Compiling Techniques

Lecture 10: Type Analysis

Types and Type System

A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.

Benjamin Pierce Types and Programming Languages

Types and Type System

We operate on the AST

A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.

Benjamin Pierce Types and Programming Languages

Types and Type System

We operate on the AST

Detecting of Semantic Errors

A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.

Benjamin Pierce Types and Programming Languages

Types and Type System

We operate on the AST

Detecting of Semantic Errors

A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.

Benjamin Pierce Types and Programming Languages

That's what we call ***Types***

ChocoPy Type System

- ChocoPy is a *statically typed* language
  - We verify that programs are well-typed by analyzing the program's syntax without executing it
- ChocoPy is a *strongly typed* language
  - Programs with type errors are rejected and there are no implicit type conversions
- ChocoPy has *subtyping*
  - Types form a hierarchy and a value of a subtype can safely be used in a context where a value of the supertype is expected

Types of ChocoPy

```
…
type := type_name | `[` type `]`
type_name := int | bool | str | object
…
```

- The grammar of ChocoPy contains the syntax of the following types:
  - int - representing integer values
  - `bool` - representing the two values `True` and `False`
  - `str` - representing strings
  - `object` - the *top type*, i.e., every value has this type
  - [T] - representing a list with elements of type T, where T is itself a type

- ChocoPy defines three more types that cannot be written by the user:
  - `<Empty>` - representing an empty list
  - `<None>` - representing the value `None`
  - ⊥ - the *bottom type*, i.e., the type that has no value

Type Hierarchy of ChocoPy

- The types form a hierarchy:

```
                        object
                           |
   ┌────────┬───────┬────────┼──────┬─────────┬─────────┐
   |        |       |        |      |         |         |
  int     bool     str    [int]  …  [[bool]]  <Empty>  <None>
   |        |       |        |      |         |         |
   └────────┴───────┴────────┼──────┴─────────┴─────────┘
                           |
                           ⊥
```

- The type hierarchy is precisely defining by a *subtyping relationship* (≤), where:
  - `T ≤ T` for all types `T`
  - `T ≤ object` for all types `T`
  - `⊥ ≤ T` for all types `T`
  - If none of the three cases above apply, then the types are not related by subtyping, for example: `[int]` and `[bool]` are not related by subtyping

Type Checking

- The type checking process verifies and enforces the type system

- The type system is defined by a set of formal typing rules that describe under what conditions a syntactic construct is well-typed (*“has a valid type”*)

- To perform type checking, we process a syntactically well-formed program and apply the typing rules to check if we can justify that every definition, statement, and expression is well-typed

Typing rules are inference rules

- A typing rule is a form of a logical inference rule

- We write a typing rule like this:

$$\frac{\vdots}{O \vdash e : T}\ [\mathrm{NAME}]$$

- Each typing rule contains:
    - a `[NAME]`,
    - zero, one, or multiple *premises* above the line,
    - a *conclusion* below the line.

- The rule states, that if the premises are true, then the conclusion is true as well.
  In other words: to check the conclusion, we must check that all premises are true

Typing Judgement

- `O ⊢ e : T` is a *typing judgement*, where the turnstile symbol (⊢) separates the *typing environment* on the left from the proposition on the right.

- This judgement should be read as:

  "In the type environment O the expression e is well typed and has type T"

- Why do we need a typing environment?
- Why can't we just say: *"The expression e is well typed and has type T"*?

Typing Environment

- Consider: `⊢ x + 4 : int` Is this a valid typing judgement?

- We cannot say without knowing the type of `x`!
  - If `x` has type `int`, then the judgement seems valid
  - If `x` has not the type `int`, then the judgement seems wrong

- The *typing environment* records the type of all variables and functions that are in scope when type checking a definition, statement, or expression

- Given a typing environment, we can always conclude if a typing judgement is valid:

  `{x: int} ⊢ x + 4 : int`

Typing Environment of ChocoPy

- For type checking ChocoPy, we use a *local environment* O that contains:
  - The types of all variables in scope
    We write `O(v) = T` to indicate that variable `v` is in the local environment and has type `T`

  - Information about all functions in scope
    We write $O(f) = \{T_1 \times \dots \times T_n \rightarrow T_0; x_1, \dots, x_n; v_1: T'_1, \dots, v_m: T'_m\}$
    to indicate that function `f` is in the local environment and
    - has a function type with
      - $T_1, \dots, T_n$ the types of the function parameters
      - T_0 the function return type
    - has function parameters with names $x_1, \dots, x_n$
    - has identifiers and types $v_1: T'_1, \dots, v_m: T'_m$ of variables declared in the body of `f`
- We also record the return type R of the current function in the environment

First ChocoPy Typing Rules

$$\frac{}{O, R \vdash \texttt{False} : \texttt{bool}} \quad \texttt{[BOOL-FALSE]}$$

First ChocoPy Typing Rules

```
______________________________ [BOOL-FALSE]
O, R ⊢ False : bool
```

"There is no premise that must be true, so we can directly conclude that in the type environment `O` and `R` the expression **`False`** *is well typed and has type* **`bool`***"*

First ChocoPy Typing Rules

```
———————————————————— [BOOL-FALSE]
 O, R ⊢ False : bool
```

"There is no premise that must be true, so we can directly conclude that in the type environment `O` *and* `R` *the expression* **`False`** *is well typed and has type* **`bool`***"*

```
———————————————————— [BOOL-TRUE]
 O, R ⊢ True : bool
```

First ChocoPy Typing Rules

$$\frac{}{O, R \vdash \texttt{False} : \texttt{bool}} \text{[BOOL-FALSE]}$$

"There is no premise that must be true, so we can directly conclude that in the type environment O and R the expression **`False`** *is well typed and has type* **`bool`***"*

$$\frac{}{O, R \vdash \texttt{True} : \texttt{bool}} \text{[BOOL-TRUE]}$$

"There is no premise that must be true, so we can directly conclude that in the type environment O and R the expression **`True`** *is well typed and has type* **`bool`***"*

First ChocoPy Typing Rules

$$\frac{O, R \vdash e_1 : bool \qquad O, R \vdash e_2 : bool}{O, R \vdash e_1 \text{ and } e_2 : bool} \text{[AND]}$$

"If e_1 has type **bool** in the type environment O and R, and
if e_2 has type **bool** in the same type environment O and R,
then we can conclude that
in the same type environment O and R
the expression e_1 **and** e_2 *is well typed and has type* **bool***"*

Example of Type Checking

```
────────────────────────────────────────────[?]
O, R ⊢ False and (True and False) : bool
```

```
──────────────────── [BOOL-FALSE]
O, R ⊢ False : bool
```

```
──────────────────── [BOOL-TRUE]
O, R ⊢ True : bool
```

```
O, R ⊢ e₁ : bool
O, R ⊢ e₂ : bool
──────────────────────── [AND]
O, R ⊢ e₁ and e₂ : bool
```

Example of Type Checking

```
O, R ⊢ False : bool

O, R ⊢ True and False : bool
──────────────────────────────────────────── [AND]
O, R ⊢ False and (True and False) : bool
```

```
──────────────────── [BOOL-FALSE]
O, R ⊢ False : bool
```

```
──────────────────── [BOOL-TRUE]
O, R ⊢ True : bool
```

```
O, R ⊢ e₁ : bool
O, R ⊢ e₂ : bool
──────────────────────── [AND]
O, R ⊢ e₁ and e₂ : bool
```

Example of Type Checking

```
O, R ⊢ False : bool

                  O, R ⊢ True and False : bool
------------------------------------------------ [AND]
O, R ⊢ False and (True and False) : bool
```

```
------------------------ [BOOL-FALSE]
O, R ⊢ False : bool
```

```
------------------------ [BOOL-TRUE]
O, R ⊢ True : bool
```

```
O, R ⊢ e₁ : bool
O, R ⊢ e₂ : bool
------------------------------ [AND]
O, R ⊢ e₁ and e₂ : bool
```

Example of Type Checking

```
──────────────────────── [?]
O, R ⊢ False : bool

                          ──────────────────────────────────── [?]
                          O, R ⊢ True and False : bool
─────────────────────────────────────────────────────────────── [AND]
O, R ⊢ False and (True and False) : bool
```

```
───────────────────────── [BOOL-FALSE]
O, R ⊢ False : bool
```

```
───────────────────────── [BOOL-TRUE]
O, R ⊢ True : bool
```

```
O, R ⊢ e₁ : bool
O, R ⊢ e₂ : bool
─────────────────────────────── [AND]
O, R ⊢ e₁ and e₂ : bool
```

Example of Type Checking

$$\dfrac{\dfrac{}{O, R \vdash \texttt{False} : \texttt{bool}}[\text{BOOL-FALSE}] \qquad \dfrac{}{O, R \vdash \texttt{True and False} : \texttt{bool}}[?]}{O, R \vdash \texttt{False and (True and False)} : \texttt{bool}}[\text{AND}]$$

$$\dfrac{}{O, R \vdash \texttt{False} : \texttt{bool}}[\text{BOOL-FALSE}]$$

$$\dfrac{}{O, R \vdash \texttt{True} : \texttt{bool}}[\text{BOOL-TRUE}]$$

$$\dfrac{O, R \vdash e_1 : \texttt{bool} \quad O, R \vdash e_2 : \texttt{bool}}{O, R \vdash e_1 \texttt{ and } e_2 : \texttt{bool}}[\text{AND}]$$

Example of Type Checking

```
———————————————————[BOOL-FALSE]
 O, R ⊢ False : bool

                    O, R ⊢ True  : bool

                    O, R ⊢ False : bool
                    ——————————————————————————————[AND]
                    O, R ⊢ True and False : bool
——————————————————————————————————————————————————— [AND]
 O, R ⊢ False and (True and False) : bool
```

```
——————————————————————— [BOOL-FALSE]
 O, R ⊢ False : bool
```

```
——————————————————————— [BOOL-TRUE]
 O, R ⊢ True : bool
```

$O, R \vdash e_1 : bool$
$O, R \vdash e_2 : bool$
——————————————————— [AND]
$O, R \vdash e_1 \text{ and } e_2 : bool$

Example of Type Checking

```
———————————————————[BOOL-FALSE]
 O, R ⊢ False : bool

                     O, R ⊢ True  : bool

                                O, R ⊢ False : bool
                     ——————————————————————————————[AND]
                     O, R ⊢ True and False : bool
——————————————————————————————————————————————————— [AND]
 O, R ⊢ False and (True and False) : bool
```

```
———————————————————— [BOOL-FALSE]
 O, R ⊢ False : bool
```

```
———————————————————— [BOOL-TRUE]
 O, R ⊢ True : bool
```

```
 O, R ⊢ e₁ : bool
 O, R ⊢ e₂ : bool
————————————————————————— [AND]
 O, R ⊢ e₁ and e₂ : bool
```

Example of Type Checking

```
──────────────────────[BOOL-FALSE]
 O, R ⊢ False : bool

                 ──────────────────────[?]
                      O, R ⊢ True   : bool

                                  ─────────────[?]
                                   O, R ⊢ False : bool
                     ─────────────────────────────────[AND]
                      O, R ⊢ True and False : bool
──────────────────────────────────────────────────────── [AND]
 O, R ⊢ False and (True and False) : bool
```

$$\frac{}{O,\ R \vdash \texttt{False} : \texttt{bool}}\ [\text{BOOL-FALSE}]$$

$$\frac{}{O,\ R \vdash \texttt{True} : \texttt{bool}}\ [\text{BOOL-TRUE}]$$

$$\frac{O,\ R \vdash e_1 : \texttt{bool} \qquad O,\ R \vdash e_2 : \texttt{bool}}{O,\ R \vdash e_1 \texttt{ and } e_2 : \texttt{bool}}\ [\text{AND}]$$

Example of Type Checking

```
─────────────────────────[BOOL-FALSE]
 O, R ⊢ False : bool

                     ─────────────────────────[BOOL-TRUE]
                           O, R ⊢ True   : bool

                                          ───────────────[?]
                                           O, R ⊢ False : bool
                              ─────────────────────────────────────[AND]
                              O, R ⊢ True and False : bool
───────────────────────────────────────────────────────────────── [AND]
 O, R ⊢ False and (True and False) : bool
```

```
─────────────────────────  [BOOL-FALSE]
 O, R ⊢ False : bool
```

```
─────────────────────────  [BOOL-TRUE]
 O, R ⊢ True : bool
```

```
 O, R ⊢ e₁ : bool
 O, R ⊢ e₂ : bool
───────────────────────────────  [AND]
 O, R ⊢ e₁ and e₂ : bool
```

Example of Type Checking

```
───────────────────────[BOOL-FALSE]
 O, R ⊢ False : bool

                   ────────────────────────[BOOL-TRUE]
                        O, R ⊢ True   : bool

                                     ───────────────[BOOL-FALSE]
                                       O, R ⊢ False : bool
                        ─────────────────────────────────────[AND]
                        O, R ⊢ True and False : bool
──────────────────────────────────────────────────────────────[AND]
 O, R ⊢ False and (True and False) : bool
```

```
────────────────────────── [BOOL-FALSE]
 O, R ⊢ False : bool
```

```
────────────────────────── [BOOL-TRUE]
 O, R ⊢ True : bool
```

```
 O, R ⊢ e₁ : bool
 O, R ⊢ e₂ : bool
──────────────────────────── [AND]
 O, R ⊢ e₁ and e₂ : bool
```