An algebraic theory of named threads (work in progress)

Cristina Matache

GALOP 2024

University of Edinburgh

Algebraic effects

- Effects can be modelled using **strong monads** on cartesian closed categories [Moggi'91].
- and analyzed using **algebraic theories** [Plotkin & Power] in terms of **operations** and program **equations**.
- The correspondence between **algebraic theories** and **strong monads** on Set can be used to recover Moggi's monads.

Question

Effects that dynamically allocate resources (local) need more sophisticated algebraic theories/monads. E.g. local state.

[Plotkin & Power'02], [Power'06], [Melliès'10,'14], [Staton'13]

Question

Can concurrency (forking threads and waiting for them) be axiomatized as a local algebraic effect?

Ongoing work using **parameterized algebraic theories** [Staton'13].

Outline

1. Parameterized algebraic theories
2. Parameterized theory of threads with names
3. Operational semantics for threads with names

Parameterized algebraic theories [Staton FOSSACS'13, LICS'13, POPL'15]

- Uniform framework for axiomatizing local effects:

Example	Parameters
local state	location names

read(a, x, y) read the bit stored in location a and continue as either x or y
a is a free parameter

new$_0(a.x(a))$ create a new location a, containing 0
a is a fresh parameter, bound

$+$ other operations and equations

- Extend algebraic theories by allowing binding of abstract parameters.

Parameterized algebraic theories [Staton FOSSACS'13, LICS'13, POPL'15]

- Uniform framework for axiomatizing local effects:

Example	Parameters
local state	location names
π-calculus (fragment)	communication channels
first-order logic	individuals
quantum computation	qubits

- Extend algebraic theories by allowing binding of abstract parameters.
- Correspondence to monads on a functor category.

π-calculus (fragment): does not contain parallel composition as an operation

see also [Stark'08], [van Glabbeek & Plotkin'10]

Outline

Parameterized theory of threads with names

Parameters = thread IDs

Operations:

$\mathsf{fork}(a.x(a),\ y)$	x = parent thread; variable standing for another term y = child thread; variable a = ID of child; bound name that x can use, but y can't
$\mathsf{wait}(a,\ x)$	wait for the thread named a to finish, continue as x
stop	this thread has finished (no continuation)
$\mathsf{print}_s(x)$	print s (observable behaviour), continue as x

Forking and waiting are similar to the ones in Unix.

(Tentative) Equations for the parameterized theory of threads

$$y : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{stop},\, y) = y$$

(Tentative) Equations for the parameterized theory of threads

$$y : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{stop},\, y) = y \qquad x : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{wait}(a, x),\, \mathsf{stop}) = x$$

(Tentative) Equations for the parameterized theory of threads

$$y : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{stop},\ y) = y \qquad x : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{wait}(a, x),\ \mathsf{stop}) = x$$

$$x : 2, y : 1 \mid b \vdash \mathsf{fork}(a.\mathsf{wait}(b, x(b, a)),\ \mathsf{wait}(b, y(b))) = \mathsf{wait}(b, \mathsf{fork}(a.x(b, a),\ y(b)))$$

(Tentative) Equations for the parameterized theory of threads

$$y : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{stop},\ y) = y \qquad x : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{wait}(a, x),\ \mathsf{stop}) = x$$

$$x : 2, y : 1 \mid b \vdash \mathsf{fork}(a.\mathsf{wait}(b, x(b, a)),\ \mathsf{wait}(b, y(b))) = \mathsf{wait}(b, \mathsf{fork}(a.x(b, a),\ y(b)))$$

And many more:

$$x : 0 \mid - \vdash \mathsf{fork}(a.x,\ \mathsf{stop}) = x \qquad y : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{wait}(a, \mathsf{stop}), y) = y$$

$$x : 1 \mid a \vdash \mathsf{wait}(a, \mathsf{wait}(a, x(a))) = \mathsf{wait}(a, x(a))$$

$$x : 2 \mid a, b \vdash \mathsf{wait}(a, \mathsf{wait}(b, x(a, b))) = \mathsf{wait}(b, \mathsf{wait}(a, x(a, b)))$$

$$x : 2, y_1, y_2 : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{fork}(b.x(a, b),\ y_2),\ y_1) = \mathsf{fork}(b.\mathsf{fork}(a.x(a, b),\ y_1),\ y_2)$$

$$x, y : 0 \mid - \vdash \mathsf{fork}(a.x,\ y) = \mathsf{fork}(a.y,\ x)$$

$$x, y : 1, z : 0 \mid - \vdash \mathsf{fork}(a.x(a),\ \mathsf{fork}(b.y(b),\ z)) = \mathsf{fork}(b.\mathsf{fork}(a.x(a),\ y(b)),\ z)$$

(Tentative) Equations for the parameterized theory of threads

$$y : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{stop},\ y) = y \qquad x : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{wait}(a, x),\ \mathsf{stop}) = x$$

$$x : 2, y : 1 \mid b \vdash \mathsf{fork}(a.\mathsf{wait}(b, x(b, a)),\ \mathsf{wait}(b, y(b))) = \mathsf{wait}(b, \mathsf{fork}(a.x(b, a),\ y(b)))$$

And many more:

$$x : 0 \mid - \vdash \mathsf{fork}(a.x,\ \mathsf{stop}) = x \qquad y : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{wait}(a, \mathsf{stop}), y) = y$$

$$x : 1 \mid a \vdash \mathsf{wait}(a, \mathsf{wait}(a, x(a))) = \mathsf{wait}(a, x(a))$$

$$x : 2 \mid a, b \vdash \mathsf{wait}(a, \mathsf{wait}(b, x(a, b))) = \mathsf{wait}(b, \mathsf{wait}(a, x(a, b)))$$

$$\ldots$$

Goal

Compare the equations with an operational semantics.

Outline

Operational semantics for threads with names

Configuration T = a set of running (named) threads

Labels = printed symbols

Labelled transition system:

$$T \uplus \{[a]\mathsf{fork}(b.t_1,\ t_2)\} \to T \uplus \{[a]t_1,\ [b]t_2\} \qquad b \text{ fresh}$$

$$T \uplus \{[a]\mathsf{wait}(b, t),\ [b]\mathsf{stop}\} \to T \uplus \{[a]t,\ [b]\mathsf{stop}\}$$

$$T \uplus \{[a]\mathsf{print}_s(t)\} \xrightarrow{s} T \uplus \{[a]t\}$$

Terms t_1 and t_2 are **contextually equivalent** if they have the same sets of traces in all contexts.

Operational semantics for threads with names

Goal

(1) Does equality in the theory imply contextual equivalence?

(2) And vice-versa?

(1) No, unless we remove some equations.

$$\sout{y : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{stop},\ y) = y} \qquad x : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{wait}(a, x),\ \mathsf{stop}) = x$$

$$x : 2, y : 1 \mid b \vdash \mathsf{fork}(a.\mathsf{wait}(b, x(b, a)),\ \mathsf{wait}(b, y(b))) = \mathsf{wait}(b, \mathsf{fork}(a.x(b, a),\ y(b)))$$

$$x : 0 \mid - \vdash \mathsf{fork}(a.x,\ \mathsf{stop}) = x \qquad y : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{wait}(a, \mathsf{stop}), y) = y$$

$$x : 1 \mid a \vdash \mathsf{wait}(a, \mathsf{wait}(a, x(a))) = \mathsf{wait}(a, x(a))$$

$$x : 2 \mid a, b \vdash \mathsf{wait}(a, \mathsf{wait}(b, x(a, b))) = \mathsf{wait}(b, \mathsf{wait}(a, x(a, b)))$$

$$x : 2, y_1, y_2 : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{fork}(b.x(a, b),\ y_2),\ y_1) = \mathsf{fork}(b.\mathsf{fork}(a.x(a, b),\ y_1),\ y_2)$$

$$\sout{x, y : 0 \mid - \vdash \mathsf{fork}(a.x,\ y) = \mathsf{fork}(a.y,\ x)}$$

$$x, y : 1, z : 0 \mid - \vdash \mathsf{fork}(a.x(a),\ \mathsf{fork}(b.y(b),\ z)) = \mathsf{fork}(b.\mathsf{fork}(a.x(a),\ y(b)),\ z)$$

Operational semantics for threads with names

Goal

(1) Does equality in the theory imply contextual equivalence?

(2) And vice-versa?

(1) is hard to prove because of the quantification over all contexts.

Trace equivalence equates too many programs.

Question

What should we replace contextual equivalence with? What is a good notion of trace?

Trace equivalence equates too many programs

$$\sout{y : 0 \mid - \vdash \mathsf{fork}(a.\mathsf{stop},\ y) = y}$$

$$t_1 = \mathsf{fork}(a.\mathsf{stop},\ \mathsf{print}_1(\mathsf{stop})) \qquad \{1\}$$

$$t_2 = \mathsf{print}_1(\mathsf{stop}) \qquad \{1\}$$

Terms t_1 and t_2 are trace equivalent, but not contextually equivalent:

$$C = \mathsf{fork}(b.\mathsf{wait}(b,\ \mathsf{print}_2(\mathsf{stop})),\ \square)$$

$$C[t_1] \qquad \{21,\ 12\}$$

$$C[t_2] \qquad \{12\}$$

Summary

Work in progress about:

- axiomatizing Unix fork and wait
- as an algebraic theory, parameterized by thread ID's
- and comparing to an operational semantics

Question

What is an appropriate notion of program equivalence? How does it compare to the axiomatization?