Vérification de programmes C concurrents avec Cubicle : Enfoncer les barrières

JFLA
9 janvier 2014

David Declerck

Université Paris-Sud

Travail réalisé conjointement avec :

- Sylvain Conchon, Alain Mebsout (Université Paris-Sud)
- Luc Maranget (INRIA)

Synchronization Barriers

Data structure used to synchronize the execution of a group of threads at a program point.

POSIX libraries implement barriers defined as

- a data type `barrier_t`
- an initialization function `barrier_init`
- a synchronization function `barrier_wait`

Sense-Reversing Barriers

Demo.

Safe barrier

Good synchronization:

- There does not exist a thread before the barrier and another thread after the barrier

Annotations in C source of these program points with

```
///exclusive
wait_barrier(...);
///exclusive
```

Contributions of this work

Proving safety of synchronization barriers

- written in C
- for any number of threads
- automatically by model checking

We assume sequential consistency :

- Interleaving semantics
- Preservation of operations order

Compiling (a fragment of) C to Cubicle

A limited fragment of C (basically, just what we need for the implementation of our benchmarks)

- `int` and `void` data types
- restricted usage of structures
- pointers limited to passing by reference
- loops, assignments, conditionals,
- arithmetic and relational operations

Target language

- Transition systems with states described by global variables (of type int, bool and enumerations) and infinite arrays indexed by thread identifiers

- Transitions are encoded by logical formulas and they can be parameterized by thread identifiers (existential quantification)

$$\begin{aligned}\exists i.\quad & \mathtt{T}[i] = \mathsf{true} \land \mathtt{X} \leq 100 \land \forall k.\ k \neq i \implies \mathtt{T}[k] = \mathsf{false} \\ & \land\ \mathtt{X}' = \mathtt{X} + 1 \land \mathtt{T}'[i] = \mathsf{false}\end{aligned}$$

(here $\mathtt{T}'$ and $\mathtt{X}'$ denote respectively the value of array $\mathtt{T}$ and variable $\mathtt{X}$ after the execution of the transition)

We can only check safety properties characterized by bad states

(very simple) Memory Model

A set of global variables shared between threads, and for each thread `i`

- a program counter `PC[i]` of type `t`, where

  ```
  type t = Idle | End | L1 | L2 | ...
  ```

- a stack represented by a set of k global variables `STACK_`j`[i]`

Compilation schema: Example

$$\texttt{x = x + 1} \parallel \ldots$$

corresponds to the following instructions

$$
\begin{aligned}
&\text{L0} : \texttt{STACK\_0[i]} \leftarrow \texttt{x}\\
&\text{L1} : \texttt{STACK\_0[i]} \leftarrow \texttt{STACK\_0[i] + 1}\\
&\text{L2} : \texttt{x} \leftarrow \texttt{STACK\_0[i]}
\end{aligned}
$$

which are compiled as three transitions

$$\exists i.\ \texttt{PC}[i] = \texttt{L0} \land \texttt{STACK\_0}'\texttt{[i]} = \texttt{x} \land \texttt{PC}'[i] = \texttt{L1}$$

$$\exists i.\ \texttt{PC}[i] = \texttt{L1} \land \texttt{STACK\_0}'\texttt{[i]} = \texttt{STACK\_0}[\texttt{i}] + 1 \land \texttt{PC}'[i] = \texttt{L2}$$

$$\exists i.\ \texttt{PC}[i] = \texttt{L2} \land \texttt{x}' = \texttt{STACK\_0[i]}$$

Compiling Thread Counters

How to encode the arbitrary number of threads `N` ?

`#define N ...` `int cpt = N;`	... | 1 | 1 | 1 | ...	$\forall i.\ \texttt{cpt}'[i] = 1$
`cpt--;`	... | 1 | 1 | 1 | ... $\downarrow$ ... | 1 | 0 | 1 | ...	$\exists i.\ \texttt{cpt}[i] = 1 \land \texttt{cpt}'[i] = 0$
`if (cpt == 0) ...`	... | 0 | 0 | 0 | ...	$\forall i.\ \texttt{cpt}[i] = 0$

Proving the Sense-Reversing Barrier

Demo.

Benchmarks

	Nodes	Inv.	Restarts	Time
`sb_alt.c`	598	180	53	11m27s
`sb.c`	414	156	34	5m21s
`sb_nice.c`	303	139	49	28m8s
`sb_single.c`	174	99	54	17m44s
`sb_loop.c`	-	-	-	TO

Optimizations

1. We designed a (simple) typing analysis to determine when a variable of type `int` is used as a Boolean
   - SMT solver more efficient on booleans
   - Invariant generation of model checker is not good with integers
2. Elimination of spurious traces arising from crash failure model present to handle universal quantifiers of thread counters' encoding
   - Reduce number of backtracking (restarts) in model checker

Benchmarks: typing optimization

	with typing			without typing		
	Inv.	Restarts	Time	Inv.	Restarts	Time
`sb_alt.c`	152	7	**7.64s**	180	53	11m27s
`sb.c`	226	10	**20.7s**	156	34	5m21s
`sb_nice.c`	106	9	**11.6s**	139	49	28m8s
`sb_single.c`	115	5	**3.11s**	99	54	17m44s
`sb_loop.c`	1577	33	**14m49**	-	-	TO

Benchmarks: *crash failure model* optimization

	Refinement			No refinement		
	Inv.	Restarts	Time	Inv.	Restarts	Time
`sb_alt`	37	1	**2,17s**	152	7	7,64s
`sb.c`	64	1	**3,99s**	226	10	20,7s
`sb_nice.c`	51	1	**2,54s**	106	9	11,6s
`sb_single.c`	36	1	**1,12s**	115	5	3,11s
`sb_single_us.c`	–	0	**4,94s**	–	0	5,06s
`sb_loop.c`	275	1	**59,8s**	1577	33	14m49s

Future work

- Experiment with other types of synchronization barriers
- Larger subset of C
- C11 standard (semantic for concurrent programs)
- Improve Cubicle's invariants generation mechanism for numerical candidates

Merci.

Optimization 1: Integer as Booleans

We designed a (simple) typing analysis to determine when a variable of type `int` is used as a Boolean

Optimization 1: Integer as Booleans

We designed a (simple) typing analysis to determine when a variable of type `int` is used as a Boolean

```
int x, y, z;
x = 0;
y = x;
z = y;
if (z) { x = x + 1; }
```

Optimization 1: Integer as Booleans

We designed a (simple) typing analysis to determine when a variable of type `int` is used as a Boolean

```
int x, y, z;
x = 0;                    x is int or bool
y = x;
z = y;
if (z) { x = x + 1; }
```

Optimization 1: Integer as Booleans

We designed a (simple) typing analysis to determine when a variable of type `int` is used as a Boolean

```
int x, y, z;
x = 0;                    x is int or bool
y = x;
z = y;                    x, y and z have the same type
if (z) { x = x + 1; }
```

Optimization 1: Integer as Booleans

We designed a (simple) typing analysis to determine when a variable of type `int` is used as a Boolean

`int x, y, z;`	
`x = 0;`	`x` is `int` or `bool`
`y = x;`	
`z = y;`	`x`, `y` and `z` have the same type
`if (z) { x = x + 1; }`	`z` is `bool`, and `x` is `int`

Optimization 1: Integer as Booleans

We designed a (simple) typing analysis to determine when a variable of type `int` is used as a Boolean

`int x, y, z;`	
`x = 0;`	`x` is `int` or `bool`
`y = x;`	
`z = y;`	`x`, `y` and `z` have the same type
`if (z) { x = x + 1; }`	`z` is `bool`, and `x` is `int`

The program is rejected

Optimization 1: Integer as Booleans

We designed a (simple) typing analysis to determine when a variable of type `int` is used as a Boolean

```
int x, y;
y = 0;
if (y == 0) { x = 0; }
```

Optimization 1: Integer as Booleans

We designed a (simple) typing analysis to determine when a variable of type `int` is used as a Boolean

```
int x, y;
y = 0;                          y is int or bool
if (y == 0) { x = 0; }
```

Optimization 1: Integer as Booleans

We designed a (simple) typing analysis to determine when a variable of type `int` is used as a Boolean

`int x, y;`	
`y = 0;`	`y` is `int` or `bool`
`if (y == 0) { x = 0; }`	`y` is `int` and `x` is `int` or `bool`

Optimization 1: Integer as Booleans

We designed a (simple) typing analysis to determine when a variable of type `int` is used as a Boolean

`int x, y;`	
`y = 0;`	y is int or bool
`if (y == 0) { x = 0; }`	y is int and x is int or bool

The program is well typed

`x:int` (for safety reasons)
`y:int`

Optimization 1: Integer as Booleans

We designed a (simple) typing analysis to determine when a variable of type `int` is used as a Boolean

```
int x, y, z;
x = 0 && 1;
if (x != y && y !=z && x != z) ...
```

Optimization 1: Integer as Booleans

We designed a (simple) typing analysis to determine when a variable of type `int` is used as a Boolean

```
int x, y, z;
x = 0 && 1;                                    x is bool
if (x != y && y !=z && x != z) ...
```

Optimization 1: Integer as Booleans

We designed a (simple) typing analysis to determine when a variable of type `int` is used as a Boolean

```
int x, y, z;
x = 0 && 1;                                    x is bool
if (x != y && y !=z && x != z) ...
```

`x`, `y` and `z` are `bool`, but only `x` is initialized

Optimization 1: Integer as Booleans

We designed a (simple) typing analysis to determine when a variable of type `int` is used as a Boolean

```
int x, y, z;
x = 0 && 1;
if (x != y && y !=z && x != z) ...
```

`x` is `bool`

`x`, `y` and `z` are `bool`, but only `x` is initialized

The program is rejected

Optimization 2: Elimination Spurious Traces

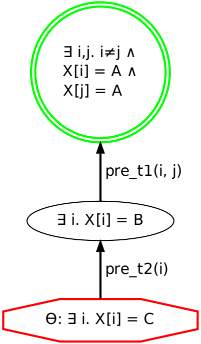
Crash Failure Model

$$\begin{cases} \texttt{type t = A | B | C} \\ \forall i.\ \texttt{X}[i] = \texttt{A} \quad (\text{inital states}) \\ t_1 : \exists i, j.\ i \neq j \land \texttt{X}[i] = \mathsf{A} \land \texttt{X}[j] = \mathsf{A} \land\ \texttt{X}'[i] = \mathsf{B} \\ t_2 : \exists i.\ \texttt{X}[i] = \mathsf{B} \land \forall j.\ j \neq i \implies \texttt{X}[j] \neq \mathsf{A} \land\ \texttt{X}'[i] = \mathsf{C} \end{cases}$$

Optimization 2: Elimination Spurious Traces

Crash Failure Model

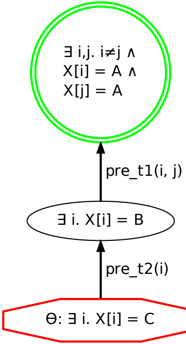
$$\begin{cases} \texttt{type t = A | B | C} \\ \forall i.\ \texttt{X}[i] = \texttt{A} \quad (\text{inital states}) \\ t_1 : \exists i, j.\ i \neq j \wedge \texttt{X}[i] = \mathsf{A} \wedge \texttt{X}[j] = \mathsf{A} \wedge\ \texttt{X}'[i] = \mathsf{B} \\ t_2 : \exists i.\ \texttt{X}[i] = \mathsf{B} \wedge \forall j.\ j \neq i \implies \texttt{X}[j] \neq \mathsf{A} \wedge\ \texttt{X}'[i] = \mathsf{C} \end{cases}$$



Optimization 2: Elimination Spurious Traces

Crash Failure Model

$$\begin{cases} \texttt{type t = A | B | C} \\ \forall i.\ \texttt{X}[i] = \texttt{A} \quad (\text{inital states}) \\ t_1 : \exists i, j.\ i \neq j \wedge \texttt{X}[i] = \mathsf{A} \wedge \texttt{X}[j] = \mathsf{A} \wedge\ \texttt{X}'[i] = \mathsf{B} \\ t_2 : \exists i.\ \texttt{X}[i] = \mathsf{B} \wedge \forall j.\ j \neq i \implies \texttt{X}[j] \neq \mathsf{A} \wedge\ \texttt{X}'[i] = \mathsf{C} \end{cases}$$

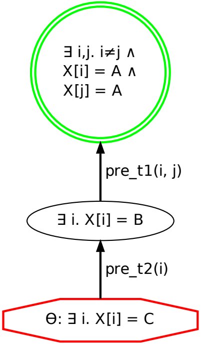


the program could contain only one thread

Optimization 2: Elimination Spurious Traces

Crash Failure Model

$$\begin{cases} \texttt{type t = A | B | C} \\ \forall i.\ \texttt{X}[i] = \texttt{A} \quad (\text{inital states}) \\ t_1 : \exists i, j.\ i \neq j \land \texttt{X}[i] = \mathsf{A} \land \texttt{X}[j] = \mathsf{A} \land\ \texttt{X}'[i] = \mathsf{B} \\ t_2 : \exists i.\ \texttt{X}[i] = \mathsf{B} \land \boxed{\forall j.\ j \neq i \implies \texttt{X}[j] \neq \mathsf{A}} \land\ \texttt{X}'[i] = \mathsf{C} \end{cases}$$



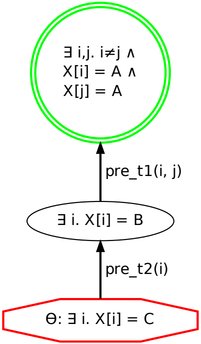
the program contains at least two threads

the program could contain only one thread

Optimization 2: Elimination Spurious Traces

Crash Failure Model

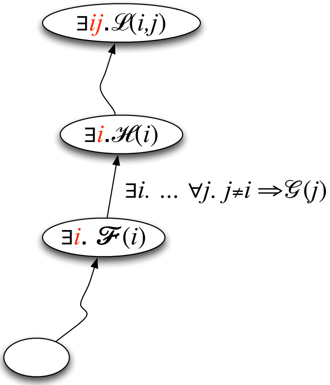
$$\begin{cases} \texttt{type t = A | B | C} \\ \forall i.\ \texttt{X}[i] = \texttt{A} \quad (\text{inital states}) \\ t_1 : \exists i, j.\ i \neq j \wedge \texttt{X}[i] = \mathsf{A} \wedge \texttt{X}[j] = \mathsf{A} \wedge\ \texttt{X}'[i] = \mathsf{B} \\ t_2 : \exists i.\ \texttt{X}[i] = \mathsf{B} \wedge \forall j.\ j \neq i \implies \texttt{X}[j] \neq \mathsf{A} \wedge\ \texttt{X}'[i] = \mathsf{C} \end{cases}$$



the program contains at least two threads

the program could contain only one thread
thus, t_2 is not possible from that state

How to Refine the Crash Failure Model ?

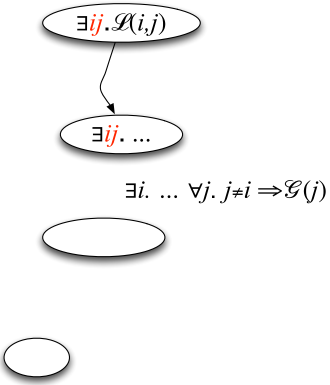


How to Refine the Crash Failure Model ?

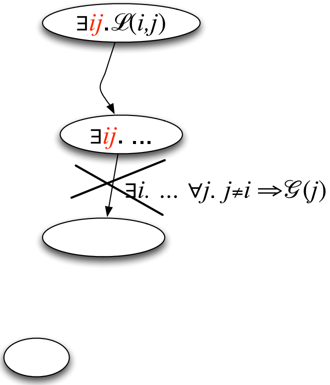
$\exists ij.\mathscr{L}(i,j)$

$\exists i. \; \ldots \; \forall j.\; j \neq i \Rightarrow \mathscr{G}(j)$

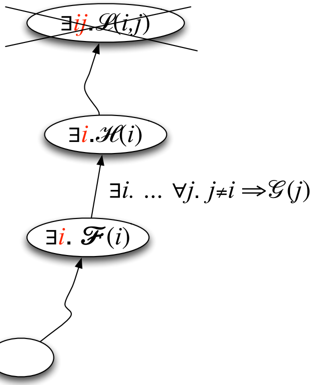
How to Refine the Crash Failure Model ?



How to Refine the Crash Failure Model ?



How to Refine the Crash Failure Model ?



How to Refine the Crash Failure Model ?