CS 367 Compilers

Instructor R. Muller

mini-C: Syntax and Type System

Mini-C is a clipped version of the C programming language. This document is a draft specification of the syntax and type system for the language. Like C programs, mini-C programs are collections of procedure definitions. By convention, a mini-C program has one procedure named main. An example, mini-C program with 3 procedure definitions is:

```
int fact(int n) {
  if (n == 0)
    return 1;
  else
    return n * fact(n - 1);
}

int factIter(int n) {
  int answer;

  answer = 1;
  while (n > 0) {
    answer = answer * n;
    n = n - 1;
    }
  return answer;
}

void main() {
  print fact(3) + factIter(3);
  return ();
}
```

1 Syntax

The syntax of mini-C programs is specified by the grammar in Figure 1.

2 Type System

The type system for mini-C is specified in Figures 2 and 3. The specification makes use of *type environments* Γ of the form:

$$\Gamma ::= \epsilon \mid \Gamma[\mathsf{id} : \mathsf{type}]$$

In the interest of modularity and to simplify the implementation of the type checker, it will be convenient to use an initial type environment Γ_0, sometimes called a *static basis*, which contains the type signatures of all of the primitive operators in the language.

$$\Gamma_0 = \epsilon[+ : \mathsf{int} * \mathsf{int} \to \mathsf{int}] \ldots [>: \mathsf{int} * \mathsf{int} \to \mathsf{bool}]$$

type	::=	int $\mid$ bool $\mid$ void $\mid$ type $\rightarrow$ type $\mid$ type $*$ type
digit	::=	$0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
integer	::=	digit $\mid$ digit integer
boolean	::=	True $\mid$ False
value	::=	integer $\mid$ boolean $\mid$ ()
op	::=	$+ \mid - \mid * \mid / \mid \% \mid ** \mid < \mid <= \mid == \mid >= \mid >$
id	=	identifier
program	::=	procedure $\mid$ program procedure
procedure	::=	type id (parameters) block
parameters	::=	ϵ $\mid$ binding $\mid$ binding , parameters
binding	::=	type id
block	::=	{ declarations statements }
	$\mid$	{ statements }
declarations	::=	binding ; $\mid$ declarations binding ;
statement	::=	id = expression ;
	$\mid$	id (expressions) ;
	$\mid$	return expression ;
	$\mid$	print expression ;
	$\mid$	while (expression) statement
	$\mid$	if (expression) statement else statement
	$\mid$	block
expression	::=	value $\mid$ id $\mid$ (expression) $\mid$ id (expressions)
	$\mid$	expression op expression $\mid$ not expression $\mid$ $-$ expression
	$\mid$	expression and expression $\mid$ expression or expression
expressions	::=	ϵ $\mid$ expression $\mid$ expression , expressions

Figure 1: The syntax of programs in mini-C.

The formal rules for type-checking procedures use judgements of the form:

$$\Gamma \vdash \mathsf{procedure}\ \ldots\ \mathsf{procedure} : \mathsf{ok}$$

and

$$\vdash \mathsf{procedure} \Rightarrow \mathsf{id} : \mathsf{type}.$$

The rules for type-checking statements use judgements of the form:

$$\Gamma, \mathsf{type} \vdash \mathsf{statement} : \mathsf{ok}.$$

The formal rules for deriving all three of these judgements are specified in Figure 2.

Rule (Prog) codifies when a collection of procedures (i.e., a program) is well-typed. The first antecedent of the rule

$$\forall i \in \{1..n\} \vdash \mathsf{procedure}_i \Rightarrow \mathsf{id}_i : \mathsf{type}_i$$

uses the (Sig) rule to gather the type signatures of each procedure in the program. This type information is then used to extend the type environment from the consequent so that each procedure can be type-checked by the other antecedent

$$\forall i \in \{1..n\}\ \Gamma[\mathsf{id}_1 : \mathsf{type}_1] \ldots [\mathsf{id}_n : \mathsf{type}_n] \vdash \mathsf{procedure}_i : \mathsf{ok}.$$

Judgements of the form $\Gamma, \mathsf{type} \vdash \mathsf{statement} : \mathsf{ok}$ codify when a mini-C statement is well-typed. There are 7 rules, one for each kind of statement. Note that the symbol type in the statement judgement $\Gamma, \mathsf{type} \vdash \mathsf{statement} : \mathsf{ok}$ plays an essential role in ensuring that procedures return values of the expected type.

The symbol type in the procedure definition $\mathsf{type}\ \mathsf{id}\ (\mathsf{type}_1\ \mathsf{id}_1, \ldots, \mathsf{type}_n\ \mathsf{id}_n)\ \mathsf{block}$ is the expected return type of the function. All statements within the procedure body block of the form return expression ; must be checked to ensure that the type of expression agrees with the expected return type of the procedure in which it is contained.

The typing rules for expressions use judgements of the form

$$\Gamma \vdash \mathsf{expression} : \mathsf{type}$$

The rules for deriving judgements of this form are specified in Figure 3.

Procedures

$$\frac{\forall i \in \{1..n\} \;\; \vdash \mathsf{procedure}_i \Rightarrow \mathsf{id}_i : \mathsf{type}_i; \quad \Gamma[\mathsf{id}_1 : \mathsf{type}_1]\ldots[\mathsf{id}_n : \mathsf{type}_n] \vdash \mathsf{procedure}_i : \mathsf{ok}}{\Gamma \vdash \mathsf{procedure}_1 \;\ldots\; \mathsf{procedure}_n : \mathsf{ok}} \;(\mathsf{Prog})$$

$$\frac{\Gamma[\mathsf{id}_1 : \mathsf{type}_1]\ldots[\mathsf{id}_n : \mathsf{type}_n], \mathsf{type} \vdash \mathsf{block} : \mathsf{ok}}{\Gamma \vdash \mathsf{type}\ \mathsf{id}\ (\mathsf{type}_1\ \mathsf{id}_1, \ldots, \mathsf{type}_n\ \mathsf{id}_n)\ \mathsf{block} : \mathsf{ok}} \;(\mathsf{Proc})$$

$$\frac{}{\vdash \mathsf{type}\ \mathsf{id}\ (\mathsf{type}_1\ \mathsf{id}_1, \ldots, \mathsf{type}_n\ \mathsf{id}_n)\ \mathsf{block} \Rightarrow \mathsf{id} : \mathsf{type}_1 * \ldots * \mathsf{type}_n \to \mathsf{type}} \;(\mathsf{Sig})$$

Statements

$$\frac{\Gamma[\mathsf{id}_1 : \mathsf{type}_1]\ldots[\mathsf{id}_n : \mathsf{type}_n], \mathsf{type} \vdash \mathsf{statements} : \mathsf{ok}}{\Gamma, \mathsf{type} \vdash \{\ \mathsf{type}_1\ \mathsf{id}_1;\ \ldots;\ \mathsf{type}_n\ \mathsf{id}_n;\ \mathsf{statements}\} : \mathsf{ok}} \;(\mathsf{Block})$$

$$\frac{\forall i \in \{1..n\} \;\; \Gamma, \mathsf{type} \vdash \mathsf{statement}_i : \mathsf{ok}}{\Gamma, \mathsf{type} \vdash \mathsf{statement}_1 \;\ldots\; \mathsf{statement}_n : \mathsf{ok}} \;(\mathsf{Statements})$$

$$\frac{\Gamma \vdash \mathsf{id} : \mathsf{type}';\ \Gamma \vdash \mathsf{expression} : \mathsf{type}'}{\Gamma, \mathsf{type} \vdash \mathsf{id} = \mathsf{expression}\ ; : \mathsf{ok}} \;(\mathsf{Assign})$$

$$\frac{\Gamma(\mathsf{id}) = \mathsf{type}_1 * \ldots * \mathsf{type}_n \to \mathsf{void};\ \forall i \in \{1 \ldots n\}.\Gamma \vdash \mathsf{expression}_i : \mathsf{type}_i}{\Gamma, \mathsf{type} \vdash \mathsf{id}(\mathsf{expression}_1, \ldots,\ \mathsf{expression}_n) : \mathsf{ok}} \;(\mathsf{Call})$$

$$\frac{\Gamma \vdash \mathsf{expression} : \mathsf{type}}{\Gamma, \mathsf{type} \vdash \mathsf{return}\ \mathsf{expression}\ ; : \mathsf{ok}} \;(\mathsf{Return}) \qquad \frac{\Gamma \vdash \mathsf{expression} : \mathsf{int}}{\Gamma, \mathsf{type} \vdash \mathsf{print}\ \mathsf{expression}\ ; : \mathsf{ok}} \;(\mathsf{Print})$$

$$\frac{\Gamma \vdash \mathsf{expression} : \mathsf{bool};\ \Gamma, \mathsf{type} \vdash \mathsf{statement} : \mathsf{ok}}{\Gamma, \mathsf{type} \vdash \mathsf{while}\ (\ \mathsf{expression}\ )\ \mathsf{statement} : \mathsf{ok}} \;(\mathsf{While})$$

$$\frac{\begin{array}{c}\Gamma \vdash \mathsf{expression} : \mathsf{bool} \\ \Gamma, \mathsf{type} \vdash \mathsf{statement}_1 : \mathsf{ok};\ \Gamma, \mathsf{type} \vdash \mathsf{statement}_2 : \mathsf{ok}\end{array}}{\Gamma, \mathsf{type} \vdash \mathsf{if}\ (\ \mathsf{expression}\ )\ \mathsf{statement}_1\ \mathsf{else}\ \mathsf{statement}_2 : \mathsf{ok}} \;(\mathsf{If})$$

Figure 2: A type system for procedures and statements in mini-C.

Expressions

$$\frac{}{\Gamma \vdash \mathsf{integer} : \mathsf{int}}\ (\mathsf{int}) \qquad \frac{}{\Gamma \vdash \mathsf{boolean} : \mathsf{bool}}\ (\mathsf{bool})$$

$$\frac{}{\Gamma \vdash () : \mathsf{void}}\ (\mathsf{void}) \qquad \frac{\Gamma(\mathsf{id}) = \mathsf{type}}{\Gamma \vdash \mathsf{id} : \mathsf{type}}\ (\mathsf{Var})$$

$$\frac{\Gamma \vdash \mathsf{expression}_1 : \mathsf{type}_1;\ \Gamma \vdash \mathsf{expression}_2 : \mathsf{type}_2;\ \Gamma(\mathsf{op}) = \mathsf{type}_1 * \mathsf{type}_2 \rightarrow \mathsf{type}}{\Gamma \vdash \mathsf{expression}_1\ \mathsf{op}\ \mathsf{expression}_2 : \mathsf{type}}\ (\mathsf{op})$$

$$\frac{\Gamma(\mathsf{id}) = \mathsf{type}_1 * \ldots * \mathsf{type}_n \rightarrow \mathsf{type};\ \forall i \in \{1 \ldots n\}.\Gamma \vdash \mathsf{expression}_i : \mathsf{type}_i}{\Gamma \vdash \mathsf{id}(\mathsf{expression}_1, \ldots,\ \mathsf{expression}_n) : \mathsf{type}}\ (\mathsf{App})$$

$$\frac{\Gamma \vdash \mathsf{expression}_1 : \mathsf{bool};\ \Gamma \vdash \mathsf{expression}_2 : \mathsf{bool}}{\Gamma \vdash \mathsf{expression}_1\ \mathsf{and}\ \mathsf{expression}_2 : \mathsf{bool}}\ (\mathsf{And})$$

$$\frac{\Gamma \vdash \mathsf{expression}_1 : \mathsf{bool};\ \Gamma \vdash \mathsf{expression}_2 : \mathsf{bool}}{\Gamma \vdash \mathsf{expression}_1\ \mathsf{or}\ \mathsf{expression}_2 : \mathsf{bool}}\ (\mathsf{Or})$$

$$\frac{\Gamma \vdash \mathsf{expression} : \mathsf{bool}}{\Gamma \vdash \mathsf{not}\ \mathsf{expression} : \mathsf{bool}}\ (\mathsf{Not})$$

Figure 3: A type system for expressions in mini-C.