Under consideration for publication in the Journal of Functional Programming

Appendix for "Calculating Correct Compilers"

PATRICK BAHR
Department of Computer Science, University of Copenhagen, Denmark

GRAHAM HUTTON
School of Computer Science, University of Nottingham, UK

A Global State

Recall the specification of the compiler:

$$\begin{aligned} exec\ (comp'\ x\ c)\ (s,q) = \mathbf{case}\ &eval\ x\ q\ \mathbf{of} \\ &(Just\ n,q') \quad \rightarrow exec\ c\ (VAL\ n : s,q') \\ &(Nothing,q') \rightarrow fail\ (s,q') \end{aligned} \tag{12}$$

Below we give the full calculations for the *Add* and *Catch* cases.

$$\begin{array}{ll}
& exec\ (comp'\ (Add\ x\ y)\ c)\ (s,q) \\
= & \{ \text{ specification (12) } \} \\
& \mathbf{case}\ eval\ x\ q\ \mathbf{of} \\
& \quad (Just\ n,q') \quad \rightarrow \mathbf{case}\ eval\ y\ q'\ \mathbf{of} \\
& \qquad\qquad (Just\ m,q'') \quad \rightarrow exec\ c\ (VAL\ (n+m) : s,q'') \\
& \qquad\qquad (Nothing,q'') \rightarrow fail\ (s,q'') \\
& \quad (Nothing,q') \rightarrow fail\ (s,q') \\
= & \{ \text{ define: } exec\ (ADD\ c)\ (VAL\ m : VAL\ n : s,q'') = exec\ c\ (VAL\ (n+m) : s,q'') \ \} \\
& \mathbf{case}\ eval\ x\ q\ \mathbf{of} \\
& \quad (Just\ n,q') \quad \rightarrow \mathbf{case}\ eval\ y\ q'\ \mathbf{of} \\
& \qquad\qquad (Just\ m,q'') \quad \rightarrow exec\ (ADD\ c)\ (VAL\ m : VAL\ n : s,q'') \\
& \qquad\qquad (Nothing,q'') \rightarrow fail\ (s,q'') \\
& \quad (Nothing,q') \rightarrow fail\ (s,q') \\
= & \{ \text{ define: } fail\ (VAL\ n : s,q'') = fail\ (s,q'') \ \} \\
& \mathbf{case}\ eval\ x\ q\ \mathbf{of} \\
& \quad (Just\ n,q') \quad \rightarrow \mathbf{case}\ eval\ y\ q'\ \mathbf{of} \\
& \qquad\qquad (Just\ m,q'') \quad \rightarrow exec\ (ADD\ c)\ (VAL\ m : VAL\ n : s,q'') \\
& \qquad\qquad (Nothing,q'') \rightarrow fail\ (VAL\ n : s,q'') \\
& \quad (Nothing,q') \rightarrow fail\ (s,q') \\
= & \{ \text{ induction hypothesis for } y \ \} \\
& \mathbf{case}\ eval\ x\ q\ \mathbf{of} \\
& \quad (Just\ n,q') \quad \rightarrow exec\ (comp'\ y\ (ADD\ c))\ (VAL\ n : s,q') \\
& \quad (Nothing,q') \rightarrow fail\ (s,q') \\
= & \{ \text{ induction hypothesis for } x \ \} \\
& exec\ (comp'\ x\ (comp'\ y\ (ADD\ c)))\ (s,q)
\end{array}$$

$$
\begin{aligned}
&\mathit{exec}\ (\mathit{comp}'\ (\mathit{Catch}\ x\ h)\ c)\ (s,q)\\
=&\quad \{\ \text{specification (12)}\ \}\\
&\textbf{case}\ \mathit{eval}\ x\ q\ \textbf{of}\\
&\quad (\mathit{Just}\ n,q') \quad\ \to \mathit{exec}\ c\ (\mathit{VAL}\ n : s,q')\\
&\quad (\mathit{Nothing},q') \to \textbf{case}\ \mathit{eval}\ h\ q'\ \textbf{of}\\
&\qquad\qquad\qquad\qquad (\mathit{Just}\ m,q'') \quad\ \to \mathit{exec}\ c\ (\mathit{VAL}\ m : s,q'')\\
&\qquad\qquad\qquad\qquad (\mathit{Nothing},q'') \to \mathit{fail}\ (s,q'')\\
=&\quad \{\ \text{induction hypothesis for } h\ \}\\
&\textbf{case}\ \mathit{eval}\ x\ q\ \textbf{of}\\
&\quad (\mathit{Just}\ n,q') \quad\ \to \mathit{exec}\ c\ (\mathit{VAL}\ n : s,q')\\
&\quad (\mathit{Nothing},q') \to \mathit{exec}\ (\mathit{comp}'\ h\ c)\ (s,q')\\
=&\quad \{\ \text{define: } \mathit{fail}\ (\mathit{HAN}\ c' : s,q') = \mathit{exec}\ c'\ (s,q')\ \}\\
&\textbf{case}\ \mathit{eval}\ x\ q\ \textbf{of}\\
&\quad (\mathit{Just}\ n,q') \quad\ \to \mathit{exec}\ c\ (\mathit{VAL}\ n : s,q')\\
&\quad (\mathit{Nothing},q') \to \mathit{fail}\ (\mathit{HAN}\ (\mathit{comp}'\ h\ c) : s,q')\\
=&\quad \{\ \text{define: } \mathit{exec}\ (\mathit{UNMARK}\ c)\ (\mathit{VAL}\ n : \mathit{HAN}\ \_ : s,q') = \mathit{exec}\ c\ (\mathit{VAL}\ n : s,q')\ \}\\
&\textbf{case}\ \mathit{eval}\ x\ q\ \textbf{of}\\
&\quad (\mathit{Just}\ n,q') \quad\ \to \mathit{exec}\ (\mathit{UNMARK}\ c)\ (\mathit{VAL}\ n : \mathit{HAN}\ (\mathit{comp}'\ h\ c) : s,q')\\
&\quad (\mathit{Nothing},q') \to \mathit{fail}\ (\mathit{HAN}\ (\mathit{comp}'\ h\ c) : s,q')\\
=&\quad \{\ \text{induction hypothesis for } x\ \}\\
&\mathit{exec}\ (\mathit{comp}'\ x\ (\mathit{UNMARK}\ c))\ (\mathit{HAN}\ (\mathit{comp}'\ h\ c) : s,q)\\
=&\quad \{\ \text{define: } \mathit{exec}\ (\mathit{MARK}\ c''\ c')\ (s,q) = \mathit{exec}\ c'\ (\mathit{HAN}\ c'' : s,q)\ \}\\
&\mathit{exec}\ (\mathit{MARK}\ (\mathit{comp}'\ h\ c)\ (\mathit{comp}'\ x\ (\mathit{UNMARK}\ c)))\ (s,q)
\end{aligned}
$$

B Local State

We now consider the *local* approach to combining exceptions and state, in which the current state is discarded when an exception is thrown. This idea is reflected in the type for evaluation by moving the output state 'inside' the *Maybe* type:

$$\mathit{eval} :: \mathit{Expr} \to \mathit{State} \to \mathit{Maybe}\ (\mathit{Int},\mathit{State})$$

That is, if evaluation succeeds then *eval* returns an integer value and a new state, and if an exception is thrown it returns *Nothing*. The definition for *eval* is similar to the previous section except there is now no state to propagate when evaluation fails, and in the case for *Catch* the handler uses the state from when the catch was entered:

$$
\begin{aligned}
&\mathit{eval}\ (\mathit{Val}\ n)\ q && = \mathit{Just}\ (n,q)\\
&\mathit{eval}\ (\mathit{Add}\ x\ y)\ q && = \textbf{case}\ \mathit{eval}\ x\ q\ \textbf{of}\\
& && \qquad \mathit{Just}\ (n,q') \to \textbf{case}\ \mathit{eval}\ y\ q'\ \textbf{of}\\
& && \qquad\qquad \mathit{Just}\ (m,q'') \to \mathit{Just}\ (n+m,q'')\\
& && \qquad\qquad \mathit{Nothing} \quad\ \ \to \mathit{Nothing}\\
& && \qquad \mathit{Nothing} \quad\ \to \mathit{Nothing}\\
&\mathit{eval}\ \mathit{Throw}\ q && = \mathit{Nothing}\\
&\mathit{eval}\ (\mathit{Catch}\ x\ h)\ q && = \textbf{case}\ \mathit{eval}\ x\ q\ \textbf{of}\\
& && \qquad \mathit{Just}\ (n,q') \to \mathit{Just}\ (n,q')
\end{aligned}
$$

$$
\begin{array}{lll}
 & & \mathit{Nothing} \quad \to \mathit{eval}\ h\ q \\
\mathit{eval}\ \mathit{Get}\ q & = \mathit{Just}\ (q,q) & \\
\mathit{eval}\ (\mathit{Put}\ x\ y)\ q & = \textbf{case}\ \mathit{eval}\ x\ q\ \textbf{of} & \\
 & & \mathit{Just}\ (n,q') \to \mathit{eval}\ y\ n \\
 & & \mathit{Nothing} \quad \to \mathit{Nothing}
\end{array}
$$

For the purposes of the derivation of the compilation function $comp' :: Expr \to Code \to Code$ we use the same types as for the global state semantics:

$$
\begin{array}{l}
\mathit{exec} :: \mathit{Code} \to \mathit{Conf} \to \mathit{Conf} \\
\textbf{type}\ \mathit{Conf} = (\mathit{Stack}, \mathit{State}) \\
\textbf{type}\ \mathit{Stack} = [\mathit{Elem}] \\
\textbf{data}\ \mathit{Elem} = \mathit{VAL}\ \mathit{Int}
\end{array}
$$

The specification for the desired behaviour of $comp'$ is essentially the same as for global state, except that when evaluation fails we no longer have an output state to consider and hence the function *fail* only takes a stack as argument:

$$
\begin{array}{ll}
\mathit{exec}\ (\mathit{comp'}\ e\ c)\ (s,q) = \textbf{case}\ \mathit{eval}\ e\ q\ \textbf{of} & \\
\qquad \mathit{Just}\ (n,q') \to \mathit{exec}\ c\ (\mathit{VAL}\ n : s, q') & \\
\qquad \mathit{Nothing} \quad \to \mathit{fail}\ s &
\end{array}
\tag{14}
$$

However, to ensure type correctness of the specification, *fail* must still return a configuration, i.e. $fail :: Stack \to Conf$. An alternative would be to supply the input state q as an argument to *fail*, which is a valid choice that would lead to a different compiler. We start the derivation for $comp'$ with the cases for *Val n*, *Throw* and *Get*, which are easy:

$$
\begin{array}{ll}
 & \mathit{exec}\ (\mathit{comp'}\ (\mathit{Val}\ n)\ c)\ (s,q) \\
= & \{\ \text{specification (14)}\ \} \\
 & \mathit{exec}\ c\ (\mathit{VAL}\ n : s, q) \\
= & \{\ \text{define: } \mathit{exec}\ (\mathit{PUSH}\ n\ c)\ (s,q) = \mathit{exec}\ c\ (\mathit{VAL}\ n : s, q)\ \} \\
 & \mathit{exec}\ (\mathit{PUSH}\ n\ c)\ (s,q)
\end{array}
$$

$$
\begin{array}{ll}
 & \mathit{exec}\ (\mathit{comp'}\ \mathit{Throw}\ c)\ (s,q) \\
= & \{\ \text{specification (14)}\ \} \\
 & \mathit{fail}\ s \\
= & \{\ \text{define: } \mathit{exec}\ \mathit{FAIL}\ (s,q) = \mathit{fail}\ s\ \} \\
 & \mathit{exec}\ \mathit{FAIL}\ (s,q)
\end{array}
$$

$$
\begin{array}{ll}
 & \mathit{exec}\ (\mathit{comp'}\ \mathit{Get}\ c)\ (s,q) \\
= & \{\ \text{specification (14)}\ \} \\
 & \mathit{exec}\ c\ (\mathit{VAL}\ q : s, q) \\
= & \{\ \text{define: } \mathit{exec}\ (\mathit{LOAD}\ c)\ (s,q) = \mathit{exec}\ c\ (\mathit{VAL}\ q : s, q)\ \} \\
 & \mathit{exec}\ (\mathit{LOAD}\ c)\ (s,q)
\end{array}
$$

The case for *Add* follows the now familiar pattern:

$$
\begin{array}{ll}
& \mathit{exec}\ (\mathit{comp}'\ (\mathit{Add}\ x\ y)\ c)\ (s,q) \\
= & \quad \{\ \text{specification (14)}\ \} \\
& \textbf{case}\ \mathit{eval}\ x\ q\ \textbf{of} \\
& \quad \mathit{Just}\ (n,q') \rightarrow \textbf{case}\ \mathit{eval}\ y\ q'\ \textbf{of} \\
& \qquad\qquad\qquad\qquad \mathit{Just}\ (m,q'') \rightarrow \mathit{exec}\ c\ (\mathit{VAL}\ (n+m) : s,q'') \\
& \qquad\qquad\qquad\qquad \mathit{Nothing} \qquad \rightarrow \mathit{fail}\ s \\
& \quad \mathit{Nothing} \qquad \rightarrow \mathit{fail}\ s \\
= & \quad \{\ \text{define: } \mathit{exec}\ (\mathit{ADD}\ c)\ (\mathit{VAL}\ m : \mathit{VAL}\ n : s,q'') = \mathit{exec}\ c\ (\mathit{VAL}\ (n+m) : s,q'')\ \} \\
& \textbf{case}\ \mathit{eval}\ x\ q\ \textbf{of} \\
& \quad \mathit{Just}\ (n,q') \rightarrow \textbf{case}\ \mathit{eval}\ y\ q'\ \textbf{of} \\
& \qquad\qquad\qquad\qquad \mathit{Just}\ (m,q'') \rightarrow \mathit{exec}\ (\mathit{ADD}\ c)\ (\mathit{VAL}\ m : \mathit{VAL}\ n : s,q'') \\
& \qquad\qquad\qquad\qquad \mathit{Nothing} \qquad \rightarrow \mathit{fail}\ s \\
& \quad \mathit{Nothing} \qquad \rightarrow \mathit{fail}\ s \\
= & \quad \{\ \text{define: } \mathit{fail}\ (\mathit{VAL}\ n : s) = \mathit{fail}\ s\ \} \\
& \textbf{case}\ \mathit{eval}\ x\ q\ \textbf{of} \\
& \quad \mathit{Just}\ (n,q') \rightarrow \textbf{case}\ \mathit{eval}\ y\ q'\ \textbf{of} \\
& \qquad\qquad\qquad\qquad \mathit{Just}\ (m,q'') \rightarrow \mathit{exec}\ (\mathit{ADD}\ c)\ (\mathit{VAL}\ m : \mathit{VAL}\ n : s,q'') \\
& \qquad\qquad\qquad\qquad \mathit{Nothing} \qquad \rightarrow \mathit{fail}\ (\mathit{VAL}\ n : s) \\
& \quad \mathit{Nothing} \qquad \rightarrow \mathit{fail}\ s \\
= & \quad \{\ \text{induction hypothesis for } y\ \} \\
& \textbf{case}\ \mathit{eval}\ x\ q\ \textbf{of} \\
& \quad \mathit{Just}\ (n,q') \rightarrow \mathit{exec}\ (\mathit{comp}'\ y\ (\mathit{ADD}\ c))\ (\mathit{VAL}\ n : s,q') \\
& \quad \mathit{Nothing} \qquad \rightarrow \mathit{fail}\ s \\
= & \quad \{\ \text{induction hypothesis for } x\ \} \\
& \mathit{exec}\ (\mathit{comp}'\ x\ (\mathit{comp}'\ y\ (\mathit{ADD}\ c)))\ (s,q)
\end{array}
$$

The case for *Catch* is more interesting this time. In the calculation for the global state semantics it was straightforward to bring the configuration arguments into the right form to apply the induction hypotheses. With local state, however, when an exception handler is invoked we require access to the state that was in place when the enclosing *Catch* was entered, which information we communicate via the stack:

$$
\begin{array}{ll}
& \mathit{exec}\ (\mathit{comp}'\ (\mathit{Catch}\ x\ h)\ c)\ (s,q) \\
= & \quad \{\ \text{specification (14)}\ \} \\
& \textbf{case}\ \mathit{eval}\ x\ q\ \textbf{of} \\
& \quad \mathit{Just}\ (n,q') \rightarrow \mathit{exec}\ c\ (\mathit{VAL}\ n : s,q') \\
& \quad \mathit{Nothing} \qquad \rightarrow \textbf{case}\ \mathit{eval}\ h\ q\ \textbf{of} \\
& \qquad\qquad\qquad\qquad \mathit{Just}\ (m,q'') \rightarrow \mathit{exec}\ c\ (\mathit{VAL}\ m : s,q'') \\
& \qquad\qquad\qquad\qquad \mathit{Nothing} \qquad \rightarrow \mathit{fail}\ s \\
= & \quad \{\ \text{induction hypothesis for } h\ \} \\
& \textbf{case}\ \mathit{eval}\ x\ q\ \textbf{of} \\
& \quad \mathit{Just}\ (n,q') \rightarrow \mathit{exec}\ c\ (\mathit{VAL}\ n : s,q') \\
& \quad \mathit{Nothing} \qquad \rightarrow \mathit{exec}\ (\mathit{comp}'\ h\ c)\ (s,q) \\
= & \quad \{\ \text{define: } \mathit{fail}\ (\mathit{HAN}\ c'\ q : s) = \mathit{exec}\ c'\ (s,q)\ \} \\
& \textbf{case}\ \mathit{eval}\ x\ q\ \textbf{of} \\
& \quad \mathit{Just}\ (n,q') \rightarrow \mathit{exec}\ c\ (\mathit{VAL}\ n : s,q')
\end{array}
$$

$$
\begin{array}{ll}
& \quad \mathit{Nothing} \quad \rightarrow \mathit{fail}\ (\mathit{HAN}\ (\mathit{comp}'\ h\ c)\ q : s) \\
= & \{ \text{ define: } \mathit{exec}\ (\mathit{UNMARK}\ c)\ (\mathit{VAL}\ n : \mathit{HAN}\ \_\ \_ : s, q') = \mathit{exec}\ c\ (\mathit{VAL}\ n : s, q') \ \} \\
& \textbf{case}\ \mathit{eval}\ x\ q\ \textbf{of} \\
& \quad \mathit{Just}\ (n, q') \rightarrow \mathit{exec}\ (\mathit{UNMARK}\ c)\ (\mathit{VAL}\ n : \mathit{HAN}\ (\mathit{comp}'\ h\ c)\ q : s, q') \\
& \quad \mathit{Nothing} \quad \rightarrow \mathit{fail}\ (\mathit{HAN}\ (\mathit{comp}'\ h\ c)\ q : s) \\
= & \{ \text{ induction hypothesis for } x \ \} \\
& \mathit{exec}\ (\mathit{comp}'\ x\ (\mathit{UNMARK}\ c))\ (\mathit{HAN}\ (\mathit{comp}'\ h\ c)\ q : s, q) \\
= & \{ \text{ define: } \mathit{exec}\ (\mathit{MARK}\ c''\ c')\ (s, q) = \mathit{exec}\ c'\ (\mathit{HAN}\ c''\ q : s, q) \ \} \\
& \mathit{exec}\ (\mathit{MARK}\ (\mathit{comp}'\ h\ c)\ (\mathit{comp}'\ x\ (\mathit{UNMARK}\ c)))\ (s, q)
\end{array}
$$

Note that the new constructor *HAN* added to the *Elem* type within this calculation now has two arguments: one for the handler code (as in previous calculations), and one for the state to be used if the handler is invoked (for local state). We conclude the calculation with the case for *Put*, which proceeds in the same manner as for global state:

$$
\begin{array}{ll}
& \mathit{exec}\ (\mathit{comp}'\ (\mathit{Put}\ x\ y)\ c)\ (s, q) \\
= & \{ \text{ specification (14) } \} \\
& \textbf{case}\ \mathit{eval}\ x\ q\ \textbf{of} \\
& \quad \mathit{Just}\ (n, q') \rightarrow \textbf{case}\ \mathit{eval}\ y\ n\ \textbf{of} \\
& \qquad\qquad\qquad\qquad \mathit{Just}\ (m, q'') \rightarrow \mathit{exec}\ c\ (\mathit{VAL}\ m : s, q'') \\
& \qquad\qquad\qquad\qquad \mathit{Nothing} \quad \rightarrow \mathit{fail}\ s \\
& \quad \mathit{Nothing} \quad \rightarrow \mathit{fail}\ s \\
= & \{ \text{ induction hypothesis for } y \ \} \\
& \textbf{case}\ \mathit{eval}\ x\ q\ \textbf{of} \\
& \quad \mathit{Just}\ (n, q') \rightarrow \mathit{exec}\ (\mathit{comp}'\ y\ c)\ (s, n) \\
& \quad \mathit{Nothing} \quad \rightarrow \mathit{fail}\ s \\
= & \{ \text{ define: } \mathit{exec}\ (\mathit{SAVE}\ c')\ (\mathit{VAL}\ n : s, q') = \mathit{exec}\ c'\ (s, n) \ \} \\
& \textbf{case}\ \mathit{eval}\ x\ q\ \textbf{of} \\
& \quad \mathit{Just}\ (n, q') \rightarrow \mathit{exec}\ (\mathit{SAVE}\ (\mathit{comp}'\ y\ c))\ (\mathit{VAL}\ n : s, q') \\
& \quad \mathit{Nothing} \quad \rightarrow \mathit{fail}\ s \\
= & \{ \text{ induction hypothesis for } x \ \} \\
& \mathit{exec}\ (\mathit{comp}'\ x\ (\mathit{SAVE}\ (\mathit{comp}'\ y\ c)))\ (s, q)
\end{array}
$$

In summary, collecting together everything that we have learned in the process of the above calculations, we obtained the following definitions.

Target language:

$$
\begin{array}{l}
\textbf{data}\ \mathit{Code} = \mathit{HALT} \mid \mathit{PUSH}\ \mathit{Int}\ \mathit{Code} \mid \mathit{ADD}\ \mathit{Code} \mid \\
\qquad\qquad\quad \mathit{FAIL} \mid \mathit{MARK}\ \mathit{Code}\ \mathit{Code} \mid \mathit{UNMARK}\ \mathit{Code} \mid \\
\qquad\qquad\quad \mathit{LOAD}\ \mathit{Code} \mid \mathit{SAVE}\ \mathit{Code}
\end{array}
$$

Compiler:

$$
\begin{array}{ll}
\mathit{comp} & :: \mathit{Expr} \rightarrow \mathit{Code} \\
\mathit{comp}\ x & = \mathit{comp}'\ x\ \mathit{HALT} \\
\mathit{comp}' & :: \mathit{Expr} \rightarrow \mathit{Code} \rightarrow \mathit{Code} \\
\mathit{comp}'\ (\mathit{Val}\ n)\ c & = \mathit{PUSH}\ n\ c
\end{array}
$$

$$
\begin{aligned}
&comp'\ (Add\ x\ y)\ c &&= comp'\ x\ (comp'\ y\ (ADD\ c))\\
&comp'\ Throw\ c &&= FAIL\\
&comp'\ (Catch\ x\ h)\ c &&= MARK\ (comp'\ h\ c)\ (comp'\ x\ (UNMARK\ c))\\
&comp'\ Get\ c &&= LOAD\ c\\
&comp'\ (Put\ x\ y)\ c &&= comp'\ x\ (SAVE\ (comp'\ y\ c))
\end{aligned}
$$

Virtual machine:

$$
\begin{aligned}
&\textbf{data}\ Elem &&= VAL\ Int \mid HAN\ Code\ State\\
&exec &&:: Code \to Conf \to Conf\\
&exec\ HALT\ (s,q) &&= (s,q)\\
&exec\ (PUSH\ n\ c)\ (s,q) &&= exec\ c\ (VAL\ n : s,q)\\
&exec\ (ADD\ c)\ (VAL\ m : VAL\ n : s,q) &&= exec\ c\ (VAL\ (n+m) : s,q)\\
&exec\ FAIL\ (s,q) &&= fail\ s\\
&exec\ (MARK\ h\ c)\ (s,q) &&= exec\ c\ (HAN\ h\ q : s,q)\\
&exec\ (UNMARK\ c)\ (VAL\ n : HAN\ \_\ \_ : s,q) &&= exec\ c\ (VAL\ n : s,q)\\
&exec\ (LOAD\ c)\ (s,q) &&= exec\ c\ (VAL\ q : s,q)\\
&exec\ (SAVE\ c)\ (VAL\ n : s,q) &&= exec\ c\ (s,n)\\
&fail &&:: Stack \to Conf\\
&fail\ [\,] &&= ([\,],0)\\
&fail\ (VAL\ n : s) &&= fail\ s\\
&fail\ (HAN\ h\ q : s) &&= exec\ h\ (s,q)
\end{aligned}
$$

Note that, as previously, we added an equation to *fail* for the case when the stack is empty in order to make the definition complete. Because *fail* does not take a state as an argument, we can only give a fixed output state as the result, for which purposes we simply return the value 0. As before, the choice for this additional equation has no impact on the correctness of the above calculations because they do not depend on this choice.