Second Exam
CS 3366 Programming Languages

KEY

Friday April 7, 2017

Instructor Muller
Boston College
Spring 2017

Before reading further, please arrange to have an empty seat on either side of you. Now that you are seated, please write your name **on the back** of this exam.

This is a closed-notes and closed-book exam. Computers, calculators, and books are prohibited.

- Partial credit will be given so be sure to show your work.
- Feel free to write helper functions if you need them.
- **Please write neatly.**

Problem	Points	Out Of
Expressions		6
Statements		8
Procedures		6
Total		20

```
type       ::= int | bool | type → type | type * type | void | ok

boolean    ::= True | False
value      ::= integer | boolean | ()
op         ::= ...

procedures ::= procedure | procedure procedures
procedure  ::= type id (type id, ..., type id) block
block      ::= { type id; ...; type id; statement; ...; statement }
statement  ::= id = expression
             | id(expression, ..., expression)
             | return expression
             | while ( expression ) statement
             | if ( expression ) statement else statement
             | block
expression ::= value | id | expression op expression | id(expression, ..., expression)
```

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

This exam has essentially one question: define a type system for the subset of miniC/Mars whose abstract syntax is shown in Figure **??**. As you know, Mini-C is a clipped version of the C programming language. This exam deals with an even simpler form. Your task is to define a type system for the language shown.

There are essentially three forms that your type system has to account for: procedures, statements and expressions — you'll need a system for each of them. By "system", we mean 1. **judgements** of one kind or another together with 2. **axioms and inference rules** for deriving judgements.

Notes

1. You should start this exam by carefully reading the grammar, starting with the definition of procedures. As you can see, a miniC program is a sequence of procedure definitions

   ```
   procedure1 procedure2 ... procedureN
   ```

   where each $\mathsf{procedure}_k$ is free to make use of $\mathsf{procedure}_j$ for $j \leq k$. NB: procedures may be recursive but they are not mutually recursive. For example:

   ```
   int mod(int m, int n) {
     if (m < n)
       return m;
     else
       return mod(m - n, n);
   }

   int gcd(int m, int n) {
     if (m == 0)
       return n;
     else {
       int rem;
       rem = mod(n, m);
       return gcd(rem, m);
       }
   }
   ```

   Your type system should account for this ordering, gathering the type signatures of procedure definitions from first to last. In the example, the type environment used in typing `gcd` should include the type signature of `mod`.

2. Statements do not have value types in the sense of an expression type. Instead, they are either ok or they are not ok. Your judgements, axioms and inference rules should account for this. For example, the following block statement is not ok.

   ```
   { bool x; x = 5; }
   ```

3. There are 6 different statement forms and 4 different expression forms. The latter should be familiar to you and would be a good place to start defining your judgements, axioms and inference rules.

Expressions

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

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

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

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

Figure 2: Axioms and inference rules for expressions in mini-C. Judgements of the form $\Gamma \vdash \mathsf{expression} : \mathsf{type}$.

Statements

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

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

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

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

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

Figure 3: Inference rules for statements in mini-C. Judgements of the form $\Gamma, \mathsf{type} \vdash \mathsf{statement} : \mathsf{ok}$.

Procedures

$$\frac{\Gamma \vdash \mathsf{procedure} \Rightarrow \Gamma';\quad \Gamma' \vdash \mathsf{procedures} : \mathsf{ok}}{\Gamma \vdash \mathsf{procedure\ procedures} : \mathsf{ok}}\ (\mathsf{Procedures}_1)$$

$$\frac{\Gamma[\mathsf{id} : \mathsf{type}_1 * \ldots * \mathsf{type}_n \to \mathsf{type}][\mathsf{id}_1 : \mathsf{type}_1] \ldots [\mathsf{id}_n : \mathsf{type}_n], \mathsf{type} \vdash \mathsf{block} : \mathsf{ok}}{\Gamma \vdash \mathsf{type\ id}\ (\mathsf{type}_1\ \mathsf{id}_1, \ldots, \mathsf{type}_n\ \mathsf{id}_n)\ \mathsf{block} : \mathsf{ok}}\ (\mathsf{Procedures}_2)$$

$$\frac{\Gamma[\mathsf{id} : \mathsf{type}_1 * \ldots * \mathsf{type}_n \to \mathsf{type}][\mathsf{id}_1 : \mathsf{type}_1] \ldots [\mathsf{id}_n : \mathsf{type}_n],\ \mathsf{type} \vdash \mathsf{block} : \mathsf{ok}}{\Gamma \vdash \mathsf{type\ id}\ (\mathsf{type}_1\ \mathsf{id}_1, \ldots, \mathsf{type}_n\ \mathsf{id}_n)\ \mathsf{block} \Rightarrow \Gamma[\mathsf{id} : \mathsf{type}_1 * \ldots * \mathsf{type}_n \to \mathsf{type}]}\ (\mathsf{Signature})$$

Figure 4: Inference rules for procedures in mini-C. Judgements of the form $\Gamma \vdash \mathsf{procedures} : \mathsf{ok}$ and $\Gamma \vdash \mathsf{procedure} \Rightarrow \Gamma$.

Scrap