

Computer Supported Modeling and Reasoning

David Basin, Achim D. Brucker, Jan-Georg Smaus, and
Burkhardt Wolff

April 2005

<http://www.infsec.ethz.ch/education/permanent/csmr/>

Higher-Order Logic Application: Denotational Semantics for Functional Languages

Burkhart Wolff

Global Outline (1)

- Foundations
 - Foundational Axioms, Methodology, Historical Background, Principia Structure
 - Fixpoints and Inductive Sets
 - Well-founded Orders and Recursors
 - Arithmetic, Data-Types

Global Outline (2)

- Embeddings
 - Foundations,
Functional Languages and Denotational Semantics
 - Imperative Languages, Refinement Calculus
 - Z and Data-Refinement,
CSP and Process-Refinement
 - Object-oriented Languages (Java-Light . . .)

Global Outline (2)

- Embeddings
 - Foundations,
Functional Languages and Denotational Semantics
 - Imperative Languages, Refinement Calculus
 - Z and Data-Refinement,
CSP and Process-Refinement
 - Object-oriented Languages (Java-Light . . .)

Motivation

- Current stage of our course:
 - we have a logical framework for computer science
 - with set theory, total function recursion theory
 - proof support for: inductive sets, datatypes, primitive recursion definition
 - rich library

Motivation

- Current stage of our course:
 - we have a logical framework for computer science
 - with set theory, total function recursion theory
 - proof support for: inductive sets, datatypes, primitive recursion definition
 - rich library

⇒ how can we apply this framework to
specification and programming languages?

Representation Techniques for Semantics

Outline:

- Representing Languages in HOL

Outline:

- Representing Languages in HOL
 - shallow
 - deep

Outline:

- Representing Languages in HOL
 - shallow
 - deep
- Foundation for Functional Programming
 - sets and relations
 - cpo's

Outline:

- Representing Languages in HOL
 - shallow
 - deep
- Foundation for Functional Programming
 - sets and relations
 - cpo's
- Deriving Operational Semantics . . .

Question:

What is the *Meaning* of a
“Language”?

Syntax and Semantics

- syntax: language = set of symbols

Syntax and Semantics

- syntax: language = set of symbols
- semantics:
 - set of denotations, the “semantic *domain*”
 - meaning function (or: *interpretation*) relating these two

Syntax and Semantics

A Language:

- set of “words” (strings)
 - ⇒ concrete syntax
 - definition techniques:
 - inductive sets of strings in HOL

Syntax and Semantics

A Language:

- set of “words” (strings)
 - ⇒ concrete syntax
 - definition techniques:
 - inductive sets of strings in HOL
- set of “trees” (terms)
 - ⇒ abstract syntax
 - definition techniques:
 - ▷ abstract data types
 - ▷ constant definitions in HOL

Representation Techniques: An Example

Example: Regular Expressions

- The Language:
 - concrete syntax in a BNF-grammar:

rex ::= char "—" char

rex ::= char

rex ::= "."

rex ::= "(" rex ")"

rex ::= "[" rex "]"

rex ::= rex "*"

rex ::= rex "|" rex

rex ::= rex rex

Example: Regular Expressions

- The Language:

- concrete syntax: BNF-grammar as inductive definition (**Version 1**):

consts rex :: string set

inductive "rex"

intros

range $[(x :: \text{char})] @ \text{"-"} @ [(y :: \text{char})] \in \text{rex}$

char $[(x :: \text{char})] \in \text{rex}$

dot $\text{"."} \in \text{rex}$

par $r_1 \in \text{rex} \implies \text{"("} @ r_1 @ \text{"}")} \in \text{rex}$

bracket $r_1 \in \text{rex} \implies \text{"["} @ r_1 @ \text{"]"} \in \text{rex}$

star $r_1 \in \text{rex} \implies r_1 @ \text{"*"} \in \text{rex}$

alt $[[r_1 \in \text{rex}; r_2 \in \text{rex}]] \implies r_1 @ \text{"|"} @ r_2 \in \text{rex}$

seq $[[r_1 \in \text{rex}; r_2 \in \text{rex}]] \implies r_1 @ r_2 \in \text{rex}$

Example: Regular Expressions

- The Language:
 - Well-known **problems**: grammars not deterministic, . . .

Example: Regular Expressions

- The Language:
 - Well-known **problems**: grammars not deterministic, . . .
 - therefore precedences, auxiliary non-terminals, . . .

rex ::= sx [" | " rx]

sx ::= tx [sx]

tx ::= ax ["*" | "+" | "?"]

ra ::= char " - " char

mx ::= ra [mx]

ax ::= char

ax ::= "."

ax ::= "(" rex ")"

ax ::= "[" mx "]"

Example: Regular Expressions

- The Language:
 - (well-known) **solution**: *abstract* syntaxes implemented as data-type (**Version 2**):

Example: Regular Expressions

- The Language:
 - (well-known) **solution**: *abstract* syntaxes implemented as data-type (**Version 2**):

```
datatype rex =  
range char char (" _—_")  
| char char (" (-)")  
| dot (" .")  
| bracket rex (" [_]")  
| star rex (" _*")  
| alt rex rex (" _ | _")  
| seq rex rex (" _ _")
```

Example: Regular Expressions

- The Language:
 - (well-known) **solution**: *abstract* syntaxes implemented as data-type (**Version 2**):

```
datatype rex =  
range char char (" _-_" )  
| char char (" (-)" )  
| dot (" ." )  
| bracket rex (" [_]" )  
| star rex (" _*" )  
| alt rex rex (" _ |_" )  
| seq rex rex (" _ _" )
```

- Note:
 - no "par"-variant necessary!
 - priorities omitted!

Example: Regular Expressions

- The Language:
 - (well-known) **solution**: *abstract* syntaxes implemented as signature (**Version 3**):

Example: Regular Expressions

- The Language:

- (well-known) **solution**: *abstract* syntaxes implemented as signature (**Version 3**):

type rex

consts

range :: [char, char] ⇒ rex (" _—_ ")

char :: char ⇒ rex (" _ ")

dot :: rex (" . ")

bracket :: rex ⇒ rex (" [_] ")

star :: rex ⇒ rex (" _* ")

alt :: [rex, rex] ⇒ rex (" _ | _ ")

seq :: [rex, rex] ⇒ rex (" _ _ ")

Example: Regular Expressions

- The Language:

- Input into Isabelle:

can be identical for all three versions,

but highly different in their internal representation!

$$a(c - d)^*$$

(provided a, c and d are the usual character constants . . .)

Question:

How can we represent semantics?

Semantic Representation

- Deep Embeddings:
 - syntax as *explicit* datatype (e.g. Version 2)
 - interpretation as explicit function mapping each element of the language to a value

Semantic Representation

- Deep Embeddings:
 - syntax as *explicit* datatype (e.g. Version 2)
 - interpretation as explicit function mapping each element of the language to a value

- Shallow Embedding:
 - syntax implicit in *notation* for operators on the semantic domain (based on Version 3)

Semantic Representation: Example

- Deep Embeddings (based on Version 2):
 - semantic function L primitive recursive:

Semantic Representation: Example

- Deep Embeddings (based on Version 2):

- semantic function L primitive recursive:

consts $L :: \text{rex} \Rightarrow \text{string set}$

primrec L

$$L(\text{range } x \ y) = \{[a] \mid a. x \leq a \wedge a \leq y\}$$

$$L(\text{char } x) = \{[x]\}$$

$$L(\text{dot}) = \{[x] \mid x. \text{True}\}$$

$$L(\text{bracket } r) = \{[]\} \cup L(r)$$

$$L(\text{star } r) = \text{lfp}(\lambda X. \{[]\} \cup \{x@y \mid x,y. x \in L(r) \wedge y \in X\})$$

$$L(\text{alt } r_1 \ r_2) = L(r_1) \cup L(r_2)$$

$$L(\text{seq } r_1 \ r_2) = \{x @ y \mid x,y. x \in L(r_1) \wedge y \in L(r_2)\}$$

where $\{f \ a \mid a. P \ a\} \equiv f' \{a. P \ a\}$

Representation

- Deep Embeddings: Question

Why does Version 1 does not work here
for use with primitive recursion?

Semantic Representation: Example

- Shallow Embeddings (based on Version 3)

- Operators are directly interpreted in domain:

type rex = string set

defs

range_def $\text{range } x \ y \equiv \{[a] \mid a. x \leq a \wedge a \leq y\}$

char_def $\text{char } x \equiv \{[x]\}$

dot_def $\text{dot} \equiv \{[x] \mid x. \text{True}\}$

bracket_def $\text{bracket } r \equiv \{[]\} \cup r$

star_def $\text{star } r \equiv \text{lfp } (\lambda X. \{[]\} \cup \{x@y \mid x,y. x \in r \wedge y \in X\})$

alt_def $\text{alt } r_1 \ r_2 \equiv r_1 \cup r_2$

seq_def $\text{seq } r_1 \ r_2 \equiv \{x @ y \mid x,y. x \in r_1 \wedge y \in r_2\}$

where $\{f \ a \mid a. P \ a\} \equiv f' \{a. P \ a\}$

Semantic Representation: Example

- Shallow Embeddings (based on Version 3):
 - Can we induce over shallow embeddings?

Semantic Representation: Example

- Shallow Embeddings (based on Version 3):
 - Can we induce over shallow embeddings?
 - Yes, if we can give the domain an inductive structure, e.g.:

Semantic Representation: Example

- Shallow Embeddings (based on Version 3):

consts REX :: string set set

inductive "REX"

intros

range range $x\ y \in \text{REX}$

char char $x \in \text{REX}$

dot dot $\in \text{REX}$

bracket $r_1 \in \text{REX} \implies \{\text{bracket } x \mid x. x \in r_1\} \in \text{REX}$

star $r_1 \in \text{REX} \implies \{\text{star } x \mid x. x \in r_1\} \in \text{REX}$

alt $\llbracket r_1 \in \text{REX}; r_2 \in \text{REX} \rrbracket \implies$
 $\{\text{alt } x\ y \mid x,y. x \in r_1 \wedge y \in r_2\} \in \text{REX}$

seq $\llbracket r_1 \in \text{REX}; r_2 \in \text{REX} \rrbracket \implies$
 $\{\text{seq } x\ y \mid x,y. x \in r_1 \wedge y \in r_2\} \in \text{REX}$

- from *REX* to *rex* we can go via type definition . . .

Semantic Representation: Example

- Shallow Embeddings (based on Version 3):
 - The main advantage: we inherit the binding structure of HOL!

Semantic Representation: Example

- Shallow Embeddings (based on Version 3):
 - The main advantage: we inherit the binding structure of HOL!
 - Example: We can add

$$\mu X.f X = lfp f$$

into the rex-language and have:

$$\mu X.a[X]a = \{a^{2^n}.n\}!!!$$

Semantic Representation: Example

- Shallow Embeddings (based on Version 3):

- Note: In

$$\mu X.a[X]a = \{a^{2^n}.n\},$$

X is a HOL-Variable, a a constant, etc.

In a deep embedding, we would have to introduce a constant set (e.g. X_1, X_2, \dots) for variables and handle them in own substitution functions

Semantic Representation: Example

- Shallow Embeddings (based on Version 3):

- Note: In

$$\mu X.a[X]a = \{a^{2^n}.n\},$$

X is a HOL-Variable, a a constant, etc.

In a deep embedding, we would have to introduce a constant set (e.g. $X\ 1, X\ 2, \dots$) for variables and handle them in own substitution functions

and take care of name clashes and name captures ourselves !

Semantic Representation: Example

- Shallow Embeddings (based on Version 3):
 - The main advantage:
we inherit the binding structure of HOL!
 - In deep embeddings, this requires extra

Semantic Representation: Example

- Shallow Embeddings (based on Version 3):
 - The main advantage:
we inherit the binding structure of HOL!
 - In deep embeddings, this requires extra
variable symbols,

Semantic Representation: Example

- Shallow Embeddings (based on Version 3):

- The main advantage:
we inherit the binding structure of HOL!
- In deep embeddings, this requires extra

variable symbols,

substitution functions,

Semantic Representation: Example

- Shallow Embeddings (based on Version 3):

- The main advantage:
we inherit the binding structure of HOL!
- In deep embeddings, this requires extra

variable symbols,

substitution functions,

typing functions, . . .

Semantic Representation: Example

- Shallow Embeddings (based on Version 3):
 - Drawbacks:
 - ▷ binding structure may be too tight
(blockstructured, λ -calculus oriented)

Semantic Representation: Example

- Shallow Embeddings (based on Version 3):
 - Drawbacks:
 - ▷ binding structure may be too tight
(blockstructured, λ -calculus oriented)
 - ▷ typing may be too tight

Semantic Representation: Example

- Shallow Embeddings (based on Version 3):
 - Drawbacks:
 - ▷ binding structure may be too tight
(blockstructured, λ -calculus oriented)
 - ▷ typing may be too tight
 - ▷ the semantic domain may **not** have
an inductive structure ($\alpha, \alpha \Rightarrow \beta, \alpha \text{ set}$)

Semantic Representation: Example

- Shallow Embeddings (based on Version 3):
 - Drawbacks:
 - ▷ binding structure may be too tight (blockstructured, λ -calculus oriented)
 - ▷ typing may be too tight
 - ▷ the semantic domain may **not** have an inductive structure ($\alpha, \alpha \Rightarrow \beta, \alpha$ set)
 - ▷ unfortunately, this rules out some crucial meta-language proofs (e.g. completeness proofs)

Semantic Representation

- Further Examples:
 - Deep Embeddings:

Semantic Representation

- Further Examples:
 - Deep Embeddings:
 - ▷ Theory/Collection “Lambda” (Isabelle99)

Semantic Representation

- Further Examples:
 - Deep Embeddings:
 - ▷ Theory/Collection “Lambda” (Isabelle99)
 - ▷ Languages IMP, NanoJava, ProofPower-Z
 - Shallow Embeddings:

Semantic Representation

- Further Examples:
 - Deep Embeddings:
 - ▷ Theory/Collection “Lambda” (Isabelle99)
 - ▷ Languages IMP, NanoJava, ProofPower-Z
 - Shallow Embeddings:
 - ▷ HOL itself !!! (λ , \exists , $@$, . . .)

Semantic Representation

- Further Examples:
 - Deep Embeddings:
 - ▷ Theory/Collection “Lambda” (Isabelle99)
 - ▷ Languages IMP, NanoJava, ProofPower-Z
 - Shallow Embeddings:
 - ▷ HOL itself !!! (λ , \exists , $@$, . . .)
 - ▷ HOLCF ([MNOS99]; see Isabelle distribution)

Semantic Representation

- Further Examples:

- Deep Embeddings:

- ▷ Theory/Collection “Lambda” (Isabelle99)

- ▷ Languages IMP, NanoJava, ProofPower-Z

- Shallow Embeddings:

- ▷ HOL itself !!! (λ , \exists , $@$, . . .)

- ▷ HOLCF ([MNOS99]; see Isabelle distribution)

- ▷ HOL-Z [BRW03], HOL-CSP [TW97]

Semantic Representation

- Further Examples:

- Deep Embeddings:

- ▷ Theory/Collection “Lambda” (Isabelle99)
- ▷ Languages IMP, NanoJava, ProofPower-Z

- Shallow Embeddings:

- ▷ HOL itself !!! (λ , \exists , $@$, . . .)
- ▷ HOLCF ([MNOS99]; see Isabelle distribution)
- ▷ HOL-Z [BRW03], HOL-CSP [TW97]
- ▷ MiniML (discussed in the following)

Semantic Representation

- Further Examples:

- Deep Embeddings:

- ▷ Theory/Collection “Lambda” (Isabelle99)
- ▷ Languages IMP, NanoJava, ProofPower-Z

- Shallow Embeddings:

- ▷ HOL itself !!! (λ , \exists , $@$, . . .)
- ▷ HOLCF ([MNOS99]; see Isabelle distribution)
- ▷ HOL-Z [BRW03], HOL-CSP [TW97]
- ▷ MiniML (discussed in the following)

- In-between: MicroJava [NOP00]

Scott's Approach to Denotational Semantics

- Overall aim: Having a structure with a fixpoint property:

$$\mu X.E(X) = E(\mu X.E(X))$$

- Topological Approach
(denotational, Scott/Strachey Approach):

(\sqsubseteq, C) is a “complete partial order” (CPO)

- Idea: We represent (\sqsubseteq, C) as subclass of order!

Denotational Semantics

- Some basic definitions:

upper bound $S \leq x \equiv \forall y. y \in S \rightarrow y \leq x$

is_least_ub $S \ll x \equiv S \leq x \wedge (\forall u. S \leq u \rightarrow x \leq u)$

lub $\text{lub}(S) \equiv \text{THE } x. S \ll x$

directed $X \equiv (X \neq \{\}) \wedge$
 $(\forall a \in X, b \in X. \exists c \in X. a \leq c \wedge b \leq c)$

fix $f \equiv \text{lub}(\text{range}(\lambda i. \text{iterate } i \ F \ \perp))$

Denotational Semantics

- Some basic definitions:

classes cpo0 < order

consts $\perp :: 'a :: \text{cpo0}$ (**In cpo's there is a constant Bottom**)

axclass

cpo < cpo0

least " $\perp \leq x$ " (** ... which is least**)

complete "(directed X) $\implies (\exists b. X \ll b)$ "

Denotational Semantics

- Some basic definitions:

- continuity:

$$\text{cont } f \equiv \forall Y. Y \subseteq A \wedge \text{directed } Y \rightarrow f'Y \ll |f(\text{lub } Y)$$

where $f :: 'a :: \text{cpo} \Rightarrow 'b :: \text{cpo}$

Denotational Semantics

- Some basic definitions:

- continuity:

$$\text{cont } f \equiv \forall Y. Y \subseteq A \wedge \text{directed } Y \rightarrow f'Y \ll f(\text{lub } Y)$$

where $f :: 'a :: \text{cpo} \Rightarrow 'b :: \text{cpo}$

- admissibility:

$$\text{adm}(P) \equiv$$

$$\forall Y. Y \subseteq A \wedge \text{directed } Y \rightarrow (\forall x \in Y. P \ x) \rightarrow P(\text{lub } Y)$$

where $P :: 'a :: \text{cpo} \Rightarrow \text{bool}$

Denotational Semantics

- Main Theorems:

- recursion (Knaster-Tarski):

$$\text{cont } f \implies \mathbf{fix} \ f = f \ (\mathbf{fix} \ f)$$

where $f :: 'a :: \text{cpo} \Rightarrow 'a :: \text{cpo}$

- fixpoint induction:

$$\llbracket \text{cont } f; \text{adm}(P); P(\perp); \forall x. P(x) \implies P(f \ x) \rrbracket \implies P(\mathbf{fix} \ f)$$

Denotational Semantics

- Minor Theorems:

$$(\forall x. \text{cont}(f \ x)) \implies \text{cont}(\lambda x. \mathbf{fix} \ (f \ x))$$

$$\text{cont}(\text{Pair}), \text{cont}(\text{fst}), \text{cont}(\text{snd})$$

$$\text{cont}(\lambda x. c)$$

$$\text{cont}(\lambda x. x)$$

$$\text{cont}(f) \wedge \text{cont}(g) \implies \text{cont}(f \circ g)$$

$$\text{cont}(u) \wedge \text{cont}(v) \implies \text{adm}(\lambda x. u \ x \leq v \ x)$$

$$\text{cont}(u) \wedge \text{cont}(v) \implies \text{adm}(\lambda x. u \ x \leq v \ x)$$

$$\text{adm } P \wedge \text{adm } Q \implies \text{adm}(\lambda x. P \ x \wedge Q \ x)$$

$$\text{adm } P \wedge \text{adm } Q \implies \text{adm}(\lambda x. P \ x \vee Q \ x)$$

Denotational Semantics

- Constructions: The “flat cpo”

datatype 'a up = lift 'a | down

instance up :: (term) ord

constdefs

drop :: 'a :: cpo up \Rightarrow 'a

”drop x \equiv case x **of** lift v \longrightarrow v | down $\longrightarrow \perp$ ”

le_up_def ”x \leq y \equiv case x **of**

lift v \longrightarrow (case y **of** lift u \longrightarrow v = u
| down \longrightarrow False)

| down \longrightarrow True”

Denotational Semantics

- Extremely Nifty Constructions: Products, Function Space.

instance

" \times " :: (ord, ord) ord

defs

le_pair_def " $x \leq y \equiv ((fst\ x \leq fst\ y) \wedge (snd\ x \leq snd\ y))$ "

arities

fun :: (term, order) order (* *fun* = \longrightarrow *)

defs

le_fun_def " $f \leq g \equiv (\forall x. f\ x \leq g\ x)$ "

Denotational Semantics

- Extremely Nifty Consequences

Dom2 = Dom1 +

instance up :: (term) cpo (⊥_up_def, le_up_least ,
le_up_complete)

instance

"×" :: (cpo,cpo) cpo (⊥_pair_def, le_pair_least ,

instance

fun :: (term,cpo) cpo (le_fun_least , le_fun_complete)

⇒ CPO-checking is type checking!

Example: Shallow Embedding of MiniML

- Core Definitions of MiniML [Win96]:
 one exception (EXN), apply, closure, if_then_else, . . .
- EXN_def EXN $\equiv \perp$
- APPLY_def F ^! x \equiv if x = \perp then \perp
 else if F = \perp then \perp else (drop F) x
- LAM_def Lam f \equiv lift f (* LAM x. f x*)
- REC_def REC f \equiv **fix** f
- IF_def (IF x THEN y ELSE z) \equiv (case x **of**
 lift v \Rightarrow if v then y else z
 | down $\Rightarrow \perp$)
- LET_def LET s f \equiv f ^! s

Denotational Semantics

- More Definitions of MiniML [Win96]:
Basic Operations, Global Declarations.

types Int = int up Bool = bool up

constdefs TIMES :: [Int, Int] \Rightarrow Int
 "TIMES \equiv strictify ($\lambda x :: \text{nat.}$ strictify ($\lambda y.$ lift ($x * y$)))"
 DIV :: [Int, Int] \Rightarrow Int
 "DIV \equiv strictify ($\lambda x :: \text{nat.}$ strictify ($\lambda y.$
 if y = lift 0 then \perp else lift ($x * y$))))"

constdefs VAL :: "'a, 'a] \Rightarrow bool"
 "VAL f E \equiv (f = E)"

$$\text{FUN} :: \text{'' } [\text{'a} :: \text{cpo}, \text{'a} \Rightarrow \text{'a}] \Rightarrow \text{bool''}$$
$$\text{'' FUN } f \text{ F} \equiv (f = \text{REC}(F)) \wedge_{\text{cont}} F\text{''}$$

Denotational Semantics

- Now we can derive the operational semantics of MiniML: evaluation relation, canonical forms [Win96]

constdefs $\text{eval} :: [\text{'a} :: \text{cpo}, \text{'a}] \Rightarrow \text{bool}$

"eval s t \equiv (s = t)"

$\text{cf} :: [\text{'a} :: \text{cpo}] \Rightarrow \text{bool}$

"cf t \equiv (t \sim = \perp)"

syntax " -A -> " :: ['a, 'a]}] \Rightarrow bool (**infixl** 50)

translations

"s -A -> t" \equiv "eval s t"

Denotational Semantics

- Now we can **derive** the operational semantics of MiniML: strict beta-reduction, if_then_else with EXN ($= \perp$).

$$\begin{aligned} & \llbracket \text{cf } c; \text{ cf } c_2; \\ & \quad t_1 \text{ } \text{-A} \text{ } \text{->} (\text{LAM } x. t \ x); \\ & \quad t_2 \text{ } \text{-A} \text{ } \text{->} c_2; (t(c_2)) \text{ } \text{-A} \text{ } \text{->} c \rrbracket \\ \implies & (t_1 \text{ } \hat{!} \ t_2) \text{ } \text{-A} \text{ } \text{->} c \end{aligned}$$

$$\begin{aligned} & \llbracket t_1 \text{ } \text{-A} \text{ } \text{->} \text{EXN}; t_2 \text{ } \text{-A} \text{ } \text{->} c_2 \rrbracket \\ \implies & (t_1 \text{ } \hat{!} \ t_2) \text{ } \text{-A} \text{ } \text{->} \text{EXN} \end{aligned}$$

$$\llbracket t_1 \text{ } \text{-A} \text{ } \text{->} (\text{LAM } x. t \ x);$$

$$\begin{aligned} & \llbracket t_2 \text{ -A -> EXN } \rrbracket \\ \implies & (t_1 \text{ ^! } t_2) \text{ -A -> EXN} \end{aligned}$$

$$\begin{aligned} & \llbracket \text{cf } c_2; t_1 \text{ -A -> TRUE;} \\ & \quad t_2 \text{ -A -> } c_2 \rrbracket \\ \implies & (\text{IF } t_1 \text{ THEN } t_2 \text{ ELSE } t_3) \text{ -A -> } c_2 \end{aligned}$$

$$\begin{aligned} & \llbracket \text{cf } c_2; t_1 \text{ -A -> FALSE;} t_3 \text{ -A -> } c_2 \rrbracket \\ \implies & (\text{IF } t_1 \text{ THEN } t_2 \text{ ELSE } t_3) \text{ -A -> } c_2 \end{aligned}$$

$$\begin{aligned} & \llbracket t_1 \text{ -A -> EXN } \rrbracket \\ \implies & (\text{IF } t_1 \text{ THEN } t_2 \text{ ELSE } t_3) \text{ -A -> EXN} \end{aligned}$$

Denotational Semantics

- Now we can **derive** the operational semantics of MiniML: recursion, basic operations.

$$\begin{aligned} \llbracket \text{cont}(\lambda X. \text{lift } (f X)) \rrbracket &\implies \\ \text{REC}(\lambda X. (\text{LAM } x_1. f X x_1)) \text{ --A -->} & \\ (\text{LAM } x_1. f (\text{REC}(\lambda X. (\text{LAM } x_1. f X x_1)))) x_1 & \end{aligned}$$

$$\begin{aligned} \llbracket \text{cf } c_1; \text{ cf } c_2; t_1 \text{ --A -->} c_1; t_2 \text{ --A -->} c_2 \rrbracket &\implies \\ (\text{TIMES } t_1 t_2) \text{ --A -->} \text{lift}(c_1 * c_2) & \end{aligned}$$

Summary

- Isabelle/HOL is a powerful framework for embedding programming languages
 - via deep embeddings
 - via shallow embeddings
 - denotational semantics can be used as definitional basis for operational semantics !!!

Summary

- Isabelle/HOL is a powerful framework for embedding programming languages
 - via deep embeddings
 - via shallow embeddings
 - denotational semantics can be used as definitional basis for operational semantics !!!

- First Principle for language embeddings:
KEEP IT AS SHALLOW AS POSSIBLE !!!

References

- [BRW03] Achim D. Brucker, Frank Rittinger, and Burkhart Wolff. Hol-z 2.0: A proof environment for z-specifications. *Journal of Universal Computer Science*, 9(2):152–172, February 2003.
- [MNOS99] Olaf Müller, Tobias Nipkow, David von Oheimb, and Oskar Slotosch. HOLCF = HOL + LCF. *Journal of Functional Programming*, 9:191–223, 1999.
- [NOP00] Tobias Nipkow, David von Oheimb, and Cornelia Pusch. μ Java: Embedding a programming language in a theorem prover. In F.L. Bauer and R. Steinbrüggen, editors, *Foundations of Secure Computation. Proc. Int. Summer School Marktoberdorf 1999*, pages 117–144. IOS Press, 2000.
- [TW97] H. Tej and B. Wolff. A corrected failure-divergence model for csp in isabelle/hol. In J. Fitzgerald, C.B. Jones, and P. Lucas, editors, *Proceedings of the FME 97 — Industrial Applications and Strengthened Foundations of Formal Methods*, LNCS 1313, pages 318–337. Springer Verlag, 1997.

- [Win96] Glynn Winskel. *The Formal Semantics of Programming Languages – An Introduction*. MIT Press, 1996. 3rd ed.