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 where 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

Processor 0	*Processor 1*
`mov r1, [_ x] // M1` `mov [_ y], 1 // M2`	`mov r2, [_ y] // M3` `mov [_ x], 1 // M4`
Initially `x == y == 0` `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

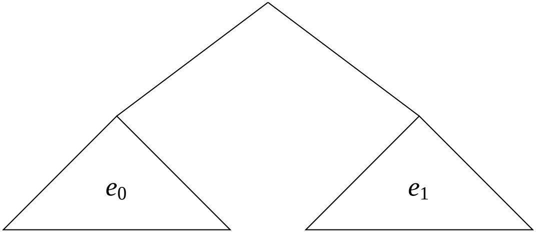
$$
\begin{array}{lcll}
e & ::= & v \mid (e_0 e_1) & (\mathit{expressions}) \\
 & \mid & (\mathrm{ref}\, e) \mid (!e) \mid (e_0 := e_1) & \\
 & \mid & (\mathrm{thread}\, e) \mid (\mathrm{with}\, l\, \mathrm{do}\, e) & \\
v & ::= & x \mid (\lambda x e) \mid () & (\mathit{values}) \\
r & ::= & (\lambda x e_0 e_1) \mid (!\, a) \mid (a := v) & (\mathit{redexes})
\end{array}
$$

$$
\begin{array}{lcll}
\mathbf{E} & ::= & [] \mid \mathbf{E}[\mathbf{F}] & (\mathit{evaluation\ contexts}) \\
\mathbf{F} & ::= & ([]e) \mid (v[]) & (\mathit{frames}) \\
 & \mid & (\mathrm{ref}[]) \mid (![]) \mid ([] := e) & \\
 & \mid & (v := []) \mid (\mathrm{holding}\, l\, \mathrm{do}[]) &
\end{array}
$$

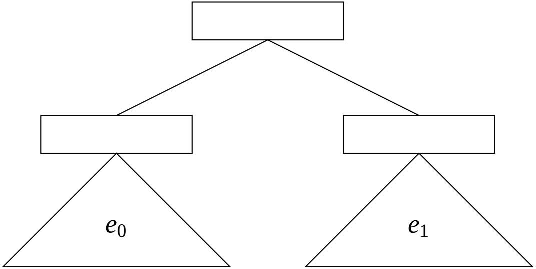
Strong Semantics: Interleaving

$$
\begin{array}{rcll}
(S, L, \mathbf{T}[\mathbf{E}[(\lambda x e v)]]) & \longrightarrow & (S, L, \mathbf{T}[\mathbf{E}[\{x \mapsto v\}e_0]]) & \\
(S, L, \mathbf{T}[\mathbf{E}[(\mathrm{ref}\, v)]]) & \longrightarrow & (S\{p \mapsto v\}, L, \mathbf{T}[\mathbf{E}[x]]) & p \notin dom(S) \\
(S, L, \mathbf{T}[\mathbf{E}[(!p)]]) & \longrightarrow & (S, L, \mathbf{T}[\mathbf{E}[v]]) & S(p) = v \\
(S, L, \mathbf{T}[\mathbf{E}[(p := v)]]) & \longrightarrow & (S\{p \mapsto v\}, L, \mathbf{T}[\mathbf{E}[()]]) & \\
(S, L, \mathbf{T}[\mathbf{E}[(\mathrm{thread}\ \ e)]]) & \longrightarrow & (S, L, \mathbf{T}[(\mathbf{E}[()] \parallel e)]) & \\
(S, L, \mathbf{T}[\mathbf{E}[(\mathrm{with}\, l\, \mathrm{do}\, e)]]) & \longrightarrow & (S, L \cup \{l\}, \mathbf{T}[\mathbf{E}[(\mathrm{holding}\, l\, \mathrm{do}\, e)]]) & l \notin L \\
(S, L, \mathbf{T}[\mathbf{E}[(\mathrm{holding}\, l\, \mathrm{do}\, v)]]) & \longrightarrow & (S, L - \{l\}, \mathbf{T}[\mathbf{E}[v]]) &
\end{array}
$$

Adding Write Buffers



Adding Write Buffers

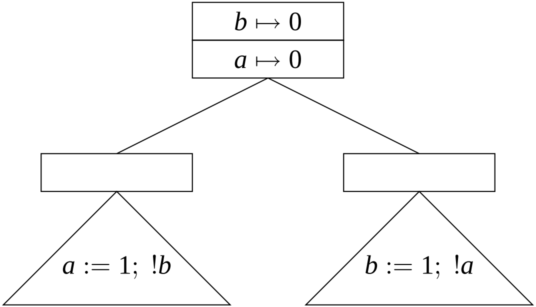


Weak Semantics: Write buffers

$$
\begin{array}{rcl}
(S, L, \mathbf{B}[\mathbf{E}[(!p)]]) & \rightsquigarrow & (S, L, \mathbf{B}[\mathbf{E}[v]]) \\
 & & \lfloor \mathbf{B} \rfloor (p) = \epsilon, S(p) = v \\
(S, L, \mathbf{B}[\mathbf{E}[(p := v)]]) & \rightsquigarrow & (S, L, \mathbf{B}[\mathbf{E}[(\{p \mapsto v\}, \mathbf{E}[()])]]) \\
(S, L, \mathbf{B}[\mathbf{E}[(\mathrm{holding}\, l\, \mathrm{do}\, v)]]) & \rightsquigarrow & (S, L - \{l\}, \mathbf{B}[\mathbf{E}[v]]) \\
 & & \mathbf{B}\dagger \\
(S, L, \mathbf{B}[(b, (b', B))]) & \rightsquigarrow & (S, L, \mathbf{B}[(put(b, p, v), (pop(b', p), B))]) \\
 & & p \in dom(b'), b'(p) = v.q \\
(S, L, (b, B)) & \rightsquigarrow & (S\{p \mapsto v\}, L, (pop(b, x), B)) \\
 & & p \in dom(b), b(p) = v.q
\end{array}
$$

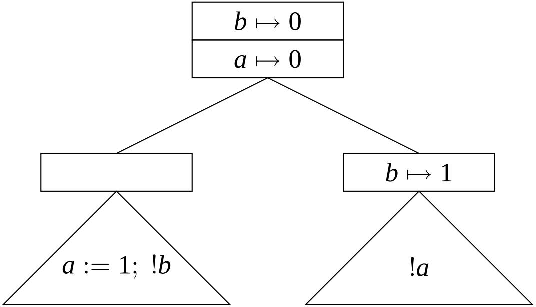
Racy example: Weak behavior

$$a := 1;\ !b \ \| \ b := 1;\ !a$$



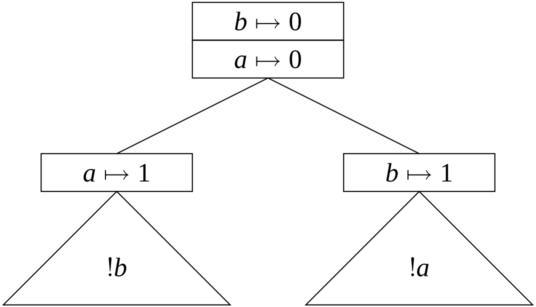
Racy example: Weak behavior

$$a := 1;\ !b \parallel b := 1;\ !a$$



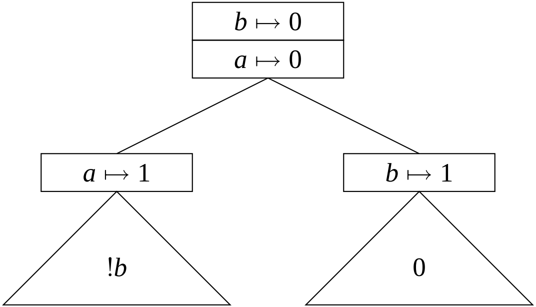
Racy example: Weak behavior

$$a := 1;\ !b \ \parallel\ b := 1;\ !a$$



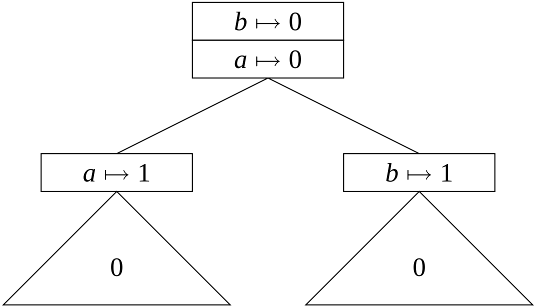
Racy example: Weak behavior

$$a := 1;\ !b \ \| \ b := 1;\ !a$$



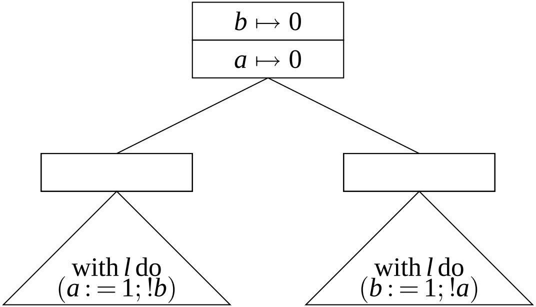
Racy example: Weak behavior

$$a := 1;\ !b \;\|\; b := 1;\ !a$$



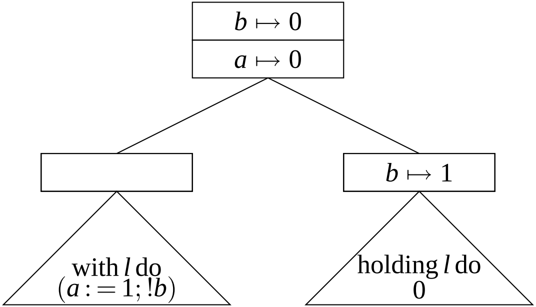
Correctly Synchronized: Strong behavior

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



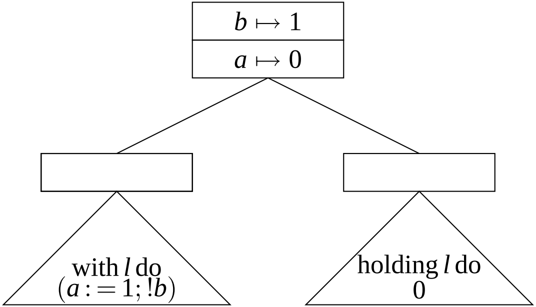
Correctly Synchronized: Strong behavior

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



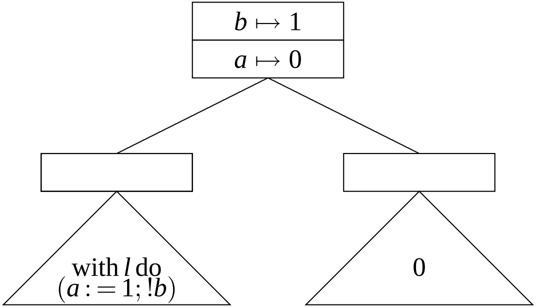
Correctly Synchronized: Strong behavior

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



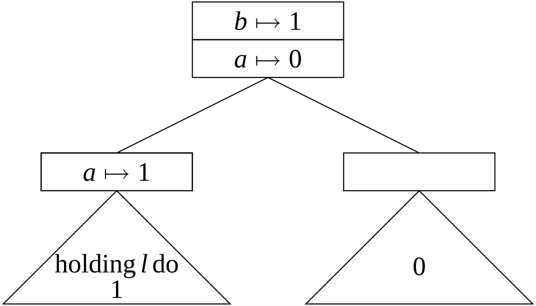
Correctly Synchronized: Strong behavior

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



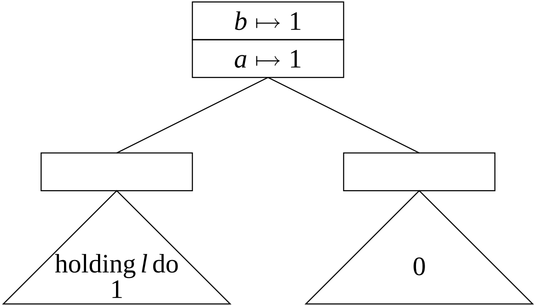
Correctly Synchronized: Strong behavior

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



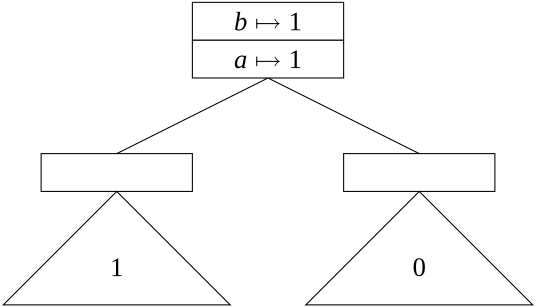
Correctly Synchronized: Strong behavior

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



Correctly Synchronized: Strong behavior

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



Dekker's mutual exclusion

$$
\begin{array}{l|l}
\mathit{flag}_1 := \mathit{ff}\,; & \mathit{flag}_2 := \mathit{ff}\,; \\
\text{if } (!\mathit{flag}_2) \text{ then} & \text{if } (!\mathit{flag}_1) \text{ then} \\
\quad \texttt{Critical Section} & \quad \texttt{Critical Section}
\end{array}
$$

Dekker's mutual exclusion

$$
\begin{array}{l||l}
\mathit{flag}_1 := \mathit{ff}\,; & \mathit{flag}_2 := \mathit{ff}\,; \\
\text{if } (!\mathit{flag}_2) \text{ then} & \text{if } (!\mathit{flag}_1) \text{ then} \\
\quad \texttt{Critical Section} & \quad \texttt{Critical Section}
\end{array}
$$

Not Safe

Publication

$$\begin{array}{l} data := 8; \\ flag_1 := ff \end{array} \;\Big\|\; \begin{array}{l} \text{if } (!flag_1) \text{ then} \\ \quad r_1 := (!data) \end{array}$$

Publication

$$\begin{array}{l} \mathit{data} := 8; \\ \mathit{flag}_1 := \mathit{ff} \end{array} \;\Big\|\; \begin{array}{l} \text{if } (!\mathit{flag}_1) \text{ then} \\ \quad r_1 := (!\mathit{data}) \end{array}$$

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

Correctly Synchronized Programs

CS key property

Concurrent conflicting events are dependent in every execution.

So we need to define:

- Event
- Conflict of events ($\#$)
- Concurrency of events ($\smile$)
- Dependency of events ($<$)

True concurrency: Event

- We want to derive events from the semantics
- Annotate the semantics with actions
  - Action type
  - Who performed the action

Strong Semantics: annotated

$$
\begin{array}{rcll}
(S, L, \mathbf{T}[\mathbf{E}[(\lambda x e v)]]) & \xrightarrow[@\mathbf{T}]{\beta} & (S, L, \mathbf{T}[\mathbf{E}[\{x \mapsto v\}e_0]]) & \\
(S, L, \mathbf{T}[\mathbf{E}[(\mathrm{ref}\, v)]]) & \xrightarrow[@\mathbf{T}]{\nu_p} & (S\{p \mapsto v\}, L, \mathbf{T}[\mathbf{E}[x]]) & p \notin dom(S) \\
(S, L, \mathbf{T}[\mathbf{E}[(!x)]]) & \xrightarrow[@\mathbf{T}]{\mathrm{rd}_p} & (S, L, \mathbf{T}[\mathbf{E}[v]]) & S(p) = v \\
(S, L, \mathbf{T}[\mathbf{E}[(p := v)]]) & \xrightarrow[@\mathbf{T}]{\mathrm{wr}_p} & (S\{p \mapsto v\}, L, \mathbf{T}[\mathbf{E}[()]]) & \\
(S, L, \mathbf{T}[\mathbf{E}[(\mathrm{thread}\ \ e)]]) & \xrightarrow[@\mathbf{T}]{\mathrm{spw}} & (S, L, \mathbf{T}[(\mathbf{E}[()] \parallel e)]) & \\
(S, L, \mathbf{T}[\mathbf{E}[(\mathrm{with}\, l\, \mathrm{do}\, e)]]) & \xrightarrow[@\mathbf{T}]{\overset{\curvearrowright}{l}} & (S, L \cup \{l\}, \mathbf{T}[\mathbf{E}[(\mathrm{holding}\, l\, \mathrm{do}\, e)]]) & l \notin L \\
(S, L, \mathbf{T}[\mathbf{E}[(\mathrm{holding}\, l\, \mathrm{do}\, v)]]) & \xrightarrow[@\mathbf{T}]{\overset{\curvearrowleft}{l}} & (S, L - \{l\}, \mathbf{T}[\mathbf{E}[v]]) &
\end{array}
$$

Weak Semantics: annotated

$$
\begin{array}{rcl}
(S, L, \mathbf{B}[\mathbf{E}[(!p)]]) & \underset{@\mathbf{B}}{\overset{\mathrm{rd}_p}{\leadsto}} & (S, L, \mathbf{B}[\mathbf{E}[v]]) \\
& & \lfloor \mathbf{B} \rfloor (p) = \epsilon, S(p) = v \\
(S, L, \mathbf{B}[\mathbf{E}[(p := v)]]) & \underset{@\mathbf{B}}{\overset{\mathrm{wr}_p}{\leadsto}} & (S, L, \mathbf{B}[\mathbf{E}[(\{p \mapsto v\}, \mathbf{E}[()])]]) \\
(S, L, \mathbf{B}[\mathbf{E}[(\mathrm{holding}\, l \,\mathrm{do}\, v)]]) & \underset{@\mathbf{B}}{\overset{\curvearrowleft l}{\leadsto}} & (S, L - \{l\}, \mathbf{B}[\mathbf{E}[v]]) \\
& & \mathbf{B}\dagger \\
(S, L, \mathbf{B}[(b, (b', B))]) & \underset{@\mathbf{B}}{\leadsto} & (S, L, \mathbf{B}[(put(b, p, v), (pop(b', p), B))]) \\
& & p \in dom(b'),\ b'(p) = v.q \\
(S, L, (b, B)) & \leadsto & (S\{p \mapsto v\}, L, (pop(b, x), B)) \\
& & p \in dom(b),\ b(p) = v.q
\end{array}
$$

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

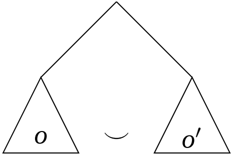
True concurrency: Conflict

Standard definition:

- Actions on the same memory location (at least one write)
  - $\mathrm{wr}_p \# \mathrm{rd}_p$
  - $\mathrm{wr}_p \# \mathrm{wr}_p$
- Locking actions on the same lock
  - $\overset{\curvearrowright}{l} \# \overset{\curvearrowright}{l}$
  - $\overset{\curvearrowright}{l} \# \overset{\curvearrowleft}{l}$
  - $\overset{\curvearrowleft}{l} \# \overset{\curvearrowleft}{l}$

True concurrency: Concurrency

Two occurrences are concurrent if neither is prefix of the other: $o \smile o'$



True concurrency: Asynchrony

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

- $C \xrightarrow[o_1]{a_1} C_1 \xrightarrow[o_2]{a_2} C_2$
- 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

- $C \xrightarrow[o_2]{a_2} C_1' \xrightarrow[o_1]{a_1} C_2$

$$\begin{tikzcd}
 & C_0 \arrow[dl, dash, "a_1"'] \arrow[dr, dash, "a_2"] & \\
C_1 \arrow[dr, dash, "a_2"'] & & C_1' \arrow[dl, dash, "a_1"] \\
 & C_2 &
\end{tikzcd}$$

True concurrency: Asynchrony

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

- $C \xrightarrow[o_1]{a_1} C_1 \xrightarrow[o_2]{a_2} C_2$
- 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

- $C \xrightarrow[o_2]{a_2} C_1' \xrightarrow[o_1]{a_1} C_2$

Equivalence by Permutation: $C \xrightarrow[o_1]{a_1} C_1 \xrightarrow[o_2]{a_2} C_2 \simeq C \xrightarrow[o_2]{a_2} C_1' \xrightarrow[o_1]{a_1} C_2$

True concurrency: Ordering

$(a, o)_i \leq_\gamma (a', o')_j$

In any execution γ' 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