Types

Two big questions:

1. What can we say about a term **without running it**? (Static Analysis)
2. Can we tell a term will get **stuck** without running it? (Types)

A *type* is a means of *classifying terms*. We will want these to "play well" with the **reduction relation**.

Typing Rules for Booleans

Like our operational semantics, the typing relation is defined using a set of **inference rules**.

$\mathbb{B}$ *(typed)* — *Extends* **B** *(3-1)*

New syntactic forms

$\mathtt{T}$ ::=		*types:*
	$\mathtt{Bool}$	*type of booleans*

New typing rules $\boxed{\mathtt{t : T}}$

$$\mathtt{true : Bool} \qquad \text{(T-True)}$$

$$\mathtt{false : Bool} \qquad \text{(T-False)}$$

$$\frac{\mathtt{t_1 : Bool} \qquad \mathtt{t_2 : T} \qquad \mathtt{t_3 : T}}{\mathtt{if}\ \mathtt{t_1}\ \mathtt{then}\ \mathtt{t_2}\ \mathtt{else}\ \mathtt{t_3} : \mathtt{T}} \qquad \text{(T-If)}$$

Typing If

Note the form of the rule T-If.

- If both t_2 and t_3 have *the same type* T, then complete expression has type T.
- Otherwise, the expression **has no type**

A term which can be typed is called **typable**, or **well-typed**. A term which can't be typed is called **untypable**.

Another way to say it: the type relation is **not total** on terms.

Typing If

Note the form of the rule T-If.

- If both t_2 and t_3 have *the same type* T, then complete expression has type T.
- Otherwise, the expression **has no type**

A term which can be typed is called **typable**, or **well-typed**. A term which can't be typed is called **untypable**.

Another way to say it: the type relation is **not total** on terms.

The following evaluates to a value, but is untypeable:

$$\texttt{if true then false else 0} \tag{1}$$

Natural Numbers

$\mathbb{B}$ $\mathbb{N}$ *(typed)* — *Extends* **NB** *(3-2) and 8-1*

New syntactic forms

T	::=	...	*types:*
		Nat	*type of natural numbers*

New typing rules $\boxed{t : T}$

$$0 : \mathsf{Nat} \qquad \text{(T-Zero)}$$

$$\frac{t_1 : \mathsf{Nat}}{\mathsf{succ}\ t_1 : \mathsf{Nat}} \qquad \text{(T-Succ)}$$

$$\frac{t_1 : \mathsf{Nat}}{\mathsf{pred}\ t_1 : \mathsf{Nat}} \qquad \text{(T-Pred)}$$

$$\frac{t_1 : \mathsf{Nat}}{\mathsf{iszero}\ t_1 : \mathsf{Bool}} \qquad \text{(T-IsZero)}$$

Definition of the Typing Relation

The **typing relation** for arithmetic expressions is the smallest binary relation between terms and types satisfying all the typing rules given in the last two figures.

- A term t is **well-typed** if there is some T such that $t : T$

When talking about types, we will often make statements like:

- If a term of the form $\texttt{succ}\ \ t_1$ has any type at all, then it has type $\texttt{Nat}$.

There is a sort of *information flow*, up and down the AST, of typing information.

Inversion of the Typing Relation

The following inversion rules are immediately derivable from our typing rules:

LEMMA: [Inversion of the Typing Relation]

$$\texttt{true} : R \implies R = Bool \tag{2}$$

$$\texttt{false} : R \implies R = Bool \tag{3}$$

$$\texttt{if}\ \ t_1\ \ \texttt{then}\ \ t_2\ \ \texttt{else}\ \ t_3 : R \implies t_1 : Bool \wedge t_2 : R \wedge t_3 : R \tag{4}$$

$$0 : R \implies R = Nat \tag{5}$$

$$\texttt{succ}\ \ t_1 : R \implies R = Nat \wedge t_1 : Nat \tag{6}$$

$$\texttt{pred}\ \ t_1 : R \implies R = Nat \wedge t_1 : Nat \tag{7}$$

$$\texttt{iszero}\ \ t_1 : R \implies R = Bool \wedge t_1 : Nat \tag{8}$$

Typing Derivations

Consider the term `if iszero 0 then 0 else pred 0`
Let's draw (on board) a **typing derivation** for it.

Uniqueness of Types

THEOREM [Uniqueness of Types]
Each term t has at most one type. That is, if t is well-typed, then its type is unique. Additionally, there is only one derivation of this type, based on our inference rules.

- Proof is by structural induction on t, and uses inversion.

Note that *induction over typing derivations* is also a valid means to prove certain properties.

Type Safety

The most important property of any type system: **safety**.

- Slogan: Well-typed terms can't go wrong
- i.e., if a term is well typed, it can't get stuck.

We break safety down into two pieces:

$$\textbf{Safety} = \textbf{Progress} + \textbf{Preservation}$$

Progress + Preservation

THEOREM [Progress of Typed Arithmetic Expressions]
A well-typed term is not stuck. That is, either it is a value, or one of our evaluation rules can be applied.

THEOREM [Preservation of Typed Arithmetic Expressions]
If a well-typed term takes a step of evaluation, then the resulting term is also well-typed.

- Taken together, we can say that any well-typed term will eventually evaluate to a well-typed value without getting stuck.
- We can argue this inductively over evaluation derivations.

Canonical Forms

The **canonical forms** of a type are the values which have that type.

LEMMA [Canonical Forms]

1. If v is a value of type $Bool$, then v is either `true` or `false`
2. If v is a value of type Nat, then v is a numeric value.
   - That is, v is either 0, or `succ` nv, where nv is also a numeric value.

Canonical Form of Bool, Nat

If v is a value of type $Bool$, then v is either `true` or `false`

By analysis of all values forms: `true`, `false`, 0, and `succ` nv.

- For `true` and `false`, get Bool from inversion.
- For 0, get Nat from inversion.
- For `succ` nv inversion gives that term must have type Nat, not $Bool$.

If v is a value of type Nat, then v is either `0` or `succ(nv)` where nv is a value of type Nat.
Argument is very similar to above.

Proof of Progress I

THEOREM : [Progress]
Suppose $t : T$. Then t is either a value, or else there is some t' such that $t \to t'$.

By Induction on typing derivations:

- T-True, T-False, and T-Zero, all apply if t is a value.

Proof of Progress II

- T-If:

$$t = \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \tag{9}$$

  By inversion:

$$t_1 : Bool \qquad t_2 : T \qquad t_3 : T$$

  - By the induction hypothesis, t_1 is either a value, or there is some t_1' such that $t_1 \to t_1'$
    - If a value, t_1 must be `true` or `false`, via the canonical forms lemma. In these cases either E-IfTrue or E-IfFalse apply to t respectively.
    - If $t_1 \to t_1'$, then E-If is applicable to t.

Proof of Progress III

- T-Succ. Inversion gives $t = \texttt{succ}\ \ t_1 \wedge t_1 : \mathit{Nat}$
  - IH: either t_1 value, or $\exists\ t_1'$ such that $t_1 \to t_1'$
    - ★ If t_1 is a value, must be numeric value (cannonical forms lemma).
    - ★ If $t_1 \to t_1'$, Then E-Succ is applicable.
- T-Pred. Inversion gives $t = \texttt{pred}\ \ t_1 \wedge t_1 : \mathit{Nat}$
  - IH: t_1 is either a value, or $\exists\ t_1'$ such that $t_1 \to t_1'$
    - ★ If t_1 is a value, it must be a numeric value via the canonical forms lemma.
      - If $t_1 = 0$, E-PredZero applies to t.
      - If $t_1 = \texttt{succ}\ \ t_2$, E-PredSucc applies to t.
    - ★ If $t_1 \to t_1'$, the congruency rule E-Pred applies to t.
- T-IsZero. Inverse gives $t = \texttt{isZero}\ \ t_1 \wedge t_1 : \mathit{Nat}$
  - IH: t_1 is either a value, or $\exists\ t_1'$ such that $t_1 \to t_1'$
    - ★ If t_1 is a value, must be NV by canonical from lemma.
      - If $t_1 = 0$, E-IsZeroZero applies to t.
      - If $t_1 = \texttt{succ}\ \ t_2$, E-IsZeroSucc applies to t.
    - ★ If $t_1 \to t_1'$, the congruency rule E-IsZero applies to t.

Proof of Preservation I

THEOREM [Preservation of Typed Arithmetic Expressions]

$$t : T \wedge t \to t' \implies t' : T \tag{10}$$

Induction on typing derivations; if last step was:

T-True: $t = \texttt{true} \wedge T = Bool$, so $t \not\to t'$.

T-False, T-Zero: same.

T-Succ: $t = \texttt{succ}\ \ t_1 \wedge T = Nat \wedge t_1 : Nat$

- only one rule, E-Succ:, thus $t_1 \to t_1'$.
- Plus $t_1 : Nat$ implies $t_1' : Nat$.
- From $t' = \texttt{succ}\ \ t_1'$ and $t_1' : Nat$, typing says $t' : Nat$

Proof of Preservation II

T-If: $t = \texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3$, $t_1 : Bool \wedge t_2 : T \wedge t_3 : T$
Now case analysis on evaluation rules for `if`:

- E-IfTrue: $t_1 = \texttt{true}$ and $t' = t_2 \Longrightarrow t' : T$.
- E-IfFalse: $t_2 = \texttt{false}$ and $t' = t_3 \Longrightarrow t' : T$.
- E-If: $t_1 \to t_1'$ and $\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \to \texttt{if}\ t_1'\ \texttt{then}\ t_2\ \texttt{else}\ t_3$.
    - IH: $t_1 : T \wedge t_1 \to t_1' \implies t_1' : T$.
        - $t_1 : Bool$ (via typing relation case analysis)
        - $t_1 \to t_1'$ (via evaluation relation case analysis)
        - Thus $t_1' : Bool$ by IH
    - As $t_1' : Bool$, $t_2 : T$ and $t_3 : T$, typing gives $\texttt{if}\ t_1'\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : T$.

(and so on; T-Pred does require more care)