Types

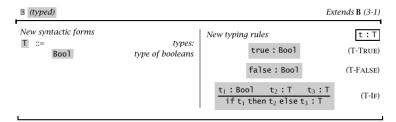
Two big questions:

- What can we say about a term without running it? (Static Analysis)
- 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.



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

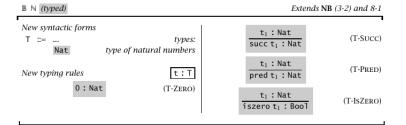
A term which can be typed is called **typable**, or **well-typed**. A term which can't be typed is called **untypable**.

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

The following evaluates to a value, but is untypeable:

if true then false else 0 (1)

Natural Numbers



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:
 - ullet If a term of the form succ t_1 has any type at all, then it has type 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]

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

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

if
$$t_1$$
 then t_2 else $t_3: R \implies t_1: Bool \land t_2: R \land t_3: R$ (4)

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

$$succ t_1: R \implies R = Nat \wedge t_1: Nat$$
 (6)

$$pred t_1: R \implies R = Nat \wedge t_1: Nat$$
 (7)

iszero
$$t_1: R \implies R = Bool \land t_1: Nat$$
 (8)

Typing Derivations

Consider the term if iszero 0 then 0 else pred 0 Let's draw (on board) a ${f typing}$ 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:

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

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]

- 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.
 - \blacktriangleright That is, ν is either 0, or succ $n\nu$, where $n\nu$ 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:

$$t = \text{if } t_1 \text{ then } t_2 \text{ else } t_3$$
 (9)

By inversion:

$$t_1$$
: Bool t_2 : T 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 $t_1 \rightarrow t'_1$, then E-If is applicable to t.

Proof of Progress III

- T-Succ. Inversion gives $t = \verb+succ+ t_1 \wedge t_1 : Nat$
 - ▶ 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).
 - \star If $t_1 \rightarrow t_1'$, Then E-Succ is applicable.
- T-Pred. Inversion gives $t = \text{pred } t_1 \wedge t_1 : Nat$
 - ▶ IH: t_1 is either a value, or $\exists t_1'$ such that $t_1 \to t_1'$
 - \star 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 $t_1 = succ \ t_2$, E-PredSucc applies to t.
 - \star If $t_1 o t_1'$, the congruency rule E-Pred applies to t.
- T-IsZero. Inverse gives $t = isZero t_1 \wedge t_1 : Nat$
 - ▶ 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 $t_1 = succ \ t_2$, E-IsZeroSucc applies to t.
 - **★** If $t_1 \rightarrow t'_1$, the congruency rule E-IsZero applies to t.

Proof of Preservation I

THEOREM [Preservation of Typed Arithmetic Expressions]

$$t: T \wedge t \to t' \implies t': T$$
 (10)

Induction on typing derivations; if last step was:

T-True: $t = \text{true} \land T = Bool$, so $t \nrightarrow t'$.

T-False, T-Zero: same.

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

- ullet only one rule, E-Succ:, thus $t_1
 ightarrow t_1'$.
- Plus t_1 : Nat implies t'_1 : Nat.
- From $t' = succ \ t'_1$ and $t'_1 : Nat$, typing says t' : Nat

Proof of Preservation II

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

- E-IfTrue: $t_1 = \text{true}$ and $t' = t_2 \Longrightarrow t' : T$.
- E-IfFalse: $t_2 = \text{false} \text{ and } t' = t_3 \Longrightarrow t' : T$.
- ullet E-If: $t_1 o t_1'$ and if t_1 then t_2 else $t_3 o$ if t_1' then t_2 else t_3 .
 - ▶ IH: $t_1: T \land t_1 \rightarrow t'_1 \implies t'_1: T$.
 - ★ t_1 : Bool (via typing relation case analysis)
 - \star $t_1
 ightarrow t_1'$ (via evaluation relation case analysis)
 - ★ Thus t'_1 : Bool by IH
- As t'_1 : Bool, t_2 : T and t_3 : T, typing gives if t'_1 then t_2 else t_3 : T. (and so on; T-Pred does require more care)