SmallC Formal Type Checking Rules

November 14, 2025

1 Introduction

This document presents the type checking rules for SmallC. SmallC as presented here does not perform type inference, only type checking.

2 Preliminaries

2.1. Context. Type rules define judgments that make use of a context Γ (Stylized as G). A context is a mapping between variable names and their type. $G(x)$ means to look up the value of x maps to in the context G. This operation is undefined if there is no mapping for x in G. The second operation is written $G[x : t]$. It defines a new context that is the same as G but maps x to t. It thus overrides any prior mapping for x in G.

2.2. Syntax. In this document, we have simplified the presentation of the syntax, so it may not correspond exactly to the files that your interpreter will read in. For example, we write `while` $e\,s$ to represent the syntax of a while-loop, where e is the guard and s is the body. This corresponds to `While of expr * stmt` in the AST file. Hopefully, the connection between what we show here at that file is clear enough from context.

2.3. Error conditions. The semantics here define only *correct* evaluations. It says nothing about what happens when, say, you have a type error. For example, for the rules below, there is no type t for which you can prove the judgment $\Gamma \vdash 1 + \textbf{true} : t$. In your actual implementation, erroneous programs will cause an exception to be raised, as indicated in the project README.

2.4. Unknown Types. In the rules below, some things (most notably values in the AST represented by $Value$) will type to UT, Unknown Type. This represents, for instance, user input which does not have a known type at compile time. In the project, UTs are associated with numbers - however, in most cases below, the numbers are not relevant and so are omitted.

2.5. Subtypes. Anywhere a value of type t is expected, if the value instead has type k where $k <: t$ (k is a subtype of t), this will also work. The subtyping rules are given as axioms.

3 Type Checking Rules

3.1. Subtypes.

$$\text{subtype-bool}\,\frac{}{UT <: bool} \qquad \text{subtype-int}\,\frac{}{UT <: int} \qquad \text{subtype-ut}\,\frac{}{UT(n) <: UT(m)}$$

3.2. Expression Axioms.

$$\text{var-lookup}\,\frac{G(x) = t}{G \vdash x : t} \qquad \text{int}\,\frac{}{G \vdash n : int} \qquad \text{bool}\,\frac{}{G \vdash b : bool} \qquad \text{value}\,\frac{}{G \vdash read() : UT}$$

3.3. Non-Axiom Expressions.

$$\text{add}\,\frac{G \vdash e_1 : int \quad G \vdash e_2 : int}{G \vdash e_1 + e_2 : int} \qquad \text{sub}\,\frac{G \vdash e_1 : int \quad G \vdash e_2 : int}{G \vdash e_1 - e_2 : int}$$

$$\text{mult}\,\frac{G \vdash e_1 : int \quad G \vdash e_2 : int}{G \vdash e_1 * e_2 : int} \qquad \text{div}\,\frac{G \vdash e_1 : int \quad G \vdash e_2 : int}{G \vdash e_1 / e_2 : int}$$

$$\text{and}\,\frac{G \vdash e_1 : bool \quad G \vdash e_2 : bool}{G \vdash e_1 \;\&\&\; e_2 : bool} \qquad \text{or}\,\frac{G \vdash e_1 : bool \quad G \vdash e_2 : bool}{G \vdash e_1 \;||\; e_2 : bool} \qquad \text{not}\,\frac{G \vdash e_1 : bool}{G \vdash \texttt{not}\ e_1 : bool}$$

$$\text{equal}\,\frac{G \vdash e_1 : t \quad G \vdash e_2 : t}{G \vdash e_1 == e_2 : bool} \qquad \text{not-equal}\,\frac{G \vdash e_1 : t \quad G \vdash e_2 : t}{G \vdash e_1\ ! = e_2 : bool} \qquad \text{greater-equal}\,\frac{G \vdash e_1 : t \quad G \vdash e_2 : t}{G \vdash e_1 >= e_2 : bool}$$

$$\text{less-equal}\,\frac{G \vdash e_1 : t \quad G \vdash e_2 : t}{G \vdash e_1 \Leftarrow e_2 : bool} \qquad \text{greater}\,\frac{G \vdash e_1 : t \quad G \vdash e_2 : t}{G \vdash e_1 > e_2 : bool} \qquad \text{less}\,\frac{G \vdash e_1 : t \quad G \vdash e_2 : t}{G \vdash e_1 < e_2 : bool}$$

3.4. Statements.

In SmallC, statements can modify the context. $\rightarrow G'$ indicates that this statement updated the context to G'. In addition, they themselves do not have a meaningful type in the context of this project, but their subsections still need to be typechecked. This is indicated in the below rules by each one producing a "type" ().

$$\text{seq}\,\frac{G \vdash s_1 : () \rightarrow G' \quad G' \vdash s_2 : () \rightarrow G''}{G \vdash s_1; s_2 : () \rightarrow G''} \qquad \text{print}\,\frac{G \vdash e : t}{G \vdash \texttt{printf}(e) : () \rightarrow G}$$

$$\text{if}\,\frac{G \vdash e_1 : bool \quad G \vdash s_1 : () \rightarrow G' \quad G \vdash s_2 : () \rightarrow G''}{G \vdash \texttt{if}\ e\ s_1\ \ s_2 : () \rightarrow G' \cup G''} \qquad \text{while-loop}\,\frac{G \vdash e : bool \quad G \vdash s : () \rightarrow G'}{G \vdash \texttt{while}\ e\ s : () \rightarrow G'}$$

$$\text{for-loop-given-predefined-guard-var}\,\frac{G(x) : int \quad G \vdash e_1 : int \quad G \vdash e_2 : int \quad G \vdash s : () \rightarrow G'}{G \vdash \texttt{for}\ x\ e_1\ e_2\ s : () \rightarrow G'}$$

$$\text{for-loop-given-guard-var-is-new}\,\frac{G \vdash e_1 : int \quad G \vdash e_2 : int \quad G[x : int] \vdash s : () \rightarrow G'}{G \vdash \texttt{for}\ x\ e_1\ e_2\ s : () \rightarrow G'}$$

$$\text{assign}\,\frac{G \vdash e : t_1 \quad (t_1 <: t_0\ \texttt{or}\ t_0 <: t_1)}{G \vdash x\ (\texttt{type}\ t_0) = e : () \rightarrow G[x : t_0]}$$