Type derivation for extended example

Steffen Zschaler

April 26, 2014

This document is an appendix to the paper "Towards Constraint-Based Model Types: A Generalised Formal Foundation for Model Genericity" showing the type derivation of the second example. This type derivation (see Fig. 1) shows that we can then also pass in instances of `Mandatory-Start`. We use the following abbreviations:

$$
\begin{aligned}
Cons_2 &= \neg isAbstract(StateMachine) \wedge \neg isAbstract(State) \\
Cons_2' &= Cons_2 \wedge \forall e : (StateMachine, \_, \_) : lower(e) = 0 \\
Cons_2'^* &= Cons_2 \wedge \forall e : (State, \_, \_) : lower(e) = 0 \\
Cons_2'' &= Cons_2'^* \vee \forall e : (\_, initialState, State) : lower(e) \leq 1 \\
T_2 &= (\{StateMachine, State\}, \\
&\quad \{(StateMachine, initialState, State)\}, \\
&\quad Cons_2) \\
T_2' &= (\{StateMachine, State\}, \\
&\quad \{(StateMachine, initialState, State)\}, \\
&\quad Cons_2') \\
T_2'^* &= (\{StateMachine, State\}, \\
&\quad \{(StateMachine, initialState, State)\}, \\
&\quad Cons_2'^*) \\
T_2'' &= (\{StateMachine, State\}, \\
&\quad \{(StateMachine, initialState, State)\}, \\
&\quad Cons_2'') \\
d_2 &= [s \mapsto (\top, T_2)] \\
d_2' &= d_2\,[sm \mapsto (StateMachine, T_2')] \\
d_2'' &= d_2'\,[sm \mapsto (StateMachine, T_2'')] \\
d_2''' &= d_2''\,[s \mapsto (\top, T_2'')] \\
d_2'''' &= [s \mapsto (\top, T_2'')]
\end{aligned}
$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{StateMachine \in \{StateMachine, State\}}\quad\dfrac{}{Cons_2 \Rightarrow \neg isAbstract(StateMachine)}}{d_2 \vdash \texttt{StateMachine} : \{c : Class | \neg isAbstract(c)\}}\ \textsc{New}}{d_2 \vdash \texttt{new StateMachine()} : (StateMachine, T_2')}\ \textsc{Var}}{\vdash \texttt{sm := new StateMachine()} \ : d_2 \to d_2'}\quad\dfrac{}{d_2' \vdash \texttt{sm} : (StateMachine, T_2')}\ \textsc{Ctx}_1\quad\dfrac{}{canNavigate(StateMachine, T_2', initialState, State)}}{\vdash \texttt{sm.initialState += new State()} \ : d_2' \to d_2''}\quad\dfrac{\dfrac{\dfrac{}{State \in \{StateMachine, State\}}\quad\dfrac{}{Cons_2 \Rightarrow \neg isAbstract(State)}}{d_2' \vdash \texttt{State} : \{c : Class | \neg isAbstract(c)\}}\ \textsc{New}}{d_2' \vdash \texttt{new State()} : (State, T_2'^*)}\ \textsc{Apnd}_2}{\vdash \texttt{sm.initialState += new State()} \ ; \texttt{s += sm} \ : d_2' \to d_2'''}\ \textsc{Seq}$$

$$\dfrac{\dfrac{\dfrac{}{d_2'' \vdash \texttt{sm} : (StateMachine, T_2'')}\ \textsc{Ctx}_1}{\vdash \texttt{s += sm} \ : d_2'' \to d_2'''}\ \textsc{Apnd}_1}{\vdash \texttt{sm.initialState += new State() ; s += sm} \ : d_2' \to d_2'''}\ \textsc{Seq}$$

$$\dfrac{\vdash \begin{array}{l}\texttt{sm := new StateMachine() ;}\\ \texttt{sm.initialState += new State() ;}\\ \texttt{s += sm} \ : d_2 \to d_2'''\end{array}}{\vdash \begin{array}{l}\texttt{mmo NewSM (s:}T_2''\texttt{) \{}\\ \quad \texttt{sm := new StateMachine() ;}\\ \quad \texttt{sm.initialState += new State() ;}\\ \quad \texttt{s += sm}\\ \texttt{\} :} T_2''\end{array}}\ \textsc{OpDef}$$

Figure 1: Type derivation for an extended version of the `NewSM` program