CSC2/455 Software Analysis and Improvement
Type Inference - II

Sreepathi Pai
March 20, 2024
URCS
Type Systems for Realistic Languages

Type Checking ChocoPy

Postscript
Type Systems for Realistic Languages

Type Checking ChocoPy

Postscript
The type system we studied in the last class was for a functional language

- Programs consist of expressions
- All well-formed expressions have a type

Some languages, notably imperative languages (Python, C, etc.), differentiate between statements and expressions

- Expressions have types
- Statements do not have types
- “Procedures” do not have return values
Additional judgements for imperative languages

- $\Gamma \vdash C$
  - $C$ is a well-formed command (or statement) in $\Gamma$

In general, block-structured languages may have $C$ be a block as well.
Subtyping

- Most languages have a subtype relation
  - C has \textit{unsigned short} is a subtype of \textit{unsigned int}
  - Subtypes can be used wherever the ”supertype” is used
- Usually denoted by $A \leq B$
  - $A$ is a subtype of $B$
  - $A \leq A$, obviously
- Type hierarchy, a tree of types
  - Usually \texttt{object} as the root
  - A edge from $B$ to $A$ if $A < B$
Type Systems for Realistic Languages

Type Checking ChocoPy

Postscript
ChocoPy is a simpler version of Python3
  - Developed as a “toy” language for compiler courses
- Fully-specified
  - Syntax (BNF)
  - Type System
  - Semantics (Operational Semantics)
- We will only look at its Type System
ChocoPy Types

- Basic types: object, int, bool, str
- List type: [T], where T is another type
  - e.g. [int]
  - T be a list as well, this is an example of a recursive type
  - [[[int]]]
- Special types
  - <None>
  - <Empty>, indicating an empty list []
ChocoPy Type Hierarchy

- $A \leq A$, for all types $A$
- $C \leq A$ if $C$ is a subclass of $A$
  - int, bool, str are all subclasses of object
- $[T] \leq \text{object}$
  - but no relationship between two different list types
- $\langle \text{None} \rangle \leq \text{object}$
- $\langle \text{Empty} \rangle \leq \text{object}$
A type $T_1$ is assignment compatible ($\leq_a$) to $T_2$ iff values of type $T_1$ can be used wherever values of $T_2$ are expected.

- $T_1 \leq T_2$ (note: this is subtyping $\leq$)
- $T_1$ cannot be $<$None$>$ if $T_2$ is int, bool or str
- $T_1$ can be $<$Empty$>$ only if $T_2$ is $[T]$
- $T_1$ can be $[<$None$>]$ if $T_2$ is $[T]$, where $<$None$> \leq_a T$
  - Note that means $T$ can only be object ($*$)
The join \( \sqcup \) of two types \( A \) and \( B \) denoted as \( A \sqcup B \) is defined as:

- \( B \) if \( A \leq_a B \)
  - This is commutative, so \( B \sqcup A \) is also \( B \) in this case
- \( C \) where \( C \) is the least common ancestor of \( A \) and \( B \) in the type hierarchy defined by \( \leq \)
  - So \( \text{int} \sqcup \text{str} \) is \text{object}
  - Note this is the \textit{least upper bound} or LUB (analogous to GLB in meet lattices)
Type Environment

- Judgements of the form:
  - \(O, M, C, R \vdash \ldots\)
- Where the type environment consists of:
  - \(O\), “local” environment
  - \(M\), method/attribute environment, i.e. for \(A.x\)-style code
  - Both are maps
- And:
  - \(C\), the name of the current class, can be \(\bot\) if outside class
  - \(R\), the return type of the current function, can be \(\bot\) if outside function
def contains(items:[int], x:int) -> bool:
    i:int = 0
    while i < len(items):
        if items[i] == x:
            return True
        i = i + 1
    return False

The general type for a function $f$ is $O(f) = \{ T_1 \times T_2 \times ... \times T_n \rightarrow T_0 ; x_1, x_2, ..., x_n; v_1 : T'_1, v_2 : T'_2, ..., v'_m : T'_m \}$

- $T_1 \times T_2 \times ... \times T_n \rightarrow T_0$ is the type of arguments and the return type
  - Here, [int] × int → bool
- $x_1, x_2, ..., x_n$ are names of the arguments: items, x
- $v_1 : T'_1, ..., v_m : T'_m$ are types of variables/functions declared inside the function
  - Here i : int
class Point1D:
    x: int = 0

    # the quotes are used since Point1D is being defined ...
    def distance(self: "Point1D", o: "Point1D") -> int:
        ...

- Attributes are simply $M(C, a) = T$, where $C$ is the class, $a$ is the attribute, and $T$ is its type
  - So, $M(\text{Point1D}, x) = \text{int}$
- And methods $M(C, m)$ are just like functions in the previous slide
  - So,
    
    $M(\text{Point1D}, \text{distance}) = \{\text{Point1D} \times \text{Point1D} \to \text{int}; \ldots\}$
Type Judgements in ChocoPy

- $O, M, C, R \vdash e : T$
  - Expression $e$ has type $T$ in the environment $O, M, C, R$
- $O, M, C, R \vdash b$
  - $b$ is a well-formed (block) statement in $O, M, C, R$
  - can also be a single statement, obviously
- If reading the ChocoPy reference, make sure to distinguish the colon in type judgements $e : T$ from the colon in Python syntax
  - I’ll use `:` to indicate the latter, and `:` to indicate the former
Expressions

\[
\begin{align*}
O, M, C, R \vdash \text{True} : \text{bool} & \quad \text{(BOOL-TRUE)} \\
O, M, C, R \vdash \text{False} : \text{bool} & \quad \text{(BOOL-FALSE)} \\
i \text{ is an integer literal} & \quad \text{ (INT) } \\
O, M, C, R \vdash i : \text{int} & \\
s \text{ is a string literal} & \quad \text{ (STR) } \\
O, M, C, R \vdash s : \text{str} & \\
O(id) = T & \quad T \text{ is not a function type} \quad \text{ (VAR-READ) } \\
O, M, C, R \vdash id : T & 
\end{align*}
\]
\[ O(id) = T \quad O, M, C, R \vdash e_1 : T_1 \quad T_1 \leq_a T \]

\[ O, M, C, R \vdash id : T = e_1 \quad (\text{VAR-INIT}) \]
\[ O, M, C, R \vdash s_1 \]
\[ O, M, C, R \vdash s_2 \]
\[ \vdots \]
\[ O, M, C, R \vdash s_n \]
\[ O, M, C, R \vdash s_1 \text{ NEWLINE } s_2 \ldots s_n \text{ NEWLINE} \]
Operators

\[ O, M, C, R \vdash e_1 : int \]
\[ O, M, C, R \vdash e_2 : int \]
\[ \text{op} \in \{+, -, *, \div, \mod\} \]
\[ \begin{array}{c}
\hline
O, M, C, R \vdash e_1 \text{ op } e_2 : int \\
\end{array} \]
\[ (\text{ARITH}) \]

\[ O, M, C, R \vdash e_1 : int \]
\[ O, M, C, R \vdash e_2 : int \]
\[ \sqsubseteq \in \{<, \leq, >, \geq, ==, !=\} \]
\[ \begin{array}{c}
\hline
O, M, C, R \vdash e_1 \sqsubseteq e_2 : bool \\
\end{array} \]
\[ (\text{COMPARE-INT}) \]

\[ O, M, C, R \vdash e_0 : bool \]
\[ O, M, C, R \vdash e_1 : T_1 \]
\[ O, M, C, R \vdash e_2 : T_2 \]
\[ \begin{array}{c}
\hline
O, M, C, R \vdash e_1 \text{ if } e_0 \text{ else } e_2 : T_1 \sqcup T_2 \\
\end{array} \]
\[ (\text{IF-EXPR}) \]
Return and Class Definitions

\[
\frac{O, M, C, R \vdash e : T \quad T \leq_a R}{O, M, C, R \vdash \text{return } e} \quad \text{(RETURN-E)}
\]

\[
\frac{O, M, C, R \vdash b}{O, M, \bot, R \vdash \text{class } C(S) : b} \quad \text{(CLASS-DEF)}
\]
Function Definitions

```python
x: str = "hello, world"

def somefn(x: int, y: str):
    # what is the type of x here?

- Function argument names:
  - shadow names outside the function (like x here)
  - or are introduced in the definition (like y)

- The notation for shadowing or introducing a variable is:
  - $O[T/x]$ which is defined as $O[T/x](x) = T$ and $O[T/x](y) = O(y)$ where $y \neq x$
  - i.e., $O[T/x]$ is a new map with $x$ mapped to $T$ and the other mappings unchanged
  - repeated applications are possible: $O[T_1/x][T_2/y]$
```
\[ T = \begin{cases} 
T_0, & \text{if } \rightarrow \text{ is present,} \\
<\text{None}>, & \text{otherwise.} 
\end{cases} \]

\[ O(f) = \{T_1 \times \cdots \times T_n \rightarrow T; \ x_1, \ldots, x_n; \ v_1 : T'_1, \ldots, v_m : T'_m\} \]

\[ n \geq 0 \quad m \geq 0 \]

\[ O[T_1/x_1] \ldots [T_n/x_n][T'_1/v_1] \ldots [T'_m/v_m], \ M, C, T \vdash b \]

\[ O, M, C, R \vdash \textbf{def } f(x_1:T_1, \ldots, x_n:T_n) \ [\rightarrow T_0]^?: b \]

[\text{FUNC-DEF}]
$$T = \begin{cases} T_0, & \text{if } \to \text{ is present,} \\ \langle \text{None} \rangle, & \text{otherwise.} \end{cases}$$

$$M(C, f) = \{T_1 \times \cdots \times T_n \rightarrow T; \ x_1, \ldots, x_n; \ v_1 : T'_1, \ldots, v_m : T'_m\}$$

$$n \geq 1 \quad m \geq 0$$

$$C = T_1$$

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

$$O, M, C, R \vdash \text{def } f(x_1:T_1, \ldots, x_n:T_n) [\to T_0]:b$$
Outline

Type Systems for Realistic Languages

Type Checking ChocoPy

Postscript
A self-contained introduction to type systems
  - Luca Cardelli, Type Systems, Handbook of Computer Science and Engineering, 2nd Ed
An updated version (available only through the library)
  - Stephanie Weirich, Type Systems, Handbook of Computer Science and Engineering, 3rd Ed
Rohan Padhye, Koushik Sen, Paul Hilfinger, ChocoPy v2.2: Language manual and reference
Leandro TC Melo, Rodrigo G Riberio, Breno CF Guimaraes, Fernando Magno Quintao, Type Inference for C: Applications to the Static Analysis of Incomplete Programs, ACM TOPLAS
  - The psychec tool