IST 597A: Software Security and Analysis (Spring 2015)

Lecture Note 3: Type Systems

Dinghao Wu

1 Syntax

Consider the following Mini-C language.

Program	P	::=	(D, s)
Declarations	D	::=	$x : \tau_1; D$
Types	τ	::=	$\mathsf{bool} \mid \mathsf{int}$
Variables	$x, y, \ldots$		
Statements	s	::=	$x = e \mid s_1; s_2 \mid \text{if } e \text{ then } s_1 \text{ else } s_2 \mid \text{while } e \text{ do } s$
Expressions	e	::=	$x \mid v \mid \odot e \mid e_1 \oplus e_2$
Values	v	::=	$n \mid \text{true} \mid \text{false}$
Integers	n		
Unary op	$\odot$	::=	$- \mid !$
Binary op	$\oplus$	::=	$+ \mid - \mid * \mid / \mid < \mid > \mid = \mid \wedge \mid \vee$

Exercise. Write a program in Mini-C to compute factorial.

2 Static Semantics

$$\frac{D \triangleright \Gamma \quad \Gamma \vdash s}{\vdash (D, s)} \text{ TyProg}$$

$$\frac{D \triangleright \Gamma}{x : \tau; D \triangleright \Gamma, (x, \tau)} \text{ TyDec}$$

$$\frac{}{[] \triangleright []} \text{ TyDecEmpty}$$

$$\frac{(x, \tau) \in \Gamma \quad Gamma \vdash e : \tau}{\Gamma \vdash x = e} \text{ TyStmAssign}$$

$$\frac{\Gamma \vdash s_1 \quad Gamma \vdash s_2}{\Gamma \vdash s_1; s_2} \text{ TyStmSeq}$$

$$\frac{\Gamma \vdash e : \mathsf{bool} \quad \Gamma \vdash s_1 \quad \Gamma \vdash s_2}{\Gamma \vdash \text{if } e \text{ then } s_1 \text{ else } s_2} \text{ TyStmIf}$$

$$\frac{\Gamma \vdash e : \mathsf{bool} \quad \Gamma \vdash s}{\Gamma \vdash \text{while } e \text{ do } s} \text{ TyStmWhile}$$

$$\frac{(x, \tau) \in \Gamma}{\Gamma \vdash x : \tau} \text{ TyExpVar}$$

$$\frac{}{\Gamma \vdash \text{true} : \mathsf{bool}} \text{ TyExpTrue}$$

$$\frac{}{\Gamma \vdash \text{false} : \mathsf{bool}} \text{ TyExpFalse}$$

$$\frac{}{\Gamma \vdash n : \mathsf{int}} \text{ TyExpInt}$$

$$\frac{\Gamma \vdash e : \mathsf{int}}{\Gamma \vdash -e : \mathsf{int}} \text{ TyExpNeg}$$

$$\frac{\Gamma \vdash e : \mathsf{bool}}{\Gamma \vdash !e : \mathsf{bool}} \text{ TyExpNot}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{int} \quad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1 \oplus e_2 : \mathsf{int}} \text{ TyExpIntBop} \quad (\text{where } \oplus \in \{+, -, *, /\})$$

$$\frac{\Gamma \vdash e_1 : \mathsf{bool} \quad \Gamma \vdash e_2 : \mathsf{bool}}{\Gamma \vdash e_1 \oplus e_2 : \mathsf{bool}} \text{ TyExpBoolBop} \quad (\text{where } \oplus \in \{\wedge, \vee\})$$

Exercise. Give a few examples of typing derivation.

3 Dynamic Semantics

3.1 Small Step Operational Semantics

An abstract program state (Δ, C) consists of the current computation store Δ (maps variables to values) and the code C needed for the rest of the computation. This can be viewed as an abstract machine. The machine starts with the original program P and an empty store (memory) $[\cdot]$. The machine halts when there is no code left to be executed. To make it more readable, a "$" is put there in the abstract machine to indicate it's done.

$$\frac{}{\langle \Delta, (x : \mathsf{bool}; D, s) \rangle \mapsto \langle \Delta[(x, \text{false})], (D, s) \rangle} \text{ EvalProgDeclBool}$$

$$\frac{}{\langle \Delta, (x : \mathsf{int}; D, s)\rangle \mapsto \langle \Delta[(x,0)], (D, s)\rangle}\ \text{EvalProgDeclInt}$$

$$\frac{}{\langle \Delta, (\mathsf{nil}, s)\rangle \mapsto \langle \Delta, s\rangle}\ \text{EvalProgDeclNil}$$

$$\frac{}{\langle \Delta, x = v\rangle \mapsto \langle \Delta[(x,v)], \$\rangle}\ \text{EvalStmAssignVal}$$

$$\frac{\Delta \triangleright e \mapsto e'}{\langle \Delta, x = e\rangle \mapsto \langle \Delta, x = e'\rangle}\ \text{EvalStmAssign}$$

$$\frac{\langle \Delta, s_1\rangle \mapsto \langle \Delta', s_1'\rangle}{\langle \Delta, (s_1; s_2)\rangle \mapsto \langle \Delta', (s_1'; s_2)\rangle}\ \text{EvalStmSeq}$$

$$\frac{}{\langle \Delta, \text{if true then } s_1 \text{ else } s_2\rangle \mapsto \langle \Delta, s_1\rangle}\ \text{EvalStmIfTrue}$$

$$\frac{}{\langle \Delta, \text{if false then } s_1 \text{ else } s_2\rangle \mapsto \langle \Delta, s_2\rangle}\ \text{EvalStmIfFalse}$$

$$\frac{\Delta \triangleright e \mapsto e'}{\langle \Delta, \text{if } e \text{ then } s_1 \text{ else } s_2\rangle \mapsto \langle \Delta, \text{if true then } s_1 \text{ else } s_2\rangle}\ \text{EvalStmIf}$$

$$\frac{\Delta \triangleright e \mapsto^* \text{true}}{\langle \Delta, \text{while } e \text{ do } s\rangle \mapsto \langle \Delta, s \text{ ; while } e' \text{ do } s\rangle}\ \text{EvalStmWhileTrue}$$

$$\frac{\Delta \triangleright e \mapsto \text{false}}{\langle \Delta, \text{while } e \text{ do } s\rangle \mapsto \langle \Delta, \$\rangle}\ \text{EvalStmWhileFalse}$$

Exercise. Work out the small-step typing rules for experssions.

3.2 Big-Step Operational Semantics

See Harper [3, Ch. 7].

4 Type Safety = Preservation + Progress

See Chapter 6 of Harper [3] and Chapter 8 of Pierce [5].

5 Homework 3

Problem 1. Add a "halt" statement in the language. Work out typing and evaluation rules and augment the safety proof.

Problem 2. Add a "ref τ" type for pointers and a pointer dereference operator "*" as in C, but with no pointer arithmetic. Work out their typing rules and prove the type safety theorem.

Problem 3. Add a security type qualifier "red τ" which stands for sensitive datatypes. You can use red color instead of text "red." Work out their typing rules and prove the type safety theorem.

6 Bibliography Notes and Further Reading

Informally speaking, the type safety states that well-typed programs do not "go wrong." The slogan "safety is preservation plus progress" is due to Harper.

The small-step style operational semantics is sometimes called *structural operational semantics* (as it operates on the structures of terms) introduced by Plotkin [6]. The big-step style is due to Kahn [4].

Harper [3] and Pierce [5] are two excellent textbooks on this topic. Cardelli [1] and Cardelli [2] are excellent introduction to the topic too.

References

[1] Luca Cardelli. Type systems. In *The Computer Science and Engineering Handbook*, pages 2208–2236. CRC Press, 1997.

[2] Luca Cardelli and Peter Wegner. On understanding types, data abstraction, and polymorphism. *ACM Comput. Surv.*, 17(4):471–523, December 1985.

[3] Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, New York, NY, USA, 2012. `http://www.cs.cmu.edu/~rwh/plbook/book.pdf`.

[4] Gilles Kahn. Natural semantics. In *STACS 87, Proceedings of the 4th Annual Symposium on Theoretical Aspects of Computer Science*, pages 22–39, Passau, Germany, 1987.

[5] Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, Cambridge, MA, USA, 2002.

[6] Gordon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, Aarhus, Denmark, 1981.