UNIVERSITY OF Southampton
School of Electronics and Computer Science

COMP2212 PROGRAMMING LANGUAGE CONCEPTS

Julian Rathke and Pawel Sobocinski

TYPE SAFETY

A STATEMENT OF TYPE SAFETY

- Recall type safety slogan "well-typed programs never go wrong"
  - we understand what it means to be well-typed for a program: inductive typing relation between programs and types.
  - we know what it means for a program to "go" : by inductively defining operational semantics for that program.

- We need to understand what "wrong" means
- Notice that the operational semantics we gave for the Toy language did not depend on the terms being well-typed.
  - e.g. `if(if(true) then false else 0) then true else 34` is ill-typed but according to the big and small step semantics, it would evaluate to the value 34.
  - consider instead `if(if(false) then false else 0) then true else 34` . This term is also ill-typed but if one tries to evaluate it, it gets 'stuck'

```
if(if(false) then false else 0) then true else 34
→ if(0) then true else 34
→ ???
```

STUCK TERMS AND DIVERGENCE

- For the small step semantics, we can model our run time errors as 'stuck' terms. That is, terms that can not evaluate any further and are not values of the language.
- For the big step semantics, we could do the same, but unfortunately, were we to introduce recursion or while loops to the language then big step semantics could not distinguish between stuck and divergent terms.
  - to model run time errors in a big step semantics we must explicitly model either errors. This would be done by introducing an error relation E ↯ that describes when E has reached an error state.
  - e.g. if (n) then E else E' ↯ would hold for any literal n.
  - we won't go in to details of all of the error cases, let's just assume ↯ is defined.
- This means that we could now distinguish between a stuck term and a divergent term in the big step semantics in the presence of recursion.
- Unfortunately, this still doesn't allow us to state type safety for the big step semantics.
- The big step semantics only considers 'complete' computations. It cannot refer to what happens during computation.
- All that we can easily specify with a big step semantics is a weak notion of type correctness: preservation.

A WEAK STATEMENT OF TYPE SAFETY - PRESERVATION

A typed language satisfies **(weak) preservation** if the following holds:

for all closed well-typed terms E :

$$\text{if } \vdash E : T \text{ and } E \Downarrow V \text{ then } \vdash V : T$$

This states that programs do not change their type after run time. More precisely it says that that values that programs produce actually are of the expected type.

The situation for small-step semantics is much better:

A typed language satisfies **preservation** if the following holds:

for all closed well-typed terms E :

$$\text{if } \vdash E : T \text{ and } E \rightarrow E' \text{ then } \vdash E' : T$$

This states that, at every step of evaluation, programs do not change their type.

PROGRESS

- Preservation alone is not a sufficient property to describe type safety.
- It does not capture the fact that well-typed terms stay in semantically 'good' states.
- For example, if we have a (broken) type system that allows a stuck term to be well-typed, then this stuck term will trivially satisfy preservation.
  - There are no reductions from the stuck term so types are preserved !
- **To capture type safety more fully we need to know that well-typed terms never become stuck.**
- This property is what we call progress.

A typed language satisfies **progress** if the following holds:

for all closed well-typed terms E :

if $\vdash E : T$ then either $E \rightarrow E'$ (for some E') or E is a value V

Type safety = preservation and progress.

PROVING PROGRESS FOR TOY LAMBDA

Theorem: if $\vdash$ E : T then E $\rightarrow$ E' or E is a value V

Proof: We use a proof by induction over the typing derivation $\vdash$ E : T

Base Cases: (Let's remind ourselves of the typing rules)

THE TYPE SYSTEM IN FULL (FOR REFERENCE)

$$\frac{}{\Gamma \vdash n : \mathsf{Int}}\ \textsc{TInt} \qquad \frac{}{\Gamma \vdash b : \mathsf{Bool}}\ \textsc{TBool} \qquad \frac{x : T \in \Gamma}{\Gamma \vdash x : T}\ \textsc{TVar}$$

$$\frac{\Gamma \vdash E_1 : \mathsf{Int} \quad \Gamma \vdash E_2 : \mathsf{Int}}{\Gamma \vdash E_1 < E_2 : \mathsf{Bool}}\textsc{TLt} \qquad \frac{\Gamma \vdash E_1 : \mathsf{Int} \quad \Gamma \vdash E_2 : \mathsf{Int}}{\Gamma \vdash E_1 + E_2 : \mathsf{Int}}\textsc{TAdd}$$

$$\frac{\Gamma \vdash E_b : \mathsf{Bool} \quad \Gamma \vdash E_1 : T \quad \Gamma \vdash E_2 : T}{\Gamma \vdash \mathsf{if}\ E_b\ \mathsf{then}\ E_1\ \mathsf{else}\ E_2 : T}\ \textsc{TIf}$$

$$\frac{\Gamma \vdash E_1 : T \quad \Gamma, x : T \vdash E_2 : U}{\Gamma \vdash \mathsf{let}\ (x : T)\ =\ E_1\ \mathsf{in}\ E_2 : U}\textsc{TLet}$$

$$\frac{\Gamma, x : T \vdash E : U}{\Gamma \vdash \lambda(x : T)E : T \to U}\textsc{TLam}$$

$$\frac{\Gamma \vdash E_1 : T \to U \quad \Gamma \vdash E_2 : T}{\Gamma \vdash E_1\ E_2 : U}\textsc{TApp}$$

PROVING PROGRESS FOR TOY LAMBDA

Theorem: if $\vdash E : T$ then $E \rightarrow E'$ or E is a value V

Proof: We use a proof by induction over the typing derivation $\vdash E : T$

Base Cases: if the proof trees consist of just the rule TInt or TBool then E is a value.

Inductive Cases: (we'll show TIf as an example)

Suppose the last rule used to derive the type of `E` is TIf. Then we know that `E` is of the form `if E₁ then E₂ else E₃` where $\vdash$ `E₁` : Bool and $\vdash$ `E₂` , `E₃` : T.

Now, if `E₁` is not a value, then by the inductive hypothesis we must have that `E₁ → E₁'` for some `E₁`'. This allows us to derive

```
  if E₁ then E₂ else E₃ →  if E₁' then E₂ else E₃
```

as required.

Suppose then that `E₁` is a value. We know that the only values of type Bool are true and false. In either case we get a reduction

```
 if E₁ then E₂ else E₃ → E₂ or
 if E₁ then E₂ else E₃ → E₃.
```

This means, whatever the form of E1, we have a reduction and hence progress.

We must consider every type rule in the language and perform similar reasoning in each case

PROVING PRESERVATION FOR TOY

Theorem:

If $\vdash$ E : T and E $\rightarrow$ E' then $\vdash$ E' : T

Proof: We use a proof by induction over the typing derivation $\vdash$ E : T

Base Cases: rule TInt or TBool , E is a value so the hypothesis is trivially satisfied

Inductive Cases: (we'll show rule TIf as an example)

Suppose the last rule used to derive the type of **E** is TIf. Then we know that **E** is of the form **`if E₁ then E₂ else E₃`** where $\vdash$ **E₁** : Bool and $\vdash$ **E₂** , **E₃** : T.

Now, consider the possible reductions that can originate from **E** : either

(i) it is a reduction of **E₁** to some **E₁'**, that is **E₁** $\rightarrow$ **E₁'** (so that **E'** is **`if E₁' then E₂ else E₃`**) or

(ii) **E₁** is a boolean literal and **`if E₁ then E₂ else E₃`** $\rightarrow$ **E₂** (or similar for **E₃**) so that **E'** is **E₂** (or **E₃**). In case (ii), we know that both **E₂** and **E₃** have type T so therefore $\vdash$ **E'** : T as required.

In case (i) we apply the inductive hypothesis to **E₁**. We know $\vdash$ **E₁** : Bool

and by induction, we see that $\vdash$ **E₁'** : Bool also. Hence $\vdash$ **`if E₁' then E₂ else E₃`** : T as required.

NEXT LECTURE: STRUCTURED TYPES