Typed Arithmetic Expressions

Principles of Programming Languages

CSE 526

1 Typed Arithmetic Expressions
2 Simply-Typed $\lambda$-Calculus
Types

- Types are a way to classify terms (programs).
- Meaningful terms (e.g., those that do not get stuck) should have a type.
- A typing relation relates terms to types.
- Two ways to define semantics:
  - *Curry-style*: Define terms and their semantics, then define types to reject those terms whose semantics are problematic.
  - *Church-style*: Define terms and a typing relation, then define semantics only for well-typed terms.
Typed arithmetic expressions

\[ t ::= \text{true} \quad \text{Terms} \]
\[ \quad | \quad \text{false} \]
\[ \quad | \quad \text{if}(t, t, t) \]
\[ \quad | \quad 0 \]
\[ \quad | \quad \text{succ}\ t \]
\[ \quad | \quad \text{pred}\ t \]
\[ \quad | \quad \text{iszero}\ t \]

\[ T ::= \quad \text{Types} \]
\[ \quad | \quad \text{Bool} \]
\[ \quad | \quad \text{Nat} \]
Typing relation for arithmetic expressions

The smallest binary relation “:” between types and terms satisfying all instances of the following inference rules:

- \( \text{true} : \text{Bool} \quad \text{T-TRUE} \)
- \( \text{false} : \text{Bool} \quad \text{T-FALSE} \)
- \( t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T \quad \frac{}{\text{if}(t_1, t_2, t_3) : T} \quad \text{T-IF} \)
- \( 0 : \text{Nat} \quad \text{T-ZERO} \)
- \( t_1 : \text{Nat} \quad \frac{}{\text{succ} t_1 : \text{Nat}} \quad \text{T-SUCC} \)
- \( t_1 : \text{Nat} \quad \frac{}{\text{pred} t_1 : \text{Nat}} \quad \text{T-PRED} \)
- \( t_1 : \text{Nat} \quad \frac{}{\text{iszero} t_1 : \text{Bool}} \quad \text{T-ISZERO} \)
A term \( t \) is said to be *well-typed* if there is a type \( T \) such that \( t : T \).

- **Uniqueness of types:** Each term \( t \) has at most one type \( T \) such that \( t : T \).
Properties of the typing relation

A term $t$ is said to be \textit{well-typed} if there is a type $T$ such that $t : T$.

- **Uniqueness of types:** Each term $t$ has at most one type $T$ such that $t : T$.

- **Progress:** For every well-typed term $t$, either $t$ is a value or there is a $t'$ such that $t \rightarrow t'$.
Properties of the typing relation

A term $t$ is said to be well-typed if there is a type $T$ such that $t : T$.

- **Uniqueness of types:** Each term $t$ has at most one type $T$ such that $t : T$.
- **Progress:** For every well-typed term $t$, either $t$ is a value or there is a $t'$ such that $t \rightarrow t'$.
- **Preservation:** If $t : T$ and $t \rightarrow t'$ then $t' : T$. 
Properties of the typing relation

A term $t$ is said to be well-typed if there is a type $T$ such that $t : T$.

- **Uniqueness of types:** Each term $t$ has at most one type $T$ such that $t : T$.
- **Progress:** For every well-typed term $t$, either $t$ is a value or there is a $t'$ such that $t \rightarrow t'$.
- **Preservation:** If $t : T$ and $t \rightarrow t'$ then $t' : T$.
- **Safety = Progress + Preservation**
Inversion of the typing relation

- If true : R then $R = \text{Bool}$
  (similarly, if false : R then $R = \text{Bool}$)
- If if($t_1, t_2, t_3$) : R then $t_1 : \text{Bool}$, $t_2 : R$ and $t_3 : R$.
- If 0 : R then $R = \text{Nat}$
- If succ $t_1$ : R then $R = \text{Nat}$ and $t_1 : \text{Nat}$
  (similarly for pred $t_1$ : R)
- If iszero $t_1$ : R then $R = \text{Bool}$ and $t_1 : \text{Nat}$

These results follow from the definition of the typing relation, and are used to prove progress and preservation.
Recall booleans, numbers and operations on them can be encoded in the pure λ-calculus.

Nevertheless, it is convenient to include primitive data types in the calculus as well.

\( \lambda B \) is an enriched calculus with boolean data types `true` and `false`, and operation `if`.

\( \lambda x. \lambda y. \text{if}(x, y, x) \) is a term in \( \lambda B \).

\( \lambda NB \) is a similarly enriched calculus with numbers and booleans.

\( \lambda x. \lambda y. \text{if}(\text{iszero}(x), \text{succ}(y), x) \) is a term in \( \lambda NB \).
Simply-Typed $\lambda$-Calculus

Syntax:

$$t ::= x \quad \text{Variable}$$

$$\lambda x : T . t \quad \text{Abstraction}$$

$$t t \quad \text{Application}$$
Simply-Typed $\lambda$-Calculus

Syntax:

$$t ::= \begin{array}{l}
\text{Terms} \\
\text{Variable} \\
\text{Abstraction} \\
\text{Application} \\
\end{array}
\begin{array}{l}
x \\
\lambda x : T . t \\
t t \\
\end{array}
$$

$$T ::= \begin{array}{l}
\text{Types} \\
\text{Base types} \\
\text{type of functions} \\
\end{array}
\begin{array}{l}
A \\
T \rightarrow T \\
\end{array}$$
Simply-Typed $\lambda$-Calculus

Syntax:

$$
t ::= \begin{array}{l}
x \quad \text{Variable} \\
\lambda x : T. t \quad \text{Abstraction} \\
t t \quad \text{Application} \\
\end{array}
$$

$$
T ::= \begin{array}{l}
A \quad \text{Base types} \\
T \rightarrow T \quad \text{type of functions} \\
\end{array}
$$

$$
\Gamma ::= \begin{array}{l}
\emptyset \quad \text{Empty Context} \\
\Gamma, x : T \quad \text{Variable Binding} \\
\end{array}
$$
Small-Step Evaluation Relation for simply-typed $\lambda$-calculus:

$$
\begin{align*}
&t_1 \to t'_1 \\
&\frac{t_1 \ t_2 \to t'_1 \ t_2}{t_1 \ t_2 \to t'_1 \ t_2} & \text{E-APP1} \\
&t_2 \to t'_2 \\
&\frac{v_1 \ t_2 \to v_1 \ t'_2}{v_1 \ t_2 \to v_1 \ t'_2} & \text{E-ABS2} \\
&\left(\lambda x : T \cdot t_1\right) v_2 \to [x \mapsto v_2]t_1 & \text{E-APPABS}
\end{align*}
$$
Typing Relation

\[ \frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad \text{T-VAR} \]
Typed Arithmetic Expressions

Simply-Typed λ-Calculus

Typing Relation

\[
\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad \text{T-VAR}
\]

\[
\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1. \ t_2 : T_1 \rightarrow T_2} \quad \text{T-ABS}
\]
Typing Relation

\[
\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad \text{T-VAR}
\]

\[
\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1. \ t_2 : T_1 \rightarrow T_2} \quad \text{T-ABS}
\]

\[
\frac{\Gamma \vdash s : T_1 \rightarrow T_2 \quad \Gamma \vdash t : T_1}{\Gamma \vdash (s \ t) : T_2} \quad \text{T-APP}
\]
Properties of the typing relation

A term $t$ is said to be well-typed in context $\Gamma$ if there is a type $T$ such that $t : T$.

- **Uniqueness of types:** In a context $\Gamma$, each term $t$ has at most one type $T$ such that $t : T$.
- **Progress:** For every closed, well-typed term $t$, either $t$ is a value or there is a $t'$ such that $t \rightarrow t'$.
- **Preservation under substitution:** If $\Gamma, x : S \vdash t : T$ and $\Gamma \vdash s : S$, then $\Gamma \vdash [x \mapsto s]t : T$.
- **Preservation:** If $\Gamma \vdash t : T$ and $t \rightarrow t'$ then $\Gamma \vdash t' : T$.
- **Safety = Progress + Preservation**
**Erasure and Typability**

`erase` is a function that maps simply-typed λ-terms to untyped λ-terms.

\[
\begin{align*}
    erase(x) &= x \\
    erase(\lambda x : T. t) &= \lambda x. \text{erase}(t) \\
    erase(t_1 t_2) &= \text{erase}(t_1) \text{erase}(t_2)
\end{align*}
\]

- If \( t \rightarrow t' \) under typed evaluation relation, then \( \text{erase}(t) \rightarrow \text{erase}(t') \)
Erasure and Typability

erase is a function that maps simply-typed \( \lambda \)-terms to untyped \( \lambda \)-terms.

\[
\begin{align*}
erase(x) &= x \\
erase(\lambda x : T. t) &= \lambda x. \ erase(t) \\
erase(t_1 \ t_2) &= \ erase(t_1) \ erase(t_2)
\end{align*}
\]

- If \( t \to t' \) under typed evaluation relation, then \( \ erase(t) \to \ erase(t') \)
- If \( \ erase(t) \to m' \), then there is a simply-typed term \( t' \) such that \( t \to t' \) (under typed evaluation relation) and \( \ erase(t') = m' \)
Erasure and Typability

*erase* is a function that maps simply-typed $\lambda$-terms to untyped $\lambda$-terms.

$$
erase(x) = x$$

$$
erase(\lambda x : T. t) = \lambda x. erase(t)$$

$$
erase(t_1 t_2) = erase(t_1) erase(t_2)$$

- If $t \rightarrow t'$ under typed evaluation relation, then $erase(t) \rightarrow erase(t')$
- If $erase(t) \rightarrow m'$, then there is a simply-typed term $t'$ such that $t \rightarrow t'$ (under typed evaluation relation) and $erase(t') = m'$
- An untyped term $m$ is **typable** if there is some simply-typed term $t$ and type $T$ and context $\Gamma$ such that $erase(t) = m$ and $\Gamma \vdash t : T$. 
Erasure and Typability

`erase` is a function that maps simply-typed \( \lambda \)-terms to untyped \( \lambda \)-terms.  

\[
erase(x) = x 
\]
\[
erase(\lambda x : T. \ t) = \lambda x. \ erase(t) 
\]
\[
erase(t_1 \ t_2) = erase(t_1) \ erase(t_2) 
\]

- If \( t \xrightarrow{} t' \) under typed evaluation relation, then \( erase(t) \xrightarrow{} erase(t') \)
- If \( erase(t) \xrightarrow{} m' \), then there is a simply-typed term \( t' \) such that \( t \xrightarrow{} t' \) (under typed evaluation relation) and \( erase(t') = m' \)
- An untyped term \( m \) is **typable** if there is some simply-typed term \( t \) and type \( T \) and context \( \Gamma \) such that \( erase(t) = m \) and \( \Gamma \vdash t : T \).
- Not every untyped lambda term is typable!
  Example: \( (x \ x) \)