Weak Memory Models: an Operational Theory

Gérard Boudol and Gustavo Petri

INRIA Sophia Antipolis

9th June 2008
Memory models, what are they good for?

- Hardware optimizations
- Contract between hardware and software
- Defines which values can be read from the memory
- Semantics of concurrency
Memory models, what do they specify?

- Define memory actions (events)
- Ordering constraints
- Visibility constraints
- Atomicity constraints
- Determines allowed optimizations
Memory models, how are they specified?

- (Lamport 79)
  
  ... result of any execution is the same as if the operations of all the processors were executed ...  

- (Java Memory Model 2005)

  An action \( a \) is described by a tuple \( \langle t, k, v, u \rangle \), comprising: \( t \) - the thread performing the action ...

- (Intel 64 2007)

  Stores are not reordered with older loads

<table>
<thead>
<tr>
<th>Processor 0</th>
<th>Processor 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov r1, [x] // M1</td>
<td>mov r2, [y] // M3</td>
</tr>
<tr>
<td>mov [y], 1 // M2</td>
<td>mov [x], 1 // M4</td>
</tr>
</tbody>
</table>

Initially \( x == y == 0 \)
\r
\( r1 == 1 \) and \( r1 == 1 \) is not allowed
An operational theory

Standard memory models build around
- single processor ordering
- happens before order (Lamport78)
- rules to restrict reordering of instructions

Our proposal: **Standard Operational Semantics Techniques**
- a small step interleaving semantics
- a small step weak semantics (write buffers)
- **true concurrency** techniques to define: events, conflict, concurrency, dependency
- bisimulation to prove DRF guarantee
A simple calculus

\[ e ::= v \mid (e_0 e_1) \mid (\text{ref } e) \mid (!e) \mid (e_0 := e_1) \mid (\text{thread } e) \mid (\text{with } l \text{ do } e) \]

\[ v ::= x \mid (\lambda x e) \mid () \]

\[ r ::= (\lambda x e_0 e_1) \mid (!a) \mid (a := v) \]

\[ E ::= [] \mid E[F] \]

\[ F ::= ([] e) \mid ([v]) \]

\[ \quad \mid ([\text{ref}]) \mid (![]) \mid ([] := e) \]

\[ \quad \mid ([v] := []) \mid (\text{holding } l \text{ do } []) \]
Weak Memory Models: an Operational Theory

Semantics

Strong Semantics: Interleaving

\[(S, L, T[E[(\lambda x e)]])) \rightarrow (S, L, T[E[x \mapsto v]e_0])\]

\[(S, L, T[E[\text{ref } v]]) \rightarrow (S\{p \mapsto v\}, L, T[E[x]]) \quad p \notin \text{dom}(S)\]

\[(S, L, T[E[(!p)]])) \rightarrow (S, L, T[E[v]]) \quad S(p) = v\]

\[(S, L, T[E[(p := v)]])) \rightarrow (S\{p \mapsto v\}, L, T[E[\text{()}]])\]

\[(S, L, T[E[\text{thread } e]]) \rightarrow (S, L, T[E[\text{()} \parallel e]])\]

\[(S, L, T[E[(\text{with } l \text{ do } e)]])) \rightarrow (S, L \cup \{l\}, T[E[\text{(holding } l \text{ do } e)]])) \quad l \notin L\]

\[(S, L, T[E[(\text{holding } l \text{ do } v)]])) \rightarrow (S, L - \{l\}, T[E[v]])\]
Adding Write Buffers
Adding Write Buffers

\[ e_0 \quad \quad e_1 \]
Weak Semantics: Write buffers

\[(S, L, B[E[(!p)]] \Rightarrow (S, L, B[E[v]]) \quad [B](p) = \epsilon, S(p) = v\]

\[(S, L, B[E[(p := v)]] \Rightarrow (S, L, B[E[(\{p \mapsto v\}, E[()])]])\]

\[(S, L, B[E[(\text{holding } l \text{ do } v)]] \Rightarrow (S, L - \{l\}, B[E[v]])\]

\[B^\dagger (S, L, B[(b, (b', B))]) \Rightarrow (S, L, B[(\text{put}(b, p, v), (\text{pop}(b', p), B))])\]

\[p \in \text{dom}(b'), b'(p) = v.q\]

\[(S, L, (b, B)) \Rightarrow (S[\{p \mapsto v\}], L, (\text{pop}(b, x), B))\]

\[p \in \text{dom}(b), b(p) = v.q\]
Racy example: Weak behavior

\[
a := 1; \; \text{!}b \parallel b := 1; \; \text{!}a
\]

\[
\begin{array}{c}
b \mapsfrom 0 \\
a \mapsfrom 0
\end{array}
\]
Racy example: Weak behavior

\[ a := 1; \! b \parallel b := 1; \! a \]

\[
\begin{array}{c}
b \mapsto 0 \\
a \mapsto 0
\end{array}
\]

\[
\begin{array}{c}
a := 1; \! b \\
\end{array}
\]

\[
\begin{array}{c}
b \mapsto 1 \\
\! a
\end{array}
\]
Racy example: Weak behavior

\[ a := 1; \ !b \parallel b := 1; \ !a \]

\[
\begin{array}{c}
b \mapsto 0 \\
a \mapsto 0
\end{array}
\]

\[
\begin{array}{c}
a \mapsto 1 \\
!b
\end{array}
\]

\[
\begin{array}{c}
b \mapsto 1 \\
!a
\end{array}
\]
Racy example: Weak behavior

\[ a := 1; \, !b \quad \| \quad b := 1; \, !a \]

\[
\begin{array}{c}
b \mapsto 0 \\
\hline
a \mapsto 0
\end{array}
\]
Racy example: Weak behavior

\[ a := 1; !b \parallel b := 1; !a \]

\[
\begin{array}{c}
b \rightarrow 0 \\
\hline
a \rightarrow 0
\end{array}
\]

\[
\begin{array}{c}
a \rightarrow 1 \\
0
\end{array} \quad \begin{array}{c}
b \rightarrow 1 \\
0
\end{array}
\]
Correctly Synchronized: Strong behavior

```
with l do (a := 1; !b) || with l do (b := 1; !a)
```

- $b \mapsto 0$
- $a \mapsto 0$

```
with l do (a := 1; !b)  
with l do (b := 1; !a)
```

$a \mapsto 0$

$b \mapsto 0$
Correctly Synchronized: Strong behavior

$$\text{with } l \text{ do } (a := 1; !b) \parallel \text{with } l \text{ do } (b := 1; !a)$$

- $b \mapsto 0$
- $a \mapsto 0$

- with $l \text{ do } (a := 1; !b)$
- holding $l \text{ do } 0$

- $b \mapsto 1$
Correctly Synchronized: Strong behavior

\[ \text{with } l \text{ do } (a := 1; !b) \parallel \text{with } l \text{ do } (b := 1; !a) \]

\[
\begin{array}{c}
b \mapsto 1 \\
a \mapsto 0
\end{array}
\]

with \(l\) do \((a := 1; !b)\) \hspace{1cm} \text{holding } l \text{ do } 0

\(a \mapsto 0\)
Correctly Synchronized: Strong behavior

\[
\text{with } l \text{ do}(a := 1; !b) \parallel \text{with } l \text{ do}(b := 1; !a)
\]

\[
\begin{array}{c}
  b \mapsto 1 \\
  a \mapsto 0
\end{array}
\]
Correctly Synchronized: Strong behavior

\[
\text{with } l \text{ do}(a := 1; !b) \parallel \text{with } l \text{ do}(b := 1; !a)
\]

\[
\begin{array}{c}
b \mapsto 1 \\
a \mapsto 0
\end{array}
\]

\[
\text{holding } l \text{ do} 1 \\
0
\]
Correctly Synchronized: Strong behavior

\[
\text{with } l \text{ do}\,(a := 1; !b) \parallel \text{with } l \text{ do}(b := 1; !a)
\]

\[
\begin{array}{c}
b \mapsto 1 \\
a \mapsto 1
\end{array}
\]

holding \( l \) do 1

0
Correctly Synchronized: Strong behavior

\[ \text{with } l \text{ do } (a : = 1; !b) \parallel \text{with } l \text{ do } (b : = 1; !a) \]

\[
\begin{array}{c}
\begin{array}{c}
b \mapsto 1 \\
a \mapsto 1
\end{array}
\end{array}
\]

1

0
Dekker’s mutual exclusion

\[\text{flag}_1 := \text{ff} ; \]
\[\text{if} \ (\neg \text{flag}_2) \ \text{then} \]
\[\text{Critical Section} \quad \text{Critical Section} \]

\[\text{flag}_2 := \text{ff} ; \]
\[\text{if} \ (\neg \text{flag}_1) \ \text{then} \]
Dekker’s mutual exclusion

\[ flag_1 := ff; \]
\[ \text{if } (!flag_2) \text{ then } \]
\[ \text{Critical Section} \]
\[ \text{Not Safe} \]
\[ flag_2 := ff; \]
\[ \text{if } (!flag_1) \text{ then } \]
\[ \text{Critical Section} \]
Publication

\[
data := 8; \quad \text{if } (!flag_1) \text{ then } \quad r_1 := (!data)
\]
data := 8; \quad \text{if} \ (\neg \text{flag}_1) \ \text{then} \quad r_1 := (!\text{data})

Not safe: \ r_1 \neq 8
Some definitions

- **Correctly Synchronized**: A program is correctly synchronized if all its strong executions are free of data races.
- **DRF Guarantee**: If a program is correctly synchronized, all its behaviors in the weak semantics are sequentially consistent.
- **Our approach to DRF**: The strong and weak semantics are bisimilar for correctly synchronized programs.
Concurrently conflicting events are dependent in every execution.

So we need to define:

- Event
- Conflict of events ($\#$)
- Concurrency of events ($\sim$)
- Dependency of events ($<$)
We want to derive events from the semantics

- Annotate the semantics with actions
  - Action type
  - Who performed the action
Strong Semantics: annotated

\[
\begin{align*}
(S, L, T[E[(\lambda xev)]] & \xrightarrow{\beta} (@T) (S, L, T[E[x \mapsto v]e_0]]) \\
(S, L, T[E[(\text{ref } v)]] & \xrightarrow{\nu_p} (@T) (S \{ p \mapsto v \}, L, T[E[x]])) \quad \text{if } p \notin \text{dom}(S) \\
(S, L, T[E[(!x)]] & \xrightarrow{\text{rd}_p} (@T) (S, L, T[E[v]]) \\
(S, L, T[E[(p := v)]] & \xrightarrow{\text{wr}_p} (@T) (S \{ p \mapsto v \}, L, T[E[()]]) \\
(S, L, T[E[(\text{with } l \text{ do } e)]] & \xrightarrow{l} (@T) (S, L \cup \{l\}, T[E[(\text{holding } l \text{ do } e)]])) \quad l \notin L \\
(S, L, T[E[(\text{holding } l \text{ do } v)]] & \xrightarrow{l} (@T) (S, L - \{l\}, T[E[v]])
\end{align*}
\]
Weak Semantics: annotated

\[(S, L, B[E[(!p)]])) \sim_{\text{rd}_p}^{\text{rd}} \atop \to \]
\[(S, L, B[E[(p : = v)]])) \sim_{\text{wr}_p}^{\text{wr}} \atop \to \]
\[(S, L, B[E[(\text{holding } l \text{ do } v)]])) \sim_{\text{l}}^{\text{l}} \atop \to \]
\[(S, L, (b, (b', B))) \]]\]

\[(S, L, B[E[(\text{holding } l \text{ do } v)]])) \sim_{\text{rd}_p}^{\text{rd}} \atop \to \]
\[(S, L, B[E[v]])\]

\[(S, L, B[E[(p : = v)]])) \sim_{\text{wr}_p}^{\text{wr}} \atop \to \]
\[(S, L, B[E[(\{p \mapsto v\}, E[()])]])\]

\[(S, L, B[E[(\text{holding } l \text{ do } v)]])) \sim_{\text{l}}^{\text{l}} \atop \to \]
\[(S, L - \{l\}, B[E[v]])\]

\[B^\dagger\]

\[(S, L, B[(\text{put}(b, p, v), (\text{pop}(b', p), B))])\]

\[p \in \text{dom}(b'), b'(p) = v.q\]

\[(S\{p \mapsto v\}, L, (\text{pop}(b, x), B)))\]

\[p \in \text{dom}(b), b(p) = v.q\]
True concurrency: Event

- We derive events from the semantics
- Annotate the semantics with actions
  - Action type
  - Which thread performed it
- Events: \((a, o)_i\)
  - \(a\) is the action
  - \(o\) is the occurrence in the pool (i.e. Thread ID)
  - \(i\) is the number of repetitions of that action in the execution so far
True concurrency: Conflict

Standard definition:

- Actions on the same memory location (at least one write)
  - $\text{wr}_p \# \text{rd}_p$
  - $\text{wr}_p \# \text{wr}_p$

- Locking actions on the same lock
  - $\sim l \# \sim l$
  - $\sim l \# \sim l$
  - $\sim l \# \sim l$
True concurrency: Concurrency

Two occurrences are concurrent if neither is prefix of the other: $o \bowtie o'$
True concurrency: Asynchrony

If we have two events \((a, o)_i, (a, o)_j\) such that

- \(C \xrightarrow{a_1}{o_1} C_1 \xrightarrow{a_2}{o_2} C_2\)
- \(a_1\) and \(a_2\) are not conflicting
- \(o_1\) and \(o_2\) are concurrent

then there is a unique configuration \(C'_1\) such that

\[ C \xrightarrow{a_2}{o_2} C'_1 \xrightarrow{a_1}{o_1} C_2 \]
True concurrency: Asynchrony

If we have two events \((a, o)_i, (a, o)_j\) such that

- \(a_1\) and \(a_2\) are not conflicting
- \(o_1\) and \(o_2\) are concurrent

then there is a unique configuration \(C'_1\) such that

\[
\begin{align*}
&C \xrightarrow{a_1}{o_1} C_1 \xrightarrow{a_2}{o_2} C_2 \\
&\text{Equivalence by Permutation: } C \xrightarrow{a_1}{o_1} C_1 \xrightarrow{a_2}{o_2} C_2 \simeq C \xrightarrow{a_2}{o_2} C_1 \xrightarrow{a_1}{o_1} C_2
\end{align*}
\]
True concurrency: Ordering

\((a, o)_i \leq_{\gamma} (a', o')_j\)

In any execution \(\gamma'\) such that \(\gamma' \simeq \gamma\), we have that \((a, o)_i\) precedes \((a', o')_j\).
The DRF guarantee

- We define all these concepts for both semantics
- We define a relation between configurations
- We prove that our relation is a bisimulation for correctly synchronized programs
Conclusions

- A strong and a **weak** Small Step Operational semantics
- Instantiation of true concurrency definition to memory models
- **Bisimulation** proof of DRF for the weak semantics

**Interesting observation**
- The proof carries over if we allow **reading from buffers** (Cache)

**Future work**
- Formalize the proofs in Coq
- Experiment with different flavors of the weak semantics