YASL Typechecking

Brian Howard

Fall 2015, Version 1.1

This is a full version of the type checking rules described in class on October 29.

Type check an entire program by creating an empty symbol table that maps identifiers to `Type`, then type check the block inside a new scope whose name is the program name:

$$\mathcal{T}[\![\texttt{program ID ; Block .}]\!] = \begin{cases} \texttt{t = new SymbolTable[Type]} \\ \texttt{t.enter(ID)} \\ \mathcal{T}[\![\texttt{Block}]\!](t) \\ \texttt{t.exit()} \end{cases}$$

Type check a block by type checking in turn each of the declarations, followed by each of the statements of the block. All procedure declarations have their headers added to the symbol table before any of them are type checked, to allow for mutual recursion:

$$\mathcal{T}[\![\texttt{ConstDecls VarDecls ProcDecls begin Stmts end}]\!](t) = \begin{cases} \texttt{foreach cd} \in \texttt{ConstDecls: } \mathcal{T}[\![\texttt{cd}]\!](t) \\ \texttt{foreach vd} \in \texttt{VarDecls: } \mathcal{T}[\![\texttt{vd}]\!](t) \\ \texttt{foreach pd} \in \texttt{ProcDecls:} \\ \quad \texttt{t.bind(pd.ID, new ProcVal(pd.Params)} \\ \texttt{foreach pd} \in \texttt{ProcDecls: } \mathcal{T}[\![\texttt{pd}]\!](t) \\ \texttt{foreach s} \in \texttt{Stmts: } \mathcal{T}[\![\texttt{s}]\!](t) \end{cases}$$

Type check a constant declaration by binding the identifier to the type `IntVal`. It is an error if the identifier is already bound in the current scope:

$$\mathcal{T}[\![\texttt{const ID = NUM ;}]\!](t) = \texttt{t.bind(ID, IntVal)}$$

Type check a variable declaration by binding the identifier to the type `IntVar` or `BoolVar`. It is an error if the identifier is already bound in the current scope:

$$\mathcal{T}[\![\texttt{var ID : int ;}]\!](t) = \texttt{t.bind(ID, IntVar)}$$
$$\mathcal{T}[\![\texttt{var ID : bool ;}]\!](t) = \texttt{t.bind(ID, BoolVar)}$$

Type check a procedure declaration by binding the identifier to a new `ProcVal` holding the parameters (this already happened above), then type check the block in a new local scope where each of the parameters has been bound to the appropriate type (`IntVar` or `BoolVar`). It is an error if an identifier is already bound in the current scope:

$$\mathcal{T}[\![\texttt{proc ID Params ; Block ;}]\!](t) = \begin{cases} \texttt{t.enter(ID)} \\ \texttt{foreach param} \in \texttt{Params:} \\ \quad \texttt{if param.type is int:} \\ \quad\quad \texttt{t.bind(param.id, IntVar)} \\ \quad \texttt{else:} \\ \quad\quad \texttt{t.bind(param.id, BoolVar)} \\ \mathcal{T}[\![\texttt{Block}]\!](t) \\ \texttt{t.exit()} \end{cases}$$

Type check an assignment statement by looking up the identifier on the left, type checking the expression on the right, and then checking that the value on the left is a variable of the same type as the value on the right:

$$\mathcal{T}[\![\texttt{ID = Expr ;}]\!](t) = \begin{cases} \texttt{lhs = t.lookup(ID)} \\ \texttt{rhs =} \ \mathcal{T}[\![\texttt{Expr}]\!](t) \\ \texttt{check lhs.isVar} \\ \texttt{check lhs.type is rhs.type} \end{cases}$$

Type check a procedure call by looking up the identifier to get a `ProcVal`, then type checking the list of arguments to get a list of types. It is an error if the number of parameters does not match the number of arguments, if the type of a parameter does not match the type of its argument, or if a `var` parameter is not given a variable argument:

$$\mathcal{T}[\![\texttt{ID Args ;}]\!](t) = \begin{cases} \texttt{ProcVal(params) = t.lookup(ID)} \\ \texttt{args = foreach e} \in \texttt{Args: yield} \ \mathcal{T}[\![\texttt{e}]\!](t) \\ \texttt{match(params, args) // see below} \end{cases}$$

The `match` function is defined by the following cases (note that `Nil` is an empty list, while

`head :: tail` is a list with the given `head` element and rest of the list `tail`):

$$\begin{cases}
\texttt{match(Nil, Nil) = ok} \\
\texttt{match((ID : int) :: params, arg :: args) =} \\
\quad \texttt{check arg.type is int; match(params, args)} \\
\texttt{match((ID : bool) :: params, arg :: args) =} \\
\quad \texttt{check arg.type is bool; match(params, args)} \\
\texttt{match((var ID : int) :: params, arg :: args) =} \\
\quad \texttt{check arg is IntVar; match(params, args)} \\
\texttt{match((var ID : bool) :: params, arg :: args) =} \\
\quad \texttt{check arg is BoolVar; match(params, args)}
\end{cases}$$

Type check a compound statement by type checking each of the contained statements in turn:

$$\mathcal{T}[\![\texttt{begin Stmts end ;}]\!](t) = \texttt{foreach s} \in \texttt{Stmts:}\ \mathcal{T}[\![\texttt{s}]\!](t)$$

Type check an if-then statement by type checking the test, which should give a boolean value, then type checking the statement:

$$\mathcal{T}[\![\texttt{if Expr then Stmt}]\!](t) = \begin{cases}
\texttt{test =}\ \mathcal{T}[\![\texttt{Expr}]\!](t) \\
\texttt{check test.type is bool} \\
\mathcal{T}[\![\texttt{Stmt}]\!](t)
\end{cases}$$

Type check an if-then-else statement by type checking the test, which should give a boolean value, then type checking each of the statements:

$$\mathcal{T}[\![\texttt{if Expr then Stmt1 else Stmt2}]\!](t) = \begin{cases}
\texttt{test =}\ \mathcal{T}[\![\texttt{Expr}]\!](t) \\
\texttt{check test.type is bool} \\
\mathcal{T}[\![\texttt{Stmt1}]\!](t) \\
\mathcal{T}[\![\texttt{Stmt2}]\!](t)
\end{cases}$$

Type check a while statement by type checking the test, which should give a boolean value, then type checking the statement:

$$\mathcal{T}[\![\texttt{while Expr do Stmt}]\!](t) = \begin{cases}
\texttt{test =}\ \mathcal{T}[\![\texttt{Expr}]\!](t) \\
\texttt{check test.type is bool} \\
\mathcal{T}[\![\texttt{Stmt}]\!](t)
\end{cases}$$

Type check a prompt statement by checking that the identifier, if present, is an integer variable:

$$\mathcal{T}[\![\texttt{prompt STRING ;}]\!](t) = \texttt{ok}$$

$$\mathcal{T}[\![\texttt{prompt STRING , ID ;}]\!](t) = \begin{cases} \texttt{lhs = t.lookup(ID)} \\ \texttt{check lhs is IntVar} \end{cases}$$

Type check a print statement by type checking each of its items–strings are always OK, and expressions must be of integer type:

$$\mathcal{T}[\![\texttt{print Items ;}]\!](t) = \begin{cases} \texttt{foreach it} \in \texttt{Items:} \\ \quad \texttt{if it is an Expr:} \\ \quad\quad \texttt{x =} \ \mathcal{T}[\![\texttt{it}]\!](t) \\ \quad\quad \texttt{check x.type is int} \\ \quad \texttt{else: // it is a STRING} \\ \quad\quad \texttt{ok} \end{cases}$$

Type check a binary operation expression by type checking the left and right operands, then checking that their types match the indicated operation and returning the resulting type:

$$\mathcal{T}[\![\texttt{Expr1 op Expr2}]\!](t) = \begin{cases} \texttt{lhs =} \ \mathcal{T}[\![\texttt{Expr1}]\!](t) \\ \texttt{rhs =} \ \mathcal{T}[\![\texttt{Expr2}]\!](t) \\ \texttt{switch op:} \\ \quad \texttt{case AND, OR:} \\ \quad\quad \texttt{check lhs.type is bool and rhs.type is bool} \\ \quad\quad \texttt{return BoolVal} \\ \quad \texttt{case EQ, NE, LE, LT, GE, GT:} \\ \quad\quad \texttt{check lhs.type is int and rhs.type is int} \\ \quad\quad \texttt{return BoolVal} \\ \quad \texttt{case PLUS, MINUS, STAR, DIV, MOD:} \\ \quad\quad \texttt{check lhs.type is int and rhs.type is int} \\ \quad\quad \texttt{return IntVal} \end{cases}$$

Type check a unary operation expression by type checking the operands, then checking

that its type matches the indicated operation and returning the resulting type:

$$\mathcal{T}[\![\texttt{op Expr}]\!](t) = \begin{cases} \texttt{x = } \mathcal{T}[\![\texttt{Expr}]\!](t) \\ \texttt{switch op:} \\ \quad \texttt{case MINUS:} \\ \quad\quad \texttt{check x.type is int; return IntVal} \\ \quad \texttt{case NOT:} \\ \quad\quad \texttt{check x.type is bool; return BoolVal} \end{cases}$$

Type check an integer or boolean literal expression by returning the corresponding type:

$$\mathcal{T}[\![\texttt{NUM}]\!](t) = \texttt{return new IntVal}$$
$$\mathcal{T}[\![\texttt{true}]\!](t) = \texttt{return new BoolVal}$$
$$\mathcal{T}[\![\texttt{false}]\!](t) = \texttt{return new BoolVal}$$

Type check a variable expression by returning the type bound to the identifier in the symbol table. It is an error if the variable is undefined:

$$\mathcal{T}[\![\texttt{ID}]\!](t) = \texttt{return t.lookup(ID)}$$

There are five kinds of types that may be stored in the symbol table:

- `IntVal`: the operation `IntVal.type` returns `int`, and `IntVal.isVar` is false;
- `BoolVal`: the operation `BoolVal.type` returns `bool`, and `BoolVal.isVar` is false;
- `IntVar`: the operation `IntVar.type` returns `int`, and `IntVar.isVar` is true;
- `BoolVar`: the operation `BoolVar.type` returns `bool`, and `BoolVar.isVar` is true;
- `ProcVal(params)`: described above when type checking procedure declarations and calls; the operations `.type` and `.isVar` on a `ProcVal` are undefined.