

Compiling Techniques

Lecture 11: Type Analysis (Part 2)

First ChocoPy Typing Rule that uses the Environment

$$\frac{O(\text{id}) = T; \text{ where } T \text{ is not a function type.}}{O, R \vdash \text{id} : T} \quad [\text{VAR-READ}]$$

*“If the variable **id** is in the type environment O with type T , and T is not a function type then we can conclude that in the same type environment O and R the expression **id** is well typed and has type T ”*

Example of Type Checking with Environment

$O(\text{id}) = T$; where T is not
a function type.
————— [VAR-READ]
 $O, R \vdash \text{id} : T$

$O, R \vdash e : \text{int}$
————— [NEGATE]
 $O, R \vdash -e : \text{int}$

————— [?]
 $\{x: \text{int}\}, R \vdash -x : \text{int}$

Example of Type Checking with Environment

$O(id) = T$; where T is not
a function type.

————— [VAR-READ]

$O, R \vdash id : T$

$O, R \vdash e : int$

————— [NEGATE]

$O, R \vdash -e : int$

————— [?]
 $\{x: int\}, R \vdash x : int$

————— [NEGATE]
 $\{x: int\}, R \vdash -x : int$

Example of Type Checking with Environment

$\{x: \text{int}\}(x) = \text{int}$; where int is not
a function type

_____ [VAR-READ]
 $\{x: \text{int}\}, R \vdash x : \text{int}$

_____ [NEGATE]
 $\{x: \text{int}\}, R \vdash -x : \text{int}$

$O(\text{id}) = T$; where T is not
a function type.

_____ [VAR-READ]
 $O, R \vdash \text{id} : T$

$O, R \vdash e : \text{int}$

_____ [NEGATE]
 $O, R \vdash -e : \text{int}$

Another ChocoPy Typing Rule that uses the Environment

$$\frac{\begin{array}{l} O(id) = T \\ O, R \vdash e_1 : T_1 \\ T_1 \leq_a T \end{array}}{O, R \vdash id = e_1} \quad [\text{VAR-ASSIGN-STMT}]$$

Another ChocoPy Typing Rule that uses the Environment

$$\frac{\begin{array}{l} O(id) = T \\ O, R \vdash e_1 : T_1 \\ T_1 \leq_a T \end{array}}{O, R \vdash id = e_1} \quad [\text{VAR-ASSIGN-STMT}]$$

What is this?

Assignment compatibility

- Besides the subtyping relationship, ChocoPy introduces another relation between two types: *assignment compatibility* (\leq_a)
- The idea is that we may assign a value of type T_1 to something of type T_2 iff T_1 is assignment compatible with T_2
- $T_1 \leq_a T_2$, iff at least one of the following is true:
 - $T_1 \leq T_2$ (i.e., T_1 is a subtype of T_2)
 - T_1 is $\langle \text{None} \rangle$ and T_2 is not `int`, `bool`, or `str`
 - T_2 is a list type $[T]$ and T_1 is $\langle \text{Empty} \rangle$
 - T_2 is a list type $[T]$ and T_1 is $[\langle \text{None} \rangle]$, where $\langle \text{None} \rangle \leq_a T$

Another ChocoPy Typing Rule that uses the Environment

$$\frac{\begin{array}{l} \mathcal{O}(\text{id}) = T \\ \mathcal{O}, R \vdash e_1 : T_1 \\ T_1 \leq_a T \end{array}}{\mathcal{O}, R \vdash \text{id} = e_1} \quad [\text{VAR-ASSIGN-STMT}]$$

*“If the variable **id** is in the type environment \mathcal{O} with type T , and expression e_1 has type T_1 in the same type environment \mathcal{O} and R , and T_1 is assignment compatible with T , then we can conclude that in the same type environment \mathcal{O} and R the expression $\text{id} = e_1$ is well typed”*

Note: we are checking a statement that has no type!

ChocoPy Typing Rule for Conditional Expressions

$$\frac{\begin{array}{l} 0, R \vdash e_0 : \text{bool} \\ 0, R \vdash e_1 : T_1 \\ 0, R \vdash e_2 : T_2 \end{array}}{0, R \vdash e_1 \text{ **if** } e_0 \text{ **else** } e_2 : T_1 \sqcup T_2} \text{ [COND]}$$

ChocoPy Typing Rule for Conditional Expressions

$$\frac{\begin{array}{l} 0, R \vdash e_0 : \text{bool} \\ 0, R \vdash e_1 : T_1 \\ 0, R \vdash e_2 : T_2 \end{array}}{0, R \vdash e_1 \text{ if } e_0 \text{ else } e_2 : T_1 \sqcup T_2} \quad [\text{COND}]$$

What is this?

Join of Types

- Sometimes (e.g, when type checking a conditional expression), we need to find a single type that can be used to represent the two original types. For this, we define the *join* operator
- The join of two types T_1 and T_2 (written as $T_1 \sqcup T_2$) is:
 - T_2 if $T_1 \leq_a T_2$
 - T_1 if $T_2 \leq_a T_1$
 - object otherwise, as it is the *least common ancestor* of T_1 and T_2

ChocoPy Typing Rule for Conditional Expressions

$$\frac{\begin{array}{l} O, R \vdash e_0 : \text{bool} \\ O, R \vdash e_1 : T_1 \\ O, R \vdash e_2 : T_2 \end{array}}{O, R \vdash e_1 \text{ if } e_0 \text{ else } e_2 : T_1 \sqcup T_2} \quad [\text{COND}]$$

*“If the expression e_0 has type **bool** in the type environment O and R , and the expression e_1 has type T_1 in the same type environment O and R , and the expression e_2 has type T_2 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 **if** e_0 **else** e_2 is well typed and has type $T_1 \sqcup T_2$.”*

Example of Type Checking for Conditional Expressions

```
O, R ⊢ [True] if True else [] : [bool] ⊔ <Empty>
```

[bool]

object

$T_1 \leq_a T_2$

- $T_1 \leq T_2$ (i.e., T_1 is a subtype of T_2)
- T_1 is <None> and T_2 is not int, bool, or str
- T_2 is a list type [T] and T_1 is <Empty>
- T_2 is a list type [T] and T_1 is [<None>], where <None> \leq_a T

Example of Type Checking for Conditional Expressions

```
O, R ⊢ [True] if True else [] : [bool] ⊔ <Empty>
```

[bool]

object

```
O, R ⊢ [True] if True else None : [bool] ⊔ <None>
```

[bool]

object

$T_1 \leq_a T_2$

- $T_1 \leq T_2$ (i.e., T_1 is a subtype of T_2)
- T_1 is <None> and T_2 is not int, bool, or str
- T_2 is a list type [T] and T_1 is <Empty>
- T_2 is a list type [T] and T_1 is [<None>], where <None> \leq_a T

Example of Type Checking for Conditional Expressions

```
O, R ⊢ [True] if True else [] : [bool] ⊔ <Empty>
```

[bool]

object

```
O, R ⊢ [True] if True else None : [bool] ⊔ <None>
```

[bool]

object

```
O, R ⊢ [True] if True else [None] : [bool] ⊔ [<None>]
```

[bool]

object

$T_1 \leq_a T_2$

- $T_1 \leq T_2$ (i.e., T_1 is a subtype of T_2)
- T_1 is <None> and T_2 is not int, bool, or str
- T_2 is a list type [T] and T_1 is <Empty>
- T_2 is a list type [T] and T_1 is [<None>], where <None> \leq_a T

Example of Type Checking for Conditional Expressions

```
O, R ⊢ [True] if True else [] : [bool] ⊔ <Empty>
```

[bool]

object

```
O, R ⊢ [True] if True else None : [bool] ⊔ <None>
```

[bool]

object

```
O, R ⊢ [True] if True else [None] : [bool] ⊔ [<None>]
```

[bool]

object

$T_1 \leq_a T_2$

- $T_1 \leq T_2$ (i.e., T_1 is a subtype of T_2)
- T_1 is $\langle \text{None} \rangle$ and T_2 is not `int`, `bool`, or `str`
- T_2 is a list type $[T]$ and T_1 is $\langle \text{Empty} \rangle$
- T_2 is a list type $[T]$ and T_1 is $[\langle \text{None} \rangle]$, where $\langle \text{None} \rangle \leq_a T$

ChocoPy Function Definition Typing Rule

$T = T_o$ if return type is present, $\langle \text{None} \rangle$ otherwise

$O(f) = \{T_1 \times \dots \times T_n \rightarrow T; x_1, \dots, x_n; v_1: T'_1, \dots, v_m: T'_m\}$

$O[T_1/x_1] \dots [T_n/x_n][T'_1/v_1] \dots [T'_m/v_m], T \vdash b$

[FUNC-DEF]

$O, R \vdash \text{def } f(x_1: T_1, \dots, x_n: T_n) [\Rightarrow T_o]? : b$

ChocoPy Function Definition Typing Rule

1. Set T to be the return type, or $\langle \text{None} \rangle$

$T = T_o$ if return type is present, $\langle \text{None} \rangle$ otherwise

$O(f) = \{T_1 \times \dots \times T_n \rightarrow T; x_1, \dots, x_n; v_1: T'_1, \dots, v_m: T'_m\}$

$O[T_1/x_1] \dots [T_n/x_n][T'_1/v_1] \dots [T'_m/v_m], T \vdash b$

[FUNC-DEF]

$O, R \vdash \text{def } f(x_1: T_1, \dots, x_n: T_n) [\Rightarrow T_o]? : b$

ChocoPy Function Definition Typing Rule

1. Set T to be the return type, or $\langle \text{None} \rangle$

2. Get information about f from the environment

$T = T_o$ if return type is present, $\langle \text{None} \rangle$ otherwise

$O(f) = \{T_1 \times \dots \times T_n \rightarrow T; x_1, \dots, x_n; v_1: T'_1, \dots, v_m: T'_m\}$

$O[T_1/x_1] \dots [T_n/x_n][T'_1/v_1] \dots [T'_m/v_m], T \vdash b$

[FUNC-DEF]

$O, R \vdash \text{def } f(x_1: T_1, \dots, x_n: T_n) [\Rightarrow T_o]? : b$

ChocoPy Function Definition Typing Rule

1. Set \mathbf{T} to be the return type, or $\langle \mathbf{None} \rangle$

2. Get information about \mathbf{f} from the environment

$\mathbf{T} = \mathbf{T}_o$ if return type is present, $\langle \mathbf{None} \rangle$ otherwise

$\mathcal{O}(\mathbf{f}) = \{\mathbf{T}_1 \times \dots \times \mathbf{T}_n \rightarrow \mathbf{T}; \mathbf{x}_1, \dots, \mathbf{x}_n; \mathbf{v}_1: \mathbf{T}'_1, \dots, \mathbf{v}_m: \mathbf{T}'_m\}$

$\mathcal{O}[\mathbf{T}_1/\mathbf{x}_1] \dots [\mathbf{T}_n/\mathbf{x}_n][\mathbf{T}'_1/\mathbf{v}_1] \dots [\mathbf{T}'_m/\mathbf{v}_m], \mathbf{T} \vdash \mathbf{b}$

[FUNC-DEF]

$\mathcal{O}, \mathbf{R} \vdash \text{def } \mathbf{f}(\mathbf{x}_1: \mathbf{T}_1, \dots, \mathbf{x}_n: \mathbf{T}_n) \llbracket \rightarrow \mathbf{T}_o \rrbracket? : \mathbf{b}$

3. Type check function body \mathbf{b} with an adjusted environment, where

- \mathbf{x}_i has type \mathbf{T}_i and \mathbf{v}_i has type \mathbf{T}'_i (notation: $\mathcal{O}[\mathbf{T}/\mathbf{c}](\mathbf{c}) = \mathbf{T}$; $\mathcal{O}[\mathbf{T}/\mathbf{c}](\mathbf{d}) = \mathcal{O}(\mathbf{d})$ if $\mathbf{d} \neq \mathbf{c}$)
- \mathbf{T} is used instead of \mathbf{R}

Implementing ChocoPy Typing Rules

Basic idea

- Implement one Python function for each typing rule, e.g.:

```
# [NEGATE] rule
#  $O, R, \vdash -e : \text{int}$ 
def negate_rule(o: LocalEnvironment, r: Type, e: Operation) → Type:
    #  $O, R, \vdash e : \text{int}$ 
    check_type(check_expr(o, r, e), expected=int_type)
    return int_type
```

$$\frac{O, R \vdash e : \text{int}}{O, R \vdash -e : \text{int}} \quad [\text{NEGATE}]$$

- Have a *dispatch* function that decides which typing rule to invoke.

Implementing dispatch function

Basic idea

- Implement one Python function for each typing rule.
- Have a *dispatch* function that decides which typing rule to invoke:

```
def check_expr(o: LocalEnvironment, r: Type, op: Operation) → Type:
    if isinstance(op, choco_ast.UnaryExpr):
        unary_expr = op
        op = unary_expr.op.data
        e = unary_expr.value.blocks[0].ops[0]
        if op == "-":
            return negate_rule(o, r, e)
        else:
            raise Exception("Not implemented yet")
    else:
        raise Exception("Not implemented yet")
```

Dispatch of Typing Rules

- There are three different dispatch functions:
 - `def check_stmt_or_def_list(o, r, ops: List[Operation])` for list of statements and definitions
 - `def check_stmt_or_def(o, r, op: Operation)` for statements and definitions
 - `def check_expr(o, r, op: Operation) → Type` for expressions
- Challenge:

The syntax alone is not always enough to decide which typing rule to invoke!

$$\frac{\begin{array}{l} 0, R \vdash e_1 : \text{int} \\ 0, R \vdash e_2 : \text{int} \\ \text{op} \in \{+, -, *, //, \%\} \end{array}}{\quad} \text{[ARITH]}$$
$$0, R \vdash e_1 \text{ op } e_2 : \text{int}$$
$$\frac{\begin{array}{l} 0, R \vdash e_1 : \text{str} \\ 0, R \vdash e_2 : \text{str} \end{array}}{\quad} \text{[STR-CONCAT]}$$
$$0, R \vdash e_1 + e_2 : \text{str}$$

To decide which rule to invoke, I need to know the type of **e1** or **e2**!