Extending the herd Memory model simulator

Simon Colin

July 13, 2018

Memory models

Thread 0	Thread 1
$x \leftarrow 1$	$y \leftarrow 1$
$r0 \leftarrow y$	$r1 \leftarrow x$

Can we have $r0 = 0$ and $r1 = 0$?
We expect the instructions to be executed in order, so this shouldn't be possible.
On modern processors, this behaviour can happen.

Memory events

Programs are made of instructions.

Instructions involve memory events : reads from and writes to the shared memory.

Instructions	Events
`x = 1;`	`Store(x, 1)`
`a = x;`	`Read(x)`
`fence;`	`Fence`
`y++;`	`Read(y, s0)`
	`Store(y, s0 + 1)`

x and y are shared variables, a is a local variable

A simple model : sequential consistency

Two properties :

- No local reordering : the events occur in the same order as in the program.
- Instantaneous memory : when a thread writes to memory all threads see the value instantly.

Weaker memory models

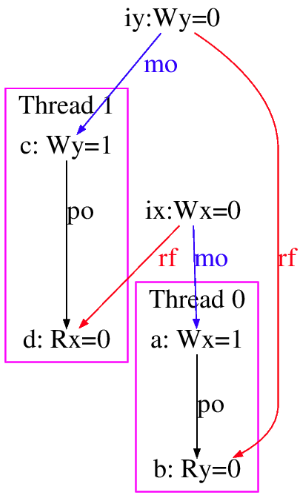
Modern memory models allow more behaviors than sequential consistency.

Such behaviours result from :

- out of order execution
- speculative execution
- writes don't become visible to every thread at the same time

Executions

Executions are a set of events and relations.



- program order `po`
- memory order `mo`
- read from `rf`
- many more relations ...

RC11

An RC11[1] consistent execution satisfies :

- `irreflexive (hb; eco?)` (coherence)
- `empty (rmw & (rb; mo)).` (atomicity)
- `acyclic (po | rf)` (no-thin-air)
- `acyclic psc` (sequential-consistency)

[1]Repairing sequential consistency in C/C++11, Proceedings of the 38th ACM SIGPLAN Conference

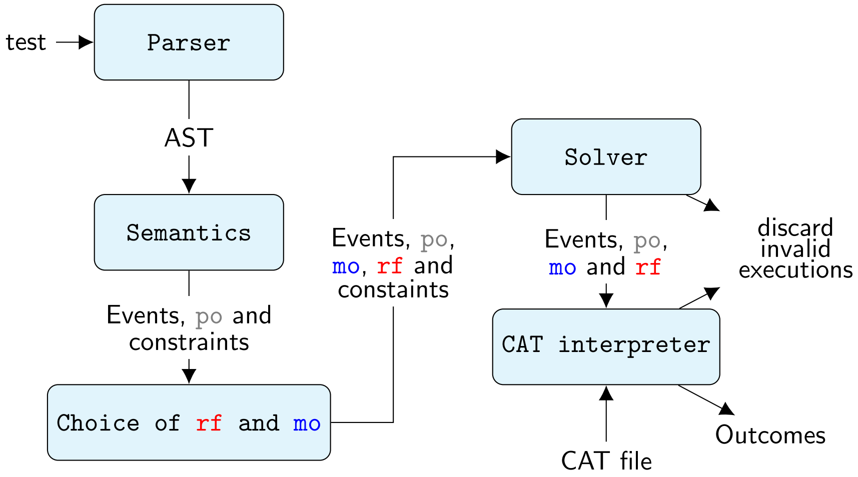
no-thin-air

```
acyclic (po | rf)
```

- If we read from a write, the write has to have taken place first.
- If a write depends on a read, the read has to have taken place first.

`dep`; `rf` causality loops are possible.
Since `dep` $\subseteq$ `po` we forbid `dep`; `rf` loops by simply banning `po`; `rf` loops.

Herd


test
Parser
AST
Semantics
Events, po and constraints
Choice of rf and mo
Events, po, mo, rf and constaints
Solver
Events, po, mo and rf
discard invalid executions
CAT interpreter
CAT file
Outcomes

Herd

```
herd(execution E)
{
  forall mo on E
  {
    forall rf on execution E
    {
      if consistent(CAT model, E, mo, rf)
      then add(outcomes, E, mo, rf)
    }
  }
```

Herd

Generates every combination of `rf` and `mo`, this is expensive.

Dealing with large sets of executions is a common problem for model checkers.

Solution : a stateless incremental algorithm[2] that avoids building most inconsistent executions.

[2]Effective stateless model checking for C/C++ concurrency, Proceedings of the ACM on Programming Languages, Colume 2 Issue POPL

My project

Implement RC11 in CAT.

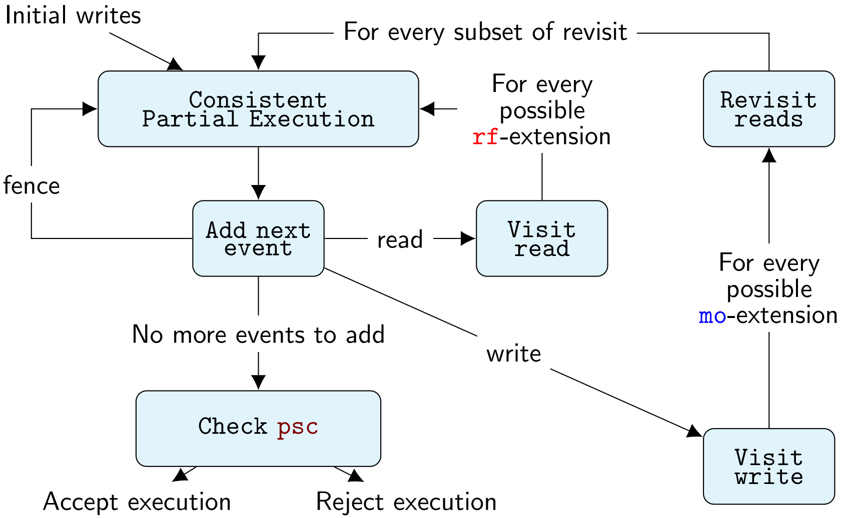
Implement the RC11 check directly into Herd.

Implement the stateless algorithm in Herd.

The stateless algorithm

```
visit(execution $E$) {
  if has_events_to_add($E$)
  then
  {
    $e$ = next_event($E$)
    $execs$ = possible_extensions($E$, $e$)
    for $ex$ in $execs$
    {
      if rc11_consistent($ex$)
      then visit($ex$)
    }
  } else {
    if acyclic($E$.psc)
    then add(outcomes, $E$)
  }}
```

How the stateless algorithm works



Conclusion

My algorithm works for CAS-free programs.
It is faster than Herd for larger tests, but slower for shorter ones.

The treatment of CAS is not finished yet, this is difficult because of differences in the handling of semantics between Herd and the reference.

Further work

- finish the implementation
- test the incremental algorithm against the RC11 CAT file
- optimize the performance
- work on SMT-based memory model checking