(Method) Contracts

(updated on July 22, 2008)

by Christoph Gladisch

The following open proof tree shows how to derive the method contract rule. Assume that p represents the method that is “replaced” by the contract and q is some program that follows p.

$$\dfrac{\dfrac{\dfrac{\mathrm{pre}_{\mathrm{pq}} \Rightarrow \mathrm{pre}_p}{\mathrm{pre}_{\mathrm{pq}} \Rightarrow \mathrm{pre}_p, \langle p;q\rangle \mathrm{post}_{\mathrm{pq}}} \qquad \dfrac{\dfrac{\mathrm{post}_p \Rightarrow \langle q\rangle \mathrm{post}_{\mathrm{pq}}}{\langle p\rangle \mathrm{post}_p \Rightarrow \langle p\rangle\langle q\rangle \mathrm{post}_{\mathrm{pq}}}}{\langle p\rangle \mathrm{post}_p, \mathrm{pre}_{\mathrm{pq}} \Rightarrow \langle p\rangle\langle q\rangle \mathrm{post}_{\mathrm{pq}}}}{\mathrm{pre}_p \rightarrow \langle p\rangle \mathrm{post}_p, \mathrm{pre}_{\mathrm{pq}} \Rightarrow \langle p\rangle\langle q\rangle \mathrm{post}_{\mathrm{pq}}}}{\underbrace{\mathrm{pre}_p \rightarrow \langle p\rangle \mathrm{post}_p}_{\text{Contract}} \Rightarrow \underbrace{\mathrm{pre}_{\mathrm{pq}} \rightarrow \langle p\rangle\langle q\rangle \mathrm{post}_{\mathrm{pq}}}_{\text{Proof obligation}}}$$

The open proof branches are the the branches of the method contract rule:

Basic method contract rule withoud modifies clause

Here we assume that $\mathrm{pre}_p \rightarrow \langle p\rangle \mathrm{post}_p$ is a correct contract.

$$\dfrac{\mathrm{pre}_{\mathrm{pq}} \Rightarrow \mathrm{pre}_p \qquad \mathrm{post}_p \Rightarrow \langle q\rangle \mathrm{post}_{\mathrm{pq}}}{\mathrm{pre}_p \rightarrow \langle p\rangle \mathrm{post}_p \Rightarrow \mathrm{pre}_{\mathrm{pq}} \rightarrow \langle p\rangle\langle q\rangle \mathrm{post}_{\mathrm{pq}}}$$

In KeY the contract $\mathrm{pre}_p \rightarrow \langle p\rangle \mathrm{post}_p$ is not explicit in the sequent but is extracted from the jml contracts of the current java file where $\mathrm{pre}_{\mathrm{pq}} \rightarrow \langle p\rangle\langle q\rangle \mathrm{post}_{\mathrm{pq}}$ stemm from.

Invariant rule construction

The Invariant rule is similar to the method contract rule. We make no statement about termination therefore the box-operator “[ ]” is used.

$$\dfrac{\mathrm{pre}_{\mathrm{pq}} \Rightarrow \mathrm{Inv} \qquad \overbrace{\mathrm{Inv}, \neg c}^{\mathrm{post}_{\mathrm{loop}}} \Rightarrow \langle q\rangle \mathrm{post}_{\mathrm{pq}}}{\underbrace{\mathrm{Inv} \wedge c}_{\mathrm{pre}_{\mathrm{loop}}} \rightarrow [b]\mathrm{Inv} \Rightarrow \mathrm{pre}_{\mathrm{pq}} \rightarrow [\mathrm{while}(c)\{b\}]\langle q\rangle \mathrm{post}_{\mathrm{pq}}}$$

Usually we don’t assume $\mathrm{Inv} \wedge c \rightarrow [b]\mathrm{Inv}$ is correct. Therefore, proving it yields the loop invariant rule:

$$\dfrac{\mathrm{pre}_{\mathrm{pq}} \Rightarrow \mathrm{Inv} \qquad \mathrm{Inv} \wedge c \Rightarrow [b]\mathrm{Inv} \qquad \mathrm{Inv}, \neg c \Rightarrow \langle q\rangle \mathrm{post}_{\mathrm{pq}}}{\Rightarrow \mathrm{pre}_{\mathrm{pq}} \rightarrow [\mathrm{while}(c)\{b\}]\langle q\rangle \mathrm{post}_{\mathrm{pq}}}$$

In KeY without modifies clause:

$$\dfrac{\Gamma \Rightarrow \{U\}\mathrm{inv} \qquad \Rightarrow \mathrm{inv} \rightarrow ([\mathtt{b=c}](b = \mathrm{true}) \rightarrow [\mathtt{body}]\mathrm{inv}) \qquad \Rightarrow \mathrm{inv} \rightarrow \neg c \rightarrow \mathrm{Post}}{\Gamma \Rightarrow \{U\}[\mathtt{while(c)\{body\}}]\mathrm{Post}}$$

Note that Γ (which has all the useful information that we migth need for a proof) is not present in the second and third branch.

In KeY with modifies clause:

Here $\{M\}$ represents a so-called “anonymous update” that is created from a modifier set M (in KeY these updates look like this: “$\{ * := * 1\}$”). The modifier set M is a set of all program variables (or non-rigid function symbols) that may be modified by the loop body. $\{M\}$ replaces all symbols that could be modified by new symbols (skolem functions). In this way modified symbols are not in “conflict” with symbols that are constrained by Γ. For instance assume that $i = 0$ before loop execution and the body computes $i++$, then without $\{M\}$ we get the “conflict” $i = 0 \wedge i = 1$. However if $\{M\}$ represents, e.g., the anonymous update $\{i := i_{\mathrm{sk}}\}$, where i_{sk} is a new function symbol, then we get $i = 0 \wedge \{M\}i = 1$ yields $i = 0 \wedge i_{\mathrm{sk}} = 1$.

$$\dfrac{\boldsymbol{\Gamma} \Rightarrow \{U\}\mathrm{inv} \qquad \boldsymbol{\Gamma} \Rightarrow \{U\}\{M\}(\mathrm{inv} \rightarrow ([\mathtt{b=c}](b = \mathrm{true}) \rightarrow [\mathtt{body}]\mathrm{inv})) \qquad \boldsymbol{\Gamma} \Rightarrow \{U\}\{M\}(\mathrm{inv} \rightarrow \neg c \rightarrow \mathrm{Post})}{\boldsymbol{\Gamma} \Rightarrow \{U\}[\mathtt{while(c)\{body\}}]\mathrm{Post}}$$