510: Programming Languages

TinyC: A Fragment of C

David Walker

TinyC Overview

TinyC is C-like:

- Imperative programming model based on assignment.

- A single type of integers (words).

- Block-structured allocation of variables.

Example: Factorial Function

```
int fact (int n) {
  int r = 1;
  int m = 0;
  /* invariant: r = m! */
  while (m != n) {
    m := m+1;
    r := m*r;
  }
  /* r = n! */
  return r;
}
```

TinyC Programs

A **program** is a **global declaration** and a **statement**.

- Declarations define functions, global variables.
- Statement is executed at start-up (typically, `main()`).

$$prog \quad p \quad ::= \quad g \ s$$

TinyC Statements

A **statement** is either

- An **expression**, executed for its effect on variables.

- A `return` statement.

- A **conditional** statement.

- A **loop** statement.

- A **block** of local declarations and statements.

TinyC Statements

$$
\begin{array}{llll}
\textit{stmt} & s & ::= & e\,; \mid \texttt{if}\ (e)\ s_1\ \texttt{else}\ s_2 \mid \texttt{return}\ e; \mid \\
 & & & \texttt{while}\ (e)\ s \mid \{\ l\ ss\ \} \\
\\
\textit{sseq} & ss & ::= & \circ \mid s\ ss
\end{array}
$$

TinyC Expressions

An **expression** has both a **value** and an **effect**.

- Evaluates to an integer (if it terminates).
- Perhaps assigns to some variables.

$$\mathit{exp} \quad e \quad ::= \quad v \mid x \mid e_1 \texttt{+} e_2 \mid e_1 \texttt{-} e_2 \mid x \texttt{=} e \mid f(\mathit{es})$$

TinyC Declarations

A **global declaration** defines zero or more functions and variables.

$$gdec \quad g \quad ::= \quad \circ \mid vd \;\; g \mid fd \;\; g$$

A **local declaration** defines zero or more variables.

$$ldec \quad l \quad ::= \quad \circ \mid vd \;\; l$$

TinyC Declarations

A **variable declaration** defines a variable with an initial value.

$$\mathit{vdec} \quad \mathit{vd} \quad ::= \quad \texttt{int}\ x\ \texttt{=}\ v\ \texttt{;}$$

A **function declaration** defines a function with zero or more arguments.

$$\mathit{fdec} \quad \mathit{fd} \quad ::= \quad \texttt{int}\ f(\mathit{as})\ s$$

TinyC Declarations

An **argument declaration** defines the type of a variable.

$$
\begin{array}{llcl}
\mathit{arg} & a & ::= & \texttt{int}\ x \\
\mathit{args} & \mathit{as} & ::= & a \mid a\ \mathit{as}
\end{array}
$$

Binding and Scope

As usual, we **identify** programs that differ only in the names of bound variables.

- Function parameters.

- Local and global variables.

- Function names.

Getting this right is a bit involved (see notes), but the ideas are the same as before.

Static Semantics

The static semantics enforces these requirements:

- All variables and functions must be defined before use.

- Functions must be called with the correct number of arguments.

The semantics maintains two contexts:

- A set of active variables, V.

- A mapping of active functions to their arities, F.

Static Semantics

Main judgements:

$\vdash p$ ok	*p is well-formed*
$F, V \vdash s$ ok	*s is well-formed wrt F and V*
$F, V \vdash e$ ok	*e is well-formed wrt F and V*
$F, V \vdash g$ decs F', V'	*g declares functions F' and variables V'*
$F, V \vdash l$ decs V'	*l declares variables V'*

Static Semantics: Programs

$$\frac{\emptyset, \emptyset \vdash g \ \mathsf{decs}\ F, V \qquad F, V \vdash s \ \ \mathsf{ok}}{\vdash g \ \ s \ \ \mathsf{ok}}$$

Static Semantics: Statements

Expressions:

$$\frac{F, V \vdash e \ \mathsf{ok}}{F, V \vdash e\ \texttt{;}\ \mathsf{ok}}$$

Conditionals:

$$\frac{F, V \vdash e \ \mathsf{ok} \quad F, V \vdash s_1 \ \mathsf{ok} \quad F, V \vdash s_2 \ \mathsf{ok}}{F, V \vdash \texttt{if}\ (e)\ s_1\ \texttt{else}\ s_2 \ \mathsf{ok}}$$

Static Semantics: Statements

While loops:

$$\frac{F, V \vdash e \ \mathsf{ok} \qquad F, V \vdash s \ \mathsf{ok}}{F, V \vdash \texttt{while} \ (e) \ s \ \mathsf{ok}}$$

Return statements:

$$\frac{F, V \vdash e \ \mathsf{ok}}{F, V \vdash \texttt{return} \ e\texttt{;} \ \mathsf{ok}}$$

Static Semantics: Statements

Blocks of statements:

$$\frac{F, V \vdash l \ \mathsf{decs}\ V' \qquad F, V \cup V' \vdash ss\ \mathsf{ok} \qquad (V \cap V' = \emptyset)}{F, V \vdash \{\ l\ \ ss\ \}\ \mathsf{ok}}$$

Static Semantics: Expressions

Values:

$$\frac{}{F, V \vdash v \ \mathsf{ok}}$$

Variables must be declared:

$$\frac{(x \in V)}{F, V \vdash x \ \mathsf{ok}}$$

Arithmetic:

$$\frac{F, V \vdash e_1 \ \mathsf{ok} \quad F, V \vdash e_2 \ \mathsf{ok}}{F, V \vdash e_1 \texttt{+} e_2 \ \mathsf{ok}}$$

Static Semantics: Expressions

Assignment:

$$\frac{F, V \vdash e \ \mathsf{ok} \qquad (x \in V)}{F, V \vdash x \,\mathsf{=}\, e \ \mathsf{ok}}$$

Function call:

$$\frac{F, V \vdash es\,\mathsf{ok}\,n \qquad (F(f) = n)}{F, V \vdash f(es) \ \mathsf{ok}}$$

Static Semantics: Declarations

Global declarations:

$$\frac{}{F, V \vdash \circ \ \mathsf{decs}\ \emptyset}$$

$$\frac{F, V \vdash vd \ \mathsf{decs}\ x \qquad F, V \cup \{\, x \,\} \vdash g \ \mathsf{decs}\ F', V' \qquad (x \notin V')}{F, V \vdash vd \;\; g \ \mathsf{decs}\ F', \{\, x \,\} \cup V'}$$

$$\frac{\begin{array}{c} F, V \vdash fd \ \mathsf{decs}\ f : n \\ F \cup \{\, f : n \,\}, V \vdash g \ \mathsf{decs}\ F', V' \qquad (f \notin \mathsf{dom}(F)) \end{array}}{F, V \vdash fd \;\; g \ \mathsf{decs}\ \{\, f : n \,\} \cup F', V'}$$

Static Semantics: Declarations

Local declarations:

$$\frac{}{F, V \vdash \circ \ \mathsf{decs}\ \emptyset}$$

$$\frac{F, V \vdash vd\ \mathsf{decs}\ x \qquad F, V \cup \{\, x \,\} \vdash l\ \mathsf{decs}\ V' \qquad (x \notin V')}{F, V \vdash vd\ \ l\ \mathsf{decs}\ \{\, x \,\} \cup V'}$$

Static Semantics: Declarations

Variable declarations:

$$\frac{(x \notin V)}{F, V \vdash \texttt{int}\ x\ \texttt{=}\ v\ \texttt{;}\ \mathsf{decs}\ x}$$

Function declarations:

$$\frac{\begin{array}{c} F \cup \{\, f : n \,\}, V \cup V' \vdash s\ \mathsf{ok} \\ F, V \vdash as\ \mathsf{decs}\ V' \quad (||V'|| = n,\ f \notin \mathsf{dom}(F)) \end{array}}{F, V \vdash \texttt{int}\ f(as)\ s\ \mathsf{decs}\ f : n}$$

Dynamic Semantics

Main judgement forms:

Program execution	$p \mapsto p'$
Statement execution	$(g, s) \mapsto (g', s')$
Expression execution	$(g, e) \mapsto (g', e')$

States of the form $(g, v\ ;)$ are final.

Dynamic Semantics

Key ideas:

- Execution **modifies** global declarations!
- "Global" is relative to the current block.

Dynamic Semantics: Programs

$$\frac{(g, s) \mapsto (g', s')}{g \ \ s \mapsto g' \ \ s'}$$

Dynamic Semantics: Statements

An expression as a statement:

$$\frac{(g, e) \mapsto (g', e')}{(g, e\ \mathbf{;}) \mapsto (g', e'\ \mathbf{;})}$$

Value is irrelevant, only its effect.

Dynamic Semantics: Statements

Conditional:

$$\frac{(g, e) \mapsto (g', e')}{(g, \texttt{if}\ (e)\ s_1\ \texttt{else}\ s_2) \mapsto (g', \texttt{if}\ (e')\ s_1\ \texttt{else}\ s_2)}$$

$$\frac{}{(g, \texttt{if}\ (\texttt{0})\ s_1\ \texttt{else}\ s_2) \mapsto (g, s_2)}$$

$$\frac{(n \neq \texttt{0})}{(g, \texttt{if}\ (n)\ s_1\ \texttt{else}\ s_2) \mapsto (g, s_1)}$$

Dynamic Semantics: Statements

Loop:

$$\frac{}{\begin{array}{l}(g, \texttt{while}\ (e)\ s) \mapsto \\ (g, \texttt{if}\ (e)\ \{\ \ s\ \texttt{while}\ (e)\ s\ \}\ \texttt{else}\ \{\ \ \texttt{0}\,;\ \})\end{array}}$$

Dynamic Semantics: Statements

Return:

$$\frac{(g, e) \mapsto (g', e')}{(g, \texttt{return } e\texttt{;}) \mapsto (g', \texttt{return } e'\texttt{;})}$$

Dynamic Semantics: Block

$$\frac{(g \cdot l, ss) \mapsto^* (g' \cdot l', n\ ;)}{(g, \{\ l\ \ ss\ \}) \mapsto (g', n\ ;)}$$

$$\frac{(g \cdot l, ss) \mapsto^* (g' \cdot l', \texttt{return}\ v\texttt{;})}{(g, \{\ l\ \ ss\ \}) \mapsto (g', \texttt{return}\ v\texttt{;})}$$

Dynamic Semantics: Blocks

Key points:

- Multi-step evaluation in premise to complete execution of block.

- Extension of globals with locals of the block.

Dynamic Semantics: Sequences

$$\frac{}{(g, \circ) \mapsto (g, 0\ ;)}$$

$$\frac{(g, s) \mapsto (g', s')}{(g, s\ \ ss) \mapsto (g', s'\ \ ss)}$$

$$\frac{}{(g, v\ ;\ \ ss) \mapsto (g, ss)}$$

$$\frac{}{(g, \texttt{return}\ v;\ \ ss) \mapsto (g, \texttt{return}\ v;)}$$

Dynamic Semantics: Expressions

Variables:

$$\frac{g \mathbin{@} x = v}{(g, x) \mapsto (g, v)}$$

Dynamic Semantics: Expressions

Arithmetic:

$$\frac{(g, e_1) \mapsto (g', e_1')}{(g, e_1\texttt{+}e_2) \mapsto (g', e_1'\texttt{+}e_2)}$$

$$\frac{(g, e_2) \mapsto (g', e_2')}{(g, n_1\texttt{+}e_2) \mapsto (g', n_1\texttt{+}e_2')}$$

$$\frac{(n = n_1 + n_2)}{(g, n_1\texttt{+}n_2) \mapsto (g, n)}$$

Dynamic Semantics: Expressions

Assignment:

$$\frac{(g, e) \mapsto (g', e')}{(g, x = e) \mapsto (g', x = e')}$$

$$\frac{g \mathbin{@} x \leftarrow v = g'}{(g, x = v) \mapsto (g', v)}$$

Dynamic Semantics: Expressions

Function call:

$$\frac{(g, es) \mapsto (g', es')}{(g, f(es)) \mapsto (g', f(es'))}$$

$$\frac{\begin{array}{c} g \,@\, f = \texttt{int}\ f(\texttt{int}\ x_1\texttt{,} \ldots \texttt{int}\ x_k\texttt{,} \circ)\ s \\ (g, \{\ \texttt{int}\ x_1\ \texttt{=}\ v_1\ \texttt{;}\ \cdots \texttt{int}\ x_k\ \texttt{=}\ v_k\ \texttt{;}\ \circ\ s\ \}) \mapsto \\ (g', \texttt{return}\ v\,\texttt{;}) \end{array}}{(g, f(v_1, \ldots, v_k)) \mapsto (g', v)}$$

$$\frac{\begin{array}{c} g \,@\, f = \texttt{int}\ f(\texttt{int}\ x_1\texttt{,} \ldots \texttt{int}\ x_k\texttt{,} \circ)\ s \\ (g, \{\ \texttt{int}\ x_1\ \texttt{=}\ v_1\ \texttt{;}\ \cdots \texttt{int}\ x_k\ \texttt{=}\ v_k\ \texttt{;}\ \circ\ s\ \}) \mapsto (g', v\ \texttt{;}) \end{array}}{(g, f(v_1, \ldots, v_k)) \mapsto (g', v)}$$

Type Safety

Theorem 1

If $\vdash p$ ok, *then either* p *is finished, or there exists* p' *such that* $p \mapsto p'$ *and* $\vdash p'$ ok.

Adding More Types

Adding more than one type complicates the static semantics.

- Expressions can have more than one type.
- Each argument to a function may have a different type.
- The result type of a function must be checked.

Checking Return Types

Main issues:

- Ensure that every execution path encounters a `return` statement of the specified type. This check will be conservative.

- In a sequence $s\ ss$ the statement s need not return, but ss must return.

- Any return within s must be of the right type.

Checking Return Types

Two judgements are needed:

$F, V \vdash s \,\mathsf{rets}\, \tau$	s definitely returns type τ
$F, V \vdash s \,\mathsf{ok}\, \tau$	s is consistent with returning type τ

(Plus some related auxiliary judgements.)

Interesting Variations

Stack-allocated `struct`'s.

- Allow `struct`-typed local variables.
- Extend variables to **paths**.
- Pass entire struct? Return entire struct?

Interesting Variations

Adding `break` and `continue` for loops.

- Nearest enclosing loop, if any.

- Named enclosing loop.

Interesting Variations

Stack-allocated arrays.

- Cannot pass or return entire array.
- Adds new forms of path (subscripting).
- Passing and returning arrays?

Interesting Variations

Full-scale memory allocation.

- Heap-allocated `struct`'s and arrays.
- Explicit references, but without pointer arithmetic.

Interesting Variations

Pointers.

- Pointer arithmetic? Scaling?
- Commitment to representations. Endianness?
- Casting.