Towards a Grainless Semantics for Shared Variable Concurrency

John C. Reynolds

Department of Computer Science
Carnegie Mellon University
and
BRICS
Aarhus University

POPL'04 — January 15, 2004
(corrected January 20)

john.reynolds@cs.cmu.edu

The Problem

What is the meaning of

$$\mathsf{x} := \mathsf{x} \times \mathsf{x} \parallel \mathsf{x} := \mathsf{x} + 1\,?$$

Are the assignment commands atomic, so that it is either

$$\mathsf{x} := \mathsf{x} \times \mathsf{x}\,;\mathsf{x} := \mathsf{x} + 1 \quad \text{or} \quad \mathsf{x} := \mathsf{x} + 1\,;\mathsf{x} := \mathsf{x} \times \mathsf{x}\,?$$

or are the two sides of each assignment command atomic:

$$(\mathsf{t}_1 := \mathsf{x}\,;\mathsf{x} := \mathsf{t}_1 \times \mathsf{t}_1) \parallel (\mathsf{t}_2 := \mathsf{x}\,;\mathsf{x} := \mathsf{t}_2 + 1)\,?$$

or is each lookup or assignment atomic:

$$(\mathsf{t}_1 := \mathsf{x}\,;\mathsf{t}_2 := \mathsf{x}\,;\mathsf{x} := \mathsf{t}_1 \times \mathsf{t}_2) \parallel (\mathsf{t}_3 := \mathsf{x}\,;\mathsf{x} := \mathsf{t}_3 + 1)\,?$$

or is the granularity even finer:

$$\begin{array}{l}(\mathsf{t}_1{}^{\mathrm{low}} := \mathsf{x}^{\mathrm{low}}\,;\mathsf{t}_1{}^{\mathrm{up}} := \mathsf{x}^{\mathrm{up}}\,;\mathsf{t}_2{}^{\mathrm{low}} := \mathsf{x}^{\mathrm{low}}\,;\mathsf{t}_2{}^{\mathrm{up}} := \mathsf{x}^{\mathrm{up}}\,; \\ \qquad \mathsf{x}^{\mathrm{low}} := (\mathsf{t}_1 \times \mathsf{t}_2)^{\mathrm{low}}\,;\mathsf{x}^{\mathrm{up}} := (\mathsf{t}_1 \times \mathsf{t}_2)^{\mathrm{up}}) \parallel \\ \quad (\mathsf{t}_3{}^{\mathrm{low}} := \mathsf{x}^{\mathrm{low}}\,;\mathsf{t}_3{}^{\mathrm{up}} := \mathsf{x}^{\mathrm{up}}\,;\mathsf{x}^{\mathrm{low}} := (\mathsf{t}_3 + 1)^{\mathrm{low}}\,;\mathsf{x}^{\mathrm{up}} := (\mathsf{t}_3 + 1)^{\mathrm{up}})\,?\end{array}$$

An Early Answer

In the early 70's, Hoare and Brinch-Hansen claimed that constructions such as

$$\mathsf{x} := \mathsf{x} \times \mathsf{x} \parallel \mathsf{x} := \mathsf{x} + 1$$

should be syntactically illegal.

Instead, when the same variable appears on both sides of $\parallel$, the programmer should be required to indicate the appropriate mutual exclusion explicitly by means of critical regions.

For example,

$$\mathbf{with}\ \mathsf{lock}\ \mathbf{do}\ \mathsf{x} := \mathsf{x} \times \mathsf{x} \parallel \mathbf{with}\ \mathsf{lock}\ \mathbf{do}\ \mathsf{x} := \mathsf{x} + 1$$

or

$$(\mathbf{with}\ \mathsf{lock}\ \mathbf{do}\ \mathsf{t}_1 := \mathsf{x}\,;\ \mathbf{with}\ \mathsf{lock}\ \mathbf{do}\ \mathsf{x} := \mathsf{t}_1 \times \mathsf{t}_1) \parallel$$
$$(\mathbf{with}\ \mathsf{lock}\ \mathbf{do}\ \mathsf{t}_2 := \mathsf{x}\,;\ \mathbf{with}\ \mathsf{lock}\ \mathbf{do}\ \mathsf{x} := \mathsf{t}_2 + 1).$$

The Harder Problem

What about lookup and assignment via pointers,

$$[\mathsf{x}] := [\mathsf{x}] \times [\mathsf{x}] \parallel [\mathsf{y}] := [\mathsf{y}] + 1,$$

where aliasing cannot be decided by a compiler.

Our Answer

When the addresses x and y are equal, the meaning of the above program is simply "wrong".

No further information makes sense at any level of abstraction above the machine-language implementation.

Caveat Emptor

This is a preliminary report on work in progress (and the progress has gone more slowly than I'd hoped).

It has not been carefully refereed (not even by the author).

References

[1] David M. R. Park. On the semantics of fair parallelism. In Dines Bjørner, editor, *Abstract Software Specifications*, volume 86 of *Lecture Notes in Computer Science*, pages 504–526, Berlin, 1980. Springer-Verlag.

[2] Stephen D. Brookes. Full abstraction for a shared-variable parallel language. *Information and Computation*, 127(2):145–163, June 15, 1996.

[3] Stephen D. Brookes. Semantics and separation logic for parallel programs with shared mutable data. Preliminary draft dated 7-25-03, to be presented at CONCUR'04, 2004.

Three Principles for Grainless Concurrency

- All operations except locking and unlocking have duration, and can overlap one another during execution.
- If two overlapping operations lookup or set the same location, the meaning of program execution is “wrong”.
- If, from a given starting state, execution of a program can give “wrong”, then no other possibilities need be considered.

A Simple Example

The meaning $[\![\mathsf{x} := \mathsf{x} \times \mathsf{x}]\!]$ is the set of traces

$$\mathbf{start}[\,\mathsf{x}\colon n\,]\,\mathbf{fin}[\,\mathsf{x}\colon n \times n\,],$$

for all integers n, which includes the trace

$$\mathbf{start}[\,\mathsf{x}\colon 3\,]\,\mathbf{fin}[\,\mathsf{x}\colon 9\,].$$

If $\mathsf{x} := \mathsf{x} \times \mathsf{x}$ is not running concurrently with other processes, its traces can be executed (without interruption).

For example, starting in the state $[\,\mathsf{x}\colon 3 \mid \mathsf{t}\colon 22\,]$, the above trace has the execution

$$\begin{array}{ll}
[\,\mathsf{x}\colon 3 \mid \mathsf{t}\colon 22\,] & \\
 & \mathbf{start}[\,\mathsf{x}\colon 3\,] \\
[\,\mathsf{t}\colon 22\,] & \\
 & \mathbf{fin}[\,\mathsf{x}\colon 9\,] \\
[\,\mathsf{x}\colon 9 \mid \mathsf{t}\colon 22\,]. &
\end{array}$$

On the other hand, starting in the state $[\,\mathsf{x}\colon 4 \mid \mathsf{t}\colon 22\,]$, the trace

$$\mathbf{start}[\,\mathsf{x}\colon 3\,]\,\mathbf{fin}[\,\mathsf{x}\colon 9\,]$$

has no execution. But an execution is provided by another trace, $\mathbf{start}[\,\mathsf{x}\colon 4\,]\,\mathbf{fin}[\,\mathsf{x}\colon 16\,]$ in $[\![\mathsf{x} := \mathsf{x} \times \mathsf{x}]\!]$.

An Example of Concurrency

The meaning $[\![[\mathsf{x}] := [\mathsf{x}] \times [\mathsf{x}]]\!]$ is the set of traces

$$\mathbf{start}[\,\mathsf{x}\colon n_1 \mid n_1\colon n_2\,]\,\mathbf{fin}[\,\mathsf{x}\colon n_1 \mid n_1\colon n_2 \times n_2\,],$$

for all integers n_1 and n_2. Similarly, the meaning $[\![[\mathsf{y}] := [\mathsf{y}] + 1]\!]$ is the set of traces

$$\mathbf{start}[\,\mathsf{y}\colon n_3 \mid n_3\colon n_4\,]\,\mathbf{fin}[\,\mathsf{y}\colon n_3 \mid n_3\colon n_4 + 1\,],$$

for all integers n_3 and n_4.

The meaning $[\![[\mathsf{x}]{:=}[\mathsf{x}]{\times}[\mathsf{x}] \parallel [\mathsf{y}]{:=}[\mathsf{y}]{+}1]\!]$ is the set of interleavings of the traces of each subcommand. This set includes traces of the form

$$\begin{aligned}&\mathbf{start}[\,\mathsf{x}\colon n_1 \mid n_1\colon n_2\,]\,\mathbf{start}[\,\mathsf{y}\colon n_3 \mid n_3\colon n_4\,]\\ &\quad\mathbf{fin}[\,\mathsf{y}\colon n_3 \mid n_3\colon n_4 + 1\,]\,\mathbf{fin}[\,\mathsf{x}\colon n_1 \mid n_1\colon n_2 \times n_2\,],\end{aligned}$$

in which the two assignment operations overlap.

An Example of Concurrency (continued)

When $n_1 \neq n_3$, there is no interference, and we have the execution

$$
\begin{array}{ll}
[\,\mathsf{x}\colon n_1 \mid n_1\colon n_2 \mid \mathsf{y}\colon n_3 \mid n_3\colon n_4\,] & \\
& \mathbf{start}[\,\mathsf{x}\colon n_1 \mid n_1\colon n_2\,] \\
[\,\mathsf{y}\colon n_3 \mid n_3\colon n_4\,] & \\
& \mathbf{start}[\,\mathsf{y}\colon n_3 \mid n_3\colon n_4\,] \\
[\,] & \\
& \mathbf{fin}[\,\mathsf{y}\colon n_3 \mid n_3\colon n_4 + 1\,] \\
[\,\mathsf{y}\colon n_3 \mid n_3\colon n_4 + 1\,] & \\
& \mathbf{fin}[\,\mathsf{x}\colon n_1 \mid n_1\colon n_2 \times n_2\,] \\
[\,\mathsf{y}\colon n_3 \mid n_3\colon n_4 + 1 \mid \mathsf{x}\colon n_1 \mid n_1\colon n_2 \times n_2\,]. &
\end{array}
$$

On the other hand, when $n_1 = n_3$ (which implies $n_2 = n_4$), the two processes interfere, as shown by the execution

$$
\begin{array}{ll}
[\,\mathsf{x}\colon n_1 \mid n_1\colon n_2 \mid \mathsf{y}\colon n_1\,] & \\
& \mathbf{start}[\,\mathsf{x}\colon n_1 \mid n_1\colon n_2\,] \\
[\,\mathsf{y}\colon n_1\,] & \\
& \mathbf{start}[\,\mathsf{y}\colon n_1 \mid n_1\colon n_2\,] \\
\text{wrong.} &
\end{array}
$$

Critical Regions (following Brookes)

The meaning $[\![\textbf{with}\ \mathsf{k}\ \textbf{do}\ \mathsf{x} := \mathsf{x} \times \mathsf{x}]\!]$ is the set of traces (for all integers n):

$$\mathbf{acq}(\mathsf{k})\ \mathbf{start}[\,\mathsf{x}\colon n\,]\ \mathbf{fin}[\,\mathsf{x}\colon n \times n\,]\ \mathbf{rel}(\mathsf{k}),$$
$$\mathbf{try}(\mathsf{k})\ \mathbf{acq}(\mathsf{k})\ \mathbf{start}[\,\mathsf{x}\colon n\,]\ \mathbf{fin}[\,\mathsf{x}\colon n \times n\,]\ \mathbf{rel}(\mathsf{k}),$$
$$\mathbf{try}(\mathsf{k})\ \mathbf{try}(\mathsf{k})\ \mathbf{acq}(\mathsf{k})\ \mathbf{start}[\,\mathsf{x}\colon n\,]\ \mathbf{fin}[\,\mathsf{x}\colon n \times n\,]\ \mathbf{rel}(\mathsf{k}),$$
$$\vdots$$
$$\mathbf{try}(\mathsf{k})\ \mathbf{try}(\mathsf{k})\ \mathbf{try}(\mathsf{k})\ \cdots$$

Execution is conditioned by a set κ of "closed" locks. When $\mathsf{k} \notin \kappa$ (and n is 3), only a trace of the first form has an execution:

$$\begin{array}{ll}
\kappa, [\,\mathsf{x}\colon 3 \mid \mathsf{t}\colon 22\,] & \\
& \mathbf{acq}(\mathsf{k}) \\
\kappa \cup \{\mathsf{k}\}, [\,\mathsf{x}\colon 3 \mid \mathsf{t}\colon 22\,] & \\
& \mathbf{start}[\,\mathsf{x}\colon 3\,] \\
\kappa \cup \{\mathsf{k}\}, [\,\mathsf{t}\colon 22\,] & \\
& \mathbf{fin}[\,\mathsf{x}\colon 9\,] \\
\kappa \cup \{\mathsf{k}\}, [\,\mathsf{x}\colon 9 \mid \mathsf{t}\colon 22\,] & \\
& \mathbf{rel}(\mathsf{k}) \\
\kappa, [\,\mathsf{x}\colon 9 \mid \mathsf{t}\colon 22\,]. &
\end{array}$$

On the other hand, when $\mathsf{k} \in \kappa$, only the last trace has an execution, which is nonterminating, and represents deadlock:

$$\begin{array}{ll}
\kappa, [\,\mathsf{x}\colon 3 \mid \mathsf{t}\colon 22\,] & \\
& \mathbf{try}(\mathsf{k}) \\
\kappa, [\,\mathsf{x}\colon 3 \mid \mathsf{t}\colon 22\,] & \\
& \mathbf{try}(\mathsf{k}) \\
\vdots &
\end{array}$$

Critical Regions (continued)

The remaining traces in $[\![\textbf{with}\ \mathsf{k}\ \textbf{do}\ \mathsf{x} := \mathsf{x} \times \mathsf{x}]\!]$ can only execute successfully after being interleaved with other traces that affect the same lock. For example, one possible merge of

$$\textbf{try}(\mathsf{k})\ \textbf{try}(\mathsf{k})\ \textbf{acq}(\mathsf{k})\ \textbf{start}[\,\mathsf{x}\colon 3\,]\ \textbf{fin}[\,\mathsf{x}\colon 9\,]\ \textbf{rel}(\mathsf{k})$$

with

$$\textbf{acq}(\mathsf{k})\ \textbf{start}[\,\mathsf{x}\colon 2\,]\ \textbf{fin}[\,\mathsf{x}\colon 3\,]\ \textbf{rel}(\mathsf{k})$$

is

$$\textbf{acq}(\mathsf{k})\ \textbf{start}[\,\mathsf{x}\colon 2\,]\ \textbf{try}(\mathsf{k})\ \textbf{fin}[\,\mathsf{x}\colon 3\,]\ \textbf{try}(\mathsf{k})$$
$$\textbf{rel}(\mathsf{k})\ \textbf{acq}(\mathsf{k})\ \textbf{start}[\,\mathsf{x}\colon 3\,]\ \textbf{fin}[\,\mathsf{x}\colon 9\,]\ \textbf{rel}(\mathsf{k}),$$

which executes as follows when $\mathsf{k} \notin \kappa$:

$$\begin{array}{llll}
\kappa, [\,\mathsf{x}\colon 2\,] & & \kappa \cup \{\mathsf{k}\}, [\,\mathsf{x}\colon 3\,] & \\
& \textbf{acq}(\mathsf{k}) & & \textbf{rel}(\mathsf{k}) \\
\kappa \cup \{\mathsf{k}\}, [\,\mathsf{x}\colon 2\,] & & \kappa, [\,\mathsf{x}\colon 3\,] & \\
& \textbf{start}[\,\mathsf{x}\colon 2\,] & & \textbf{acq}(\mathsf{k}) \\
\kappa \cup \{\mathsf{k}\}, [\,] & & \kappa \cup \{\mathsf{k}\}, [\,\mathsf{x}\colon 3\,] & \\
& \textbf{try}(\mathsf{k}) & & \textbf{start}[\,\mathsf{x}\colon 3\,] \\
\kappa \cup \{\mathsf{k}\}, [\,] & & \kappa \cup \{\mathsf{k}\}, [\,] & \\
& \textbf{fin}[\,\mathsf{x}\colon 3\,] & & \textbf{fin}[\,\mathsf{x}\colon 9\,] \\
\kappa \cup \{\mathsf{k}\}, [\,\mathsf{x}\colon 3\,] & & \kappa \cup \{\mathsf{k}\}, [\,\mathsf{x}\colon 9\,] & \\
& \textbf{try}(\mathsf{k}) & & \textbf{rel}(\mathsf{k}) \\
& & \kappa, [\,\mathsf{x}\colon 9\,]. &
\end{array}$$

The Programming Language

$$\langle\text{exp}\rangle ::= \langle\text{var}\rangle \mid \langle\text{constant}\rangle \mid \langle\text{exp}\rangle + \langle\text{exp}\rangle \mid [\langle\text{exp}\rangle] \mid \cdots$$

$$\langle\text{boolexp}\rangle ::= \langle\text{exp}\rangle = \langle\text{exp}\rangle \mid \cdots \mid \langle\text{boolexp}\rangle \wedge \langle\text{boolexp}\rangle \mid \cdots$$

$$\begin{aligned}
\langle\text{comm}\rangle ::= {} & \langle\text{var}\rangle := \langle\text{exp}\rangle \\
& \mid [\langle\text{exp}\rangle] := \langle\text{exp}\rangle \\
& \mid \mathbf{skip} \\
& \mid \langle\text{comm}\rangle \,;\, \langle\text{comm}\rangle \\
& \mid \mathbf{if}\ \langle\text{boolexp}\rangle\ \mathbf{then}\ \langle\text{comm}\rangle\ \mathbf{else}\ \langle\text{comm}\rangle \\
& \mid \mathbf{while}\ \langle\text{boolexp}\rangle\ \mathbf{do}\ \langle\text{comm}\rangle \\
& \mid \langle\text{comm}\rangle \parallel \langle\text{comm}\rangle \\
& \mid \mathbf{with}\ \langle\text{lock}\rangle\ \mathbf{do}\ \langle\text{comm}\rangle \\
& \mid \mathbf{with}\ \langle\text{lock}\rangle\ \mathbf{when}\ \langle\text{boolexp}\rangle\ \mathbf{do}\ \langle\text{comm}\rangle
\end{aligned}$$

States

$$
\begin{aligned}
\text{Addresses} &= \text{Integers} \\
\text{Locations} &= \text{Variables} \uplus \text{Addresses} \\
\text{States} &= \biguplus \{\, \text{States}_\delta \mid \delta \overset{\text{fin}}{\subseteq} \text{Locations} \,\} \\
\text{States}_\delta &= \delta \to \text{Integers}
\end{aligned}
$$

Metavariables

$$
\begin{aligned}
n &: \quad \text{Integers} \\
\ell &: \quad \text{Locations} \\
\delta &: \quad \text{Finite Sets of Locations} \\
\sigma &: \quad \text{States} \\
k &: \quad \text{Locks} \\
\kappa &: \quad \text{Finite Sets of Locks} \\
\tau &: \quad \text{Traces} \\
T &: \quad \text{Sets of Traces}
\end{aligned}
$$

Concepts and Notations for States

The *restriction* of a state σ to a set of locations $\delta \subseteq \operatorname{dom}\sigma$ is written $\sigma\rceil\delta$.

States σ and σ' are *compatible*, written $\sigma \smile \sigma'$, iff $\sigma \cup \sigma'$ is a function, or equivalently, iff

$$\sigma\rceil(\operatorname{dom}\sigma \cap \operatorname{dom}\sigma') = \sigma'\rceil(\operatorname{dom}\sigma \cap \operatorname{dom}\sigma').$$

We also write $\delta \perp \delta'$ when the sets δ and δ' are disjoint, and $\sigma \perp \sigma'$ when $\operatorname{dom}\sigma \perp \operatorname{dom}\sigma'$.

We write

- $[\,]$ for the function such that

$$\operatorname{dom}[\,] = \{\},$$

- $[\,\ell\colon n\,]$ for the function such that

$$\begin{aligned} \operatorname{dom}[\,\ell\colon n\,] &= \{\ell\} \\ [\,\ell\colon n\,](\ell) &= n, \end{aligned}$$

- $[\,\sigma \mid \ell\colon n\,]$ for the function such that

$$\begin{aligned} \operatorname{dom}[\,\sigma \mid \ell\colon n\,] &= \operatorname{dom}\sigma \cup \{\ell\} \\ [\,\sigma \mid \ell\colon n\,](\ell) &= n \\ [\,\sigma \mid \ell\colon n\,](\ell') &= \sigma(\ell') \text{ when } \ell \neq \ell'. \end{aligned}$$

Semantics of Expressions

$$\llbracket \langle \text{exp} \rangle \rrbracket \subseteq \text{States} \times \text{Integers}$$

$$\begin{aligned}
\llbracket 7 \rrbracket &= \{ \langle [\,], 7 \rangle \} \\
\llbracket v \rrbracket &= \{ \langle [\, v\colon n \,], n \rangle \mid n \in \text{Integers} \} \\
\llbracket e + e' \rrbracket &= \{ \langle \sigma \cup \sigma', n + n' \rangle \mid \\
&\qquad\qquad \langle \sigma, n \rangle \in \llbracket e \rrbracket, \langle \sigma', n' \rangle \in \llbracket e' \rrbracket \text{ and } \sigma \smile \sigma' \} \\
\llbracket [e] \rrbracket &= \{ \langle \sigma \cup [\, n\colon n' \,], n' \rangle \mid \\
&\qquad\qquad \langle \sigma, n \rangle \in \llbracket e \rrbracket, n \in \text{Integers and } \sigma \smile [\, n\colon n' \,] \}
\end{aligned}$$

$$\llbracket \langle \text{boolexp} \rangle \rrbracket \subseteq \text{States} \times \text{Bool}$$

$$\begin{aligned}
\llbracket e = e' \rrbracket &= \{ \langle \sigma \cup \sigma', n = n' \rangle \mid \\
&\qquad\qquad \langle \sigma, n \rangle \in \llbracket e \rrbracket, \langle \sigma', n' \rangle \in \llbracket e' \rrbracket \text{ and } \sigma \smile \sigma' \} \\
\llbracket e \wedge e' \rrbracket &= \{ \langle \sigma \cup \sigma', b = b' \rangle \mid \\
&\qquad\qquad \langle \sigma, b \rangle \in \llbracket e \rrbracket, \langle \sigma', b' \rangle \in \llbracket e' \rrbracket \text{ and } \sigma \smile \sigma' \}
\end{aligned}$$

Totality

For all σ, there are σ' and n such that $\sigma \smile \sigma'$ and $\langle \sigma', n \rangle \in \llbracket e \rrbracket$.

Traces

$$\begin{aligned}\langle\text{action}\rangle ::= {} & \mathbf{start}(\langle\text{state}\rangle) \mid \mathbf{fin}(\langle\text{state}\rangle) \\ & \mid \mathbf{try}(\langle\text{lock}\rangle) \mid \mathbf{acq}(\langle\text{lock}\rangle) \mid \mathbf{rel}(\langle\text{lock}\rangle)\end{aligned}$$

A *trace* is either a finite or infinite sequence of actions. We write ϵ for the empty trace.

Concatenation of Traces

$$\tau_1 \,;\, \tau_2 = \begin{cases} \tau_1\,\tau_2 & \text{if } \tau_1 \text{ is finite} \\ \tau_1 & \text{if } \tau_1 \text{ is infinite.} \end{cases}$$

Concatenation of Trace Sets

$$\begin{aligned} T \,;\, T' &= \{\, \tau \,;\, \tau' \mid \tau \in T, \tau' \in T' \,\} \\ T^0 &= \{\epsilon\} \\ T^{n+1} &= T \,;\, T^n \\ T^* &= \textstyle\bigcup_{n=0}^{\infty} T^n \\ T^\omega &= \{\, \tau_0 \,;\, \tau_1 \,;\, \cdots \mid \forall i \geq 0.\ \tau_i \in T \,\}. \end{aligned}$$

Filtering

$$\text{filter}(\langle\text{boolexp}\rangle) \subseteq \text{Traces}$$

$$\text{filter}(b) = \{\, \mathbf{start}(\sigma)\, \mathbf{fin}(\sigma) \mid \langle \sigma, \mathbf{true} \rangle \in [\![b]\!] \,\}$$

Totality

For all σ there is a σ' such that $\sigma \smile \sigma'$ and

$$\mathbf{start}(\sigma')\, \mathbf{fin}(\sigma') \in \text{filter}(b) \cup \text{filter}(\neg\, b).$$

Semantics of Sequential Commands

$$\llbracket \langle \text{comm} \rangle \rrbracket \subseteq \text{Traces}$$

$$\begin{aligned}\llbracket v := e \rrbracket = \{\, &\mathbf{start}(\sigma \cup [\, v\colon n_{\text{old}} \,])\, \mathbf{fin}([\, \sigma \mid v\colon n \,]) \mid \\ &\langle \sigma, n \rangle \in \llbracket e \rrbracket, \sigma \smile [\, v\colon n_{\text{old}} \,] \,\}\end{aligned}$$

$$\begin{aligned}\llbracket [e] := e' \rrbracket = \{\, &\mathbf{start}(\sigma \cup \sigma' \cup [\, n\colon n_{\text{old}} \,])\, \mathbf{fin}([\, \sigma \cup \sigma' \mid n\colon n' \,]) \mid \\ &\langle \sigma, n \rangle \in \llbracket e \rrbracket, \langle \sigma', n' \rangle \in \llbracket e' \rrbracket, \sigma \smile \sigma', (\sigma \cup \sigma') \smile [\, n\colon n_{\text{old}} \,] \,\}\end{aligned}$$

$$\llbracket \mathbf{skip} \rrbracket = \{\mathbf{start}[\,]\, \mathbf{fin}[\,]\}$$

$$\llbracket c_1 \,;\, c_2 \rrbracket = \llbracket c_1 \rrbracket \,;\, \llbracket c_2 \rrbracket$$

$$\begin{aligned}&\llbracket \mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \rrbracket \\ &\qquad = (\text{filter}(b) \,;\, \llbracket c_1 \rrbracket) \cup (\text{filter}(\neg\, b) \,;\, \llbracket c_2 \rrbracket)\end{aligned}$$

$$\llbracket \mathbf{while}\ b\ \mathbf{do}\ c \rrbracket = \big((\text{filter}(b) \,;\, \llbracket c \rrbracket)^* \,;\, \text{filter}(\neg\, b)\big) \cup (\text{filter}(b) \,;\, \llbracket c \rrbracket)^\omega$$

Concurrent Composition

$$[\![c_1 \parallel c_2]\!] = \bigcup \{\, \tau_1 \parallel \tau_2 \mid \tau_1 \in [\![c_1]\!], \tau_2 \in [\![c_2]\!] \,\}.$$

Here $\tau_1 \parallel \tau_2$ denotes the set of *fair merges* of the traces τ_1 and τ_2, i.e., the merges (or interleavings) that contain every occurrence of actions in τ_1 and τ_2 (even when τ_1 or τ_2 is infinite).

To make this concept precise, we regard a trace as a function whose domain is the finite or infinite set of nonnegative numbers less than the length of the trace.

Then τ is a *fair merge* of τ_1 and τ_2 iff there are functions ϕ_1 and ϕ_2 such that

- $\mathrm{dom}\,\phi_1 = \mathrm{dom}\,\tau_1$ and $\mathrm{dom}\,\phi_2 = \mathrm{dom}\,\tau_2$.
- ϕ_1 and ϕ_2 are strictly monotone.
- The ranges of ϕ_1 and ϕ_2 are a partition of $\mathrm{dom}\,\tau$.
- For all $i \in \mathrm{dom}\,\tau_1$, $\tau_1(i) = \tau(\phi_1(i))$.
- For all $i \in \mathrm{dom}\,\tau_2$, $\tau_2(i) = \tau(\phi_2(i))$.

Critical Regions

$$[\![\mathbf{with}\ k\ \mathbf{do}\ c]\!] = \Big(\{\mathbf{try}(k)\}^* \,;\, \{\mathbf{acq}(k)\} \,;\, [\![c]\!] \,;\, \{\mathbf{rel}(k)\}\Big) \cup \{\mathbf{try}(k)\}^\omega$$

$$[\![\mathbf{with}\ k\ \mathbf{when}\ b\ \mathbf{do}\ c]\!] =$$
$$\Big(\mathrm{wait}^* \,;\, \{\mathbf{try}(k)\}^* \,;\, \{\mathbf{acq}(k)\} \,;\, \mathrm{filter}(b) \,;\, [\![c]\!] \,;\, \{\mathbf{rel}(k)\}\Big) \cup \mathrm{wait}^\omega$$

where

$$\mathrm{wait} = \Big(\{\mathbf{try}(k)\}^* ; \{\mathbf{acq}(k)\} ; \mathrm{filter}(\neg\, b) ; \{\mathbf{rel}(k)\}\Big) \cup \{\mathbf{try}(k)\}^\omega.$$

Total Trace Sets

A set T of traces is said to be *total* whenever, if

$$\tau \, \mathbf{start}(\sigma_0) \, \tau' \in T,$$

holds for some state σ_0, then for every state σ there is a trace

$$\tau \, \mathbf{start}(\sigma') \, \tau'' \in T,$$

such that $\sigma \smile \sigma'$.

A set T of traces is *lock-total* whenever, if

$$\tau \, \mathbf{acq}(k) \, \tau' \in T,$$

then there is a trace

$$\tau \, \mathbf{try}(k) \, \tau'' \in T.$$

For all commands c, $[\![c]\!]$ is total and lock-total.

Executing Traces ("Weak Wrongness" Interpretation)

We describe the execution of traces by a small-step operational semantics:

$$\rightsquigarrow \subseteq \text{NT-Config} \times (\text{NT-Config} \cup \text{T-Config})$$

$$\text{NT-Config} = \text{Finite Sets of Locks} \times \text{States} \times \text{Traces}$$

$$\text{T-Config} = (\text{Finite Sets of Locks} \times \text{States}) \cup \{\text{wrong}, \text{impossible}\}$$

$$\kappa, \sigma, \epsilon \rightsquigarrow \kappa, \sigma$$

$$\kappa, \sigma, \mathbf{start}(\sigma')\,\tau \rightsquigarrow \begin{cases} \kappa, \sigma - \sigma', \tau & \text{if } \sigma' \subseteq \sigma \\ \text{wrong} & \text{if } \sigma \smile \sigma' \text{ and } \sigma' \not\subseteq \sigma \end{cases}$$

$$\kappa, \sigma, \mathbf{fin}(\sigma')\,\tau \rightsquigarrow \begin{cases} \kappa, \sigma \cup \sigma', \tau & \text{if } \sigma' \perp \sigma \\ \text{impossible} & \text{otherwise} \end{cases}$$

$$\kappa, \sigma, \mathbf{try}(k)\,\tau \rightsquigarrow \kappa, \sigma, \tau \quad \text{if } k \in \kappa$$

$$\kappa, \sigma, \mathbf{acq}(k)\,\tau \rightsquigarrow \kappa \cup \{k\}, \sigma, \tau \quad \text{if } k \notin \kappa$$

$$\kappa, \sigma, \mathbf{rel}(k)\,\tau \rightsquigarrow \begin{cases} \kappa - \{k\}, \sigma, \tau & \text{if } k \in \kappa \\ \text{impossible} & \text{otherwise.} \end{cases}$$

Note that there are no transitions for:

$$\kappa, \sigma, \mathbf{start}(\sigma')\,\tau \quad \text{if } \sigma \not\smile \sigma'$$

$$\kappa, \sigma, \mathbf{try}(k)\,\tau \quad \text{if } k \notin \kappa$$

$$\kappa, \sigma, \mathbf{acq}(k)\,\tau \quad \text{if } k \in \kappa.$$

Weak versus Strong Wrongness

Consider a transition from the configuration $\kappa, \sigma, \mathbf{start}(\sigma')$. If we partition σ' into

$$\sigma'_{\mathrm{in}} = \sigma' \rceil (\mathrm{dom}\, \sigma' \cap \mathrm{dom}\, \sigma) \quad \text{and} \quad \sigma'_{\mathrm{out}} = \sigma' \rceil (\mathrm{dom}\, \sigma' - \mathrm{dom}\, \sigma),$$

then we can distinguish four cases:

- If $\sigma'_{\mathrm{in}} \smile \sigma$, and σ'_{out} is empty, then
$$\kappa, \sigma, \mathbf{start}(\sigma') \leadsto \kappa, \sigma - \sigma', \tau.$$
- If $\sigma'_{\mathrm{in}} \smile \sigma$, and σ'_{out} is not empty, then
$$\kappa, \sigma, \mathbf{start}(\sigma') \leadsto \text{wrong}.$$
- If $\sigma'_{\mathrm{in}} \not\smile \sigma$, and σ'_{out} is empty, then
$$\kappa, \sigma, \mathbf{start}(\sigma') \text{ has no transition.}$$
- If $\sigma'_{\mathrm{in}} \not\smile \sigma$, and σ'_{out} is not empty, then
  - in the *weak wrongness* interpretation,
  $$\kappa, \sigma, \mathbf{start}(\sigma') \text{ has no transition,}$$
  - in the *strong wrongness* interpretation,
  $$\kappa, \sigma, \mathbf{start}(\sigma') \leadsto \text{wrong}.$$

For example:

$$\begin{array}{rl} \text{Weak wrongness:} & \kappa, [\,\mathsf{x}\colon 0\,], \mathbf{start}[\,\mathsf{x}\colon 1 \mid \mathsf{y}\colon 0\,]\,\tau \text{ has no transitions} \\ \text{Strong wrongness:} & \kappa, [\,\mathsf{x}\colon 0\,], \mathbf{start}[\,\mathsf{x}\colon 1 \mid \mathsf{y}\colon 0\,]\,\tau \leadsto \text{wrong}. \end{array}$$

Executing Traces ("Strong Wrongness" Interpretation)

$$\kappa, \sigma, \epsilon \rightsquigarrow \kappa, \sigma$$

$$\kappa, \sigma, \mathbf{start}(\sigma')\,\tau \rightsquigarrow \begin{cases} \kappa, \sigma - \sigma', \tau & \text{if } \sigma' \subseteq \sigma \\ \text{wrong} & \boxed{\text{if } \operatorname{dom} \sigma' \not\subseteq \operatorname{dom} \sigma} \end{cases}$$

$$\kappa, \sigma, \mathbf{fin}(\sigma')\,\tau \rightsquigarrow \begin{cases} \kappa, \sigma \cup \sigma', \tau & \text{if } \sigma' \perp \sigma \\ \text{impossible} & \text{otherwise} \end{cases}$$

$$\kappa, \sigma, \mathbf{try}(k)\,\tau \rightsquigarrow \kappa, \sigma, \tau \qquad \text{if } k \in \kappa$$

$$\kappa, \sigma, \mathbf{acq}(k)\,\tau \rightsquigarrow \kappa \cup \{k\}, \sigma, \tau \qquad \text{if } k \notin \kappa$$

$$\kappa, \sigma, \mathbf{rel}(k)\,\tau \rightsquigarrow \begin{cases} \kappa - \{k\}, \sigma, \tau & \text{if } k \in \kappa \\ \text{impossible} & \text{otherwise.} \end{cases}$$

Note that there are no transitions for:

$$\kappa, \sigma, \mathbf{start}(\sigma')\,\tau \quad \boxed{\text{if } \operatorname{dom} \sigma' \subseteq \operatorname{dom} \sigma \text{ and } \sigma' \not\subseteq \sigma}$$

$$\kappa, \sigma, \mathbf{try}(k)\,\tau \quad \text{if } k \notin \kappa$$

$$\kappa, \sigma, \mathbf{acq}(k)\,\tau \quad \text{if } k \in \kappa.$$

Balanced Traces

For all commands c, traces $\tau \in [\![c]\!]$, and states σ,

$$\{\}, \sigma, \tau \not\rightsquigarrow^* \text{ impossible.}$$

The Execution Function

$$\begin{aligned}
&\mathrm{Exec}(\kappa, \sigma, T) = \\
&\quad \{\, \langle \kappa', \sigma' \rangle \mid \exists \tau \in T.\ \kappa, \sigma, \tau \rightsquigarrow^* \kappa', \sigma' \,\} \\
&\quad \cup \,\mathbf{if}\ \kappa, \sigma, \tau \rightsquigarrow^* \text{wrong}\ \mathbf{then}\ \text{Wrong}\ \mathbf{else}\ \{\} \\
&\quad \cup \,\mathbf{if}\ \kappa, \sigma, \tau\ \text{diverges}\ \mathbf{then}\ \{\bot\}\ \mathbf{else}\ \{\},
\end{aligned}$$

where Wrong is the set of all terminal configurations, including "wrong". (Thus union preserves Wrong.)

Equivalence of Trace Sets

We write $T_1 \cong T_2$ iff, for all κ, σ, and trace sets T' that are total and lock-total,

$$\mathrm{Exec}(\kappa, \sigma, T_1 \parallel T') = \mathrm{Exec}(\kappa, \sigma, T_2 \parallel T').$$

A Problem with Equivalence under Weak Wrongness

Under the weak wrongness interpretation, the trace sets

$$\begin{aligned} T_1 &= \{\, \mathbf{start}[\,\mathsf{x}\colon m\,]\, \mathbf{fin}[\,\mathsf{x}\colon 0\,]\, \mathbf{start}[\,\mathsf{y}\colon n\,]\, \mathbf{fin}[\,\mathsf{y}\colon 0\,] \mid m, n \in \text{Integers}\,\} \\ &= [\![\mathsf{x} := 0\,;\, \mathsf{y} := 0]\!] \end{aligned}$$

$$T_2 = \{\, \mathbf{start}[\,\mathsf{x}\colon m \mid \mathsf{y}\colon n\,]\, \mathbf{fin}[\,\mathsf{x}\colon 0 \mid \mathsf{y}\colon 0\,] \mid m, n \in \text{Integers}\,\}$$

are not equivalent, since, when

$$\begin{aligned} \sigma &= [\,\mathsf{x}\colon 0 \mid \mathsf{y}\colon 0\,] \\ T' &= \{\mathbf{start}[\,\mathsf{x}\colon 1 \mid \mathsf{y}\colon 1\,]\, \mathbf{fin}[\,\mathsf{x}\colon 0 \mid \mathsf{y}\colon 0\,], \mathbf{start}[\,]\, \mathbf{fin}[\,]\}, \end{aligned}$$

we have

$$\begin{aligned} \mathrm{Exec}(\kappa, \sigma, T_1 \parallel T') &= \mathrm{Exec}(\kappa, \sigma, T_1) \\ \mathrm{Exec}(\kappa, \sigma, T_2 \parallel T') &= \text{Wrong}. \end{aligned}$$

A Problem with Equivalence under Strong Wrongness

Under the strong wrongness interpretation, the trace sets

$$\begin{aligned} T_1 &= \{\, \mathbf{start}[\,\mathsf{x}\colon 0\,]\,\mathbf{fin}[\,\mathsf{x}\colon 0\,]\,\mathbf{start}[\,\mathsf{y}\colon n\,]\,\mathbf{fin}[\,\mathsf{y}\colon 0\,] \mid n \in \text{Integers}\,\} \\ &\qquad \cup \{\, \mathbf{start}[\,\mathsf{x}\colon m\,]\,\mathbf{fin}[\,\mathsf{x}\colon m\,] \mid m \in \text{Integers and } m \neq 0\,\} \\ &= [\![\mathbf{if}\ \mathsf{x} = 0\ \mathbf{then}\ \mathsf{y} := 0\ \mathbf{else\ skip}]\!] \end{aligned}$$

$$\begin{aligned} T_2 &= \{\, \mathbf{start}[\,\mathsf{x}\colon 0 \mid \mathsf{y}\colon n\,]\,\mathbf{fin}[\,\mathsf{x}\colon 0 \mid \mathsf{y}\colon 0\,] \mid n \in \text{Integers}\,\} \\ &\qquad \cup \{\, \mathbf{start}[\,\mathsf{x}\colon m\,]\,\mathbf{fin}[\,\mathsf{x}\colon m\,] \mid m \in \text{Integers and } m \neq 0\,\} \end{aligned}$$

are not equivalent, since, when $\sigma = [\,\mathsf{x}\colon 1\,]$ and $T' = \{\epsilon\}$, we have

$$\begin{aligned} \mathrm{Exec}(\kappa, \sigma, T_1 \parallel T') &= \kappa, [\,\mathsf{x}\colon 1\,] \\ \mathrm{Exec}(\kappa, \sigma, T_2 \parallel T') &= \text{Wrong}. \end{aligned}$$

Future Directions

- Straighten out the notion of equivalence of trace sets.
- Introduce passivity (i.e., allow read-only interference).
- Use to model separation logic; relate to Brookes' model.