CS720

Logical Foundations of Computer Science

Lecture 17: Type systems

Tiago Cogumbreiro

What is a Type System

1. **Asserts that a term is well-formed:** eg, consider a fraction represented by two integers, assert that the denominator is not a zero; eg, all functions terminate;
2. **Asserts that a term is of a given category:** eg, an expression is numeric; eg, a file-pointer is in an open state

How does a Type System work

- Performed at compile time (a static analysis technique)
- Enforces policies to **guarantees certain properties** statically: eg, in Rust, memory is manually allocated, but no memory is leaked, no data-races errors; eg, in Java, the method of a method calls must be known at compile-time and the argument-type must match the parameter-type.

Limitations of IMP

One of the limitations of IMP is that our expressions can only have one type:

- Boolean expressions can only appear in loops/ifs
- Assignments only accept numeric expressions (no booleans)

Introducing data of different types

Let us define an expression language

$$t ::= \mathbf{true} \mid \mathbf{false} \mid \mathbf{if}\ t\ \mathbf{then}\ t\ \mathbf{else}\ t \mid 0 \mid \mathbf{succ}\ t \mid \mathbf{pred}\ t \mid \mathbf{iszero}\ t$$

Example:

```
if iszero (succ (succ(0))) then 0 else pred (succ(succ(0)))
```

Ill-formed example:

```
succ(true)
```

Values

$$\frac{}{\text{bvalue}(\texttt{true})}\,(\text{bv-true}) \qquad \frac{}{\text{bvalue}(\texttt{false})}\,(\text{bv-false})$$

$$\frac{}{\text{nvalue}(0)}\,(\text{nv-zero}) \qquad \frac{\text{nvalue}(v)}{\text{nvalue}(\text{succ}(v))}\,(\text{nv-succ})$$

$$\text{value}(v) := \text{bvalue}(v) \vee \text{nvalue}(v)$$

Value or not a value?

1. `succ(if true then succ(0) else 0)`:

Value or not a value?

1. `succc(if true then succ(0) else 0)`: Not a value.
2. `false`:

Value or not a value?

1. `succ(if true then succ(0) else 0)`: Not a value.
2. `false`: A value.
3. `iszero(0)`:

Value or not a value?

1. `succ(if true then succ(0) else 0)`: Not a value.
2. `false`: A value.
3. `iszero(0)`: Not a value.
4. `succ(0)`:

Value or not a value?

1. `succ(if true then succ(0) else 0)`: Not a value.
2. `false`: A value.
3. `iszero(0)`: Not a value.
4. `succ(0)`: A value.

Semantics

$$\frac{}{\texttt{if true then } t_1 \texttt{ else } t_2 \Rightarrow t_1}\text{(IfTrue)}$$

$$\frac{}{\texttt{if false then } t_1 \texttt{ else } t_2 \Rightarrow t_2}\text{(IfFalse)}$$

$$\frac{t_1 \Rightarrow t_1'}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \Rightarrow \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3}\text{(If)} \qquad \frac{t_1 \Rightarrow t_1'}{\texttt{succ}(t_1) \Rightarrow \texttt{succ}(t_1')}\text{(Succ)}$$

$$\frac{}{\texttt{pred}(0) \Rightarrow 0}\text{(PredZero)} \qquad \frac{\text{nvalue}(v)}{\texttt{pred}(\texttt{succ}(v)) \Rightarrow v}\text{(PredSucc)} \qquad \frac{t \Rightarrow t'}{\texttt{pred}(t) \Rightarrow \texttt{pred}(t')}\text{(PredSucc)}$$

$$\frac{}{\texttt{iszero}(0) \Rightarrow \texttt{true}}\text{(IszeroZero)} \qquad \frac{\text{nvalue}(v)}{\texttt{iszero}(\texttt{succ}(v)) \Rightarrow \texttt{false}}\text{(IszeroSucc)}$$

$$\frac{t \Rightarrow t'}{\texttt{iszero}(t) \Rightarrow \texttt{iszero}(t')}\text{(Iszero)}$$

Reduction example (1)

```
if iszero(succ(succ(0))) then 0 else pred(succ(succ(0)))
```

Reduction example (1)

```
if iszero(succ(succ(0))) then 0 else pred(succ(succ(0)))
⟹ If, IszeroSucc
if false then 0 else pred(succ(succ(0))
```

Reduction example (1)

```
if iszero(succ(succ(0))) then 0 else pred(succ(succ(0)))
⟹ If, IszeroSucc
if false then 0 else pred(succ(succ(0))
⟹ IfFalse
pred(succ(succ(0)))
```

Reduction example (1)

```
if iszero(succ(succ(0))) then 0 else pred(succ(succ(0)))
⟹ If, IszeroSucc
if false then 0 else pred(succ(succ(0))
⟹ IfFalse
pred(succ(succ(0)))
⟹ PredSucc
succ(0)
```

Reduction example (2)

```
pred(false)
```

Reduction example (2)

`pred(false)`

> How do we reduce now?

Reduction example (2)

`pred(false)`

> How do we reduce now?

Some terms are **invalid**! These are expression for which we want to consider to be malformed somehow.

> Which means our language does not enjoy the process of ***strong progress***.

Stuck terms

Stuck terms

Let us define the notion of stuck.

$$\mathrm{stuck}(t) := \neg\mathrm{value}(t) \wedge \mathrm{nf}(t)$$

> Think of it as a negation of progress (which says that a term is either a value or reduces)

Example

$$\frac{\dfrac{}{\mathrm{nf}(\texttt{pred}(\texttt{zero}))} \qquad \dfrac{}{\neg\mathrm{value}(\texttt{pred}(\texttt{zero}))}}{\mathrm{stuck}(\texttt{pred}(\texttt{zero}))}$$

Example: Stuck or not stuck?

1. $\texttt{iszero}(\texttt{if true then succ}(0)\ \texttt{else}\ 0)$

Example: Stuck or not stuck?

1. $\texttt{iszero}(\texttt{if true then succ}(0)\ \texttt{else}\ 0)$ Not stuck.

> Is it a value or does it reduce?

Example: Stuck or not stuck?

1. $\texttt{iszero}(\texttt{if true then succ}(0)\ \texttt{else}\ 0)$ Not stuck.

> Is it a value or does it reduce?

Reduces.

> What does it reduce to?

Example: Stuck or not stuck?

1. $\texttt{iszero}(\texttt{if true then succ}(0)\ \texttt{else}\ 0)$ Not stuck.

Is it a value or does it reduce?

Reduces.

What does it reduce to?

```
iszero(if true then succ(0) else 0) ⟹(IfTrue) iszero(succ(0)) ⟹(IszeroSucc) false
```

2. $\texttt{if succ}(0)\ \texttt{then true else false}$

Example: Stuck or not stuck?

1. $\texttt{iszero}(\texttt{if true then succ}(0)\ \texttt{else}\ 0)$ Not stuck.

> Is it a value or does it reduce?

Reduces.

> What does it reduce to?

`iszero(if true then succ(0) else 0)` $\Longrightarrow$`(IfTrue) iszero(succ(0))` $\Longrightarrow$`(IszeroSucc) false`

2. $\texttt{if succ}(0)\ \texttt{then true else false}$ Stuck. Why?

Example: Stuck or not stuck?

1. $\mathtt{iszero}(\mathtt{if}\ \mathtt{true}\ \mathtt{then}\ \mathtt{succ}(0)\ \mathtt{else}\ 0)$ Not stuck.

> Is it a value or does it reduce?

Reduces.

> What does it reduce to?

`iszero(if true then succ(0) else 0)` $\Longrightarrow$`(IfTrue) iszero(succ(0))` $\Longrightarrow$`(IszeroSucc) false`

2. $\mathtt{if}\ \mathtt{succ}(0)\ \mathtt{then}\ \mathtt{true}\ \mathtt{else}\ \mathtt{false}$ Stuck. Why? The `if` expects a boolean.

Type system

Type system

- A type system is a set of rules that disciplines **expression composition**.
- Our expressions can have different types: numerical or boolean
- A type system holds when an expression is of a given type

$$\vdash t \colon T$$

In our language our types are:

$$T ::= \texttt{Bool} \mid \texttt{Nat}$$

Defining a Type System (1/2)

Boolean values:

$$\frac{}{\vdash \texttt{true} : \texttt{Bool}}\,(\text{t-true}) \qquad \frac{}{\vdash \texttt{false} : \texttt{Bool}}\,(\text{t-false})$$

Defining a Type System (1/2)

Boolean values:

$$\frac{}{\vdash \texttt{true} \colon \texttt{Bool}}\text{(t-true)} \qquad \frac{}{\vdash \texttt{false} \colon \texttt{Bool}}\text{(t-false)}$$

Natural values:

$$\frac{}{\vdash 0 \colon \texttt{Nat}}\text{(t-zero)} \qquad \frac{\vdash t \colon \texttt{Nat}}{\vdash \texttt{succ}(t) \colon \texttt{Nat}}\text{(t-succ)}$$

Defining a Type System (1/2)

Boolean values:

$$\frac{}{\vdash \texttt{true} \colon \texttt{Bool}}\,(\text{t-true}) \qquad \frac{}{\vdash \texttt{false} \colon \texttt{Bool}}\,(\text{t-false})$$

Natural values:

$$\frac{}{\vdash 0 \colon \texttt{Nat}}\,(\text{t-zero}) \qquad \frac{\vdash t \colon \texttt{Nat}}{\vdash \texttt{succ}(t) \colon \texttt{Nat}}\,(\text{t-succ})$$

Composed expressions:

$$\frac{\vdash t \colon \texttt{Nat}}{\vdash \texttt{iszero}(\texttt{t}) \colon \texttt{Bool}}\,(\text{t-iszero}) \qquad \frac{\vdash t \colon \texttt{Nat}}{\vdash \texttt{pred}(\texttt{t}) \colon \texttt{Nat}}\,(\text{t-pred})$$

Defining a Type System (2/2)

- How do we write the rule for `if`?

$$\frac{\vdash t_1 : \color{red}{???} \qquad \vdash t_2 : \color{red}{???} \qquad \vdash t_3 : \color{red}{???}}{\vdash \mathtt{if}\ t_1\ \mathtt{then}\ t_2\ \mathtt{else}\ t_3 : \color{red}{???}}\,(\text{t-if})$$

Defining a Type System (2/2)

- How do we write the rule for `if`?

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

- Notice how both branches have the same type!

Examples

Example 1:

$$
\dfrac{
\dfrac{\dfrac{\dfrac{}{\vdash 0 : \mathtt{Nat}}}{\vdash \mathtt{succ}(0) : \mathtt{Nat}}}{\vdash \mathtt{succ}(\mathtt{succ}(0)) : \mathtt{Nat}}}{\vdash \mathtt{iszero}(\mathtt{succ}(\mathtt{succ}(0))) : \mathtt{Bool}}
\quad
\dfrac{}{\vdash 0 : \mathtt{Nat}}
\quad
\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash 0 : \mathtt{Nat}}}{\vdash \mathtt{succ}(0) : \mathtt{Nat}}}{\vdash \mathtt{succ}(\mathtt{succ}(0)) : \mathtt{Nat}}}{\vdash \mathtt{pred}(\mathtt{succ}(\mathtt{succ}(0))) : \mathtt{Nat}}
}{\vdash \mathtt{if}\ \mathtt{iszero}(\mathtt{succ}(\mathtt{succ}(0)))\ \mathtt{then}\ 0\ \mathtt{else}\ \mathtt{pred}(\mathtt{succ}(\mathtt{succ}(0))) : \mathtt{Nat}}
$$

Examples

Example 1:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash 0 : \texttt{Nat}}}{\vdash \texttt{succ}(0) : \texttt{Nat}}}{\vdash \texttt{succ}(\texttt{succ}(0)) : \texttt{Nat}}}{\vdash \texttt{iszero}(\texttt{succ}(\texttt{succ}(0))) : \texttt{Bool}} \quad \dfrac{}{\vdash 0 : \texttt{Nat}} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash 0 : \texttt{Nat}}}{\vdash \texttt{succ}(0) : \texttt{Nat}}}{\vdash \texttt{succ}(\texttt{succ}(0)) : \texttt{Nat}}}{\vdash \texttt{pred}(\texttt{succ}(\texttt{succ}(0))) : \texttt{Nat}}$$
$$\overline{\vdash \texttt{if}\ \texttt{iszero}(\texttt{succ}(\texttt{succ}(0)))\ \texttt{then}\ 0\ \texttt{else}\ \texttt{pred}(\texttt{succ}(\texttt{succ}(0))) : \texttt{Nat}}$$

Example 2:

$$\dfrac{}{\color{red}{\nvdash \texttt{succ}(\texttt{true})}}$$

Expected results

Expected results

Theorem. If $\vdash t : T$ and $t \Rightarrow^\star t'$, then $\neg\mathrm{stuck}(t')$.

> Type soundness tells us that all well-typed programs never reach a stuck state.

- Java and Scala's Type Systems are Unsound [OOPSLA16]
- Scala with Explicit Nulls [ECOOP20]

  > remove[s] the specific source of unsoundness identified by Amin and Tate [OOPSLA16]. This class of bugs, reported in 2016 and still present in Scala and Dotty, happens due to a combination of implicit nullability and type members with arbitrary lower and upper bounds.

- Other examples: Python's ***mypy*** or ***TypeScript***

Expected results

Theorem. If $\vdash t : T$ and $t \Rightarrow^{\star} t'$, then $\neg \mathrm{stuck}(t')$.

> Type soundness tells us that all well-typed programs never reach a stuck state.

- A framework to ensure the absence of an undesired behavior
- Type system characterizes some **desired** behaviors statically
- Type systems rejects some programs with desired behaviors (**false positives**)
- Type soundness proves the type system rejects **undesired** behavior (no false negatives)
- Type soundness is difficult to prove, because programming languages are complicated

Type soundness at UMB-SVL

Faial (UMB-SVL) checks the absence of data-races in CUDA programs

- Faial is **sound in theory** (proved in Coq), but **unsound in practice** due to
  - implementation bugs
  - unsupported CUDA features
- Faial is **incomplete** in theory and in practice, because we do not want to say that a program is free from bugs, when it does have bugs

```
__global__
void saxpy(int n, float a, float *x, float *y)
{
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n) y[i] = a*x[i] + y[i + 1];
}
```

```
** DATA RACE ERROR ***

Array:   y[1]
T1 mode: W
T2 mode: R

---------------------
 Locals       T1  T2
---------------------
 threadIdx.x  1   0
---------------------
```

UMass Boston

Progress

Progress

Theorem. If $\vdash t : T$, then $\mathrm{value}(t) \vee \exists t', t \Rightarrow t'$.

Progress vs Strong progress

1. **Theorem (Strong Progress).** $\mathrm{value}(t) \lor \exists t', t \Rightarrow t'$.
2. **Theorem (Progress).** If $\vdash t : T$, then $\mathrm{value}(t) \lor \exists t', t \Rightarrow t'$.

> What is the relation between the ***progress*** property defined here and the ***strong progress*** from `SmallStep`?

1. No difference
2. Progress implies strong progress
3. Strong progress implies progress
4. They are unrelated properties

Progress vs Strong progress

1. **Theorem (Strong Progress).** $\mathrm{value}(t) \lor \exists t', t \Rightarrow t'$.
2. **Theorem (Progress).** If $\vdash t : T$, then $\mathrm{value}(t) \lor \exists t', t \Rightarrow t'$.

> What is the relation between the ***progress*** property defined here and the ***strong progress*** from `SmallStep`?

1. No difference
2. Progress implies strong progress
3. Strong progress implies progress
4. They are unrelated properties

Strong progress implies progress.

Progress (proof)

Theorem. If $\vdash t : T$, then $\mathrm{value}(t) \lor \exists t', t \Rightarrow t'$.

The proof follows by induction on the derivation of the hypothesis. At each case we have that the simpler term is well typed and that the term is either a value or it reduces.

- In the case that the simpler term is a value, we use the canonical properties, to show that our goal is also a value.
- In the case that the simpler term can reduce, we use apply the reduction rule for the given term to reduce the goal.

```
Lemma bool_canonical : forall t,
  |- t \in TBool -> value t -> bvalue t.

Lemma nat_canonical : forall t,
  |- t \in TNat -> value t -> nvalue t.
```

Quiz

- Is every well-typed normal form is a value?

Quiz

> Is every well-typed normal form is a value?

Yes! A corollary of the progress theorem.

> Is every value is a normal form?

Quiz

> Is every well-typed normal form is a value?

Yes! A corollary of the progress theorem.

> Is every value is a normal form?

Yes!

> Is the single-step reduction relation a ***total*** function?

UMass Boston

Quiz

> Is every well-typed normal form is a value?

Yes! A corollary of the progress theorem.

> Is every value is a normal form?

Yes!

> Is the single-step reduction relation a ***total*** function?

No. Counter-example: reducing a value.

Type preservation

Type preservation

Theorem. If $\vdash t : T$ and $t \Rightarrow t'$, then $\vdash t' : T$.

> Type preservation establishes the robustness of our type system: a static (compile-time) abstraction is ensured in ***all executions*** of any accepted program. Otherwise, our type system could say an expression returns a number and upon executing that expression we find out it actually returns a boolean.

Type preservation (proof)

Theorem. If $\vdash t : T$ and $t \Rightarrow t'$, then $\vdash t' : T$.

The proof follows by induction on the derivation of the ***first*** hypothesis. At each case we must invert the hypothesis that the term reduces. The proof for each case is trivial, as we simply need to apply the typing rule for each term.

Type soundness

Type soundness

Theorem. If $\vdash t : T$ and $t \Rightarrow^\star t'$, then $\neg\mathrm{stuck}(t')$.

> Type soundness tells us that all well-typed programs never reach a stuck state.

Deterministic step

Deterministic step

Theorem. If $x \Rightarrow y_1$ and $x \Rightarrow y_2$, then $y_1 = y_2$.

Proof by induction on the derivation of the first hypothesis. At each of the 10 cases, we need to invert the second hypothesis $x \Rightarrow y_2$, which yields 22 cases. Use `auto` and `solve_by_invert` to take care of boring cases (8 cases should remain).

- At cases such as `ST_If` and `ST_Succ` we can simply use the induction hypothesis to rewrite the output term of reducing t_1.
- The remaining cases all follow the same structure: they reach a contradiction (remember to use `exfalso`). For instance, in the case for rule `ST_PredSucc`, we have that t_1 is a nat-value and that $\mathrm{succ}(t_1) \Rightarrow t_1'$. We conclude by inverting the latter, and using lemma `nvalue_no_step`.