The second reason is that this first-order continuation semantics becomes a theoretical basis or guide for a compiler for eager functional languages.

The situation is analoguous to the cps transformation that we looked at. The transformation is derived from the continuation semantics. Similarly, from the first-order semantics, we are able to derive a program (or expression) transformation sometimes called defunctionalisation, which gets rid of all higher-order functions from the given program.

By the way, this kind of connection between (denotational) semantics and compilation should not be too surprising. In a sense, a denotational semantics is a compiler of programs into phrases in mathematics. If the compiler uses only very restricted phrases, the compiled phrases can be understood as instruction sequences in a computer.

(3) Let's define the first-order semantics. It is based on the observation that when we interpret an expression e in the continuation semantics, we do not use all functions, but specific kinds of functions. In a sense, the first-order semantics replaces V_{fun}, V_{cont} and $E = V^{\langle var \rangle}$ by three sets $\widehat{V}_{fun}$, $\widehat{V}_{cont}$ and $\widehat{E}$ that consist of mathematical instructions. Then, it defines how to interpret those instructions.

We consider an eager functional language with integers and continuation values. Here are predomains and domains used in the first-order semantics.

$$\widehat{V}_* = (\widehat{V} + \{err, typerr\})_\perp$$

$$\widehat{V} \underset{\psi}{\overset{\phi}{\rightleftarrows}} V_{int} + \widehat{V}_{fun} + \widehat{V}_{cont}$$

$$V_{int} = \mathbb{Z}.$$

$$\widehat{V}_{fun} = \{abstract\} \times \langle var \rangle \times \langle exp \rangle \times \widehat{E}$$

typical element ---- $\langle abstract, v, e, \eta \rangle$

indicates this tuple represents a lambda expression $\lambda v.e$ and an environment η for the free variables in the expression.

empty

initial environment.

$$\widehat{E} = \{initenv\}$$
$$\cup \{extend\} \times \langle var \rangle \times \widehat{V} \times \widehat{E}$$
$$\cup \{recenv\} \times \widehat{E} \times \langle var \rangle \times \langle var \rangle \times \langle exp \rangle$$

$\langle extend, v, z, \eta \rangle$ environment obtained by extending η with the binding $v \mapsto z$

$\langle recenv, \eta, u, v, e \rangle$ environment obtained by extending η with the recursively defined u. (i.e., $u = \lambda v.e$).

$$\widehat{V}_{cont} = \{negate\} \times \widehat{V}_{cont}$$

→ negates its input and call conti.

$$\cup \{add_1, div_1, mul_1\} \times \langle exp \rangle \times \widehat{E} \times \widehat{V}_{cont}$$
$$\cup \{add_2, div_2, mul_2\} \times V_{int} \times \widehat{V}_{cont}$$

continuations for addition, ÷, *

$$\cup \{app_1\} \times \langle exp \rangle \times \widehat{E} \times \widehat{V}_{cont}$$
$$\cup \{app_2\} \times \widehat{V}_{fun} \times \widehat{V}_{cont}$$

continuations for function application

continuations for callcc, throw

$$\cup \{ccc\} \times \widehat{V}_{cont} \quad \cup \{thw\} \times \langle exp \rangle \times \widehat{E} \quad \cup \{initcont\}$$

Also initial continuation.