

# Formalisation of Updates in Isabelle/HOL

Philipp Rümmer  
philipp@cs.chalmers.se

12th June 2006

Talk is a short overview:

- Update syntax, semantics and calculus . . .
- . . . and their formalisation in Isabelle/HOL
- Correctness properties of calculus
  
- Updates in KeY:  
Intermediate language for storing side-effects  
(talk describes version of updates currently implemented)
- Isabelle/HOL:  
Proof assistant for higher-order logic

# Part 1: The Syntax

# Updates as a Programming Language

- Context:  
First-order vocabulary of function symbols and variables
- Derived operation not shown here: sequential composition

# Update Constructors: 1. Neutral Updates

`skip` is an update

If  $f(s_1, \dots, s_n), t$  are terms, then

$f(s_1, \dots, s_n) := t$  is an update

## Motivation

Assignments like in all imperative languages:

$$a := 42$$
$$o.attr := c + 5 \quad \equiv \quad attr(o) := c + 5$$
$$array[i] := 5 \quad \equiv \quad ar(array, i) := 5$$

# Update Constructors: 3. Parallel Composition

If  $u_1, u_2$  are updates, then

$u_1 \mid u_2$  is an update

## Motivation

Updates are executed in parallel with last-win semantics:

$$a := 42 \mid b := 4$$
$$a := b \mid b := a$$
$$a := 3 \mid a := 7 \equiv a := 7$$
$$o_1.attr := 3 \mid o_2.attr := 7$$

If  $u$  is an update and  $\phi$  a formula, then

$\text{if } \phi \{u\}$  is an update

# Update Constructors: 5. Quantification

If  $u$  is an update and  $x$  a variable, then

$\text{for } x \{u\}$  is an update

## Motivation

Update is executed for all valuations of  $x$  in parallel, with well-ordered semantics:

$$\text{for } x \{array[x] := 0\}$$
$$\text{for } x \{a := x\} \equiv a := 0$$
$$\text{for } x \{array[x^2] := x\} \equiv \text{for } x \{\text{if } x \geq 0 \{array[x^2] := x\}\}$$

Well-ordering used in KeY for integers:

$$0 < 1 < 2 < 3 < \dots < -1 < -2 < -3 < \dots$$

# Features of Updates (as Programming Language)

- Deterministic, always terminating
- “Bounded” runtime, independent from inputs
  - No loops
  - Symbolic execution only needs finitely many steps
- Updates usable as intermediate language for symbolic execution

# The Syntax of Updates in Isabelle/HOL

Terms, formulas and updates are defined as mutually inductive datatypes:

```
datatype func    = Func nat
datatype ter     = AppVar nat
                  | AppFunc func terList
                  | AppUpdTer update ter
                  | ...
and             for    = Const bool
                  | AppUpdFor update for
                  | AndFor for for
                  | ...
and             update = Skip
                  | ElUpd func terList ter
                  | ParUpd update update
                  | GuardedUpd for update
                  | QuanUpd update
```

## Part 2: Denotational Update Semantics

# Denotational Update Semantics

Starting point: usual first-order semantics:

- Non-empty universe  $U$  (a set)
- Interpretation  $I$  of function symbols  
Notation:  $I\langle f, (a_1, \dots, a_n) \rangle$
- Valuation  $\beta$  of variables
- Evaluation  $\text{val}_{S, \beta}$  of terms and formulas

# Denotational Update Semantics

Starting point: usual first-order semantics:

- Non-empty universe  $U$  (a set)
- Interpretation  $I$  of function symbols  
Notation:  $I\langle f, (a_1, \dots, a_n) \rangle$
- Valuation  $\beta$  of variables
- Evaluation  $\text{val}_{S,\beta}$  of terms and formulas

*The unusual stuff is:*

- A well-ordering  $<$  on universe is needed (quantification)  
→ Structures are  $S = (U, <, I)$
- Value of an update is a *partial interpretation*, can leave functions in certain points undefined

$$\text{val}_{S,\beta}(u) : \text{Loc} \rightarrow U$$

- Overriding operator  $\oplus$ :

$$(I \oplus I')\langle f, \bar{a} \rangle = \begin{cases} I'\langle f, \bar{a} \rangle & \text{for } I'\langle f, \bar{a} \rangle \neq \perp \\ I\langle f, \bar{a} \rangle & \text{otherwise} \end{cases}$$

# Semantics of Update Constructors

$$\text{val}_{S,\beta}(\text{skip}) = \lambda x. \perp$$

$$\text{val}_{S,\beta}(f(\bar{s}) := t) = \lambda x. \begin{cases} \text{val}_{S,\beta}(t) & \text{for } x = \langle f, \text{val}_{S,\beta}(\bar{s}) \rangle \\ \perp & \text{otherwise} \end{cases}$$

$$\text{val}_{S,\beta}(u_1 \mid u_2) = \text{val}_{S,\beta}(u_1) \oplus \text{val}_{S,\beta}(u_2)$$

$$\text{val}_{S,\beta}(\text{if } \phi \{u\}) = \begin{cases} \text{val}_{S,\beta}(u) & \text{for } \text{val}_{S,\beta}(\phi) = \text{tt} \\ \lambda x. \perp & \text{otherwise} \end{cases}$$

For  $U = \{a < b < c < \dots\}$ :

$$\text{val}_{S,\beta}(\text{for } x \{u\}) = \dots \oplus \text{val}_{S,\beta_x^c}(u) \oplus \text{val}_{S,\beta_x^b}(u) \oplus \text{val}_{S,\beta_x^a}(u)$$

- We introduce a *type class* for well-ordered, non-empty universes:

```
axclass universe < wellorder
constdefs universeEl :: "'a::universe"
  "universeEl == arbitrary"
```

# In Isabelle/HOL: Universes, Interpretations & Co

- We introduce a *type class* for well-ordered, non-empty universes:

```
axclass universe < wellorder
constdefs universeEl :: "'a::universe"
  "universeEl == arbitrary"
```

- Locations, (partial) interpretations, variable valuations and structures become datatypes:

```
types 'a location      = "func * ('a list)"
      'a interp        = "('a location) => 'a"
      'a partInterp   = "('a location)
                        => ('a option)"
      'a varValuation = "nat => 'a"
```

```
record 'a semStructure =
  Interp          :: "'a interp"
  VarValuation   :: "'a varValuation"
```

# In Isabelle/HOL: Evaluation

Evaluation of terms, formulas and updates is defined by mutual primitive recursion:

```
consts terVal :: ...
      updVal :: " (('a::universe) semStructure)
                => update => ('a partInterp) "
primrec
  "terVal s (AppVar v)           = ..."
  ...
  "updVal s (Skip)               = (%loc. None)"
  "updVal s (ElUpd f args value) = ..."
  "updVal s (ParUpd left right)
    = (let leftV  = updVal s left;
         rightV = updVal s right
         in override leftV rightV)"
  "updVal s (QuanUpd u)         = ..."
  "updVal s (GuardedUpd fo u)   = ..."
```

## Part 3: Operational Update Semantics

# Application of Updates

If  $u$  is update and  $\alpha$  term/formula/update, then

$\{u\} \alpha$  is term/formula/update

## Motivation

Updates define the interpretation under which terms/formulas/updates are evaluated:

$$\begin{aligned} \{a := f(3)\} a &\equiv f(3) \\ \{\text{for } x \{array[x^2] := x\}\} array[(-5)^2] &\equiv 5 \end{aligned}$$

- Denotational meaning is:

$$\text{val}_{S,\beta}(\{u\} \alpha) = \text{val}_{S',\beta}(\alpha)$$

where  $S = (U, <, I)$  and  $S' = (U, <, I \oplus \text{val}_{S,\beta}(u))$

# Elimination of Updates by Rewriting

- For calculi: “Mechanical” application of updates is needed

$$\{f(3) := 2\} f(a) \rightarrow \text{if } a \doteq 3 \text{ then } 2 \text{ else } f(a)$$

- Performed by a rewriting system (50 rules in total).

For instance:

$$\{u\} (t_1 \doteq t_2) \rightarrow \{u\} t_1 \doteq \{u\} t_2$$

$$\{u\} f(\bar{t}) \rightarrow \text{NON-REC}(u, f, \{u\} \bar{t})$$

$$\text{NON-REC}(f(\bar{s}) := r, f, \bar{t}) \rightarrow \text{if } \bar{t} \doteq \bar{s} \text{ then } r \text{ else } f(\bar{t})$$

- In KeY: rewriting system is called “update simplifier”

# Operational Update Semantics in Isabelle/HOL

Rewriting system is an inductively defined relation on terms, formulas and updates

Steps: Direct rewriting, monotonic closure, transitive closure

consts

```
rewrTerDirect  :: "(ter * ter) set"
rewrForDirect  :: "(for * for) set"
rewrUpdDirect  :: "(update * update) set"
```

inductive rewrForDirect

intros

...

```
uMorphFor4: "(AppUpdFor u (Eq te1 te2),
              Eq (AppUpdTer u te1)
                (AppUpdTer u te2))
              : rewrForDirect"
```

...

Part 4:  
Isabelle/HOL Theorems about Updates

# Denotation and Operation in Relation

Properties that can be proven using Isabelle/HOL:

- The rewriting rules are sound Proven  
(rewriting does not change the value of terms or formulas)

```
lemma "(te1, te2) : rewrTer ==>  
      terVal s te1 = terVal s te2"  
...
```

- Rewriting eliminates updates Proven  
(if no rules are applicable, then a term or formula does not contain updates)
- Rewriting terminates Not proven so far

Further rewriting for turning updates into normalform possible:

$$\begin{array}{l} \text{for } x_{1,1} \{ \text{for } x_{1,2} \{ \text{for } \dots \{ \text{if } \phi_1 \{ s_1 := t_1 \} \} \} \} \\ | \dots \\ | \text{for } x_{k,1} \{ \text{for } x_{k,2} \{ \text{for } \dots \{ \text{if } \phi_k \{ s_k := t_k \} \} \} \} \end{array}$$

(updates in KeY always appear in this shape)

Provable in Isabelle/HOL:

- Simplification rules for updates are correct Proven
- Normal forms of updates can be established (reduced updates are normal, termination) Not proven

# What do we get by Formalising Updates?

- Correctness of update simplifier does *not* follow directly (simplifier uses slightly different, hard-coded rules)
- Still: found one major bug in simplifier
- Verified rules *could* be implemented more directly
- Formalisation helps to understand updates, makes future experiments easier
  
- $\approx$  3500 lines Isabelle/HOL code

## In Isabelle/HOL:

- Correctness of certain taclets (?)  
*Bettina Sasse, Bernhard Beckert*
- Invariant rule  
*Florian Widmann*
- Certain assignment rules, referring to Bali formalisation of JavaCard  
*Kerry Trentelman*

## In Maude/Rewriting logic:

- Class of program transformation rules, referring to Java formalisation in Maude  
*Wolfgang Ahrendt, Andreas Roth, Ralf Sasse*

*More should be done*  
*Supposedly, many inconsistencies are waiting in KeY ...*