La stratégie de la fourchette

Simon Castellan, 8 janvier 2015
JFLA 2015
fork(II)

NAME
fork -- spawn new process

SYNOPSIS
sys fork / fork = 2.
(new process return)
(old process return)

DESCRIPTION
fork is the only way new processes are created. The new process's core image is a copy of that of the caller of fork the only distinction is the return location and the fact that r0 in the old process contains the process ID of the new process. This process ID is used by wait.

FILES

SEE ALSO
sys wait, sys exec

DIAGNOSTICS
The error bit (c-bit) is set in the old process if a new process could not be created because of lack of swap space.

BUGS
See wait for a subtle bug in process destruction.

OWNER
ken, dmr
Demo

...demo...
Syntax

- Idealized Algol + a monoid structure on each base type.

\[ A, B ::= \text{nat} \mid \text{bool} \mid \text{unit} \quad \text{(base types)} \]
\[ \mid \text{var} \quad \text{(references)} \]
\[ \mid A \Rightarrow B \quad \text{(arrow types)} \]
\[ t, u ::= x \mid \lambda x. \ t \mid t \ u \mid Y \quad \text{(simply typed \(\lambda\)-calculus + fixpoint)} \]
\[ \mid c \quad \text{(PCF constants)} \]
\[ \mid \text{new } r ::= k \text{ in } t \quad \text{(reference creation)} \]
\[ \mid t ::= u \mid !t \quad \text{(operations on references)} \]
\[ \mid t; u \mid \text{if } t \text{ then } u \text{ else } v \quad \text{(destructors)} \]
\[ \mid t \parallel u \mid \text{exit} \quad \text{(forks, only on base types)} \]
Idea: pure $\beta$-reduction + reduction at base type for effects.
(Inspired from [AM99, GM07])

- **Structural congruence** $\equiv$ to reduce at higher-order types:
  smallest congruence containing the two equations

  $$(\lambda x.t)\ u \equiv t[u/x] \quad Y\ M \equiv M\ (Y\ M)$$

- **Small-step reduction** at base types: $\Gamma \vdash (t, \rho) \rightarrow (t', \rho') : X$
  
  - $\Gamma$ contains only references
  - $t, t'$ are terms such that $\Gamma \vdash t, t' : X$ (base type)
  - $\rho, \rho' : \text{dom}(\Gamma) \rightarrow \mathbb{N}$
Operational semantics - effect reduction

\[
\beta\text{-red} \quad t' \equiv t \quad \Gamma \vdash t, \rho \rightarrow u, \rho' \quad u \equiv u'
\]

\[
\Gamma \vdash t', \rho \rightarrow u', \rho'
\]

Deref \quad \rho(r) = k

\[
\Gamma \vdash !r, \rho \rightarrow k, \rho
\]

Assign \quad \Gamma \vdash r := k, \rho \rightarrow () \cup (r \mapsto k) \cup \rho \setminus r

New1 \quad r \not\in \text{fv}(t)

\[
\Gamma \vdash \text{new } r := k \text{ in } t, \rho \rightarrow t, \rho \setminus r
\]

New2 \quad \Gamma, r : \text{var} \vdash t, \rho \rightarrow t', \rho'

\[
\Gamma \vdash \text{new } r := \rho(r) \text{ in } t, \rho \setminus r \rightarrow \text{new } r := \rho'(r) \text{ in } t', \rho' \setminus r
\]
Operational semantics - effect reduction

\[ E ::= \ (\text{base type evaluation context}) \]

\[ [] \mid \text{succ } E \mid \ldots \mid \text{if } E \text{ then } t \text{ else } u \mid x := E \mid x; E \]

**Duplication**

\[ \Gamma \vdash E[t \parallel u], \rho \rightarrow E[t] \parallel E[u], \rho \]

**Erasure**

\[ \Gamma \vdash E[\text{exit}], \rho \rightarrow \text{exit}, \rho \]

**Preemption**

\[ \Gamma \vdash t, \rho \rightarrow t', \rho' \]

\[ \Gamma \vdash (t \parallel u), \rho \rightarrow (t' \parallel u), \rho' \]

**Preemption'**

\[ \Gamma \vdash u, \rho \rightarrow u', \rho' \]

\[ \Gamma \vdash (t \parallel u), \rho \rightarrow (t \parallel u'), \rho' \]
Fork calculus = operational modelisation of the `fork` syscall
Features concurrency and non-determinism

Concurrency + Non-determinism
“true and false” + “true or false”
partial orders + binary conflict

→ Event structures.
Fork calculus = operational modelisation of the \texttt{fork} syscall

Features concurrency and non-determinism

\begin{itemize}
  \item Concurrency + Non-determinism
  \item \texttt{"true and false"} + \texttt{"true or false"}
  \item partial orders + binary conflict
\end{itemize}

\text{→ Event structures.}

\text{Now: a very high-level view of the game model based on event structures.}
On game semantics

Denotational semantics (at first of typed λ-calculus + extensions) where:

▶ types $\rightarrow$ games
▶ programs $\rightarrow$ strategies
▶ computation $\rightarrow$ interaction of strategies

Hyland-Ond game semantics: supports references [AM99] and concurrency through interleavings [GM07].
Our model: truly concurrent extension of a truly concurrent semantics to this language based on Winskel’s concurrent games.
Usual HO game semantics

Rules for a type? Arenas.

\[ (((\alpha \to \alpha) \to \alpha) \to \alpha) \to \alpha \]
Usual HO game semantics

Rules for a type? Arenas.

$\lambda f. f (\lambda x. f (\lambda y. x)) : ((\alpha \to \alpha) \to \alpha) \to \alpha$

$q^- \leftarrow$
Usual HO game semantics

Rules for a type? Arenas.

\[
(((\alpha \rightarrow \alpha) \rightarrow \alpha) \rightarrow \alpha
\]

\(q^-\)

\(q^+\)
Usual HO game semantics

Rules for a type? Arenas.

\[((\alpha \to \alpha) \to \alpha) \to \alpha\]
Usual HO game semantics

Rules for a type? Arenas.

$$(((\alpha \rightarrow \alpha) \rightarrow \alpha) \rightarrow \alpha$$
Usual HO game semantics

Rules for a type? Arenas.

$(((\alpha \rightarrow \alpha) \rightarrow \alpha) \rightarrow \alpha$
Usual HO game semantics

How to inhabit: \( \vdash \)

\[ (((\alpha \to \alpha) \to \alpha) \to \alpha) \to \alpha \]

\[ (((\alpha \to \alpha) \to \alpha) \to \alpha) \to \alpha \]

Moves are chronologically (ie. totally) ordered.

To handle concurrency, we need partial orders on plays.
Usual HO game semantics

How to inhabit: $\vdash \lambda f .
\vdash (\alpha \to \alpha) \to \alpha
\vdash (((\alpha \to \alpha) \to \alpha) \to \alpha
\vdash (((\alpha \to \alpha) \to \alpha) \to \alpha
\vdash q^-_\lambda f$.
Usual HO game semantics

How to inhabit: $\vdash \lambda f. \ f$

$(\alpha \to \alpha) \to \alpha \to \alpha$

$(\alpha \to \alpha) \to \alpha \to \alpha$
Usual HO game semantics

How to inhabit: $\vdash \lambda f.\ f\ (\lambda x.\ ) : ((\alpha \to \alpha) \to \alpha) \to \alpha$

$$(((\alpha \to \alpha) \to \alpha) \to \alpha$$
Usual HO game semantics

How to inhabit: \( \vdash \lambda f. f (\lambda x. f) \quad (\alpha \to \alpha) \to \alpha \}

\[ (((\alpha \to \alpha) \to \alpha) \to \alpha \]

\[ (((\alpha \to \alpha) \to \alpha) \to \alpha \to \alpha \]

Moves are chronologically (ie. totally) ordered.
To handle concurrency, we need partial orders on plays.
Usual HO game semantics

How to inhabit: \( \vdash \lambda f. f (\lambda x. f (\lambda y. )) : (\alpha \to \alpha) \to \alpha \to \alpha \)

\[
(((\alpha \to \alpha) \to \alpha) \to \alpha
\]

\[
(((\alpha \to \alpha) \to \alpha) \to \alpha
\]

\[
q^- \quad q^+
\]

\[
q^- \quad q^+
\]

\[
q^- \quad q^+
\]

\[
q^- \quad q^+
\]

Moves are chronologically (ie. totally) ordered.

To handle concurrency, we need partial orders on plays.
Usual HO game semantics

How to inhabit: $\vdash \lambda f.\ f\ (\lambda x.f\ (\lambda y.x)) : ((\alpha \rightarrow \alpha) \rightarrow \alpha) \rightarrow \alpha$

$$(((\alpha \rightarrow \alpha) \rightarrow \alpha) \rightarrow \alpha$$

![Diagram of game semantics]
Usual HO game semantics

How to inhabit: \( \vdash \lambda f. f (\lambda x.f (\lambda y.x)) : ((\alpha \rightarrow \alpha) \rightarrow \alpha) \rightarrow \alpha \)

\[
(((\alpha \rightarrow \alpha) \rightarrow \alpha) \rightarrow \alpha
\]

\[
(((\alpha \rightarrow \alpha) \rightarrow \alpha) \rightarrow \alpha
\]

Moves are chronologically (ie. totally) ordered.

→ To handle concurrency, we need **partial orders** on plays
Concurrency in arenas

\[(\alpha \rightarrow \alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha \rightarrow \alpha)\]
The concurrent content of terms

\[(\alpha \rightarrow \alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha \rightarrow \alpha)\]

\[q_{\lambda b}^{-}\]
The concurrent content of terms

\[ (\alpha \rightarrow \alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha \rightarrow \alpha) \]
The concurrent content of terms

\[(\alpha \rightarrow \alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha \rightarrow \alpha)\]

\[q_{\lambda b}.\]

\[ff^{-} \quad tt^{-} \quad q_{b}^{+}
\]
The concurrent content of terms

\[(\alpha \rightarrow \alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha \rightarrow \alpha)\]

Opponent: this strategy expresses its behaviour against the most concurrent Opponent.

\[\text{neg } tf = \text{ff} \quad \text{neg } tf = \text{tt} \quad \text{neg } (\text{ff} \parallel \text{tt}) = (\text{tt} \parallel \text{ff})\]
The concurrent content of terms

\[(\alpha \rightarrow \alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha \rightarrow \alpha)\]

\(\rightarrow\) evaluation order of the two negative answers is left to the Opponent: this strategy expresses its behaviour against the most concurrent Opponent.
The concurrent content of terms

\[(\alpha \rightarrow \alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha \rightarrow \alpha)\]

\[\neg tt = ff\]

→ evaluation order of the two negative answers is left to the Opponent: this strategy expresses its behaviour against the most concurrent Opponent.
The concurrent content of terms

\[(\alpha \rightarrow \alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha \rightarrow \alpha)\]

\[q^{\lambda_b}.\]

\[q_{b}^{+} \quad ff^{-} \quad tt^{−}
\]

\[ff^{+} \quad tt^{+}
\]

→ evaluation order of the two negative answers is left to the Opponent: this strategy expresses its behaviour against the most concurrent Opponent.

\[\text{neg } tt = ff \quad \text{neg } ff = tt\]
The concurrent content of terms

\[(\alpha \rightarrow \alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha \rightarrow \alpha)\]

\[q_{\lambda b}, q^+_b, q^-_b, ff^-, tt^-, ff^+, tt^+\]

→ evaluation order of the two negative answers is left to the Opponent: this strategy expresses its behaviour against the most concurrent Opponent.

\[
\text{neg } tt = ff \quad \text{neg } ff = tt \quad \text{neg } (ff \parallel tt) = (tt \parallel ff)
\]
Concurrency against non-determinism

\[(\alpha \rightarrow \alpha \rightarrow \alpha)\]

\[ff^+\rightarrow tt^+\rightarrow q^-\]

\[fork\]

\[(\alpha \rightarrow \alpha \rightarrow \alpha)\]

\[ff^+\rightarrow \sim tt^+\rightarrow q^-\]

\[choice\]
Results & conclusions

Results:

▶ Denotational semantics for the fork calculus \((t, \rho) \mapsto \llbracket t; \rho \rrbracket\)

▶ Soundness results: If \(\Gamma \vdash (t, \rho) \rightarrow (t', \rho')\) then \(\llbracket t; \rho \rrbracket\) simulates \(\llbracket t'; \rho \rrbracket\) (in [GM07], the result was full abstraction wrt trace inclusion).

▶ We retain a lot of information on the programs (causalities, non-deterministic branching point)

Perspectives:

▶ A showcase of the power of concurrent games: we can actually model complex concurrent programming languages

▶ Soundness is a first step, aim: full abstract wrt weak bisimulation because strategies retain a lot of information.

▶ Main problem: diverging branches hidden during composition:

\[
\llbracket \text{if choice then } () \text{ else } \Omega \rrbracket \sim \llbracket () \rrbracket
\]
Samson Abramsky and Guy McCusker.  
Full abstraction for idealized algol with passive expressions.  

Dan R. Ghica and Andrzej S. Murawski.  
Angelie semantics of fine-grained concurrency, 2007.