Exercises 3: Dynamic Logic

TECHNISCHE UNIVERSITÄT DARMSTADT

The solutions to the exercises will be discussed on Monday, 18th May.

Problem 1 Interpreting Dynamic Logic Formulas

What is the meaning of the following DL formulas? Are the formulas valid? Give a brief justification for your answers. We consider the type **int** to be the mathematical whole numbers, i.e. without overflow. You may assume the following definitions:

```
\programVariables {
  int i, old_i, j;
  boolean b;
}
```

a) $(\mathtt{i} > \mathtt{j}) \rightarrow \langle \mathtt{j = j - i;} \rangle (\mathtt{j} < 0)$

b) $(i > 0) \rightarrow \langle \mathtt{while\ (\ i\ !=\ 0\ )\ \{\ i\ =\ i\ -\ 2;\ \}} \rangle (\mathtt{i} \doteq 0)$

c) $[\mathtt{while\ (\ i\ !=\ 0\ )\ \{\ i\ =\ i\ -\ 2;\ \}}](\mathtt{i} \doteq 0)$

d) $(\mathtt{old\_i} \doteq \mathtt{i}) \rightarrow \langle \mathtt{j\ =\ 0;\ while\ (i\ >\ 0)\ \{\ j{+}{+};\ i\ =\ i\ -\ 1;\ \}} \rangle (\mathtt{i} \doteq 0 \rightarrow \mathtt{j} \doteq \mathtt{old\_i})$

e) $\exists\ \textbf{boolean}\ \mathtt{bool}; \big(\mathtt{b} \doteq \mathtt{bool} \rightarrow \langle \mathtt{if\ (b)\ \{\ i\ =\ 10;\ \}\ else\ \{\ j\ =\ -10;\ \}} \rangle (\mathtt{i} > \mathtt{j})\big)$

f) $\exists\ \textbf{boolean}\ \mathtt{bool}; \langle \mathtt{b\ =\ bool;\ if\ (b)\ \{\ i\ =\ 10;\ \}\ else\ \{\ j\ =\ -10;\ \}} \rangle (\mathtt{i} > \mathtt{j})$

Solution:

a) Valid. Program terminates and final value of `j` obviously less than 0.

b) Only true for states where `i` is even. Does not terminate for other states, hence, false.

c) Valid. As for positive even values the program terminates an dthe final value of `i` is 0, for all other values the program does not terminate and as we have a box modality the formula is in those cases trivially true.

d) Valid. Program terminates and property true in final state.

e) Valid choose $\mathtt{bool} \neq b$

f) Not a DL formula.

Problem 2 Semantics of Dynamic Logic

Justify formally (using the semantics definition) the following equivalence:

$$\langle \mathrm{p} \rangle \phi \text{ iff. } \neg[\mathrm{p}]\neg\phi$$

Solution:

$val_{K,s,\beta}(\langle \mathrm{p} \rangle \phi) = tt$ iff.$\rho(\mathrm{p})(s)$ defined and $val_{K,\rho(\mathrm{p})(s),\beta}(\phi) = tt \Leftrightarrow$ $\rho(\mathrm{p})(s)$ defined and not $val_{K,\rho(\mathrm{p})(s),\beta}(\phi) = ff \Leftrightarrow$ $\rho(\mathrm{p})(s)$ defined and not $val_{K,\rho(\mathrm{p})(s),\beta}(\neg\phi) = tt \Leftrightarrow$ not $\rho(\mathrm{p})(s)$ undefined and not $val_{K,\rho(\mathrm{p})(s),\beta}(\neg\phi) = tt \Leftrightarrow$ not ($\rho(\mathrm{p})(s)$ undefined or $val_{K,\rho(\mathrm{p})(s),\beta}(\neg\phi) = tt \Leftrightarrow$ not ($val_{K,s,\beta}([\mathrm{p}]\neg\phi) = tt \Leftrightarrow val_{K,s,\beta}(\neg[\mathrm{p}]\neg\phi) = tt$

Problem 3 Updates

Simplify the updates of the following formulas using the update simplification rules of the previous lecture:

- $\{x := x + y\}\{y := x + y\}\langle \mathtt{p} \rangle \phi$
- $\{x := x + y\}\{x := 3\}\langle \mathtt{p} \rangle \phi$

Assume that neither program `p` nor formula ϕ containing program variable `x`.

Solution: See files `problem3a/b.proof`

Which other simplification rule would be possible? Prove that the suggested simplification rule is sound.

Solution: E.g. $\{x := t\}\phi \rightsquigarrow \phi$ if x does not occur in ϕ

Show by structural induction over the DL formulas (and programs) that their value is independent of x if it does not occur.

Problem 4 Unwind-Loop rule

The unwindLoop rule as presented in the lecture is a simplified version of the actual one for Java as it does not consider continues, breaks, returns etc. Provide a version of the unwindLoop rule for loops with labeled break statements.

Solution:

$$\mathsf{unwindLoop}\ \frac{\Gamma \Rightarrow \langle \mathtt{outerLabel:\{}\mathbf{if}\mathtt{\ (b)\ \{\ p';\ }\mathbf{while}\mathtt{\ (b)\ p\ \};\}\ rest}\rangle \phi, \Delta}{\Gamma \Rightarrow \langle \mathbf{while}\mathtt{\ (b)\ \{\ p\ \};\ rest}\rangle \phi, \Delta}$$

`p'` is `p` where each unlabeled **`break`** which does not occur nested in another **`switch`** or loop has been replaced by **`break`** `newLabel`.