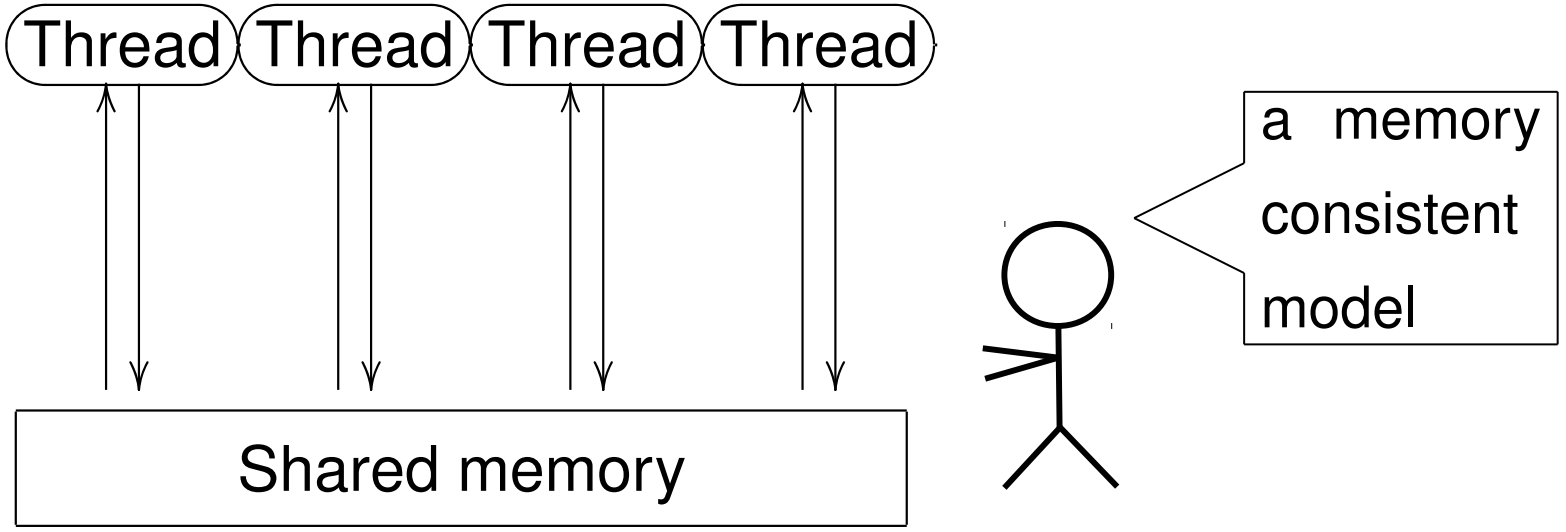
Program Verification under Formalized Memory Consistent Models

Tatsuya Abe
RIKEN AICS
(Joint Work with Toshiyuki Maeda)

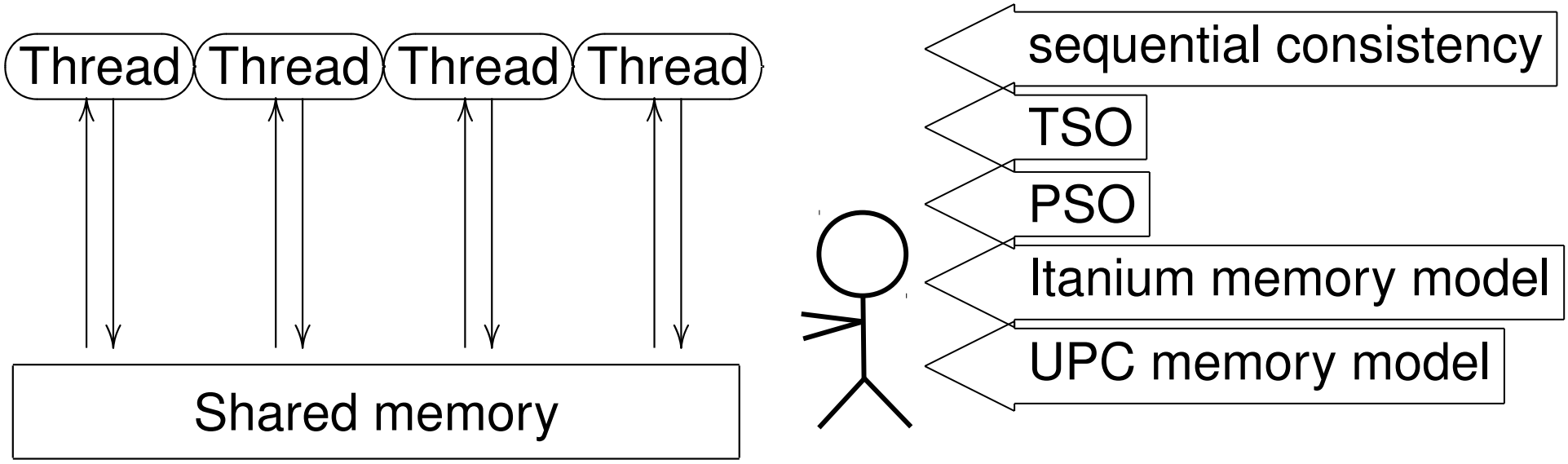
SLACS / NSA
May 26, 2014

Q. What is Memory Consistent Model (MCM)?

A. MCM is a rule to share a memory among multiple threads.



There exist many MCMs



We have to understand such MCMs
since the MCMs are rules. But,

There exist non-intuitive MCMs!

An example of curious executions

Under Itanium MCM,

Memory: `[x] == [y] == 0`

Thread 1	Thread 2
`[x] = 1;`	`r2 = [y];`
`r1 = [x];`	`r3 = [x];`
`[y] = r1;`	

`r2 == 1 && r3 == 0` is allowed!

Reordering under Itanium MCM

Under Itanium MCM,

Memory: `[x] == [y] == 0`

Thread 1	Thread 2
`[x] = 1;`	`r3 = [x];`
`r1 = [x];`	`r2 = [y];`
`[y] = r1;`	

Since `r2 == 1` and `r3 == 0` can be reordered,

`r2 == 1 && r3 == 0` is allowed!

Speculative behaviors under UPC MCM

Thread 1	Thread 2
`r1 = [x];`	`r2 = [x];`
`[x] = 2;`	`[x] = 1;`

`r1 == 1 && r2 == 2` is allowed!

Speculative behaviors under UPC MCM

Thread 1	Thread 2
`r1 = [x];`	`r2 = [x];`
`[x] = 2;`	`[x] = 1;`

`r1 == 1 && r2 == 2` is allowed!

Thread 1	Thread 2
speculate: `[x] = 1`	speculate: `[x] = 2`
`r1 = [x];`	`r2 = [x];`
`[x] = 2;`	`[x] = 1;`

Reordering depends on an MCM

Can

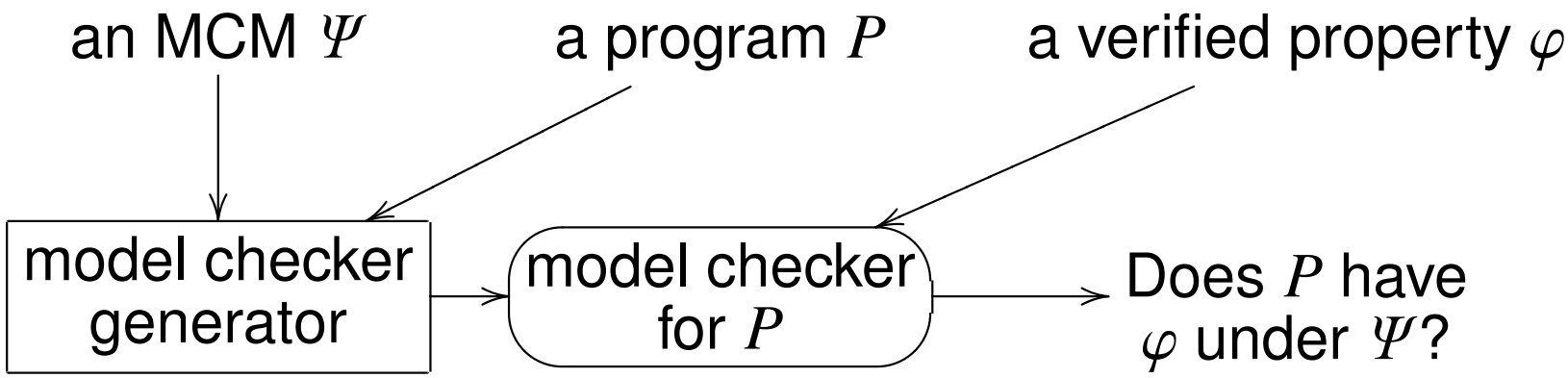
```
r2 = [y];
r3 = [x];
```

be reordered?

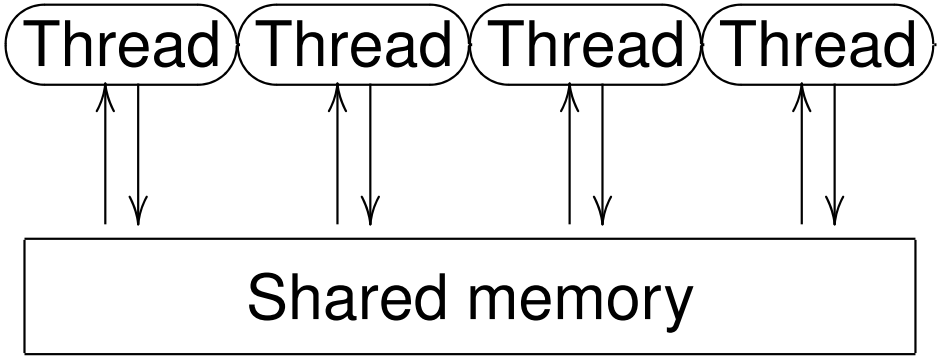
Sequential consistency:	No
Total Store Ordering:	No
Partial Store Ordering:	Yes
Itanium MCM:	Yes
UPC MCM:	No

Our approach to handle various MCMs simultaneously

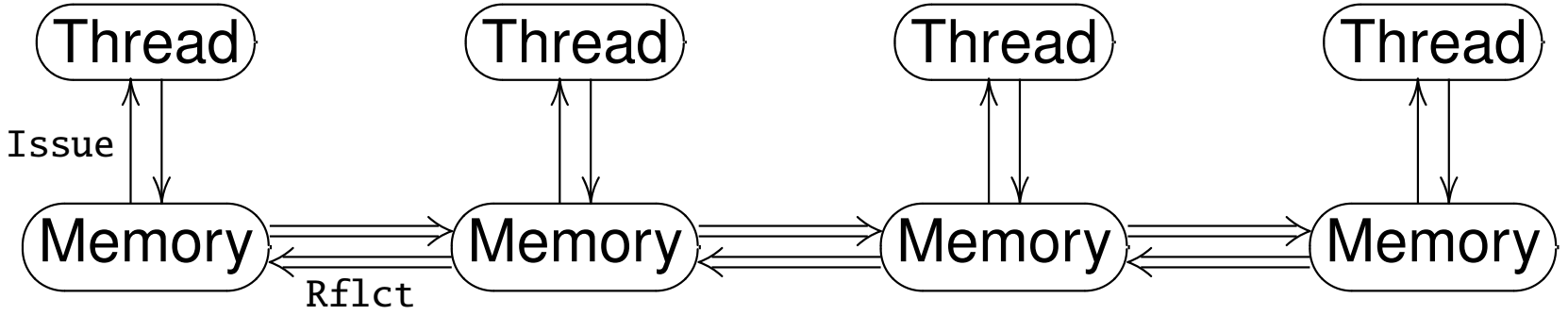
1. Give a general model (called base model),
2. define an MCM as a constraint on base model,
3. develop a model checker generator



Base model for reorderings

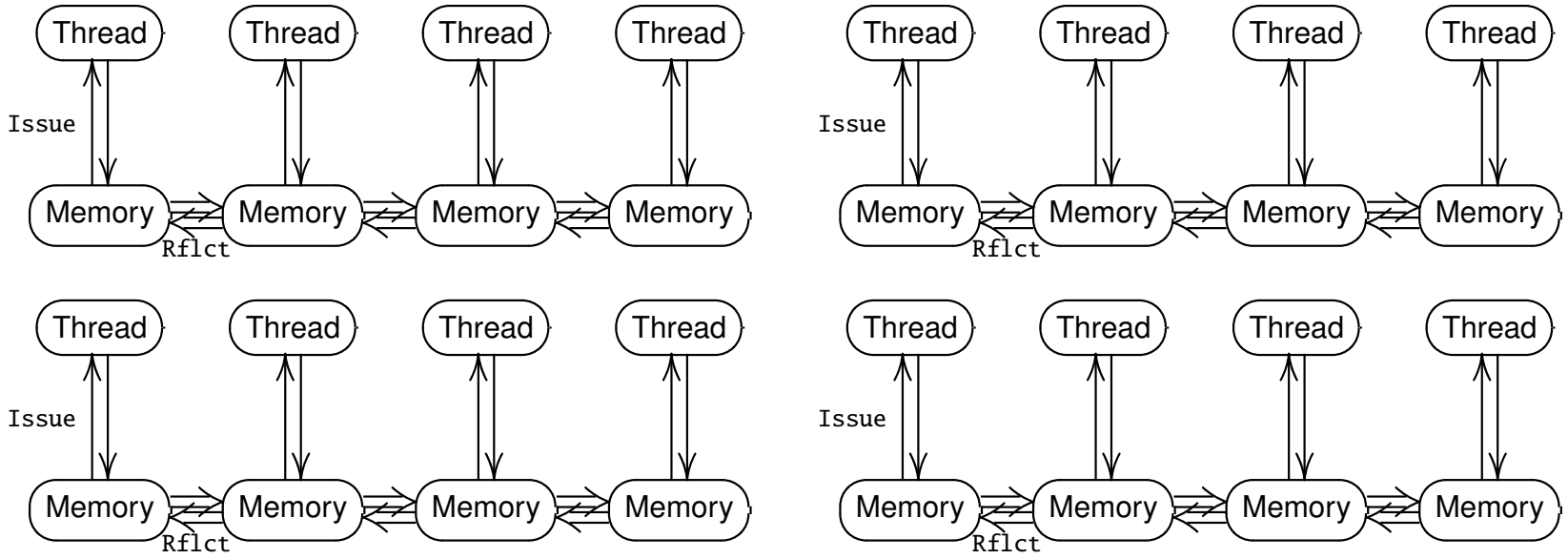


Base model



Threads have their own memories, and read-from/write-to a shared memory is simulated by communication among them.

Base model for reorderings and speculative behaviors



Each thread has its own base model (at the previous slide) for a speculation.

How to define an MCM

$$\mathtt{Issue}_1\ \mathtt{Rflct}_1\ \mathtt{Issue}_2\ \mathtt{Rflct}_2\ \mathtt{Issue}_3\ \mathtt{Rflct}_3\ \ldots$$

Each pair can be reordered on base model.

- ✓ $\mathtt{Issue}_1\ \mathtt{Issue}_2\ \mathtt{Rflct}_1\ \mathtt{Rflct}_2\ \mathtt{Issue}_3\ \mathtt{Rflct}_3\ \ldots$
- ✓ $\mathtt{Issue}_1\ \mathtt{Issue}_2\ \mathtt{Issue}_3\ \mathtt{Rflct}_1\ \mathtt{Rflct}_2\ \mathtt{Rflct}_3\ \ldots$
- ✓ ...

MCMs are constraints for base model.

- ✓ $\mathtt{Issue}_1\ \mathtt{Issue}_2\ \mathtt{Rflct}_1\ \mathtt{Rflct}_2\ \mathtt{Issue}_3\ \mathtt{Rflct}_3\ \ldots$
- × $\mathtt{Issue}_1\ \mathtt{Issue}_2\ \mathtt{Issue}_3\ \mathtt{Rflct}_1\ \mathtt{Rflct}_2\ \mathtt{Rflct}_3\ \ldots$
- ...

Formal definition of MCMs

Define an MCM as a set of formulas in mathematical logic.

Definition of formula. A formula is a combination of atomic formulas by logical connectives $\neg$ (negation), $\supset$ (implication), and $\forall$ (universal quantifier).

Atomic formulas. $o < o'$ (o' must be performed after o) where o and o' are either of the following:

- `Issue` $T\ (i,\ a)$
  T's instruction i with attributes a is issued.

- `Rflct` $[\Rightarrow T]\ (i,\ a)$
  Issue of i is reflected to T, i.e., T can observe i's issue.

Acquire and release semantics of Itanium MCM

Any instruction must wait for all reflections of an instruction with an attribute `release` that is issued before.

```
[x] = 1:release;
r1 = [x];
```

`r1 = [x]` must wait for reflections of `[x] = 1:release`.

$$\texttt{Issue}\ T\ (i,\ \{\texttt{release}\}) < \texttt{Issue}\ T\ (i',\ A\}) \supset$$
$$\texttt{Rflct}\ T\ (i,\ \{\texttt{release}\}) < \texttt{Issue}\ T\ (i',\ A)$$

We confirmed that it was possible to write Itanium MCMs.

Lock and unlock of UPC MCM

If T locks x, then T' cannot lock x until T unlocks x.

$$\texttt{Issue}\ T\ (\texttt{Nop},\ \{\texttt{lock(x)}\}) < \texttt{Issue}\ T'\ (i',\ \{\texttt{lock(x)}\}) \supset$$
$$\texttt{Issue}\ T\ (\texttt{Nop},\ \{\texttt{lock(x)}\}) < \texttt{Issue}\ T\ (\texttt{Nop},\ \{\texttt{unlock(x)}\}) \wedge$$
$$\texttt{Issue}\ T\ (\texttt{Nop},\ \{\texttt{unlock(x)}\}) < \texttt{Issue}\ T'\ (i',\ \{\texttt{lock(x)}\})$$

We confirmed that it was possible to write UPC MCMs.

Implementation: model checker generator

Skip!

Experiments: model checking under MCMs

Skip!

Related work

Relaxed memory consistency model is a hot topic.

[Yang et al. ’05] proposes an operational specification framework UMM, which cannot handle speculative behaviors.

[Saraswat et al. ’07] uses program transformations to reason about it.

[Boudol et al. ’09] uses a process calculus to reason about it.

[Shen et al. ’99] uses term rewriting to reason about it.

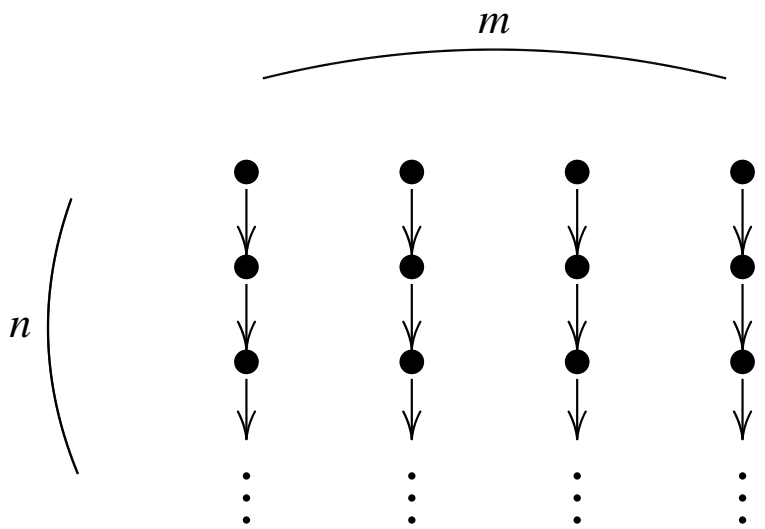
...

State explosion

A verification under an MCM suffers from state explosion.

Consider m threads with n instructions.

Under an MCM that allows interleavings,



there exist ${}_{m\cdot n}C_n \cdot {}_{(m-1)\cdot n}C_n \cdot \cdots \cdot {}_{2\cdot n}C_n \cdot {}_{n}C_n$ execution traces.

Partial order reduction based on a verified property

To check $o < o'$, use time counter,

$ABCD$ means a state $\{t_A = 1, t_B = 2, t_C = 3, t_D = 4\}$, and

$ACBD$ means a state $\{t_A = 1, t_B = 3, t_C = 2, t_D = 4\}$.

Let us use pairs of terms that occur in an MCM.

Assume a MCM is $A < B \supset C < D$.

Then, pairs of terms that occur in an MCM are $\{\langle A, B\rangle, \langle C, D\rangle\}$.

$ABCD$ means a state $\{t_{A<B} = \text{true}, t_{C<D} = \text{true}\}$, and

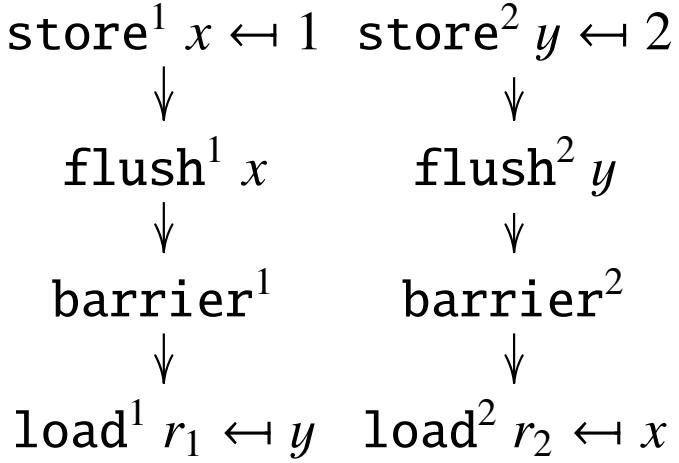
$ACBD$ means a state $\{t_{A<B} = \text{true}, t_{C<D} = \text{true}\}$, too.

Theorem proving using partial order reduction

Thread 1	Thread 2
`store` $x \leftarrow\!\shortmid 1;$	`store` $y \leftarrow\!\shortmid 2;$
`flush` $x;$	`flush` $y;$
`barrier;`	`barrier;`
`load` $r_1 \leftarrow\!\shortmid y$	`load` $r_2 \leftarrow\!\shortmid x$

Question. $r_1 = 2 \wedge r_2 = 1?$

Merging triples in backward searches



$$\frac{\Pi_1(G_1) \quad \left\{y = 2 \wedge r_2 = 1\right\} \quad \{\texttt{load}^1\ r_1 \mapsto y\} \quad \left\{r_1 = 2 \wedge r_2 = 1\right\}}{\left\{?\right\} \quad G \quad \left\{r_1 = 2 \wedge r_2 = 1\right\}}$$

$$\frac{\Pi_2(G_2) \quad \left\{r_1 = 2 \wedge x = 1\right\} \quad \{\texttt{load}^2\ r_2 \mapsto x\} \quad \left\{r_1 = 2 \wedge r_2 = 1\right\}}{\left\{?\right\} \quad G \quad \left\{r_1 = 2 \wedge r_2 = 1\right\}}$$

$$\frac{\Pi'(G') \quad \left\{y = 2 \wedge x = 1\right\} \quad \{\texttt{load}^2\ r_2 \mapsto x\} \quad \left\{y = 2 \wedge r_2 = 1\right\}}{\left\{?\right\} \quad G_1 \quad \left\{y = 2 \wedge r_2 = 1\right\}}$$

$$\frac{\Pi'(G') \quad \left\{y = 2 \wedge x = 1\right\} \quad \{\texttt{load}^1\ r_1 \mapsto y\} \quad \left\{r_1 = 2 \wedge x = 1\right\}}{\left\{?\right\} \quad G_2 \quad \left\{r_1 = 2 \wedge x = 1\right\}}$$

Semantics of programs with shared memories

$$\frac{}{\texttt{load}^i\ r \leftarrow x, \langle \sigma, s\rangle \Downarrow \langle \sigma[r := \langle\!|x|\!\rangle_\sigma], s\rangle}$$

$$\frac{}{\texttt{store}^i\ x \leftarrow e, \langle \sigma, s\rangle \Downarrow \langle \sigma, s[x := \langle\!|e|\!\rangle_\sigma]\rangle}$$

$$\frac{G, \langle \sigma, s\rangle \Downarrow \langle \sigma', s'\rangle \qquad \langle \sigma', s'\rangle \Rightarrow \langle \sigma'', s''\rangle}{G, \langle \sigma, s\rangle \Downarrow \langle \sigma'', s''\rangle}$$

The $\Rightarrow$ on $\langle \sigma, s\rangle$ is defined as the smallest reflexive and transitive closure that contains $\langle \sigma, s\rangle \Rightarrow \langle \sigma[s \upharpoonright \{x\}], s \setminus \{x\}\rangle$.

$$\frac{G \setminus \{C\}, \langle \sigma, s\rangle \Downarrow \langle \sigma', s'\rangle \qquad \{C\}, \langle \sigma', s'\rangle \Downarrow \langle \sigma'', s''\rangle \qquad C \text{ is not } \ldots}{G, \langle \sigma, s\rangle \Downarrow \langle \sigma'', s''\rangle}$$

Hoare logic for dependence graphs

Define a dependence graph from a program and an MCM.

$$E ::= r \mid x \mid \overline{x}$$

$$\Phi ::= E = E \mid E \leq E \mid \neg\, \Phi \mid \Phi \supset \Phi \mid \forall\, r.\, \Phi \mid \forall\, x.\, \Phi \mid \forall\, \overline{x}.\, \Phi$$

$$\frac{}{\{[x/r]\Phi\}\ \texttt{load}^{i}\ r \leftarrow x\ \{\Phi\}}$$

$$\frac{}{\{[e/\overline{x}](\Phi \wedge [e/x]\Phi)\}\ \texttt{store}^{i}\ x \leftarrow e\ \{\Phi\}}$$

$$\frac{\forall\, C \in L(G) \qquad \{\Phi\}\ G \setminus C\ \{\Upsilon\} \qquad \{\Upsilon\}\ C\ \{\Psi\}}{\{\Phi\}\ G\ \{\Psi\}}$$

It is sound and relatively complete to the semantics.

Related work

[Owicki & Gries 1976] gives a Hoare logic for parallel programs.

[Jones 1981] gives a compositional Hoare logic by using the so-called rely/guarantee method.

[O'Hearn 2007] gives a separation logic for concurrent programs with shared memories

[Kojima & Igarashi 2013] gives a Hoare logic for Single Instruction Multiple Data (SIMD) programs.

Summary

- Propose a base model on which we discuss MCMs,
- define a set of formulas to describe MCMs,
  - confirm possible to write Itanium and UPC MCMs.
- develop a model checker generator that takes an MCM, and
- demonstrate some experiments.

- Give semantics of programs with shared memories,
- define MCMs as translations from programs into graphs, and
- give sound and relatively complete Hoare logic for graphs.