Handles dynamic thread creation

Not easily combined with infinite data abstraction, like convex polyhedra.

Our approach

State-space of 2-threads programs

$$S = \underbrace{\text{GEnv}}_{\text{global store}} \times \underbrace{(K^1 \times \text{LEnv}^1)^+}_{\text{stack of thread 1}} \times \underbrace{(K^2 \times \text{LEnv}^2)^+}_{\text{stack of thread 2}}$$

We instrument the standard semantics

properties on executions $\Longrightarrow$ properties on reachable states

$$S_i = (K^1 \times \text{Env}^1)^+ \times (K^2 \times \text{Env}^2)^+$$

We abstract stacks into sets

$$\wp\Big((K^1 \times \text{Env}^1)^+ \times (K^2 \times \text{Env}^2)^+ \Big)$$

$$\Longleftrightarrow$$

$$\underbrace{\wp(K^1 \times K^2 \times \text{Env}^1 \times \text{Env}^2)}_{\text{pairs of stack tops}} \times \underbrace{\wp(K^1 \times \text{Env}^1)}_{\text{stack tails 1}} \times \underbrace{\wp(K^2 \times \text{Env}^2)}_{\text{stack tails 2}}$$

Defines the analysis method

Outline

Program example: synchronisation barriers

```
var go : bool,
    counter,p0,p1 : int,

initial counter==0 and go;

proc barrier(lgo:bool)
    returns (nlgo:bool)
begin
  lgo = not lgo;
  counter = counter+1;
  if counter==2 then
    counter=0; go = lgo;
  else
    assume(lgo==go);
  endif;
  nlgo = lgo;
end
```

```
thread T0:
var lgo0:bool;
begin
  p0 = 0; lgo0 = true;
  while p0<=5 do
    lgo0 = barrier(lgo0);
    p0 = p0 + 1;
  done;
end

thread T1:
var lgo1:bool;
begin
  p1 = 0; lgo1 = true;
  while p1<=10 do
    lgo1 = barrier(lgo1);
    p1 = p1 + 1;
  done;
  fail;
end
```

Program model I

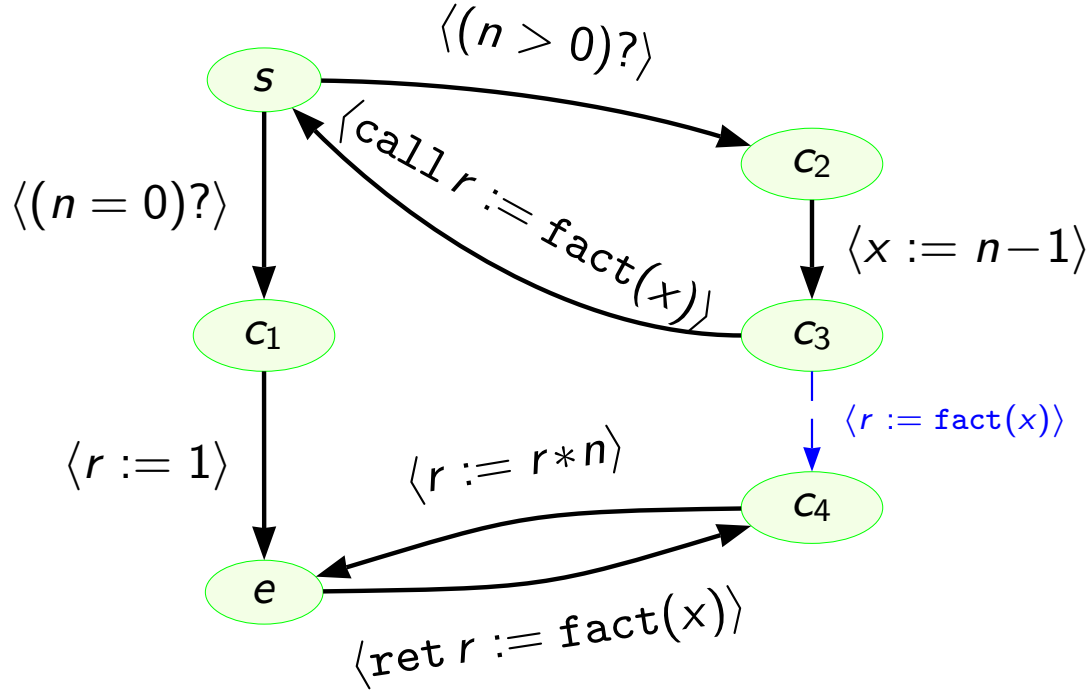
Programs composed of

- global variables $\mathbf{g}$
- procedures
  - $\mathbf{fpi}, \mathbf{fpo}$: formal input/output parameters of P
  - $\mathbf{l}$: local variables of P
- static threads communicating via global variables
  - P_0^t: main procedure of thread t

Program model II

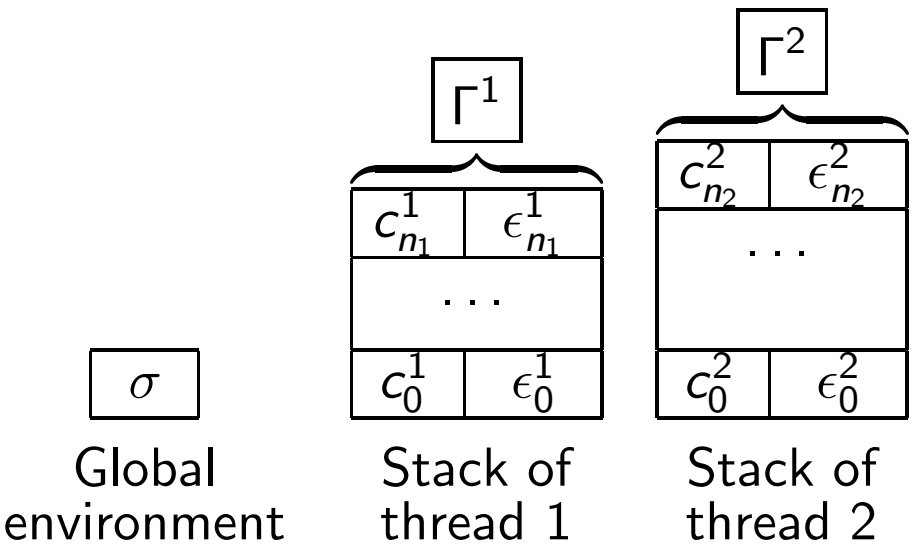
Global control flow graph G

- control points K
- s_j, e_j: start and exit points of procedure P_j
- Edges labeled with instructions



Program semantics I

Program state



with $\sigma \in \text{GEnv} = \text{GVar} \to \text{Value}$: global environments

$\epsilon \in \text{LEnv} = \text{LVar} \to \text{Value}$: local environments

Program semantics II

Operational Semantics

$$\frac{I(c, c') = \langle R \rangle \quad R(\sigma, \epsilon, \sigma', \epsilon')}{\langle \sigma,\ \Gamma \cdot \langle c, \epsilon \rangle,\ \Gamma^2 \rangle \rightarrow \langle \sigma',\ \Gamma \cdot \langle c', \epsilon' \rangle,\ \Gamma^2 \rangle} \quad \text{(Intra)}$$

$$\frac{I(c, s_j) = \langle \texttt{call}\, \mathbf{y} := P_j(\mathbf{x}) \rangle \quad R^{+}_{\mathbf{y}:=P_j(\mathbf{x})}(\sigma, \epsilon, \epsilon_j)}{\langle \sigma,\ \Gamma \cdot \langle c, \epsilon \rangle,\ \Gamma^2 \rangle \rightarrow \langle \sigma,\ \Gamma \cdot \langle c, \epsilon \rangle \cdot \langle s_j, \epsilon_j \rangle,\ \Gamma^2 \rangle} \quad \text{(Call)}$$

$$\frac{I(e_j, c) = \langle \texttt{ret}\, \mathbf{y} := P_j(\mathbf{x}) \rangle \quad R^{-}_{\mathbf{y}:=P_j(\mathbf{x})}(\sigma, \epsilon, \epsilon_j, \sigma', \epsilon')}{\langle \sigma,\ \Gamma \cdot \langle \mathit{call}(c), \epsilon \rangle \cdot \langle e_j, \epsilon_j \rangle,\ \Gamma^2 \rangle \rightarrow \langle \sigma',\ \Gamma \cdot \langle c, \epsilon' \rangle,\ \Gamma^2 \rangle} \quad \text{(Ret)}$$

Outline

Instrumenting the semantics I

Principle

- Global variables pushed on stacks
- Formal input parameters get a frozen copy
- New thread environments $\epsilon(\mathbf{g}_0, \mathbf{fpi}_0, \mathbf{g}, \mathbf{l})$
  - $\mathbf{g}_0, \mathbf{fpi}_0$: values of global and parameters at start point
  - $\mathbf{g}, \mathbf{l}$: values of global and local variables at current point

  $\mathbf{g}_0, \mathbf{fpi}_0$ tags environments with (an abstraction of) the call-context

State-space $S_i = (K^1 \times \mathrm{Env}^1)^+ \times (K^2 \times \mathrm{Env}^2)^+$

Threads must agree on the current value of global variables

Instrumenting the semantics II

Instrumented semantics: thread in isolation

$$\frac{I(c, c') = \langle R \rangle \quad R(\epsilon, \epsilon') \wedge \epsilon(\mathbf{g}_0, \mathbf{fpi}_0) = \epsilon'(\mathbf{g}_0, \mathbf{fpi}_0)}{\Gamma \cdot \langle c, \epsilon \rangle \rightarrow_i^t \Gamma \cdot \langle c', \epsilon' \rangle} \quad \text{(IntraF)}$$

$$\frac{I(c, s_j) = \langle \texttt{call}\, \mathbf{y} := P_j(\mathbf{x}) \rangle \quad R^+_{\mathbf{y}:=P_j(\mathbf{x})}(\epsilon, \epsilon_j)}{\Gamma \cdot \langle c, \epsilon \rangle \rightarrow_i^t \Gamma \cdot \langle c, \epsilon \rangle \cdot \langle s_j, \epsilon_j \rangle} \quad \text{(CallF)}$$

$$\frac{I(e_j, c) = \langle \texttt{ret}\, \mathbf{y} := P_j(\mathbf{x}) \rangle \quad R^-_{\mathbf{y}:=P_j(\mathbf{x})}(\epsilon, \epsilon_j, \epsilon')}{\Gamma \cdot \langle call(c), \epsilon \rangle \cdot \langle e_j, \epsilon_j \rangle \rightarrow_i^t \Gamma \cdot \langle c, \epsilon' \rangle} \quad \text{(RetF)}$$

Instrumenting the semantics III

Instrumented semantics: full program

Notifying update of global variables to other threads

$$\frac{\Gamma_1 \rightarrow_i^1 \Gamma_1' \qquad \Gamma_1' = \Gamma_1'' \cdot \langle c_1', \epsilon_1' \rangle \qquad \epsilon_2' = \epsilon_2[\mathbf{g} \mapsto \epsilon_1'(\mathbf{g})]}{\langle \Gamma_1,\ \Gamma_2 \cdot \langle c_2, \epsilon_2 \rangle \rangle \rightarrow_i \langle \Gamma_1',\ \Gamma_2 \cdot \langle c_2, \epsilon_2' \rangle \rangle} \quad \text{(Conc1F)}$$

$$\ldots \quad \text{(Conc2F)}$$

Properties of reachable stacks in instrumented semantics

Definition

A stack $\Gamma = \langle c_0, \epsilon_0 \rangle \dots \langle c_n, \epsilon_n \rangle \in \mathrm{Act}^+$ is *well-formed* if:

(i) c_i is a call site for the procedure $Proc(c_{i+1}) = P_j$

(ii) equality between actual and formal input parameters holds: $\epsilon_i(\mathbf{g}, \mathbf{x}) = \epsilon_{i+1}(\mathbf{g}_0, \mathbf{fpi}_0^j)$.

A state $\langle \Gamma^1, \Gamma^2 \rangle \in S_i$ is *well-formed* if Γ^1 and Γ^2 are well-formed

Theorem

Reachable states are well-formed

Strong condition for an activation record to lie below another activation record in stacks

Outline

Relational interprocedural analysis of sequential prog. I

formalized as a stack abstraction on instrumented semantics [JS04]

Galois connection $S_i = \wp(\mathrm{Act}^+) \underset{\alpha_f}{\overset{\gamma_f}{\leftrightarrows}} \wp(\mathrm{Act}) \times \wp(\mathrm{Act})$ with

$$\alpha_f \ : \ \{\Gamma = r_0 \dots r_n\} \quad \mapsto \quad \left\langle \begin{array}{c} \mathsf{hd}(\Gamma), \\ \mathsf{tl}(\Gamma) \end{array} \right\rangle = \left\langle \begin{array}{c} \{r_n\}, \\ \{r_i \mid 0 \leq i < n\} \end{array} \right\rangle$$

$$\gamma_f \ : \ \langle Y_{\mathsf{hd}}, Y_{\mathsf{tl}} \rangle \quad \mapsto \quad \left\{ \Gamma = r_0 \dots r_n \,\middle|\, \begin{array}{l} r_n \in Y_{\mathsf{hd}} \\ \forall 0 \leq i < n : r_i \in Y_{\mathsf{tl}} \\ \Gamma \text{ is a well-formed stack} \end{array} \right\}$$

Relational interprocedural analysis of sequential prog. II

Induced abstract semantics: for procedure return

$$instr(e_j, ret(c)) = \langle \texttt{ret}\ \mathbf{y} := P_j(\mathbf{x}) \rangle$$

Relational interprocedural analysis of sequential prog. II

Induced abstract semantics: for procedure return

$$instr(e_j, ret(c)) = \langle \texttt{ret}\ \mathbf{y} := P_j(\mathbf{x}) \rangle$$

$$\underset{\text{(tail)}}{Y_{\mathsf{tl}}[c](\mathbf{g}_0, \mathbf{fpi}_0,\ \mathbf{g}, \mathbf{l})} \qquad \underset{\text{(top)}}{Y_{\mathsf{hd}}[e_j](\mathbf{g}_0^j, \mathbf{fpi}_0^j,\ \mathbf{g}', \mathbf{l}')}$$

Relational interprocedural analysis of sequential prog. II

Induced abstract semantics: for procedure return

$$instr(e_j, ret(c)) = \langle \texttt{ret}\ \mathbf{y} := P_j(\mathbf{x}) \rangle$$

$$\underbrace{Y_{\mathsf{tl}}[c](\mathbf{g}_0, \mathbf{fpi}_0,\ \mathbf{g}, \mathbf{l})}_{\text{(tail)}} \qquad \underbrace{Y_{\mathsf{hd}}[e_j](\mathbf{g}_0^j, \mathbf{fpi}_0^j,\ \mathbf{g}', \mathbf{l}')}_{\text{(top)}}$$

$$(\mathbf{g}, \mathbf{x}) = (\mathbf{g}_0^j, \mathbf{fpi}_0^j)$$
$$\text{(well-formedness condition)}$$

Relational interprocedural analysis of sequential prog. II

Induced abstract semantics: for procedure return

$$instr(e_j, ret(c)) = \langle \texttt{ret}\ \mathbf{y} := P_j(\mathbf{x}) \rangle$$

$$Y_{\mathsf{tl}}[c](\mathbf{g}_0, \mathbf{fpi}_0,\ \mathbf{g}, \mathbf{l}) \qquad Y_{\mathsf{hd}}[e_j](\mathbf{g}_0^j, \mathbf{fpi}_0^j,\ \mathbf{g}', \mathbf{l}')$$

(tail) (top)

$$(\mathbf{g}, \mathbf{x}) = (\mathbf{g}_0^j, \mathbf{fpi}_0^j)$$

(well-formedness condition)

$$R^-_{\mathbf{y}:=P_j(\mathbf{x})}(\mathbf{g}, \mathbf{l}, \mathbf{g}', \mathbf{l}',\ \mathbf{g}'', \mathbf{l}'')$$

(output parameter passing)

Relational interprocedural analysis of sequential prog. II

Induced abstract semantics: for procedure return

$$instr(e_j, ret(c)) = \langle \texttt{ret}\, \mathbf{y} := P_j(\mathbf{x}) \rangle$$

$$\frac{\underset{\text{(tail)}}{Y_{\mathsf{tl}}[c](\mathbf{g}_0, \mathbf{fpi}_0,\ \mathbf{g}, \mathbf{l})} \qquad \underset{\text{(top)}}{Y_{\mathsf{hd}}[e_j](\mathbf{g}_0^j, \mathbf{fpi}_0^j,\ \mathbf{g}', \mathbf{l}')}}{\begin{array}{c}(\mathbf{g}, \mathbf{x}) = (\mathbf{g}_0^j, \mathbf{fpi}_0^j) \\ \text{(well-formedness condition)} \\ R^{-}_{\mathbf{y}:=P_j(\mathbf{x})}(\mathbf{g}, \mathbf{l}, \mathbf{g}', \mathbf{l}',\ \mathbf{g}'', \mathbf{l}'') \\ \text{(output parameter passing)}\end{array}}$$

$$\overline{Y_{\mathsf{hd}}[ret(c)](\mathbf{g}_0^j, \mathbf{fpi}_0^j,\ \mathbf{g}'', \mathbf{l}'')}$$

Relational interprocedural analysis of sequential prog. II

Induced abstract semantics: for procedure return

$$instr(e_j, ret(c)) = \langle \texttt{ret}\, \mathbf{y} := P_j(\mathbf{x}) \rangle$$

$$\frac{Y_{\mathsf{tl}}[c](\mathbf{g}_0, \mathbf{fpi}_0,\ \mathbf{g}, \mathbf{l}) \qquad Y_{\mathsf{hd}}[e_j](\mathbf{g}_0^j, \mathbf{fpi}_0^j,\ \mathbf{g}', \mathbf{l}')}{\begin{array}{c}(\text{tail}) \qquad\qquad (\text{top}) \\ (\mathbf{g}, \mathbf{x}) = (\mathbf{g}_0^j, \mathbf{fpi}_0^j) \\ (\text{well-formedness condition}) \\ R^-_{\mathbf{y}:=P_j(\mathbf{x})}(\mathbf{g}, \mathbf{l}, \mathbf{g}', \mathbf{l}',\ \mathbf{g}'', \mathbf{l}'') \\ (\text{output parameter passing}) \\ \hline Y_{\mathsf{hd}}[ret(c)](\mathbf{g}_0^j, \mathbf{fpi}_0^j,\ \mathbf{g}'', \mathbf{l}'')\end{array}}$$

Theorem (Optimality)

The abstract semantics preserves stack tops (but not full stacks)

Analysis of non-recursive concurrent prog. I

State-space: $S = \mathrm{GEnv} \times (K^1 \times \mathrm{LEnv}^1) \times (K^2 \times \mathrm{LEnv}^2)$

Principle

- One observes $\wp(S) \simeq K^1 \times K^2 \longrightarrow \wp(\mathrm{GEnv} \times \mathrm{LEnv}^1 \times \mathrm{LEnv}^2)$
- If needed, one abstracts it with $K^1 \times K^2 \longrightarrow \mathrm{Env}^\sharp$
  - $\mathrm{Env}^\sharp$: abstract environments (octagons, convex polyhedra...)

Ability to relate the local environments of different threads is fundamental !

Analysis of non-recursive concurrent prog. II

Example

- Two predicates $Y^1(g, l^1) = (g = l^1)$ and $Y^2(g, l^2) = (g = l^2 - 1)$
- Thread 1 executes $g := g + l^1$
- Effect on Y^1: $(g = 2l^1)$
- Effect on Y^2 ?
  - build $Y = Y^1 \wedge Y^2 = (g = l^1 \wedge l^1 = l^2 - 1)$
  - compute the effect of the instruction on Y
  - forget the variable l^1

$\Longrightarrow \ (g = 2l^2 - 2)$

Need for relate at least temporarily local environments of different threads

Analysis of non-recursive concurrent prog. III

Example

```
thread T1:          thread T2:
var i:int;          var j:int;
begin               begin
  i = 0;              j = 0;
  while i<=10 do      while j<=11 do
    sync a;             sync a;
    i = i+1;            j = j+1;
  done                done
end                 end
```

To establish that the thread 2 does not terminate we need to infer the invariant $i = j$ just after the synchronisation

Need for relate as long as possible local environments of different threads

Concurrent stack abstraction I

Galois connection

$$\wp(\mathsf{Act}^+ \times \mathsf{Act}^+) \underset{\alpha_c}{\overset{\gamma_c}{\leftrightarrows}} \wp(\mathsf{Act} \times \mathsf{Act}) \times \wp(\mathsf{Act}) \times \wp(\mathsf{Act}) \text{ with}$$

$$\alpha_c\Big(\big\{\langle \overbrace{r_0^1 \ldots r_{n_1}^1}^{\Gamma^1}, \overbrace{r_0^2 \ldots r_{n_2}^2}^{\Gamma^2}\rangle\big\}\Big) = \left\langle \begin{array}{c} \mathsf{hd}(\Gamma_1, \Gamma_2), \\ \mathsf{tl}(\Gamma_1), \\ \mathsf{tl}(\Gamma_2) \end{array} \right\rangle = \left\langle \begin{array}{c} \big\{\langle r_{n_1}^1, r_{n_2}^2 \rangle\big\}, \\ \big\{r_{i_1}^1 \mid 0 \leq i_1 < n_1\big\}, \\ \big\{r_{i_2}^2 \mid 0 \leq i_2 < n_2\big\} \end{array} \right\rangle$$

Concurrent stack abstraction I

Galois connection

$$\wp(\mathsf{Act}^+ \times \mathsf{Act}^+) \underset{\alpha_c}{\overset{\gamma_c}{\leftrightarrows}} \wp(\mathsf{Act} \times \mathsf{Act}) \times \wp(\mathsf{Act}) \times \wp(\mathsf{Act}) \text{ with}$$

$$\alpha_c\Big(\big\{\langle \overbrace{r_0^1 \ldots r_{n_1}^1}^{\Gamma^1}, \overbrace{r_0^2 \ldots r_{n_2}^2}^{\Gamma^2}\rangle\big\}\Big) = \left\langle \begin{array}{c} \mathsf{hd}(\Gamma_1, \Gamma_2), \\ \mathsf{tl}(\Gamma_1), \\ \mathsf{tl}(\Gamma_2) \end{array} \right\rangle = \left\langle \begin{array}{c} \big\{\langle r_{n_1}^1, r_{n_2}^2\rangle\big\}, \\ \big\{r_{i_1}^1 \mid 0 \le i_1 < n_1\big\}, \\ \big\{r_{i_2}^2 \mid 0 \le i_2 < n_2\big\} \end{array} \right\rangle$$

$$\gamma_c\big(\langle Y_{\mathsf{hd}}, Y_{\mathsf{tl}}^1, Y_{\mathsf{tl}}^2\rangle\big) =$$

$$\left\{ \langle r_0^1 \ldots r_{n_1}^1, r_0^2 \ldots r_{n_2}^2\rangle \;\middle|\; \begin{array}{c} \langle r_{n_1}^1, r_{n_2}^2\rangle \in Y_{\mathsf{hd}} \;\wedge\; \epsilon_{n_1}^1(\mathbf{g}) = \epsilon_{n_2}^2(\mathbf{g}) \\ \forall 0 \le i_1 < n_1 \;:\; r_{i_1}^1 \in Y_{\mathsf{tl}}^1 \\ \forall 0 \le i_2 < n_2 \;:\; r_{i_2}^2 \in Y_{\mathsf{tl}}^2 \\ r_0^1 \ldots r_{n_1}^1 \text{ and } r_0^2 \ldots r_{n_2}^2 \text{ are well-formed stacks} \end{array} \right\}$$

Concurrent stack abstraction I

Galois connection

$$\wp(\mathsf{Act}^+ \times \mathsf{Act}^+) \underset{\alpha_c}{\overset{\gamma_c}{\leftrightarrows}} \wp(\mathsf{Act} \times \mathsf{Act}) \times \wp(\mathsf{Act}) \times \wp(\mathsf{Act}) \text{ with}$$

$$\alpha_c\left(\left\{\langle \overbrace{r_0^1 \ldots r_{n_1}^1}^{\Gamma^1}, \overbrace{r_0^2 \ldots r_{n_2}^2}^{\Gamma^2} \rangle\right\}\right) = \left\langle \begin{array}{c} \mathsf{hd}(\Gamma_1, \Gamma_2), \\ \mathsf{tl}(\Gamma_1), \\ \mathsf{tl}(\Gamma_2) \end{array} \right\rangle = \left\langle \begin{array}{c} \left\{\langle r_{n_1}^1, r_{n_2}^2 \rangle\right\}, \\ \left\{r_{i_1}^1 \mid 0 \leq i_1 < n_1\right\}, \\ \left\{r_{i_2}^2 \mid 0 \leq i_2 < n_2\right\} \end{array} \right\rangle$$

$$\gamma_c(\langle Y_{\mathsf{hd}}, Y_{\mathsf{tl}}^1, Y_{\mathsf{tl}}^2 \rangle) =$$

$$\left\{ \langle r_0^1 \ldots r_{n_1}^1, r_0^2 \ldots r_{n_2}^2 \rangle \;\middle|\; \begin{array}{c} \langle r_{n_1}^1, r_{n_2}^2 \rangle \in Y_{\mathsf{hd}} \;\wedge\; \epsilon_{n_1}^1(\mathbf{g}) = \epsilon_{n_2}^2(\mathbf{g}) \\ \forall 0 \leq i_1 < n_1 \;:\; r_{i_1}^1 \in Y_{\mathsf{tl}}^1 \\ \forall 0 \leq i_2 < n_2 \;:\; r_{i_2}^2 \in Y_{\mathsf{tl}}^2 \\ r_0^1 \ldots r_{n_1}^1 \text{ and } r_0^2 \ldots r_{n_2}^2 \text{ are well-formed stacks} \end{array} \right\}$$

Abstract semantics: induced mechanically by abstracting the instrumented semantics with the Galois connection

Concurrent stack abstraction II

Predicate version

$$A_c \simeq \left(K^1 \times K^2 \to \wp(\mathsf{Env}^1 \times \mathsf{Env}^2)\right) \times \left(K^1 \to \wp(\mathsf{Env}^1)\right) \times \left(K^2 \to \wp(\mathsf{Env}^2)\right)$$

An abstract value: $\langle Y_{\mathsf{hd}}, Y^1_{\mathsf{tl}}, Y^2_{\mathsf{tl}} \rangle$ with

- $Y_{\mathsf{hd}}[c^1, c^2](\mathbf{g}^1_0, \mathbf{fpi}^1_0,\ \mathbf{g}^2_0, \mathbf{fpi}^2_0,\ \mathbf{g},\ \mathbf{l}^1, \mathbf{l}^2)$
  Top environments directly related
- $Y^1_{\mathsf{tl}}[c^1](\mathbf{g}^1_0, \mathbf{fpi}^1_0,\ \mathbf{g}, \mathbf{l}^1)$

  $Y^2_{\mathsf{tl}}[c^2](\mathbf{g}^2_0, \mathbf{fpi}^2_0,\ \mathbf{g}, \mathbf{l}^2)$

  Tail environments indirectly related via global variables

Abstract postcondition relies on

1. conjunctions and disjunctions
2. equality constraints between variables/dimensions
3. existential quantification
4. relations induced by intraprocedural instructions $\langle R \rangle$

Concurrent stack abstraction III

Induced semantics for procedure return (in thread 1)

$$instr(e_j, ret(c)) = \langle \texttt{ret}\ \mathbf{y} := P_j(\mathbf{x}) \rangle$$

Concurrent stack abstraction III

Induced semantics for procedure return (in thread 1)

$$instr(e_j, ret(c)) = \langle \texttt{ret}\ \mathbf{y} := P_j(\mathbf{x}) \rangle$$

$$Y^1_{\mathsf{tl}}[c, c^2](\mathbf{g}_0, \mathbf{fpi}_0,\ \mathbf{g}, \mathbf{l}) \qquad Y_{\mathsf{hd}}[e_j, c^2](\mathbf{g}^j_0, \mathbf{fpi}^j_0,\ \mathbf{g}^2_0, \mathbf{fpi}^2_0,\ \mathbf{g}', \mathbf{l}', \mathbf{l}^2)$$

(tail 1) (top)

Concurrent stack abstraction III

Induced semantics for procedure return (in thread 1)

$$instr(e_j, ret(c)) = \langle \texttt{ret}\, \mathbf{y} := P_j(\mathbf{x}) \rangle$$

$$Y^1_{\mathsf{tl}}[c, c^2](\mathbf{g}_0, \mathbf{fpi}_0,\ \mathbf{g}, \mathbf{l}) \qquad Y_{\mathsf{hd}}[e_j, c^2](\mathbf{g}^j_0, \mathbf{fpi}^j_0,\ \mathbf{g}^2_0, \mathbf{fpi}^2_0,\ \mathbf{g}', \mathbf{l}', \mathbf{l}^2)$$

(tail 1) (top)

$$(\mathbf{g}, \mathbf{x}) = (\mathbf{g}^j_0, \mathbf{fpi}^j_0)$$

(well-formedness condition for stack 1)

Concurrent stack abstraction III

Induced semantics for procedure return (in thread 1)

$$instr(e_j, ret(c)) = \langle \texttt{ret}\, \mathbf{y} := P_j(\mathbf{x}) \rangle$$

$$Y^1_{\mathsf{tl}}[c, c^2](\mathbf{g}_0, \mathbf{fpi}_0,\ \mathbf{g}, \mathbf{l}) \qquad Y_{\mathsf{hd}}[e_j, c^2](\mathbf{g}^j_0, \mathbf{fpi}^j_0,\ \mathbf{g}^2_0, \mathbf{fpi}^2_0,\ \mathbf{g}', \mathbf{l}', \mathbf{l}^2)$$

(tail 1) (top)

$$(\mathbf{g}, \mathbf{x}) = (\mathbf{g}^j_0, \mathbf{fpi}^j_0)$$

(well-formedness condition for stack 1)

$$R^-_{\mathbf{y}:=P_j(\mathbf{x})}(\mathbf{g}, \mathbf{l}, \mathbf{g}', \mathbf{l}',\ \mathbf{g}'', \mathbf{l}'')$$

(output parameter passing)

Concurrent stack abstraction III

Induced semantics for procedure return (in thread 1)

$$instr(e_j, ret(c)) = \langle \texttt{ret}\, \mathbf{y} := P_j(\mathbf{x}) \rangle$$

$$Y^1_{\mathsf{tl}}[c, c^2](\mathbf{g}_0, \mathbf{fpi}_0,\ \mathbf{g}, \mathbf{l}) \qquad Y_{\mathsf{hd}}[e_j, c^2](\mathbf{g}^j_0, \mathbf{fpi}^j_0,\ \mathbf{g}^2_0, \mathbf{fpi}^2_0,\ \mathbf{g}', \mathbf{l}', \mathbf{l}^2)$$

(tail 1) (top)

$$(\mathbf{g}, \mathbf{x}) = (\mathbf{g}^j_0, \mathbf{fpi}^j_0)$$

(well-formedness condition for stack 1)

$$R^-_{\mathbf{y}:=P_j(\mathbf{x})}(\mathbf{g}, \mathbf{l}, \mathbf{g}', \mathbf{l}',\ \mathbf{g}'', \mathbf{l}'')$$

(output parameter passing)

$$Y_{\mathsf{hd}}[ret(c)](\mathbf{g}^j_0, \mathbf{fpi}^j_0,\ \mathbf{g}^2_0, \mathbf{fpi}^2_0,\ \mathbf{g}'', \mathbf{l}'', \mathbf{l}^2)$$

Concurrent stack abstraction IV

Optimality result: concurrent stack abstraction reduces

1. to classical interprocedural analysis
   (one thread with recursion)
2. to classical analysis of concurrent programs
   (multiple threads without recursion)

In the other cases, it induces approximations.

- Otherwise it would contradict the undecidability result

Outline

Combining stack and data abstraction

We assume that we are given a Galois connection

$$\wp(\mathsf{Env}) \underset{\alpha_e}{\overset{\gamma_e}{\leftrightarrows}} \mathsf{Env}^\sharp$$

Then we can combine Galois connections:

$$\wp\left(\left(K^1 \times \mathsf{Env}\right)^+ \times \left(K^2 \times \mathsf{Env}\right)^+\right)$$

$$\underset{\alpha_c}{\overset{\gamma_c}{\leftrightarrows}} \overbrace{\left(K^1 \times K^2 \to \wp(\mathsf{Env})\right) \times \left(K^1 \to \wp(\mathsf{Env})\right) \times \left(K^2 \to \wp(\mathsf{Env})\right)}^{A_c}$$

$$\underset{\alpha_e}{\overset{\gamma_e}{\leftrightarrows}} \overbrace{\left(K^1 \times K^2 \to \mathsf{Env}^\sharp\right) \times \left(K^1 \to \mathsf{Env}^\sharp\right) \times \left(K^2 \to \mathsf{Env}^\sharp\right)}^{A_c^\sharp}$$

Example of suitable data abstractions

- Boolean programs: $\text{Env} \simeq \mathbb{B}^n$
  We have a finite lattice, no need for data abstraction
- Programs with numerical variables: $\text{Env} \simeq \mathbb{R}^n$
  Relational abstraction applicable: octagons, convex polyhedra, linear congruences, ...
  Approximations: due to $\sqcup$ and to abstraction of single instructions
- Programs with either Booleans or pointers to memory cells
  Stores represented/abstracted with 3-valued logical structures [SRW02]:

$$\wp(\text{Env}) \simeq \wp(2 - \mathit{STRUCT}) \leftrightarrows \wp(3 - \mathit{BSTRUCT})$$

  Enable the extension of [JLRS04] to concurrent programs

Complexity analysis

Program	single-thread	concurrent
single-procedure	$k \cdot \varphi(g + l)$	$k^n \cdot \varphi(g + nl)$
recursion	$2k \cdot \varphi(2g + l)$	$k^n \cdot \varphi(g + n(g + l))$

n : number of threads g : number of global variables

k : number of control points l : number of local variables

$\varphi(d)$: complexity of d-dimensional environments

Assuming $\varphi(d) = \mathcal{O}(2^d)$, the global complexity is

- polynomial in the size k of the CFGs,
- exponential in the number n of threads,
- in $\mathcal{O}(\phi(nd))$ if $d = g + l$ is the number of visible variables active in each thread

Complexity analysis

Program	single-thread	concurrent
single-procedure	$k \cdot \varphi(g + l)$	$k^n \cdot \varphi(g + nl)$
recursion	$2k \cdot \varphi(2g + l)$	$k^n \cdot \varphi(g + n(g + l))$

n : number of threads g : number of global variables

k : number of control points l : number of local variables

$\varphi(d)$: complexity of d-dimensional environments

Assuming $\varphi(d) = \mathcal{O}(2^d)$, the global complexity is

- polynomial in the size k of the CFGs,
- exponential in the number n of threads,
- in $\mathcal{O}(\phi(nd))$ if $d = g + l$ is the number of visible variables active in each thread

Well-known techniques for reducing the complexity can be reused: partial order and symmetry reduction for concurrency, Cartesian product and/or variables packing for data abstraction, ...

Outline

Implementation

- ConcurInterproc: Interproc generalized with concurrency
- Programs with finite-state and/or numerical variables
  Data abstraction: $\wp(\mathbb{B}^n \times \mathbb{R}^p) \leftrightarrows \mathbb{B}^n \to \mathrm{Pol}(\mathbb{R}^p)$
  implemented with MTBDDs (CUDD & APRON)
- Both forward and backward analysis implemented
- Choice between preemptive and cooperative scheduling

Online version available at

`http://pop-art.inrialpes.fr/interproc/concurinterprocweb.cgi`

Experiments

Goal

- Illustrate the precision of our method
- Analyze some of the approximations it induces

Test programs

Synchronisation algorithms requiring a detailed analysis of interactions between threads

- Mutual exclusion algorithms: Peterson, Kessel
- Barrier synchronisation: a protocol using counters

Mutual exclusion: Peterson

```
var b0,b1,turn:bool;
initial not b0 and not b1;

proc acquire(tid:bool) returns ()
begin
  if not tid then
    b0 = true; turn = tid;
   assume (b1==false or turn==not tid);
  else
    b1 = true; turn = tid;
   assume (b0==false or turn==not tid);
  endif;
end

proc release(tid:bool) returns ()
begin
  if not tid
  then b0 = false;
  else b1 = false; endif;
end
```

```
proc main(tid:bool) returns ()
begin
  while true do
    acquire(tid); /* C */
    release(tid);
  done;
end

thread T0:
var tid:bool;
begin
  tid = false; main(tid);
end
thread T1:
var tid:bool;
begin
  tid = true; main(tid);
end
```

Non-terminating example of [QRR04]

```
var .., g:uint[3], x,y:bool, ...;
initial .. and
       g==uint[3](0) and not x and not y,
proc foo(tid:bool,q:bool) returns ()
begin
  if not q then
    x=true; y=true; foo(tid,q);
  else
    acquire(tid);
    g = g + uint[3](1);
    release(tid);
  endif;
end
proc main(tid:bool) returns ()
var q:bool;
begin
  q = random;
  foo(tid,q);
  acquire(tid);
  if g==uint[3](0) then fail; endif;
  release(tid);
end
```

```
thread T0:
var tid:bool;
begin
  tid = false; main(tid);
end

thread T1:
var tid:bool;
begin
  tid = true; main(tid);
end
```

Synchronisation barriers with counters

```
var go : bool,
    counter,p0,p1 : int,
initial counter==0 and go;

proc barrier(lgo:bool)
    returns (nlgo:bool)
begin
  lgo = not lgo;
  counter = counter+1;
  if counter==2 then
    counter=0; go = lgo;
  else
    assume(lgo==go);
  endif;
  nlgo = lgo;
end
```

```
thread T0:
var lgo0:bool;
begin
  p0 = 0; lgo0 = true;
  while p0<=5 do
    lgo0 = barrier(lgo0);
    p0 = p0 + 1;
  done;
end
thread T1:
var lgo1:bool;
begin
  p1 = 0; lgo1 = true;
  while p1<=10 do
    lgo1 = barrier(lgo1);
    p1 = p1 + 1;
  done;
  fail;
end
```

Success if p1,p2 global, failure if they are local

Synchronisation barrier with counters: half-working

```
var go : bool, counter:int;
initial counter==0 and go;
proc barrier(lgo:bool) returns (nlgo:bool)
begin
  lgo = not lgo; counter = counter+1;
   if counter==2
   then counter=0; go = lgo;
   else assume(lgo==go); endif;
   nlgo = lgo;
end                          /* E0 */

thread T0:
var lgo0:bool;
begin
   lgo0 = true;              /* A0 */
   lgo0 = barrier(lgo0);     /* A1 */
/*lgo0 = barrier(lgo0); /* A2 */ */
end
```

```
thread T1:
var lgo1:bool;
begin
   lgo1 = true;              /* B0 */
   lgo1 = barrier(lgo1); /* B1 */
   lgo1 = barrier(lgo1); /* B2 */
   lgo1 = barrier(lgo1); /* B3 */
   fail;
end
```

Conclusion

- Unifies methods for recursive and concurrent programs

Conclusion

- Unifies methods for recursive and concurrent programs
- Technically, rather simple:
  - Classical instrumentation of the standard semantics
    - Less classical for backward semantics
  - Stack abstraction
    - collapses stacks into sets
  - Abstract semantics derived mechanically
    - Most technical aspect

Conclusion

- Unifies methods for recursive and concurrent programs
- Technically, rather simple:
  - Classical instrumentation of the standard semantics
    - Less classical for backward semantics
  - Stack abstraction
    - collapses stacks into sets
  - Abstract semantics derived mechanically
    - Most technical aspect
- Separates control and data abstraction
  - Can extend any relational interprocedural analysis to concurrent programs

Conclusion

- Unifies methods for recursive and concurrent programs
- Technically, rather simple:
  - Classical instrumentation of the standard semantics
    - Less classical for backward semantics
  - Stack abstraction
    - collapses stacks into sets
  - Abstract semantics derived mechanically
    - Most technical aspect
- Separates control and data abstraction
  - Can extend any relational interprocedural analysis to concurrent programs
- Experimental evaluation of its precision
  - Local variables handled less precisely than global variables
  - Hence, procedure inlining may improve precision

Perspective

Stack abstraction

- Better precision/Less modularity/Worse complexity: adding more information in the call-context for each thread ?
- Further abstraction to derive thread-modular analysis of [FQ03] ?
- Adapting iteration and widening techniques to a concurrent context

Application to TLM models

- Exploiting cooperative scheduling
- Encoding of TLM synchronisation concepts (events, time)
- Built-in, higher-level synchronisation primitives ?


Patrick Cousot and Radhia Cousot.
Static determination of dynamic properties of recursive procedures.
In *IFIP Conf. on Formal Description of Programming Concepts*, 1977.


M. Sharir and A. Pnueli.
Semantic foundations of program analysis.
In S.S. Muchnick and N.D. Jones, editors, *Program Flow Analysis: Theory and Applications*, chapter 7. Prentice-Hall, 1981.


J. Knoop and B. Steffen.
The interprocedural coincidence theorem.
In *Compiler Construction, CC'92*, volume 641 of *LNCS*, 1992.


Tom Reps, Susan Horwitz, and Mooly Sagiv.
Precise interprocedural dataflow analysis via graph reachability
In *Principles of Prog. Languages, POPL'95*. ACM, 1995.


Javier Esparza and Jens Knoop.
An automata-theoretic approach to interprocedural data-flow analysis.
In *Foundations of Software Science and Computation Structure, FoSSaCS '99*, volume 1578 of *LNCS*, 1999.


B. Jeannet and W. Serwe.
Abstracting call-stacks for interprocedural verification of imperative programs.
In *Int. Conf. on Algebraic Methodology and Software Technology, AMAST'04*, volume 3116 of *LNCS*, 2004.


G. Ramalingam.
Context-sensitive synchronization-sensitive analysis is undecidable
ACM Trans. on Programing Language and Systems, 22(2), 2000.


C. Flanagan and S. Qadeer.
Thread-modular model checking.
In *SPIN'03: Workshop on Model Checking Software*, volume 2648 of *LNCS*, 2003.


Shaz Qadeer, Sriram K. Rajamani, and Jakob Rehof.
Summarizing procedures in concurrent programs.
In *Principles of programming languages, POPL'04*. ACM, 2004.


Patrick Cousot and Radhia Cousot.
Static determination of dynamic properties of recursive procedures.
In *IFIP Conf. on Formal Description of Programming Concepts*, 1977.


C. Flanagan and S. Qadeer.
Thread-modular model checking.
In *SPIN'03: Workshop on Model Checking Software*, volume 2648 of *LNCS*, 2003.


B Jeannet, A. Loginov, T. Reps, and M. Sagiv.
A relational approach to interprocedural shape analysis.
In *Static Analysis Symposium, SAS'04*, volume 3148 of *LNCS*, 2004.


B. Jeannet and W. Serwe.
Abstracting call-stacks for interprocedural verification of imperative programs.
In *Int. Conf. on Algebraic Methodology and Software Technology, AMAST'04*, volume 3116 of *LNCS*, 2004.


J. Knoop and B. Steffen.
The interprocedural coincidence theorem.
In *Compiler Construction, CC'92*, volume 641 of *LNCS*, 1992.


Shaz Qadeer, Sriram K. Rajamani, and Jakob Rehof.
Summarizing procedures in concurrent programs.
In *Principles of programming languages, POPL'04*. ACM, 2004.


G. Ramalingam.
Context-sensitive synchronization-sensitive analysis is undecidable.
ACM Trans. on Programing Language and Systems, 22(2), 2000.


M. Sharir and A. Pnueli.
Semantic foundations of program analysis.
In S.S. Muchnick and N.D. Jones, editors, *Program Flow Analysis: Theory and Applications*, chapter 7. Prentice-Hall, 1981.


M. Sagiv, T. Reps, and R. Wilhelm.

Parametric shape analysis via 3-valued logic.
ACM Transactions on Prog. Languages and Systems, 24(3), 2002.