A Semantics for Lazy Assertions

Olaf Chitil

University of Kent

25th January 2011
Asserions in Functional Languages

assert nats [4,2] ⇝ [4,2]
assert nats [4,-2] ⇝ exception

Assertion application is a partial identity.

assert :: Assertion t -> t -> t
nats :: Num t => Assertion [t]

Note: Contract = Assertion + Blaming
Lazy Assertions ...

... work with non-strict functions and infinite data structures.

fibs :: [Integer]
fibs = assert nats (0 : 1 : zipWith (+) fibs (tail fibs))

Need to consider partial values:

assert nats (0:1:⊥) ⇝ 0:1:⊥
assert nats (0:1:1:⊥) ⇝ 0:1:1:⊥
assert nats (0:1:1:2:⊥) ⇝ 0:1:1:2:⊥

Any approximation of an acceptable value has to be accepted!
let x = assert equal (True, False)
in (fst x, snd x) \rightarrow exception

but

(fst (assert equal (True, False)), (True, snd (assert equal (True, False)))) \rightarrow False

Because (True, ⊥) ⊑ (True, True)
(⊥, False) ⊑ (False, False) have to be accepted.
The Problem

let x = assert equal (True,False) in (fst x, snd x) ⇝ (True, error "...") or (error "...", False)

but

(fst (assert equal (True,False)), (True, snd (assert equal (True,False)))) ⇝ False

Because

(True,⊥) ⊑ (True,True)  
(⊥,False) ⊑ (False,False)  

have to be accepted.

Hence: First define semantics, then derive an implementation.
Resulting Lazy Assertions: List of natural numbers

\[
nats :: \text{Assert} \ [\text{Integer}] \\
\text{nats} = \text{aList} (\text{pred} (\geq 0))
\]

\[
aList :: \text{Assert} \ t \rightarrow \text{Assert} \ [t] \\
aList \ a = \text{aNil} \ | \ a\text{Cons} \ a \ (a\text{List} \ a)
\]
Resulting Lazy Assertions: Minimal list length

\[
\begin{align*}
\text{lengthAtLeast} & : \text{Int} \rightarrow \text{Assert } [t] \\
\text{lengthAtLeast} 0 & = \text{aAny} \\
\text{lengthAtLeast} (n+1) & = \text{aCons aAny (lengthAtLeast n)}
\end{align*}
\]

\[
\begin{align*}
\text{initAverage} & : [\text{Int}] \rightarrow \text{Int} \\
\text{initAverage} & = \text{assert (lengthAtLeast 5 } \rightarrow \text{ aAny) initAverage'} \\
\text{initAverage'} \text{ xs} & = \text{sum (take 5 xs) 'div' 5}
\end{align*}
\]
Resulting Lazy Assertions: Logic Example

Data type for formulae:

```haskell
data Form = Imp Form Form | And Form Form |
            Or Form Form | Not Form | Atom Char
```

Assertions for conjunctive normal form:

```haskell
conjNF, disj, lit, atom :: Assert Form

conjNF = aAnd conjNF conjNF <|> disj
disj = aOr disj disj <|> lit
lit = aNot atom <|> atom
atom = aAtom aAny
```

Conjunctive normal form with left-associated operators:

```haskell
leftConjNF :: Assert Form
leftConjNF = conjNF <&> left
```
Axioms of Semantics

Write $\langle a \rangle : D \rightarrow D$ for semantics of `assert a`. Domain $D$ is directed complete partial order with $\perp$.

**Definition**

Acceptance set $\llbracket a \rrbracket := \{ v \in D \mid \langle a \rangle v = v \} \subseteq D$.

**Definition**

$a$ is lazy assertion, if

1. $\langle a \rangle : D \rightarrow D$ is a continuous function,
2. $a$ is trustworthy, that is, $\langle a \rangle v \in \llbracket a \rrbracket$ for any value $v$, (equivalent: $a$ is idempotent)
3. $\langle a \rangle$ is a partial identity, that is, $\langle a \rangle v \sqsubseteq v$ for any value $v$, and
4. $\llbracket a \rrbracket$ is a lower set.
Assertions and Projections

Definition
A function $p : D \rightarrow D$ on a domain $D$ is a projection if it is
- continuous,
- idempotent, and
- a partial identity.

Lemma
$a$ is lazy assertion $\iff \langle a \rangle$ is projection and its image is a lower set

(cf. Findler & Blume, FLOPS 2006)
Looking for Alternative Axioms

### Definition

\[ \downarrow \{ v \} := \{ v' \mid v' \sqsubseteq v \} \]

\[ A_v := \downarrow \{ v \} \cap A \]

### Theorem

\([a]_v \] is an ideal (lower & directed)

\[ \langle a \rangle v = \bigcup [a]_v \]
Alternative Axioms

**Definition**

A set $A \subseteq D$ is a lazy domain if

- $A$ is lower,
- $A$ contains the least upper bound of any directed subset, and
- $A_v = \downarrow\{v\} \cap A$ is directed for all values $v \in D$.

**Lemma**

If $a$ is a lazy assertion, then $\llbracket a \rrbracket$ is a lazy domain.

**Theorem**

If $A$ is a lazy domain, then $a$ with

$$\langle a \rangle_v := \bighat\llbracket a \rrbracket_v$$

is a lazy assertion with $\llbracket a \rrbracket = A$. 
Assertion Combinators: Minimal & Maximal

**Definition**

\[
\begin{align*}
\llbracket \text{aNone} \rrbracket & := \{ \bot \} \\
\llbracket \text{aAny} \rrbracket & := D
\end{align*}
\]

**Derived assertion applications**

\[
\begin{align*}
\langle \text{aNone} \rangle_v & = \bigsqcup \llbracket \text{aNone} \rrbracket_v = \bigsqcup \downarrow \{v\} \cap \{\bot\} = \bigsqcup \{\bot\} = \bot \\
\langle \text{aAny} \rangle_v & = \bigsqcup \llbracket \text{aAny} \rrbracket_v = \bigsqcup \downarrow \{v\} \cap D = \bigsqcup \downarrow \{v\} = v
\end{align*}
\]
Assertion Combinators: Conjunction

Definition

\[ [a \&\& b] := [a] \cap [b] \]

Lemma Conjunction of assertions is commutative and associative and has the assertion \( \text{aAny} \) as neutral element.

Lemma (Conjunction equals two assertions)

\[ \langle a\&\& b \rangle \nu = \langle a \rangle (\langle b \rangle \nu) \]
Assertion Combinators: Disjunction

Not \([a \lor b] := [a] \cup [b]\)
because \([a \lor b]_v = (\downarrow\{v\} \cap [a]) \cup (\downarrow\{v\} \cap [b])\) not directed.

**Definition**

\([a \triangleleft|> b] := \bigcap\{Y \mid [a] \cup [b] \subseteq Y, Y \text{ lazy domain}\}\)

Attention!

\[D = \{\bot, (\bot, \bot), (\text{True}, \bot), (\text{False}, \bot), \ldots, (\text{False}, \text{False})\}\]

\([\text{fstTrue}] = D \setminus \{(\text{False}, \bot), (\text{False}, \text{True}), (\text{False}, \text{False})\}\]

\([\text{fstTrue} \triangleleft|> \text{sndTrue}] = D\]

\([\text{fstTrue} \triangleleft|> (\text{sndTrue}) \triangleleft|> (\text{fstFalse} \triangleleft|> \text{sndFalse})]\] = D\]

**Lemma** Disjunction of assertions is commutative and associative and has
the assertion aNone as neutral element.
Bounded Distributive Lattice of Assertions

**Lemma (Absorption laws)**

\[ a \& (a \lor b) = a \]
\[ a \lor (a \& b) = a \]

**Lemma (Distributive laws)**

\[ a \lor (b \& c) = (a \lor b) \& (a \lor c) \]
\[ a \& (b \lor c) = (a \& b) \lor (a \& c) \]

**Theorem** Lazy assertions form a bounded distributive lattice with meet \(<\&>\), join \(<\lor>\), least element \(aNone\) and greatest element \(aAny\). The ordering is the subset-relationship on acceptance sets.

**Corollary (Idempotency laws)**

\[ a \& a = a \]
\[ a \lor a = a \]
Let $[\neg a] := \{\bot, (\bot, \bot)\}$

$a <\&> \neg a = \text{aNone}$ implies $[a] \cap [\neg a] = \{\bot\}$.

$[\neg a]$ must be a lower set.

So $[\neg a] = \{\bot\}$.

But then $[a <|> \neg a] = [a]$.

Contradiction to $a <|> \neg a = \text{aAny}$. 
Deriving an Implementation: Primitive Data Types

Flat domain, i.e., $v \sqsubseteq w$ implies $v = \bot$.

**Definition (Acceptance set of predicate assertion)**

$$[[\phi]] := \{\bot\} \cup \{v \mid \phi v = \text{True}\}$$

Derive application of assertion predicate:

$$\langle \phi \rangle v = \bigsqcup \{v\} \cap [[\phi]]$$

$$= \bigsqcup \{\bot, v\} \cap (\{\bot\} \cup \{w \mid \phi w = \text{True}\})$$

$$= \bigsqcup \{\bot\} \cup (\text{if } \phi v \text{ then } \{v\} \text{ else } \{\})$$

$$= \text{if } \phi v \text{ then } v \text{ else } \bot$$

Note: $\{\bot\} \cup \{v \mid \phi v \neq \text{False}\}$ as acceptance set is un-implementable.
Primitive Data Types: Conjunction & Disjunction

Expected definitions:

\[ \phi \triangleleft \triangleright \psi := \lambda x. \phi x \land \psi x \]
\[ \phi \triangleleft \triangleright \psi := \lambda x. \phi x \lor \psi x \]

Verify they work:

\[ \llbracket \phi \triangleleft \triangleright \psi \rrbracket = \llbracket \phi \rrbracket \cap \llbracket \psi \rrbracket = \{ v \mid \phi v \land \psi v \} \]

\[ \llbracket \phi \triangleleft \triangleright \psi \rrbracket = \bigcap \{ X \mid \llbracket \phi \rrbracket \cup \llbracket \psi \rrbracket \subseteq X, X \text{ lazy domain} \} \]
\[ = \bigcap \{ X \mid \llbracket \phi \rrbracket \cup \llbracket \psi \rrbracket \subseteq X \} \]
\[ = \llbracket \phi \rrbracket \cup \llbracket \psi \rrbracket = \{ v \mid \phi v \lor \psi v \} \]

Negation is possible:

\[ \neg \phi := \lambda x. \neg (\phi x) \]
Definition (Acceptance set for pattern assertion)

\[
[C \ a_1 \ldots a_n] := \{ \bot \} \cup \{ C \ v_1 \ldots v_n \mid v_1 \in [a_1] \ldots v_n \in [a_n]\}
\]

Lemma (Conjunction of constructor assertions)

\[
(C \ a_1 \ldots a_n) \& (C \ b_1 \ldots b_n) = C \ (a_1 \& b_1) \ldots (a_n \& b_n)
\]
\[
(C \ a_1 \ldots a_n) \& (C' \ b_1 \ldots b_n) = pNone \quad \text{if } C \neq C'
\]

Lemma (Disjunction of constructor assertions)

\[
(C \ a_1 \ldots a_n) \triangleright (C \ b_1 \ldots b_n) = C \ (a_1 \triangleright b_1) \ldots (a_n \triangleright b_n)
\]

Also if \( C \neq C' \), then

\[
[(C \ a_1 \ldots a_n) \triangleright (C' \ b_1 \ldots b_n)] = [C \ a_1 \ldots a_n] \cup [C' \ b_1 \ldots b_n]
\]
Representation of constructor assertion

\[ C_1 \bar{a}_1 < | \> C_2 \bar{a}_2 < | \> \ldots < | \> C_m \bar{a}_m \]

where  \{C_1, \ldots, C_m\} is subset of all data constructors of the type.

Application of a constructor assertion

\[
\langle C_1 \bar{a}_1 < | \ldots < | \> C_m \bar{a}_m \rangle \ (C \bar{v}) = \left\{ \begin{array}{ll}
C \ (\langle \bar{a}_j \rangle \bar{v}) & \text{if } C = C_j \\
\perp & \text{otherwise}
\end{array} \right.
\]

\[
\langle C_1 \bar{a}_1 < | \ldots < | \> C_m \bar{a}_m \rangle \perp = \perp
\]
Algebraic Data Types

Conjunction

\[
(C_{i_1} \bar{a}_{i_1} < | > \ldots < | > C_{i_m} \bar{a}_{i_m}) < & > (C_{j_1} \bar{b}_{j_1} < | > \ldots < | > C_{j_l} \bar{b}_{j_l})
= \ C_{k_1} (\bar{a}_{k_1} < & > \bar{b}_{k_1}) < | > \ldots < | > C_{k_o} (\bar{a}_{k_o} < & > \bar{b}_{k_o})
\]

where \( \{k_1, \ldots, k_o\} = \{i_1, \ldots, i_m\} \cap \{j_1, \ldots, j_l\} \)

Disjunction

\[
(C_{i_1} \bar{a}_{i_1} < | > \ldots < | > C_{i_m} \bar{a}_{i_m}) < | > (C_{j_1} \bar{b}_{j_1} < | > \ldots < | > C_{j_l} \bar{b}_{j_l})
= \ C_{k_1} \bar{z}_{k_1} < | > \ldots < | > C_{k_o} \bar{z}_{k_o}
\]

where \( \{k_1, \ldots, k_o\} = \{i_1, \ldots, i_m\} \cup \{j_1, \ldots, j_l\} \)

\[
z_{k_s} = \begin{cases} 
\bar{a}_{k_s} < | > \bar{b}_{k_s} & \text{if } k_s \in \{i_1, \ldots, i_m\} \cap \{j_1, \ldots, j_l\} \\
\bar{a}_{k_s} & \text{if } k_s \in \{i_1, \ldots, i_m\} \setminus \{j_1, \ldots, j_l\} \\
\bar{b}_{k_s} & \text{if } k_s \in \{j_1, \ldots, j_l\} \setminus \{i_1, \ldots, i_m\} 
\end{cases}
\]
What about Function Types?

Function Assertion \( a \mapsto b \)

Standard definition of assertion application:

\[
\langle a \mapsto b \rangle \delta = \lambda x. \langle b \rangle (\delta(\langle a \rangle x))
\]

But

\[
\llbracket a \mapsto b \rrbracket = \{ \delta \mid \langle b \rangle \circ \delta \circ \langle a \rangle = \delta \}
\]

is not a lower set! However,

\[
\{ \delta \mid \forall v \in \llbracket a \rrbracket. \delta v \in \llbracket b \rrbracket \}
\]

is a lazy domain.

Need two acceptance sets, for argument and context.

(cf. Findler & Blume, FLOPS 2006)
Semantics

- Acceptance sets $[a]$ are lazy domains, subdomains.
- Algebra of assertions: bounded distributive lattice.

Lazy Assertions

- Derived as library from semantics.
- Laziness restricts expressibility!
- Pattern assertions similar to algebraic data types; subtypes!
- Pattern assertions are efficient.

Future

- Assertions to express non-strictness properties.
- (Dependent?) function assertion semantics.
Example: Normalisation of $\&$.

**Formula in conjunctive normal form with left-associated binary operators:**

\[
\text{conjNF} = \text{aAnd conjNF conjNF} \lor \text{aOr disj disj} \lor \text{aNot atom} \lor \\
\text{aAtom aAny}
\]

\[
\text{left} = \text{aImp left noImp} \lor \text{aAnd left noAnd} \lor \text{aOr left noOr} \lor \\
\text{aNot left} \lor \text{aAtom aAny}
\]

**Combined:**

\[
\text{leftConjNF} = \text{conjNF} \& \text{left} = \text{aAnd (conjNF} \& \text{left) (conjNF} \& \text{noAnd)} \lor \\
\text{aOr (disj} \& \text{left) (disj} \& \text{noOr)} \lor \\
\text{aNot (atom} \& \text{left)} \lor \\
\text{aAtom (aAny} \& \text{aAny)}
\]