A Model of Cooperative Threads

Martín Abadi   Gordon Plotkin

Microsoft Research, Silicon Valley
LFCS, University of Edinburgh

Domains IX, University of Sussex 2008

Outline

Cooperative Threads and AME

- Cooperative Threads run without interruption until they yield control.
- Interest in such threads has increased recently with the introduction of Automatic Mutual Exclusion (AME) and the problem of programming multicore systems.

What we do

- We describe a simple language for cooperative threads and give it a mathematically elementary fully abstract (may) semantics of sets of traces, being *transition sequences* of, roughly, the form:

$$u = (\sigma_1, \sigma'_1) \ldots (\sigma_m, \sigma'_m)$$

  à la Abrahamson, the authors, Brookes etc, but adapted to incorporate thread spawning.
- Following the algebraic theory of effects, we characterise the semantics using a suitable inequational theory, thereby relating it to standard domain-theoretic notions of resumptions.

Syntax

$$
\begin{array}{rcrcl}
b & \in & \mathrm{BExp} & = & \ldots \\
e & \in & \mathrm{NExp} & = & \ldots \\
C, D & \in & \mathrm{Com} & = & \texttt{skip} \\
 & & & \mid & x := e \qquad (x \in \mathrm{Vars}) \\
 & & & \mid & C; D \\
 & & & \mid & \texttt{if } b \texttt{ then } C \texttt{ else } D \\
 & & & \mid & \texttt{while } b \texttt{ do } C \\
 & & & \mid & \texttt{async } C \\
 & & & \mid & \texttt{yield} \\
 & & & \mid & \texttt{block}
\end{array}
$$

Example

$$\begin{array}{l} \texttt{async}\ x := 0; \\ x := 1; \\ \texttt{yield}; \\ \texttt{if}\ x = 0\ \texttt{then}\ x := 2\ \texttt{else}\ \texttt{block} \end{array}$$

This spawns the asynchronous execution of $x := 0$, executes $x := 1$, yields, then resumes but blocks unless the predicate $x = 0$ holds, then executes $x := 2$

With respect to safety properties, the conditional blocking amounts to awaiting that $x = 0$ holds. So The last line may be paraphrased as

$$\texttt{await}\ x = 0;\ x := 2$$

Operational Semantics

$$
\begin{array}{lcl}
\langle \sigma, T, \mathcal{E}[x := e] \rangle & \longrightarrow_a & \langle \sigma[x \mapsto \sigma(e)], T, \mathcal{E}[\mathtt{skip}] \rangle \\
\langle \sigma, T, \mathcal{E}[\mathtt{skip}; C] \rangle & \longrightarrow_a & \langle \sigma, T, \mathcal{E}[C] \rangle \\
\langle \sigma, T, \mathcal{E}[\mathtt{if}\ b\ \mathtt{then}\ C\ \mathtt{else}\ D] \rangle & \longrightarrow_a & \langle \sigma, T, \mathcal{E}[C] \rangle \\
 & & \qquad (\text{if } \sigma(b) = \mathtt{true}) \\
\langle \sigma, T, \mathcal{E}[\mathtt{while}\ b\ \mathtt{do}\ C] \rangle & \longrightarrow_a & \langle \sigma, T, \mathcal{E}[C; \mathtt{while}\ b\ \mathtt{do}\ C] \rangle \\
 & & \qquad (\text{if } \sigma(b) = \mathtt{true}) \\
\langle \sigma, T, \mathcal{E}[\mathtt{async}\ C] \rangle & \longrightarrow_a & \langle \sigma, T.C, \mathcal{E}[\mathtt{skip}] \rangle \\
\langle \sigma, T, \mathcal{E}[\mathtt{yield}] \rangle & \longrightarrow_a & \langle \sigma, T.\mathcal{E}[\mathtt{skip}], \mathtt{skip} \rangle \\
\langle \sigma, T.C.T', \mathtt{skip} \rangle & \longrightarrow_c & \langle \sigma, T.T', C \rangle
\end{array}
$$

State Space and Evaluation Contexts

State Space

$$
\begin{array}{lclcl}
\Gamma & \in & \mathrm{State} & = & \mathrm{Store} \times \mathrm{ThreadPool} \times \mathrm{Com} \\
\sigma & \in & \mathrm{Store} & = & \mathrm{Vars} \rightarrow \mathrm{Nat} \\
T & \in & \mathrm{ThreadPool} & = & \mathrm{Com}^*
\end{array}
$$

where Vars is finite.

Evaluation Contexts

$$\mathcal{E} \quad = \quad [\,] \mid \mathcal{E}; C$$

Outline

Transition Sequences

- Abrahamson used transition sequences of the form:

$$u = (\sigma_1, \sigma'_1) \ldots (\sigma_m, \sigma'_m)$$

- Perhaps we need hierarchical triples for thread spawning:

$$v = (\sigma_1, u_1, \sigma'_1) \ldots (\sigma_m, u_m, \sigma'_m)$$

- Miraculously, we only need 1 embedding to 1 level, roughly:

$$v = (\sigma_1, \sigma'_1) \ldots (\sigma_m, u, \sigma'_m)$$

- Precisely, so that prefix is the right partial order, and also to allow for totality, *transition sequences* are:

$$v = (\sigma_1, \sigma'_1) \ldots (\sigma_m, \sigma'_m)[(\sigma, \sigma' \ \mathrm{return})u]$$

where $$u = (\overline{\sigma}_1, \overline{\sigma}'_1) \ldots (\overline{\sigma}_n, \overline{\sigma}'_n)[\mathrm{done}]$$

is a *pure transition sequence* (and $m, n \geq 0$).

Form of Denotational Semantics

Proc, our domain of *processes*, is $\mathcal{I}_{\neq\emptyset,\omega}(\mathrm{TSeq})$ the ω-cpo of all non-empty, countably-based ideals of transition sequences, i.e., all nonempty prefix-closed sets of transition sequences. We have:

$$[\![C]\!] \in \mathrm{Proc}$$

Pool, our domain of *thread pools*, is $\mathcal{I}_{\neq\emptyset,\omega}(\mathrm{PSeq})$ the ω-cpo of all non-empty, countably-based ideals of pure transition sequences, i.e., the ω-cpo of all non-empty prefix-closed sets of pure transition sequences. We have:

$$[\![T]\!] \in \mathrm{Pool}$$

Denotational Semantics of Commands

$$[\![\texttt{skip}]\!] = *$$

$$[\![C; D]\!] = [\![C]\!] \circ [\![D]\!]$$

$$[\![x := e]\!] = \{(\sigma, \sigma[x \mapsto n]\ \mathrm{return})\ \mathrm{done} \mid \sigma \in \mathrm{Store}, \sigma(e) = n\} \downarrow$$

$$\begin{aligned}[\![\texttt{if } b \texttt{ then } C \texttt{ else } D]\!] \quad = \quad & \{(\sigma,\tau)v \in [\![C]\!] \mid \sigma(b) = \texttt{true}\} \downarrow \ \cup \\ & \{(\sigma,\tau)v \in [\![D]\!] \mid \sigma(b) = \texttt{false}\} \downarrow\end{aligned}$$

$$[\![\texttt{while } b \texttt{ do } C]\!] = \cup_i [\![(\texttt{while } b \texttt{ do } C)_i]\!]$$

$$[\![\texttt{async } C]\!] = \mathrm{async}([\![C]\!]^c)$$

$$[\![\texttt{yield}]\!] = \mathrm{d}(*)$$

$$[\![\texttt{block}]\!] = \{\varepsilon\}$$

Sequential Composition of Processes

We give rules for composition, as it is easier to understand that way:

$$\frac{v(\sigma, \sigma'\mathrm{return})u \in P \quad (\sigma', \tau)w \in Q}{v(\sigma, \tau)(u \bowtie w) \subseteq P \circ Q}$$

$$\frac{v \in P}{v \in P \circ Q} \qquad \text{if } v \text{ does not contain } \mathrm{return}$$

It is associative with two-sided unit:

$$* = \{(\sigma, \sigma\ \mathrm{return})\ \mathrm{done} \mid \sigma \in \mathrm{Store}\}\downarrow$$

Merging transition sequences

The set of merges of a pure transition sequence u and a (pure) transition sequence w is given by:

$$u[\mathrm{done}]^1 \bowtie w[\mathrm{done}]^2 = (u \bowtie w)[\mathrm{done}]^{1 \wedge 2}$$

where the merge on the right is the standard merge of sequences and the done on the right appears only if it appears both times on the left.

Delay and Yielding

We define a continuous *delay* function $\mathrm{d} : \mathrm{Proc} \to \mathrm{Proc}$ by:

$$\mathrm{d}(P) = \{(\sigma, \sigma)v \mid \sigma \in \mathrm{Store}, v \in P\} \downarrow$$

So that:

$$[\![\texttt{yield}]\!] = \mathrm{d}(*) = \{(\sigma, \sigma)(\sigma', \sigma' \text{ return}) \; \mathit{donem}\} \downarrow$$

Spawning Threads

Recall:

$$[\![\mathrm{async}\ C]\!] = \mathrm{async}([\![C]\!]^c)$$

Where:

- $-^c : \mathrm{Proc} \to \mathrm{Pool}$ is the extension to processes of the function $-^c : \mathrm{TSeq} \to \mathrm{PSeq}$ of the same name from transition sequences to pure transition sequences which removes the marker return
- $\mathrm{async}(P) =_{\mathrm{def}} \{(\sigma, \sigma\ \mathrm{return})u \mid \sigma \in \mathrm{Store}, u \in P\}$

Note: $\mathrm{async}(P^c)$ differs from $\mathrm{d}(P)$ only in the placement of the marker return: the former replaces it at the beginning.

Denotational Semantics of Thread Pools

- $\bowtie$: $\mathrm{Pool}^2 \to \mathrm{Pool}$ is the extension to thread pools of the binary function on pure transition sequences of the same name.
- Together with $I =_{\mathrm{def}} \{\mathrm{done}\}\downarrow$ it forms a commutative monoid
- The semantics of a thread pool $C_1, \ldots, C_n$ is given by:

$$[\![C_1, \ldots, C_n]\!] = [\![C_1]\!]^c \bowtie \ldots \bowtie [\![C_n]\!]^c \quad (n \geq 0)$$

- Note that $[\![\varepsilon]\!] = I$
- Our domain of *asynchronous processes* AProc is the sub-ω-cpo of Pool none of whose elements contain done.
- We always have $[\![C]\!]^c \in \mathrm{AProc}$.

Outline

Adequacy Theorem for Pure Transition Sequences

Define:

$$\Gamma \Rightarrow \Gamma' \quad \text{iff} \quad \Gamma \longrightarrow_a{}^* \longrightarrow_c \Gamma'$$

and

$$[\![T, C]\!] = \mathrm{async}([\![T]\!]) \circ [\![C]\!]$$

Theorem

The following are equivalent:

1. $(\sigma_1, \sigma'_1) \ldots (\sigma_n, \sigma'_n)\mathrm{done} \in [\![T_1, C_1]\!]^c$ *($n > 0$)*
2. *There are T_i, C_i, ($i = 2, n$) such that*
   - $\langle \sigma_i, T_i, C_i \rangle \Rightarrow \langle \sigma'_i, T_{i+1}, C_{i+1} \rangle$, *for* $1 \leq i \leq n-1$
   - $\langle \sigma_n, T_n, C_n \rangle \longrightarrow_a{}^* \langle \sigma'_n, \varepsilon, \mathtt{skip} \rangle$.

There is an analogous statement for $(\sigma_1, \sigma'_1) \ldots (\sigma_n, \sigma'_n) \in [\![T, C]\!]^c$

Adequacy Theorem for Runs

To account for uninterrupted running, we define, for $P \in \mathrm{Pool}$:

$$\mathrm{runs}(P) = \{\sigma_1 \ldots \sigma_n[\mathrm{done}] \mid (\sigma_1, \sigma_2)(\sigma_2, \sigma_3) \ldots (\sigma_{n-1}, \sigma_n)[\mathrm{done}] \in P\}$$

These runs are our observables.

Corollary

The following are equivalent:

1. $\sigma_1 \ldots \sigma_n \mathrm{done} \in \mathrm{runs}([\![T_1, C_1]\!]) \qquad (n \geq 2)$
2. *There are T_i, C_i, $(i = 2, n-1)$ such that:*
   $\langle \sigma_1, T_1, C_1 \rangle \Rightarrow \ldots \Rightarrow \langle \sigma_{n-1}, T_{n-1}, C_{n-1} \rangle \longrightarrow_a{}^* \langle \sigma_n, \varepsilon, \mathtt{skip} \rangle$

There is an analogous statement for $\sigma_1 \ldots \sigma_n \in \mathrm{runs}([\![T_1, C_1]\!])$

Inequational Full Abstraction

Theorem

The following are equivalent, for any commands C and D:

1. $[\![C]\!] \subseteq [\![D]\!]$
2. *For every context* $\mathcal{C}$, $\mathrm{runs}([\![\mathcal{C}[C]]\!]^c) \subseteq \mathrm{runs}([\![\mathcal{C}[D]]\!]^c)$.

Overview

- Following Moggi we are interested in a monadic point of view, here using a continuous monad $T(P)$ over $\omega\mathrm{Cpo}$ to model the set of computations for elements of P. We seek such a T_{Proc} with:

$$\mathrm{Proc} = T_{\mathrm{Proc}}(1)$$

- To this end we seek a computationally interesting equational theory L_{Proc} such that T_{Proc} is the corresponding free algebra (better, free model) monad.
- This theory will be a variant of the theory for the classical resumptions monad and so we will also see how the trace model described above fits in with standard notions.

Outline

Inequational Theories

These are:

$$\mathrm{Th} = (\Sigma, \mathrm{InEq})$$

where the operation arities $f : n \to 1$ are given by Σ and InEq is a set of inequations

$$t \leq u$$

over terms formed from these operation symbols.

One then has the usual notion of Σ-algebra in $\omega\mathrm{Cpo}$ and the free model—meaning modelling the inequations—monad over $\omega\mathrm{Cpo}$ is written T_{Th}.

Two Examples

Example Nontermination: the theory L_Ω

$$\Omega \leq x$$

Here T_Ω is the usual lifting monad.

Example Hoare (Lower) Powerdomain: the theory L_H

$$x \leq x \cup y \qquad y \leq x \cup y \qquad z \cup z \leq z$$

Here T_H is the lower powerdomain monad in $\omega\mathrm{Cpo}$; $T_H(P)$ is the free ω-semilattice over P (meaning all countable sups) and it consists of of all countably generated Scott closed sets.

The Side-Effects Monad Considered Algebraically

Monad

$$T_S(P) = (\mathrm{Store} \times P)^{\mathrm{Store}}$$

Signature

$$\mathrm{lookup} : \mathrm{Nat} \to \mathrm{Vars} \quad \mathrm{update} : 1 \to \mathrm{Vars} \times \mathrm{Nat}$$

The corresponding generics are:

$$! : \mathrm{Vars} \to \mathrm{Nat} \quad := : \mathrm{Vars} \times \mathrm{Nat} \to 1$$

Sample Equations for the Side-Effects Theory $\mathcal{SE}$

$$\mathrm{update}_{l,v}(\mathrm{update}_{l',v'}(x)) = \mathrm{update}_{l',v'}(\mathrm{update}_{l,v}(x)) \text{ (if } l \neq l')$$

$$\mathrm{lookup}_l(\ldots \mathrm{update}_{l,v}(x) \ldots) = x$$

which last can be written in a finitary way as:

$$\mathrm{lookup}_l((v : \mathrm{val}).\mathrm{update}_{l,v}(x)) = x$$

Countably Infinitary Continuous Algebra

- **Signature** $\Sigma = \{f : \overrightarrow{I_1}, \ldots, \overrightarrow{I_m} \longrightarrow O_1, \ldots, O_n\}$, with $\overrightarrow{I_1}, \ldots, \overrightarrow{I_m}$ countably infinite sets, and $O_1, \ldots, O_n$ parameter spaces, being ω-cpos, giving:
  - Function symbols $f_{o_1,\ldots,o_n}$ (for $o_j \in O_j$), indexed by:

    $$O =_{\mathrm{def}} O_1 \times \ldots \times O_n$$

    of arity

    $$I =_{\mathrm{def}} (\prod \overrightarrow{I_1}) \times \ldots \times (\prod \overrightarrow{I_m})$$

  - Infinitary terms

    $$f_{\vec{o}}(\langle t_{\vec{i_1},\ldots,\vec{i_m}} \rangle_{\vec{i_1},\ldots,\vec{i_m}})$$

    Note the indexed arguments.

- **Inequations** InEq consists of inequations $t \leq u$ between the (possibly) infinitary terms formed from the function symbols.

Models

- **Algebras** Carriers, being ω-cpos A, equipped with continuous maps
$$f_A : A^I \longrightarrow A^O$$
equivalently
$$f_A : O \times A^I \longrightarrow A$$
Models are such satisfying the inequations.
- **Free Algebra Monad** We obtain T_{Th} giving the free such model; it is an $\omega\mathrm{Cpo}$-monad.

Remark There is a useful finitary notation for such infinitary theories.

Outline

The Theory for Resumptions

We define:

$$L_{\mathrm{Res}} = L_H \otimes ((L_S \otimes L_\Omega) + L_d)$$

where:

- L_d is the theory of a unary operator, d, with no axioms.
- The axioms of $L + L'$ are those of L and L' (we assume the operation symbols are disjoint).
- The axioms of $L \otimes L'$ are those of $L + L'$ together with the commutativity of the operations of the one over the operations of the other (again assuming disjointness).

Basic Q-Transition Sequences

- Q a poset
- Q-transition: $(\sigma, \sigma' x)$ where $x \in Q$
- basic Q-transition sequence: $(\sigma_1, \sigma_1), \ldots, (\sigma_n, \sigma_n)[(\sigma, \sigma' x)]$
- Q-BTrans is the partial order of Q-transition sequences where $u \leq v$ holds iff:

$$\begin{array}{ll} \text{either} & u \leq_p v \\ \text{or else} & \exists w, x \leq y.\, u \leq_p w(\sigma, \sigma' x) \wedge v = w(\sigma, \sigma' y) \end{array}$$

Characterisation Theorem for Resumptions

Theorem

1. *Viewed as an* $\mathrm{L}_{\mathrm{Res}}$*-model,* $\mathcal{I}_\omega(Q\text{-}\mathrm{BTrans})$ *is* $T_{\mathrm{Res}}(\mathcal{I}_\omega^{\uparrow}(Q))$.
2. *As a semilattice with a zero this is the solution in* $\omega\mathrm{SL}$ *of the 'domain equation'*

$$R \cong (S \times (R_\perp + Id_\omega^{\uparrow}(Q)))^S$$

equivalently

$$R \cong (S \times S) \times (R_\perp + Id_\omega^{\uparrow}(Q))$$

3. Q-BTrans *is the solution in* Pos *of:*

$$T \cong (S \times S) \times (T_\perp + Q)$$

Outline

Asynchronous Processes

L_{AProc} is L_{Res} extended by a new constant halt with the axiom:

$$\mathrm{d}(\Omega) \leq \mathrm{halt}$$

Theorem

1. AProc *is the initial* L_{AProc}*-model, i.e., it is* $T_{\mathrm{AProc}}(0)$.
2. *As a semilattice with a zero this is the solution in* $\omega\mathrm{SL}$ *of the 'domain equation'*

$$R \cong (S \times S) \times (R + 1)_{\perp}$$

Outline

The Theory for Processes

This is:

$$L_{\mathrm{Proc}} = L_{\mathrm{Res}} + L_{\mathrm{Spawn}}$$

where L_{Spawn} is the theory for **spawning** whose signature is that for L_{Res} together with two new operation symbols:

$$\mathrm{async} : 1 \longrightarrow \mathrm{AProc}$$

$$\mathrm{yield\_to} : 1 \longrightarrow \mathrm{AProc}$$

We write

$$P \cdot t \quad \text{for} \quad \mathrm{async}_P(t)$$

$$t \cdot P \quad \text{for} \quad \mathrm{yield\_to}_P(t)$$

Proc as a Proc-algebra

For $P \in \mathrm{AProc}$ and $Q \in \mathit{Proc}$ we define:

$$\begin{array}{rcl} P \cdot_{\mathrm{Proc}} Q & = & \mathrm{async}(P) \circ Q \\ & = & \bigcup \{(\sigma, \tau) u \bowtie w \mid u \in P, (\sigma, \tau) w \in Q\} \downarrow \end{array}$$

$$Q \cdot_{\mathrm{Proc}} P = \bigcup \{(\sigma, \sigma') u \bowtie v \mid (\sigma, \sigma') u \in P, v \in Q\} \downarrow$$

First Group of Equations

These concern commutation with $\cup$:

$$(P \cup_{\mathrm{AProc}} P') \cdot x = (P \cdot x) \cup (P' \cdot x)$$

$$P \cdot (x \cup y) = P \cdot x \cup P \cdot y$$

$$(x \cup y) \cdot P = x \cdot P \cup y \cdot P$$

$$x \cdot (P \cup_{\mathrm{AProc}} P') = x \cdot P \cup x \cdot P'$$

Equations for async

$$P \cdot \mathrm{update}_{l,v}(x) = \mathrm{update}_{l,v}(P \cdot x)$$

$$P \cdot \mathrm{lookup}_{l}(\langle x_v \rangle_v) = \mathrm{lookup}_{l}(\langle P \cdot x_v \rangle_v)$$

$$P \cdot \Omega = \Omega$$

$$P \cdot \mathrm{d}(x) = \mathrm{d}(P \cdot x) \cup \mathrm{d}(x \cdot P)$$

$$P \cdot (P' \cdot x) = (P \bowtie P') \cdot x$$

$$P \cdot (x \cdot P') = (P \cdot x) \cdot P' \cup (x \cdot P) \cdot P'$$

(The last is redundant.)

Equations for $\mathrm{yield\_to}$

$$x \cdot (\mathrm{update}_{\mathrm{AProc}})_{l,v}(P) = \mathrm{update}_{l,v}(x \cdot P)$$

$$x \cdot (\mathrm{lookup}_{\mathrm{AProc}})_{l}(f) = \mathrm{lookup}_{l}(\langle x \cdot f(v) \rangle_v)$$

$$x \cdot \Omega_{\mathrm{AProc}} = \Omega$$

$$x \cdot \mathrm{d}_{\mathrm{AProc}}(P) = \mathrm{d}(x \cdot P) \cup \mathrm{d}(P \cdot x)$$

$$x \cdot \mathrm{halt}_{\mathrm{AProc}} = \mathrm{d}(x)$$

Transition Sequences for Processes

- $Q\text{-Trans} =_{\text{def}} (Q \times \text{PSeq})\text{-BTrans}$
- Its elements have the form:

$$(\sigma_1, \sigma'_1) \ldots (\sigma_m, \sigma'_m)[(\sigma, \sigma'\langle x, (\overline{\sigma}_1, \overline{\sigma}'_1) \ldots (\overline{\sigma}_n, \overline{\sigma}'_n)[\text{done}]\rangle)]$$

Characterisation Theorems for Processes

Theorem

1. *Viewed as an* L_{Proc}*-model,* $\mathcal{I}_\omega(Q\text{-}\mathrm{Trans})$ *is the free model over* $\mathcal{I}^{\uparrow}_\omega(Q)$.
2. *So as a* Res*-algebra:*

$$T_{\mathrm{Proc}}(\mathcal{I}^{\uparrow}_\omega(Q)) \cong T_{\mathrm{Res}}(\mathrm{Pool} \times \mathcal{I}^{\uparrow}_\omega(Q))$$

3. *There is an isomorphism* $\theta : Q\text{-}\mathrm{Trans} \to \mathrm{TSeq}\backslash\{\varepsilon\}$*, where* $Q = \{\mathrm{return}\}$ *and so, as a* Proc*-algebra:*

$$\mathrm{Proc} \cong \mathcal{I}_\omega(\mathrm{TSeq}\backslash\{\varepsilon\}) \cong T_{\mathrm{Proc}}(1)$$

Some Algebraic Reflections

- This is applied domain theory where one is interested in particular models and, particularly, their algebraic structure.
- Having free algebras is a condition on a domain theory: cf. Martin Hyland's 'reasons for domain theory' Part 1.
- Here, some structure, particularly the semilattice structure, is 'nice' mathematically; the actions are less so.
- Still, parallel constructs are typically not even algebraic operations.
- In the Proc characterisation theorem, part 2, we do not get the correct left action structure, though there is a wrong structure as Pool is a (commutative) monoid.
- Perhaps a Hopf shuffle algebra would help for a 'rational algebraic analysis' (cf. Martin Hyland's categorical rational reconstructions in domain theory).

Possible Future Work

- Must semantics (compact sets of transition sequences)
- Add variable declaration: a challenge, at the least, for the algebraic part.
- Add higher-types. Can do as have monad, but full abstraction is another matter.
- Change notion of observations: runs with stuttering or mumbling.
- Fairness: all threads in the pool will eventually be chosen in any infinite run.
- Lower level semantics, with block treated as an exception causing a rollback; can then do C **orelse** C'.
- What equations hold not involving side-effects, conditionals or while loops? Example:
  $[\![(\texttt{async}\ (C; \texttt{async}\ (D))]\!] = [\![\texttt{async}\ (C; \texttt{yield}; D)]\!]$