Dynamic threads via algebraic effects

Cristina Matache[†]
joint work with Ohad Kammar, Jack Liell-Cock, Sam Lindley, and Sam Staton

[†]University of Edinburgh

Introduction

Goal

Denotational semantics for **concurrency** where new threads can be created dynamically. E.g. POSIX `fork`.

Main idea: think of `fork` as an algebraic effect.

We use strong monads [Moggi'91] and algebraic theories [Plotkin&Power] for semantics:

- Use algebraic laws to reason about programs.
- Potentially more easily combine with other effects [Hyland, Plotkin, Power'02].

We use an extension called parameterized algebraic theories [Staton'13].

Outline

Effects we want to model

$$\text{fork} : \text{unit} \to \text{tid} + \text{unit} \qquad \text{wait} : \text{tid} \to \text{unit} \qquad \text{stop} : \text{unit} \to \text{empty}$$

$$\text{act}_\sigma : \text{unit} \to \text{unit}$$

tid	base type of thread IDs; only introduced by fork
fork$()$	spawns new child thread, copying the parent's continuation; can check whether parent or child by looking at result of fork
wait(a)	the current thread waits for thread a to finish
stop$()$	end current thread, unblocks all threads waiting for it
act$_\sigma()$	performs observable action σ immediately

Effects we want to model

$$\text{fork} : \text{unit} \to \text{tid} + \text{unit} \qquad \text{wait} : \text{tid} \to \text{unit} \qquad \text{stop} : \text{unit} \to \text{empty}$$

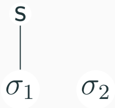
$$\text{act}_\sigma : \text{unit} \to \text{unit}$$

- A much simplified version of POSIX `fork` and `wait`.
- We consider a fine-grain call-by-value lambda calculus with these effectful operations.
- Operational semantics based on pools of threads.

Example closed programs

Can be understood as a **partial order labelled** by observable actions (pomset).

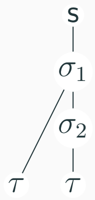
$$\text{let } y = \underline{\text{fork}}() \text{ in case } y \text{ of } \{\text{inj}_1(a) \Rightarrow \underline{\text{act}}_{\sigma_1}(); \underline{\text{stop}}(), \quad \text{inj}_2() \Rightarrow \underline{\text{act}}_{\sigma_2}(); \underline{\text{stop}}()\}$$



$$\text{let } y = \underline{\text{fork}}() \text{ in case } y \text{ of } \{\text{inj}_1(a) \Rightarrow \underline{\text{wait}}(a); \underline{\text{act}}_{\sigma_1}(); \underline{\text{stop}}(), \quad \text{inj}_2() \Rightarrow \underline{\text{act}}_{\sigma_2}(); \underline{\text{stop}}()\}$$

Example closed programs

$$\text{let } y = \underline{\text{fork}}() \text{ in case } \left(\underline{\text{act}}_\tau(); y\right)$$
$$\text{of } \{ \text{ inj}_1(a) \Rightarrow \underline{\text{wait}}(a); \underline{\text{act}}_{\sigma_1}(); \underline{\text{stop}}(),$$
$$\text{inj}_2() \Rightarrow \underline{\text{act}}_{\sigma_2}(); \underline{\text{stop}}() \}$$



$$\text{let } y = \underline{\text{fork}}() \text{ in case } y$$
$$\text{of } \{ \text{ inj}_1(a) \Rightarrow \underline{\text{wait}}(a); \underline{\text{act}}_{\sigma_1}(); \underline{\text{stop}}(),$$
$$\text{inj}_2() \Rightarrow \underline{\text{act}}_{\sigma_2}(); \underline{\text{stop}}() \}; \underline{\text{act}}_\tau()$$

Axiomatize fork, wait, stop, act_σ with 9 equations.

Outline

Generic effects vs algebraic operations

tid type of *compound* thread IDs, with a semilattice structure: e.g. $a \oplus b$, 0

Given $\underline{\text{op}} : A \to B$, the algebraic operation op takes a value of type A and B continuations.

wait$(u;\ x)$ — $\underline{\text{wait}}$: tid $\to$ unit

u is a compound thread ID; wait on all threads in u, continue as x

fork$(a.x(a),\ y)$ — $\underline{\text{fork}}$: unit $\to$ tid + unit

a is the thread ID of y; fork returns a non-compound thread ID a, bound in x

stop — $\underline{\text{stop}}$: unit $\to$ empty

has no continuation

act$_\sigma(x)$ — $\underline{\text{act}}_\sigma$: unit $\to$ unit

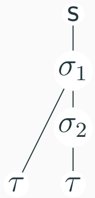
performs action σ and continues as x

Example closed programs using algebraic operations

$$\text{let } y = \underline{\text{fork}}() \text{ in case } y \text{ of } \{\text{inj}_1(a) \Rightarrow \underline{\text{wait}}(a); \underline{\text{act}}_{\sigma_1}(); \underline{\text{stop}}(), \; \text{inj}_2() \Rightarrow \underline{\text{act}}_{\sigma_2}(); \underline{\text{stop}}()\}$$

$$\text{fork}\big(a.\text{wait}(a;\ \text{act}_{\sigma_1}(\text{stop})),\ \text{act}_{\sigma_2}(\text{stop})\big)$$

$$\text{let } y = \underline{\text{fork}}() \text{ in case } \big(\underline{\text{act}}_{\tau}(); y\big) \text{ of } \{\ \text{inj}_1(a) \Rightarrow \underline{\text{wait}}(a); \underline{\text{act}}_{\sigma_1}(); \underline{\text{stop}}(), \; \text{inj}_2() \Rightarrow \underline{\text{act}}_{\sigma_2}(); \underline{\text{stop}}()\}$$



$$\text{fork}\big(a.\text{act}_{\tau}(\text{wait}(a;\ \text{act}_{\sigma_1}(\text{stop}))),\ \text{act}_{\tau}(\text{act}_{\sigma_2}(\text{stop}))\big)$$

Algebraic theory

Interaction of wait with the semilattice structure of thread IDs.

$$\text{wait}(0; x) = x \tag{1}$$

$$\text{wait}(a; \text{wait}(b; x)) = \text{wait}(a \oplus b; x) \tag{2}$$

$$\text{wait}(a; x(b)) = \text{wait}(a; x(a \oplus b)) \tag{3}$$

The term wait(a; stop) acts as a unit for fork.

$$\text{fork}(a.\text{wait}(a; \text{stop}), x) = x \tag{4}$$

$$\text{fork}(b.x(b), \text{wait}(a; \text{stop})) = x(a) \tag{5}$$

Operations wait and fork commute; fork is commutative and associative.

$$\text{wait}(b;\ \text{fork}(a.x(a),\ y)) = \text{fork}(a.\text{wait}(b;\ x(a)),\ \text{wait}(b;\ y)) \tag{6}$$

$$\text{fork}(a.\text{fork}(b.x(a, b),\ y),\ z) = \text{fork}(b.\text{fork}(a.x(a, b),\ z),\ y) \tag{7}$$

$$\text{fork}(a.x(a),\ \text{fork}(b.y(b),\ z)) = \text{fork}(b.\text{fork}(a.x(a),\ y(b)),\ z) \tag{8}$$

$$\text{act}_\sigma(x) = \text{fork}(a.\text{wait}(a, x),\ \text{act}_\sigma(\text{stop})) \tag{9}$$

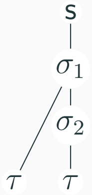
Outline

Labelled partial orders with holes

So far we only represented closed terms:

$$\mathsf{fork}\big(a.\mathsf{wait}(a;\ \mathsf{act}_{\sigma_1}(\mathsf{stop})),\ \mathsf{act}_{\sigma_2}(\mathsf{stop})\big)$$

$$\mathsf{fork}\big(a.\mathsf{act}_{\tau}(\mathsf{wait}(a;\ \mathsf{act}_{\sigma_1}(\mathsf{stop}))),\ \mathsf{act}_{\tau}(\mathsf{act}_{\sigma_2}(\mathsf{stop}))\big)$$



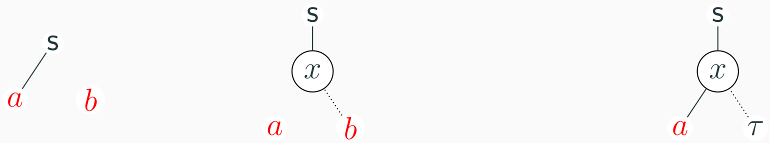
What about terms with free variables (i.e. continuations) and free **tid**'s?

E.g. $a \vdash \mathsf{fork}(b.\mathsf{wait}(a;\ x(b)),\ \mathsf{act}_{\tau}(\mathsf{stop}))$

Labelled partial orders with holes

stop $\quad x \quad$ $\mathsf{wait}(a;\ x)$ $\quad \mathsf{act}_\sigma(x)$ $\quad \mathsf{fork}(a.x(a),\ y)$

$a, b \vdash \mathsf{wait}(a;\ \mathsf{stop})$ $\quad a, b \vdash x(b)$ $\quad a \vdash \mathsf{fork}(b.\mathsf{wait}(a;\ x(b)),\ \mathsf{act}_\tau(\mathsf{stop}))$



The (non-compound) free thread ID's a, b are always minimal.

Substitution of another partial order for a hole (monadic bind)

In the term

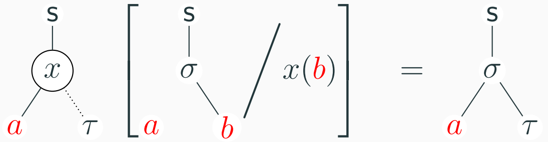
$$a \vdash \mathsf{fork}\big(b.\mathsf{wait}(a;\ x(b)),\ \mathsf{act}_\tau(\mathsf{stop})\big)$$

substitute for $x(b)$ the term

$$a, b \vdash \mathsf{wait}(b;\ \mathsf{act}_\sigma(\mathsf{stop}))$$

to get

$$a \vdash \mathsf{fork}\big(b.\mathsf{wait}(a;\ \mathsf{wait}(b;\ \mathsf{act}_\sigma(\mathsf{stop}))),\ \mathsf{act}_\tau(\mathsf{stop})\big)$$



Outline

Algebraic theory

Interaction of wait with the semilattice structure of thread IDs.

$$\mathsf{wait}(0; x) = x \tag{1}$$

$$\mathsf{wait}(a; \mathsf{wait}(b; x)) = \mathsf{wait}(a \oplus b; x) \tag{2}$$

$$\mathsf{wait}(a; x(b)) = \mathsf{wait}(a; x(a \oplus b)) \tag{3}$$

The term wait(a; stop) acts as a unit for fork.

$$\mathsf{fork}(a.\mathsf{wait}(a; \mathsf{stop}), x) = x \tag{4}$$

$$\mathsf{fork}(b.x(b), \mathsf{wait}(a; \mathsf{stop})) = x(a) \tag{5}$$

Operations wait and fork commute; fork is commutative and associative.

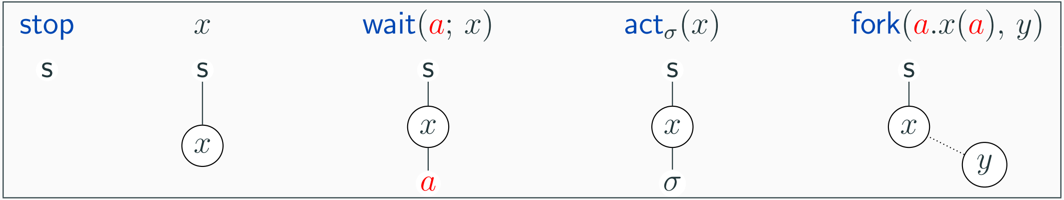
$$\mathsf{wait}(b;\ \mathsf{fork}(a.x(a),\ y)) = \mathsf{fork}(a.\mathsf{wait}(b;\ x(a)),\ \mathsf{wait}(b;\ y)) \tag{6}$$

$$\mathsf{fork}(a.\mathsf{fork}(b.x(a, b),\ y),\ z) = \mathsf{fork}(b.\mathsf{fork}(a.x(a, b),\ z),\ y) \tag{7}$$

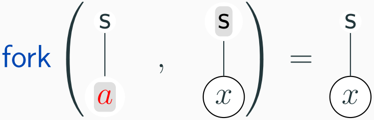
$$\mathsf{fork}(a.x(a),\ \mathsf{fork}(b.y(b),\ z)) = \mathsf{fork}(b.\mathsf{fork}(a.x(a),\ y(b)),\ z) \tag{8}$$

$$\mathsf{act}_\sigma(x) = \mathsf{fork}(a.\mathsf{wait}(a, x),\ \mathsf{act}_\sigma(\mathsf{stop})) \tag{9}$$

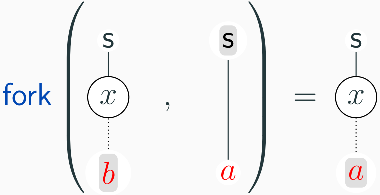
Graphical representation of equations in the algebraic theory



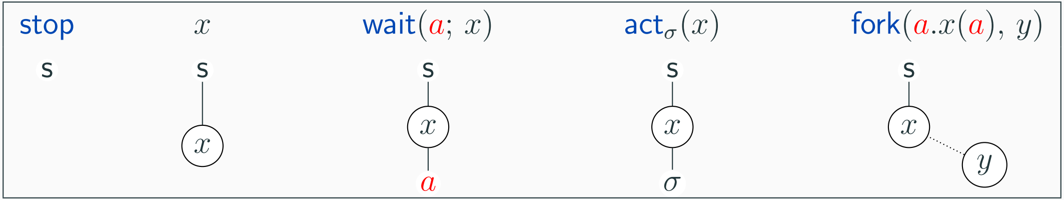
$\mathsf{fork}(a.\mathsf{wait}(a; \mathsf{stop}), x) = x \quad (4)$



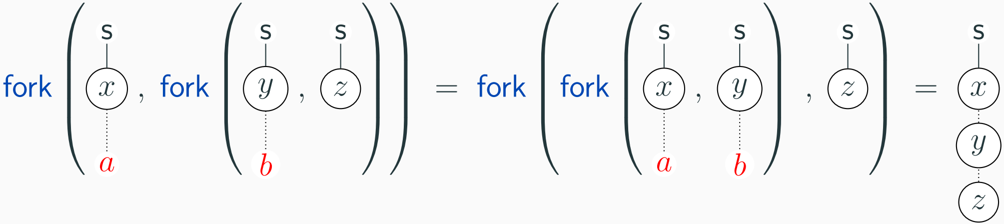
$\mathsf{fork}(b.x(b), \mathsf{wait}(a; \mathsf{stop})) = x(a) \quad (5)$



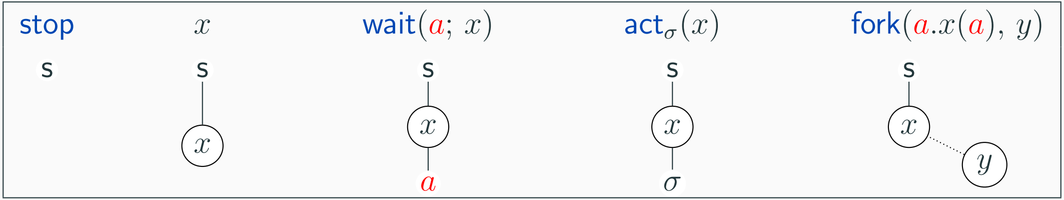
Graphical representation of equations in the algebraic theory



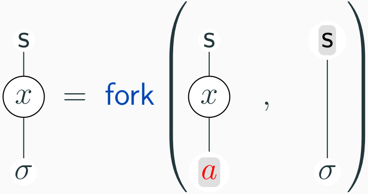
$$\mathsf{fork}(a.x(a),\ \mathsf{fork}(b.y(b),\ z)) = \mathsf{fork}(b.\mathsf{fork}(a.x(a),\ y(b)),\ z) \quad (8)$$



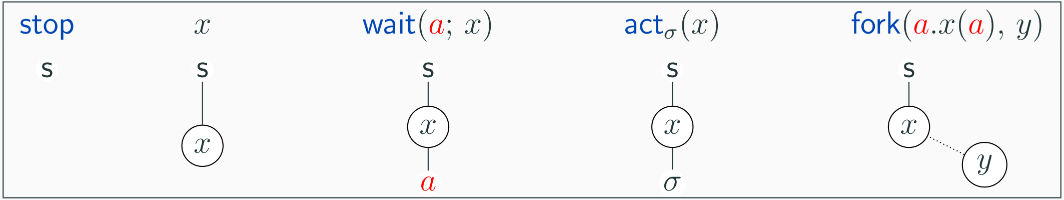
Graphical representation of equations in the algebraic theory



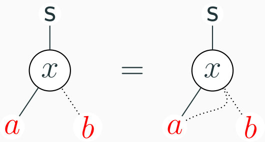
$$\mathsf{act}_\sigma(x) = \mathsf{fork}(a.\mathsf{wait}(a, x),\ \mathsf{act}_\sigma(\mathsf{stop})) \quad (9)$$



Graphical representation of equations in the algebraic theory



$$\mathsf{wait}(a; x(b)) = \mathsf{wait}(a; x(a \oplus b)) \quad (3)$$



The solid line absorbs the dotted line.

Main results

Theorem

Labelled partial orders with holes (lpoh) correspond exactly to terms in the algebraic theory. Equality of such partial orders is **sound and complete** w.r.t. equality in the algebraic theory.

Theorem (Syntactic Completeness)

If two labelled partial orders with holes (lpoh) are equal for all closing substitutions (i.e. as ordinary labelled partial orders) then they are equal.

Denotational semantics for the PL using the monad of lpoh's:

Denotational equality is **sound** for proving **contextual equivalence**, and fully abstract for first-order programs.

Summary and future work

- An algebraic theory that axiomatizes fork and wait.
- The algebraic theory induces a monad used for denotational semantics.
- We can think of programs as partial orders with labels. Reason about partial orders to show equivalence of programs.

Future work:

- Passing values from child to parent: stop $: A \to$ empty, wait $:$ tid $\to A$.
- Combine with shared state.
- Explore alternative semantics for fork and wait.