Quantified Updates

Philipp Rümmer
philipp@cs.chalmers.se

8th June 2005

Simultaneous Updates (Old Story)

- Modal operator for changing values of finitely (boundedly) many locations:

```
{ f1(s1,...):=t1, ..., fn(sn,...):=tn } phi
```

- Semantics: Parallel execution, last-win clash resolution
- KeY: Update simplifier for merging/simplifying/applying updates

```
{ x:=3 } \<{ y=x; }\> true
        ~~>
{ x:=3 } { y:=x } \<{}\> true
        ~~>
{ x:=3, y:=3 } \<{}\> true
```

Simultaneous Updates (Old Story)

```
{ f1(s1,...):=t1, ..., fn(sn,...):=tn } phi
```

- In FOL: Updates can syntactically be applied (like substitutions)
- But: Application can also be delayed efficiently
  - Important for modal logics
  - Way of storing memory contents in dynamic logic
- Insufficient in many situations: Treatment of simple loops, modifies-clauses with wildcards, etc.

```
for(int i = 0; i != a.length; ++i)
  a[i] = 0;
```

. . . would require unboundedly many assignments

Quantified Updates, Guards

Extensions of simultaneous updates (that now exist in KeY):

- Quantification of assignments:

```
{... \for S x; f(s...):=t ...} phi
{... \for (S1 x; S2 y) f(s...):=t ...} phi
```

  For instance:

```
{... \for int i; a[i]:=0 ...} phi
{... \for C obj; obj.intAttr:=0 ...} phi
```

- Guards (not discussed further here):

```
{... \if (cond) f(s...):=t ...} phi
```

Simple Loops Can Be Handled by Par. Assignments

```
for(int i = 0; i != a.length; ++i)
  a[i] = 0;
```

$\rightsquigarrow$

```
\for int i;
\if (i >= 0 & i < a.length)
   a[i]:=0
```

- Practically used for array creation

Syntactic Application of Quantified Updates

```
U = \for S1 x1; f1(s1,...):=t1,
    ...,
    \for Sn xn; fn(sn,...):=tn

{U} f(s,...)
```

Create an if-cascade:

```
\ifEx Sn xn; (condn)
\then        (tn)
\else        (... \ifEx S1 x1; (cond1)
                  \then        (t1)
                  \else        (f({U}s,...)) ...)
```

$$\texttt{condi} = \begin{cases} \texttt{\{U\}s=si \& ...} & \text{for } \texttt{fi=f} \\ \texttt{false} & \text{otherwise} \end{cases}$$

Self-Clashes of Quantified Assignments

Example: What is `x` updated to?

```
{ \for int i; x:=i } phi
```

Self-Clashes of Quantified Assignments

Example: What is `x` updated to?

```
{ \for int i; x:=i } phi
```

Solution: Sort particular assignments by well-ordering integers:

$$\cdots \succ -3 \succ -2 \succ -1 \succ \cdots \succ 3 \succ 2 \succ 1 \succ 0$$

$$i \preceq j \iff (i \geq j \wedge j < 0) \vee (0 \leq i \wedge i \leq j)$$

Meaning of update:

```
{ ..., x:=-2, x:=-1, ..., x:=2, x:=1, x:=0 } phi
```

- Domains that can be quantified over are assumed to be well-ordered (particularly integers, Java reference types)
- Mainly motivated by advantages of last-win semantics: Nice treatment of sequentiality/aliasing

Meaning of ifEx-Expressions

```
\ifEx S x; (cond) \then (then) \else (else)
```

$$\equiv$$

```
\if   (\exists S x; cond)
\then ({\subst S x; (min S x; cond)}then)
\else (else)
```

(minimum operator is never introduced explicitly in KeY)

Update Construction

Update language is (semantically) closed under the following operations (`U1, U2, U` arbitrary updates!):

- Parallel composition: `U1, U2`
- Sequential composition: `U1; U2`
- Quantification: `\for S x; U`
- Conditional execution: `\if (phi) U`

- These operations are implemented in class `logic.UpdateFactory`
- Should be preferred method for creating updates (over `logic.TermFactory`)

For instance:

```
\for int i; ( a[i]:=b[i], b[i]:=a[i] )
```

Normalised to:

```
\for int i; a[i]:=b[i], \for int i; b[i]:=a[i]
```

Possible Future Improvements

- In general interactive manipulations of updates are necessary (not yet possible in KeY)
- More efficient syntactic application of updates might be possible