COMPSCI 3MI3 - Principles of Programming Languages

Topic 7 - Introduction to Type Theory

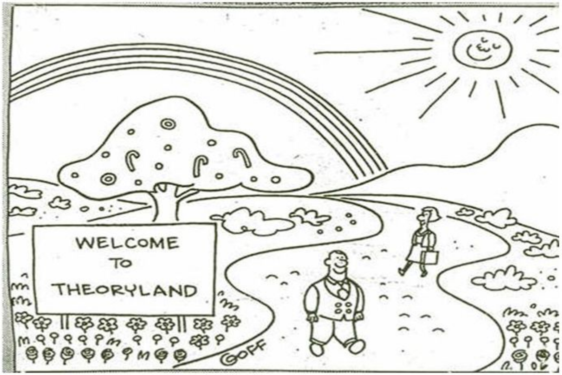
NCC Moore

McMaster University

Fall 2021

Adapted from “Types and Programming Languages” by Benjamin C. Pierce

The Typing Relation



"The road from untyped to typed universes has been followed many times, in many different fields, and largely for the same reasons."
– Luca Cardelli & Peter Wegner (1985)

Typing Arithmetic Expressions

Back in September, we developed a compact, simple language of Booleans and natural numbers.

- We will now revisit UAE, and augment it with **static types**.
- This will transform *Untyped* Arithmetic Expressions (UAE) into *Typed* Arithmetic Expressions (TAE).
- Once again, we will use this simple language to explore concepts related to type theory that will prove useful later.

Grammar of UAE

Recall the grammar of UAE.

$$
\begin{aligned}
\langle t \rangle ::= &\ \text{true} \\
\mid &\ \text{false} \\
\mid &\ \text{if } \langle t \rangle \text{ then } \langle t \rangle \text{ else } \langle t \rangle \\
\mid &\ 0 \\
\mid &\ \text{succ } \langle t \rangle \\
\mid &\ \text{pred } \langle t \rangle \\
\mid &\ \text{iszero } \langle t \rangle
\end{aligned}
$$

We also defined two subsets of our set of terms $\mathcal{T}$:

$$\mathcal{V} = \{\texttt{true}, \texttt{false}, \mathcal{NV}\} \tag{1}$$

$$\mathcal{NV} = \{0, \texttt{succ}\ \ \mathcal{NV}\} \tag{2}$$

Stuckness Revisited

Previously, we saw that UAE will either evaluate to an element of $\mathcal{V}$ or $\mathcal{NV}$, or get **stuck**.

- Roughly, stuck terms contain a type incompatability between operators and the terms they operate on.
- For example `iszero true` has no evaluation rules which apply, so no further evaluation is possible.
- We consider such programs *meaningless* or **erroneous**.

It would be useful to be able to tell if a term is going to result in an error *before* evaluating it.

Introducing... Types!

We need a way to specify what sort of value a term will produce when executed.

- We know that UAE values are divided into **Booleans** and **Natural Numbers**.
- We can therefore define the following *set of the types of UAE*.

$$T = \{Nat, Bool\} \tag{3}$$

We say that a term t "has type T" if it is obvious in some way that the result must be a value associated with the specified type.

- The type of a term should be easy to distinguish *without evaluation*.
- This is the difference between **static analysis** and **dynamic analysis**.

The Typing Relation

Effectively, we are talking about a **mapping** or **relation** between our set of terms $\mathcal{T}$ and our set of types T.

- We call this a **typing relation**, and we will using the colon symbol to indicate this relation. For example:

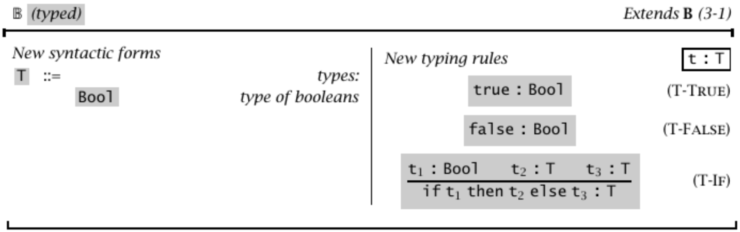
$$\texttt{true} : \mathit{Bool} \tag{4}$$

$$\texttt{false} : \mathit{Bool} \tag{5}$$

$$\texttt{0} : \mathit{Nat} \tag{6}$$

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

`T`	::=		*types:*
		`Bool`	*type of booleans*

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

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

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

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

Typing Iffiness

Note the form of the rule T-If.

- If both t_2 and t_3 have *the same type* T, then the if expression itself has type T.
- If t_2 and t_3 are of *different types*, the type of the if expression **cannot be determined!**
- The premise $t_1 : \mathit{Bool}$ also tells us that the guard term *must* be Boolean in order for the type of the expression to be determinable.

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

Non-totality in Typing Relations

Fundamentally, we want a type system which does not need to evaluate terms to yield useful information.

- It is impossible to tell which way the if expression will evaluate, unless we evaluate t_1.
- So, rather than forcing evaluation to determine typing, we say that such terms are *not in the typing relation*.

The typing relation is not total over $\mathcal{T}$, meaning...

$$dom(:) \subseteq \mathcal{T} \tag{7}$$

Wrong Values

The set of untypable expressions contains, as a subset, all of the nonsense terms we are seeking to capture, such as `succ true`. In other words,

$$\mathcal{W} \subseteq \mathcal{T} \setminus dom(:) \tag{8}$$

Where $\mathcal{W}$ is the set of all terms which are syntactically correct, but contain semantic errors.

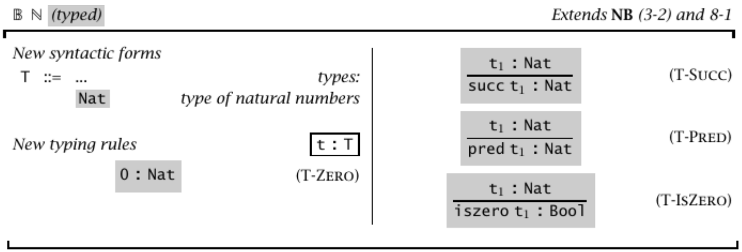
- This is the set of all terms that would evaluate to `Wrong` under UAE in Project 1.

We can't say, however, that *all* terms which are not well typed are the result of type mismatches. The following evaluates to a value, but is untypeable:

$$\texttt{if true then false else 0} \tag{9}$$

Natural Numbers

Similarily, typing rules for the natural number operations is as follows:



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

New syntactic forms

T ::= ... *types:*

Nat *type of natural numbers*

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

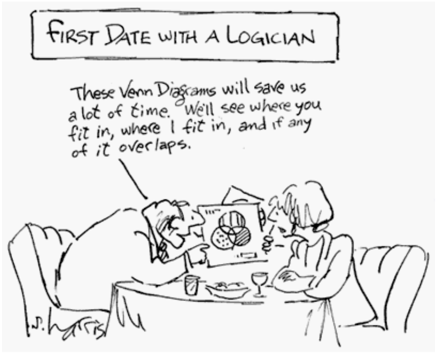
$$0 : \mathtt{Nat} \qquad \text{(T-ZERO)}$$

$$\frac{t_1 : \mathtt{Nat}}{\mathtt{succ}\ t_1 : \mathtt{Nat}} \qquad \text{(T-SUCC)}$$

$$\frac{t_1 : \mathtt{Nat}}{\mathtt{pred}\ t_1 : \mathtt{Nat}} \qquad \text{(T-PRED)}$$

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

Formalities of the UAE Type System


FIRST DATE WITH A LOGICIAN
These Venn Diagrams will save us a lot of time. We'll see where you fit in, where I fit in, and if any of it overlaps.
S. Harris

Definition of the Typing Relation

Formally, the **typing relation** for arithmetic expressions is the smallest binary relation between terms and types satisfying all the typing rules given in the previous section.

- 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 $\texttt{Nat}$.

It will be handy to be able to derive the types of subterms from their containing terms, and not just type terms by their subterms.

Inversion of the Typing Relation

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

LEMMA: [Inversion of the Typing Relation]

$$\texttt{true} : R \implies R = \mathit{Bool} \tag{10}$$

$$\texttt{false} : R \implies R = \mathit{Bool} \tag{11}$$

$$\texttt{if}\ \ t_1\ \ \texttt{then}\ \ t_2\ \ \texttt{else}\ \ t_3 : R \implies t_1 : \mathit{Bool} \land t_2 : R \land t_3 : R \tag{12}$$

$$0 : R \implies R = \mathit{Nat} \tag{13}$$

$$\texttt{succ}\ \ t_1 : R \implies R = \mathit{Nat} \land t_1 : \mathit{Nat} \tag{14}$$

$$\texttt{pred}\ \ t_1 : R \implies R = \mathit{Nat} \land t_1 : \mathit{Nat} \tag{15}$$

$$\texttt{iszero}\ \ t_1 : R \implies R = \mathit{Bool} \land t_1 : \mathit{Nat} \tag{16}$$

Typing Derivations

In the same way that our evaluation relation rules allowed us to create evaluation derivations, our typing relation rules allow us to produce **typing derivations**.

- **Statements** are formal assertions (in the Dr. Farmer sense) about the typing of programs.
- **Typing rules** are implications between statements.
- **Derivations** are deductions based on typing rules.

For example, consider the term

```
if iszero 0 then 0 else pred 0
```

Example Typing Derivation

We derive our type like so:

(1)	$\boxed{\texttt{if iszero 0 then 0 else pred 0}}$	
(2)	$\boxed{\texttt{iszero 0}}$	Hypothesis
(3)	$\boxed{0}$	Hypothesis
(4)	*Nat*	T-Zero on 3
(5)	0 : *Nat*	T-Intro on 3, 4
(6)	*Bool*	T-IsZero on 2, 5
(7)	`iszero 0` : *Bool*	T-Intro on 2, 6
(8)	$\boxed{0}$	Hypothesis
(9)	*Nat*	T-Zero on 8
(10)	0 : *Nat*	T-Intro on 8, 9

Example Typing Derivation (cont.)

Continuing our derivation...

(1)	$\boxed{\texttt{if iszero 0 then 0 else pred 0}}$	
(11)	$\boxed{\texttt{pred 0}}$	Hypothesis
(12)	$\boxed{0}$	Hypothesis
(13)	*Nat*	T-Zero on 12
(14)	$0 : \mathit{Nat}$	T-Intro on 12, 13
(15)	*Nat*	T-Pred on 11, 14
(16)	$\texttt{pred 0} : \mathit{Nat}$	T-Intro on 11, 15
(17)	*Nat*	T-If on 1, 7, 10, 15
(18)	$\texttt{if iszero 0 then 0 else pred 0} : \mathit{Nat}$	T-Intro on 1, 17

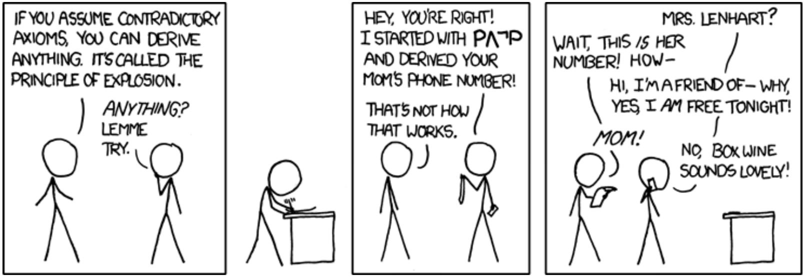
Everyone Wants to be Unique!

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.

- To prove the above, we would proceed via structural induction on t, break it into a case analysis, invoking the appropriate inversion formula (plus the induction hypothesis) in each case.

We can also use induction over typing derivations to generate proofs, the same way we used induction over evaluation relations to generate proofs under UAE.

The Safety of Type Systems


IF YOU ASSUME CONTRADICTORY AXIOMS, YOU CAN DERIVE ANYTHING. IT'S CALLED THE PRINCIPLE OF EXPLOSION.
ANYTHING? LEMME TRY.
HEY, YOU'RE RIGHT! I STARTED WITH P∧¬P AND DERIVED YOUR MOM'S PHONE NUMBER!
THAT'S NOT HOW THAT WORKS.
MRS. LENHART?
WAIT, THIS IS HER NUMBER! HOW–
HI, I'M A FRIEND OF– WHY, YES, I AM FREE TONIGHT!
MOM!
NO, BOX WINE SOUNDS LOVELY!

Type Safety

The most important property of TAE, or any other type system, is that of **safety**.

- Safety and **soundness** not quite the same thing, but the textbook uses them interchangeably.
  - In mathematical logic, a logical system is sound iff every formula that can be proved in the system is logically valid with respect to the semantics of the system.
- i.e., if a term is well typed, it can't get stuck.
  - That is, we reach a normal form without having reached a value.
- We generally prove safety by demonstrating the theorems of **Progress** and **Preservation**

Theoremception!

THEOREM [Progress of Typed Arithmetic Expressions]
A well-typed term is not (currently) 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 value without getting stuck.
- We can argue this inductively over evaluation derivations.

When Life gives you Lemmas

In order to prove the theorems on the previous slide, we will need to more tightly associate values with types. The **canonnical forms** of a type are the values in our language 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.

Pachelbel's Canonical Form

To prove the above, we first address the two clauses individually, starting with the first.

- According to UAE grammar, values have four forms: `true`, `false`, 0, and `succ` *nv*.
- For `true` and `false`, we can derive a Boolean type from clauses 1 and 2 of the inversion lemma.
- For 0, we derive *Nat*, rather than *Bool* from clause 4 of the inversion lemma.
- For `succ` *nv* we note from clause 5 of the inversion lemma that the term must have type *Nat*, not *Bool*.
  - Unlike the other terms, `succ` *t* is not always well typed.
  - Fortunately, we assume typability in our hypothesis.
- We can therefore conclude that the only Boolean values are `true` and `false`.

The argument for clause 2 is precisely symmetrical.

Proof of Progress of TAE

Proof of Progress I

We may now attempt prove the progress of TAE. If we state the theorem somewhat more rigorously...

THEOREM : [Progress]
Suppose t is a well-typed term of TAE. That is, $t : T$ for some T. We may conclude that t is either a value, or else there is some t' such that $t \to t'$.

All well-typed terms are well-typed because a typing rule applies to them. We will proceed via case analysis of our typing rules.

- For T-True, T-False, and T-Zero, all three apply if t is a value, so our theorem is satisfied.
- The rest will require induction over typing derivations.

Proof of Progress II

Let's consider T-If. Our term must be of the form:

$$t = \texttt{if} \; t_1 \; \texttt{then} \; t_2 \; \texttt{else} \; t_3 \tag{17}$$

Additionally, we get the premises of T-If for free:

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

- 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.
- So, in all cases, t is either a value, or $t \to t'$.

Proof of Progress III

Let's now consider T-Succ. We can immediately state:

$$t = \texttt{succ}\ \ t_1 \qquad\qquad t_1 : Nat$$

- By the induction hypothesis, t_1 is either a value, or there is some t_1' such that $t_1 \to t_1'$
  - If t_1 is a value, it must be a numeric value, via the cannonical forms lemma. t is therefore a value as well.
  - If $t_1 \to t_1'$, Then E-Succ is applicable.
- In all cases, therefore, t is either a value, or $t \to t'$.

Proof of Progress IV

Let's consider T-Pred. We can immediately state:

$$t = \texttt{pred}\ \ t_1 \qquad\qquad\qquad t_1 : Nat$$

- By the induction hypothesis, t_1 is either a value, or there is some 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.

    In either case, an evaluation rule applies to t.
  - If $t_1 \to t_1'$, the congruency rule E-Pred applies to t.
- Therefore, in all cases, t is either a value, or $t \to t'$.

Proof of Progress V

Finally, let's consider T-IsZero. We can immediately state:

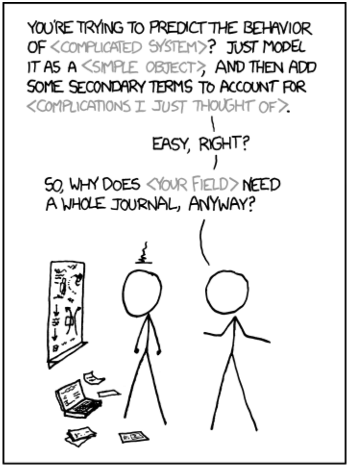
$$t = \texttt{isZero}\ \ t_1 \qquad\qquad t_1 : Nat$$

- By the induction hypothesis, t_1 is either a value, or there is some 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-IsZeroZero applies to t.
    - If $t_1 = \texttt{succ}\ \ t_2$, E-IsZeroSucc applies to t.

    In either case, an evaluation rule applies to t.
  - If $t_1 \to t_1'$, the congruency rule E-IsZero applies to t.
- Therefore, in all cases, t is either a value, or $t \to t'$.

We may therefore conclude that, for all possible well-typed terms, that the term is either a value, or can be evaluated using an evaluation rule. *QED*

Preservation


YOU'RE TRYING TO PREDICT THE BEHAVIOR OF <COMPLICATED SYSTEM>? JUST MODEL IT AS A <SIMPLE OBJECT>, AND THEN ADD SOME SECONDARY TERMS TO ACCOUNT FOR <COMPLICATIONS I JUST THOUGHT OF>.
EASY, RIGHT?
SO, WHY DOES <YOUR FIELD> NEED A WHOLE JOURNAL, ANYWAY?

LIBERAL-ARTS MAJORS MAY BE ANNOYING SOMETIMES, BUT THERE'S *NOTHING* MORE OBNOXIOUS THAN A PHYSICIST FIRST ENCOUNTERING A NEW SUBJECT.

Proof of Preservation I

More rigorously stated...
THEOREM [Preservation of Typed Arithmetic Expressions]

$$t : T \wedge t \to t' \implies t' : T \qquad (18)$$

Proof is by induction on typing derivations. That is, we will assume that the above holds for all subderivations. We will then proceed by case analysis of the "final" typing derivation.

We will be checking...

T-True T-False T-Zero

T-If T-Succ T-Pred T-IsZero

Proof of Preservation II

Consider T-True. We may immediately state:

$$t = \texttt{true} \qquad\qquad T = \mathit{Bool}$$

- In this case, $t \not\rightarrow t'$. The left-hand-side of our implication is false, so the theorem is vacuously true.

We immediately see that the exact same argument is applicable to T-False and T-Zero.

T-True T-False T-Zero

T-If T-Succ T-Pred T-IsZero

Proof of Preservation III

Consider T-If. We may immediately state:

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

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

- Since we are trying to conclude $t' : T$ from $t \to t'$, as well as $t : T$, we can also perform case analysis over the evaluation rules of if expressions.
  - For E-IfTrue, we know $t_1 = \texttt{true}$ and $t' = t_2$.
    - Since we have $t_2 : T$ above, we may conclude $t' : T$.
  - E-IfFalse proceeds as above, but with t_3 instead of t_2.
  - For E-If, we know that $t_1 \to t_1'$ and $\texttt{if}\ \ t_1\ \ \texttt{then}\ \ t_2\ \ \texttt{else}\ \ t_3 \to \texttt{if}\ \ t_1'\ \ \texttt{then}\ \ t_2\ \ \texttt{else}\ \ t_3$.

Proof of Preservation IV

- In this case, our induction hypothesis states $t_1 : T \wedge t_1 \to t_1' \implies t_1' : T$.
  - We know $t_1 : Bool$ (via typing relation case analysis)
  - We know that $t_1 \to t_1'$ (via evaluation relation case analysis)
  - We can conclude therefore that $t_1' : Bool$
- Since we have $t_1' : Bool$, $t_2 : T$ and $t_3 : T$, we conclude via our typing relation inference rule that $\texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3 : T$.

In all cases, therefore, we conclude $t' : T$.

T-True T-False T-Zero

T-If T-Succ T-Pred T-IsZero

Proof of Preservation V

Let us now consider T-Succ. We can immediately state:

$$t = \texttt{succ}\ \ t_1 \qquad T = Nat \qquad t_1 : Nat$$

- In this case, there is only one rule, E-Succ which can be used in $t \rightarrow t'$.
  - From this, we know that $t_1 \rightarrow t_1'$.
- Since we have this fact with the fact that $t_1 : Nat$, we conclude that $t_1' : Nat$.
- Since we know $t' = \texttt{succ}\ \ t_1'$ and $t_1' : Nat$, we can use our typing relation rules to conclude $t' : Nat$

T-True T-False T-Zero

T-If T-Succ T-Pred T-IsZero

Proof of Preservation VI

Our T-Pred case proceeds along similar lines. We can immediately state:

$$t = \texttt{pred}\ t_1 \qquad T : Nat \qquad t_1 : Nat$$

In this case, we must examine E-Pred, E-PredZero and E-PredSucc

- The proof of the congruence rule E-Pred is the same as the proof of E-Succ, so we won't repeat it.
- Consider E-PredZero. We can immediately state that both $t' = 0$, and $t_1 = 0$.
  - $t' : Nat$ is immediate from T-Zero.
- Consider E-PredSucc. We can immediately state that $t_1 = \texttt{succ}\ t_2$ and that $t' = t_2$.
- To finish this case, we can use clause 5 of the inversion lemma.

Proof of Preservation VII

Clause 5 of the inversion lemma:

$$\texttt{succ}\ \ s_1 : R \implies R = \mathit{Nat} \wedge s_1 : \mathit{Nat} \tag{19}$$

- If we set $s = t_2$, then $\texttt{succ}\ \ s = t_1$.
- We know that $t_1 : \mathit{Nat}$ from our case analysis.
- We can therefore conclude $t_2 : \mathit{Nat}$ from the inversion lemma.
- That is to say, $t' : \mathit{Nat}$.

Therefore, any case results in $t' : \mathit{Nat}$.

T-True T-False T-Zero

T-If T-Succ T-Pred T-IsZero

Proof of Preservation VIII

Finally, consider the case of T-IsZero. We immediately know:

$$t = \texttt{isZero}\ \ t_1 \qquad\qquad T = \mathit{Bool} \qquad\qquad t_1 : \mathit{Nat}$$

There are three evaluation rules which may apply: E-IsZero, E-IsZeroZero, E-IsZeroSucc

- The proof of E-IsZero is too similar to the congruence rules E-Pred and E-Succ to restate.
- Consider E-IsZeroZero. We immediately know $t' = \texttt{true}$.
    - We may immediately conclude from T-True that $t' : \mathit{Bool}$.
- Consider E-IsZeroSucc. We immediately know that $t' = \texttt{false}$
    - $t' : \mathit{Bool}$ is immediate from T-False.
- Therefore, $t' : \mathit{Bool}$ in all cases.

Proof of Preservation IX

T-True T-False T-Zero

T-If T-Succ T-Pred T-IsZero

We have therefore demonstrated that the property of preservation holds in all possible cases, for all possible typing relations.

QED

Last Slide Comic

I'M DONE FOR THE NIGHT. COMPUTER - LOG OUT!
COMPUTER LOGOUT!
EXIT EXIT!
LOG OUT!
PLEASE LOG OUT
RAPH_COMIC + DOGMO COMICS