CS 345H
Programming Languages: Honors
8: Type Theory

Type Systems

- Representation Independence
- Types as a safety measure in programming languages
  - Robin Milner: Well-typed programs don’t go wrong.
- Invariants and Equational properties

Example: Ariane 5

- Example: The Explosion of Ariane 5
- From the Official Report:
  - On 4 June 1996, the maiden flight of the Ariane 5 launcher ended in a failure. Only about 40 seconds after initiation of the flight sequence, at an altitude of about 3700 m, the launcher veered off its flight path, broke up and exploded.
  - The failure of the Ariane 501 was caused by the complete loss of guidance and attitude information 37 seconds after start of the main engine ignition sequence (30 seconds after lift-off). This loss of information was due to specification and design errors in the software of the inertial reference system.
  - The internal SRI software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer.

Example: Ariane 5

- The Inquiry Board recommends:
  - R5 Review all flight software (including embedded software), and in particular:
    - Identify all implicit assumptions made by the code and its justification documents on the values of quantities provided by the equipment. …
  - R9 Include external (to the project) participants when reviewing specifications, code and justification documents. Make sure that these reviews consider the substance of arguments, rather than check that verifications have been made.

http://www.ima.umn.edu/~arnold/disasters/ariane5rep.html
Type Systems

- Preventing execution errors by capturing them at compile time
  - Preventing a specific class of execution errors
  - Idea: Static analysis is easier than observing runtime behavior

- Safety
  - But... many languages are compiled to low-level unsafe code
  - Poses the challenge of compiler correctness

Typed Arithmetic Expressions

- Terms can evaluate to:
  - \( v ::= \)
    - true
    - false
    - \( nv \) numeric value

- \( nv ::= \)
  - \( \theta \) zero value
  - succ \( nv \) successor value

- Or...

Evaluation Rules

\[
\text{if } \text{true} \text{ then } t_2 \text{ else } t_3 \rightarrow t_2 \quad \text{(E-IfTrue)}
\]

\[
\text{if } \text{false} \text{ then } t_2 \text{ else } t_3 \rightarrow t_3 \quad \text{(E-IfFalse)}
\]

\[
\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \rightarrow t_1 \text{ if } t_1 \text{ then } t_2 \text{ else } t_3 \quad \text{(E-if)}
\]
### Evaluation Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 \rightarrow t'_1 )</td>
<td>(E-Succ)</td>
</tr>
<tr>
<td>( \text{succ} \ t_1 \rightarrow \text{succ} \ t'_1 )</td>
<td>(E-Succ)</td>
</tr>
<tr>
<td>( \text{pred} \ 0 \rightarrow 0 )</td>
<td>(E-PredZero)</td>
</tr>
<tr>
<td>( \text{pred} \ \text{(succ} \ n \text{)} \rightarrow n )</td>
<td>(E-PredSucc)</td>
</tr>
<tr>
<td>( t_1 \rightarrow t'_1 )</td>
<td>(E-Pred)</td>
</tr>
<tr>
<td>( \text{iszero} \ 0 \rightarrow \text{true} )</td>
<td>(E-IsZeroZero)</td>
</tr>
<tr>
<td>( \text{iszero} \ \text{(succ} \ n \text{)} \rightarrow \text{false} )</td>
<td>(E-IsZeroSucc)</td>
</tr>
<tr>
<td>( t_1 \rightarrow t'_1 )</td>
<td>(E-IsZero)</td>
</tr>
<tr>
<td>( \text{iszero} \ t_1 \rightarrow \text{iszero} \ t'_1 )</td>
<td>(E-IsZero)</td>
</tr>
</tbody>
</table>

### Example

- \( \text{pred if true then ifzero succ } 0 \text{ else } 0 \)
- \( \text{pred(if (true) then (ifzero (succ(0))) else (0))} \)
- \( \text{pred (ifzero (succ (0)))} \)
- \( \text{pred false} \)

### Stuck Term

- No further evaluation rule applies that would bring the term into a normal form
- Erroneous program

### A Type System can tell us without actual evaluation that a term does not get stuck

### Introducing a Type System

- Values can be of two different kinds in this term language
  - Boolean (Bool)
    - E.g., the result of \( \text{iszero} \)
    - The first argument of a conditional
  - Natural Numbers (Nat)
    - E.g., the result of \( \text{succ} \) and \( \text{pred} \)
    - E.g., the argument of \( \text{iszero} \)

- We can now say “a term has type \( T \)” when it obviously evaluates to a value of the appropriate form
  - Obviously = statically, without performing evaluation
  - \( \text{if true then false else true} \) has type \( \text{Bool} \)
  - \( \text{pred(succ(succ(0)))} \) hat type \( \text{Nat} \)

### Typing Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{true} : \text{Bool} )</td>
<td>(T-True)</td>
</tr>
<tr>
<td>( \text{false} : \text{Bool} )</td>
<td>(T-False)</td>
</tr>
<tr>
<td>( t_1 : \text{Bool} \quad t_2 : T \quad t_3 : T )</td>
<td>(T-If)</td>
</tr>
</tbody>
</table>

\( \text{if } t_1 \text{ then } t_2 \text{ else } t_3 \text{ : } T \)
Typing Rules

- **T-Zero**: \( \theta : \text{Nat} \) (T-Zero)
- **T-Succ**: \( \text{suc} t_1 : \text{Nat} \) (T-Succ)
- **T-Pred**: \( \text{pred} t_1 : \text{Nat} \) (T-Pred)
- **T-IsZero**: \( \text{iszero} t_1 : \text{Bool} \) (T-IsZero)

Typing Relation

- **Def**: The Typing Relation for arithmetic expressions is the smallest binary relation between terms and types satisfying all instances of the typing rules.
- **Def**: A term \( \tau \) is typeable or well-typed if there is some \( T \) such that \( \tau : T \)
- **Idea**: Answers of the sort: If the term has any type at all, then it must be type Bool.

Typing Derivation

- **if iszero \( \theta \) then \( \theta \) else suc \( \theta \) : Nat**

  \[
  \begin{array}{c}
  0 : \text{Nat} \quad \text{(T-Zero)} \\
  \text{iszero} \ 0 : \text{Bool} \quad \text{(T-IsZero)} \\
  \hline
  \text{if iszero} \ 0 \ \text{then} \ 0 \ \text{else suc} \ 0 : \text{Nat}
  \end{array}
  \]

  Every pair \((t, T)\) in the typing relation can be justified by such a derivation tree.

Inversion Lemma

- **if true : R, then R = Bool.**
- **if false : R, then R = Bool.**
- **if if \( t_1 \) then \( t_2 \) else \( t_3 \) : R, then \( t_1 : \text{Bool}, t_2 : R, \) and \( t_3 : R. \)**
- **if 0 : R, then R = Nat.**
- **if suc \( t_1 \) : R, then R = Nat and t_1 : Nat.**
- **if pred \( t_1 \) : R, then R = Nat and t_1 : Nat.**
- **if iszero \( t_1 \) : R, then R = Bool and t_1 : Nat.**
- **Practically describes an algorithm to construct a typing**
Limitations of Typing

- What is the type of the following term:
  \[
  \text{if false then true else 0}
  \]

- Well-formed versus well-typed.

- Type systems are often conservative. They do not predict the outcome of every possible program but only cover cases in which a safe approximation can be made.

Safety: Soundness of Type Systems

- Progress
  - A well-typed term \( t \) is not stuck
    - If \( t : T \) then either \( t \) is a value or \( t \rightarrow t' \) for some \( t' \).

- Preservation
  - Types are preserved by one-step evaluation
    - If \( t : T \) and \( t \rightarrow t' \) then \( t' : T \)

Canonical Forms

- Lemma:
  - If \( \nu \) is a value of type \( \text{Bool} \), then \( \nu \) is either \text{true} or \text{false}.
  - If \( \nu \) is a value of type \( \text{Nat} \), then \( \nu \) is a numeric value.

Progress

- Progress: If \( t : T \) then either \( t \) is a value or \( t \rightarrow t' \) for some \( t' \).

- Proof by induction on a derivation of \( t : T \):
  - Case T-True: \( t = \text{true} \Rightarrow t \) is a value
  - Case T-False: \( t = \text{false} \Rightarrow t \) is a value
  - Case T-Zero: \( t = 0 \Rightarrow t \) is a value
Progress

- Progress: If \( t : T \) then either \( t \) is a value or \( t \rightarrow t' \) for some \( t' \).

  - Case T-If: Inversion lemma tells us that for \( t = if \ t_1 \ then \ t_2 \ else \ t_3 \),
    \[ t_1 : \text{Bool}, t_2 : T, t_3 : T \]
    The following evaluation rules apply: E-IfTrue, E-IfFalse, and E-If
  - Assume that the theorem holds for \( t \).
    - If \( t_1 \) is a value, then \( t_1 : \text{Bool} \) and \( t_2 \) must be either true or false. In this case, the reduction can continue through E-IfTrue and E-IfFalse.
    - There exists a \( t_1' \) such that \( t_1 \rightarrow t_1' \) (since we know \( t_1 : \text{Bool} \)) and the reduction of \( t \) can be accomplished via E-If.

Preservation

- Preservation: If \( t : T \) and \( t \rightarrow t' \) then \( t' : T' \)

  - Case T-True: \( t = \text{true} \), \( T = \text{Bool} \), \( t \rightarrow t' \) is not possible.
  - Case T-False: \( t = \text{true} \), \( T = \text{Bool} \), \( t \rightarrow t' \) is not possible.
  - Case T-Zero: \( t = 0 \), \( T = \text{Nat} \), \( t \rightarrow t' \) is not possible.
  - Case T-Succ: \( t = \text{succ} \ t_1 \), \( T = \text{Nat} \), \( t_1 : \text{Nat} \)
    - Only E-Succ can derive a \( t \rightarrow t' \), since \( t_1 : \text{Nat} \) also \( t_1' : \text{Nat} \) by induction hypothesis.