Discussion 20

Type Checking

Kenneth Fang (kwf37), Newton Ni (cn279)

April 15, 2019

Agenda

1. Preview Type Checking Relation
2. Exercises
3. Midterm Projects

Type Checking Relation

The type checking relation defines what programs are “good” and what programs are “bad”

Type Checking Relation

The type checking relation defines what programs are "good" and what programs are "bad"

- We say *e* is well-typed if

```
T |- e : t
```

Type Checking Relation

The type checking relation defines what programs are "good" and what programs are "bad"

- We say *e* is well-typed if

```
T |- e : t
```

- *T* is the typing context (sometimes called Γ)
  - It is a map from variable names → types
  - A lot like the environment we saw when implementing an environment-model interpreter

Type Checking Relation

The type checking relation defines what programs are “good” and what programs are “bad”

- We say *e* is well-typed if

```
T |- e : t
```

- *T* is the typing context (sometimes called Γ)
  - It is a map from variable names → types
  - A lot like the environment we saw when implementing an environment-model interpreter
- *e* is the expression pause
- *t* is the type of the expression (sometimes called τ)

Type Checking Relation

The type checking relation defines what programs are "good" and what programs are "bad"

- We say *e* is well-typed if

```
T |- e : t
```

- *T* is the typing context (sometimes called Γ)
  - It is a map from variable names → types
  - A lot like the environment we saw when implementing an environment-model interpreter
- *e* is the expression pause
- *t* is the type of the expression (sometimes called τ)
- Read as "Expression e has type t under context T"

Static Semantics: Integer Addition

Suppose we have a Bool Type and an Int Type- here's how we can define the type relation for addition:

```
T |- e1 + e2 : int
     if  ???
```

Static Semantics

Static Semantics: Integer Addition

Suppose we have a Bool Type and an Int Type- here's how we can define the type relation for addition:

```
T |- e1 + e2 : int
     if  T |- e1 : int
     and T |- e2 : int
```

Static Semantics

```
<env, e1 + e2> => v
  if <env, e1> => v1
  and <env, e2> => v2
  and v1 + v2 = i
```

Dynamic Semantics
(Environment Model)

Static Semantics: Let Expressions

Suppose we have a Bool Type and an Int Type- here's how we can define the type relation for let expressions:

```
T |- let x = e1 in e2 : t
  if ???
```

Static Semantics

```
<env, let x = e1 in e2> => v
  if <env, e1> -->* v1
  and <env[x->v1], e2> => v
```

Dynamic Semantics
(Environment Model)

Static Semantics: Let Expressions

Suppose we have a Bool Type and an Int Type- here's how we can define the type relation for let expressions:

```
T |- let x = e1 in e2 : t
  if T |- e1 : t1
  and T[x->t1] |- e2 : t
```

Static Semantics

```
<env, let x = e1 in e2> => v
  if <env, e1> -->* v1
  and <env[x->v1], e2> => v
```

Dynamic Semantics
(Environment Model)

Static Semantics: If-Then-Else

Suppose we have a Bool Type and an Int Type- here's how we can define the type relation for if statements:

```
T |- if e1 then e2 else e3 : t
  if ???
```

Static Semantics: If-Then-Else

Suppose we have a Bool Type and an Int Type- here's how we can define the type relation for if statements:

```
T |- if e1 then e2 else e3 : t
  if T |- e1 : bool
  and e2 : t
  and e2 : t
```

Type Soundness

What does it mean for a program to be good?

Type Soundness

What does it mean for a program to be good?

Usually we want these two super-useful properties:

- **Progress:** if **e:t**, then **e** is a value or can take a step
- **Preservation:** if **e:t** and **e**→ **e'**, then **e':t**

Type Soundness

What does it mean for a program to be good?

Usually we want these two super-useful properties:

- **Progress:** Well-typed programs always run to completion
- **Preservation:** Evaluation does not change the type of an expression

Type Soundness

What does it mean for a program to be good?

Usually we want these two super-useful properties:

- **Progress:** Well-typed programs always run to completion
- **Preservation:** Evaluation does not change the type of an expression

Type Soundness

What does it mean for a program to be good?

Usually we want these two super-useful properties:

- **Progress:** Well-typed programs always run to completion
- **Preservation:** Evaluation does not change the type of an expression

If these two properties hold for a type system, we say that type system is "sound"

Type Soundness: Example

Here's an unsound example:

```
T |- if e1 then e2 else e3 : t2
     if  T |- e1 : bool
     and T |- e2 : t2
     and T |- e3 : t3
```

Does this violate Progress or Preservation (or neither)?

- **Progress:** Well-typed programs always run to completion
- **Preservation:** Evaluation does not change the type of an expression