CS 476 – Programming Language Design

William Mansky
Midterm 1 Questions

Top
Which of the following are provable with just these rules?

1. \( \{ \} \vdash 5 : \text{int} \)
2. \( \{x : \text{int}, y : \text{bool}\} \vdash y : \text{bool} \)
3. \( \{x : \text{int}, y : \text{bool}\} \vdash x + 5 : \text{int} \)
Which are provable with just the number and boolean rules?

None of them

Just 1

1 and 2

1 and 3

1, 2, and 3
Literals vs. Variables

- **(n is an integer literal)**
  - \[ \Gamma \vdash n : \text{int} \]

- **(b is a boolean literal)**
  - \[ \Gamma \vdash b : \text{bool} \]

1, 2, 3, 4, ...: integer *literals*

true, false: boolean *literals*

x, y, z: (integer- or boolean-valued) *variables*

x + 5, true || y: integer or boolean *expressions*
Imperative Languages

• Arithmetic and boolean expressions
• Variables and assignment
• Control flow (conditionals, loops)
• Variable declarations
• Exceptions and exception handling
• Function declarations and calls
Object-Oriented Imperative Languages

• Arithmetic and boolean expressions
• Variables and assignment
• Control flow (conditionals, loops)
• Variable declarations
• Exceptions and exception handling
• Function declarations and calls
• Objects and classes
Object-Oriented Programming

• An *object* is a kind of value
• Objects have *fields* (object-specific variables) and *methods* (object-specific functions)

```
o x := o.getx();
o.sety(ox);
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>5</td>
</tr>
<tr>
<td>getx()</td>
<td>return x;</td>
</tr>
<tr>
<td>sety(n)</td>
<td>y := n;</td>
</tr>
</tbody>
</table>
Object-Oriented Programming

• An object is a kind of value
• Objects have fields (object-specific variables) and methods (object-specific functions)
• Different objects may provide the same method but have different code for it (dynamic dispatch)

```
o
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>3</td>
</tr>
<tr>
<td>getx()</td>
<td>return x;</td>
</tr>
<tr>
<td>sety(n)</td>
<td>y := n;</td>
</tr>
</tbody>
</table>
```

```
o2
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>4</td>
</tr>
<tr>
<td>theta</td>
<td>pi/4</td>
</tr>
<tr>
<td>getx()</td>
<td>return r * cos(theta);</td>
</tr>
<tr>
<td>sety(n)</td>
<td>...</td>
</tr>
</tbody>
</table>
```

```
o.getx();
o2.getx();
```
Object-Oriented Programming

• An *object* is a kind of value
• Objects have *fields* (object-specific variables) and *methods* (object-specific functions)
• Different objects may provide the same method but have different code for it (dynamic dispatch)
• We can only access a field/method by going through its object
• An object may belong to a *class*, which describes a list of fields and methods the object contains
• Classes may have *subclasses*, which extend them with more fields and methods
Java-Like Language: Classes

class Square extends Shape {
    int side;
    Square(int side){
        super();
        this.side = side;
    }
    int area(){
        return this.side * this.side;
    }
}

Square s = new Square(3);

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>side</td>
<td>3</td>
</tr>
</tbody>
</table>

int x = s.area();
// x will be 9
Java-Like Language: Syntax

\[ CL ::= \text{class } <\text{id}> \text{ extends } <\text{id}> \{ \ T \ <\text{id}>, \ldots, \ T \ <\text{id}>; \ K \ M \ldots \ M \} \]

\[ K ::= <\text{id}>(<T \ <\text{id}>, \ldots, \ T \ <\text{id}>)\{
\]
\[ \text{super(<id>, ..., <id>); this.<id> = <id>; ...; this.<id> = <id>; } \]
\[ M ::= T \ <\text{id}>(<T \ <\text{id}>, \ldots, \ T \ <\text{id}>)\{ \ C \} \]

\[ E ::= \# | E + E | <\text{id}> | E.\text{id} \]

\[ C ::= <\text{id}> = E; | <\text{id}> = E.\text{id}(E, ..., E); | <\text{id}> = \text{new } <\text{id}>(E, ..., E) | C \ C \]

\[ T ::= \text{int} | <\text{id}> \]
Questions
Java-Like Language: Subtyping

class A extends Object {
    int x;
    A(int x){
        super(); this.x = x;
    }
    int getx(){
        return this.x;
    }
}

class B extends A {
    int y;
    B(int x, int y){
        super(x); this.y = y;
    }
    int gety(){
        return this.y;
    }
}
Java-Like Language: Subtyping

(new A(5)).getx();
(new B(5, 6)).getx();

• Anywhere an object of class A is expected, an object of class B should work just as well!

\[ \Gamma \vdash e : \tau_1 \quad \tau_1 \leq : \tau_2 \]
\[ \Gamma \vdash e : \tau_2 \]
How do we know whether class A is a subclass of class B?
Java-Like Language: Subtyping

\[ \Gamma \vdash C <: C \]

\[ (\Gamma(C) = \text{class } C \text{ extends } D \{ \ldots \}) \]

\[ \Gamma \vdash C <: D \]

\[ \Gamma \vdash C <: D \quad \Gamma \vdash D <: E \]

\[ \Gamma \vdash C <: E \]
Java-Like Language: Subtyping

class Shape extends Object { ... }
class Square extends Shape { ... }

\[ \Gamma \vdash C <: D \quad \Gamma \vdash D <: E \]
\[ \Gamma \vdash C <: E \]

\[ \Gamma \vdash \text{Square} <: \text{Object} \]
Java-Like Language: Subtyping

class Shape extends Object { ... }  
class Square extends Shape { ... }  

\[
\Gamma \vdash C <: D \quad \Gamma \vdash D <: E \\
\Gamma \vdash C <: E
\]

\[
\Gamma \vdash \text{Square} <: ? \quad \Gamma \vdash ? <: \text{Object} \\
\Gamma \vdash \text{Square} <: \text{Object}
\]
Java-Like Language: Subtyping

class Shape extends Object { ... }
class Square extends Shape { ... }

(\Gamma(C) = \text{class } C \text{ extends } D \{ ... \})
\quad \Gamma \vdash C <: D

\Gamma \vdash \text{Square } <: ? \quad \Gamma \vdash ? <: \text{Object}
\quad \Gamma \vdash \text{Square } <: \text{Object}
Java-Like Language: Subtyping

class Shape extends Object { ... }
class Square extends Shape { ... }

\[
\frac{(\Gamma(C) = \text{class } C \text{ extends } D \{ ... \})}{\Gamma \vdash C <: D}
\]

\[
\frac{(\Gamma(\text{Square}) = \cdots) \quad (\Gamma(\text{Shape}) = \cdots)}{\Gamma \vdash \text{Square} <: \text{Shape} \quad \Gamma \vdash \text{Shape} <: \text{Object}}
\]

\[
\frac{}{\Gamma \vdash \text{Square} <: \text{Object}}
\]
Questions
Java-Like Language: Types

• Types: int, any class name

• Rules:

  \[
  \frac{\Gamma \vdash e_1 : \text{int}}{\Gamma \vdash \text{new } C(e_1, \ldots, e_n) : \tau}
  \]

  \[
  \frac{\Gamma \vdash e : \tau_1 \quad \tau_1 \lesssim \tau_2}{\Gamma \vdash e : \tau_2}
  \]

  \[
  \frac{\Gamma \vdash e_1 + e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}}
  \]

  \[
  \frac{\Gamma \vdash e : \tau}{\Gamma \vdash e.f : \tau}
  \]

  \[
  \frac{\Gamma \vdash e_1, e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}}
  \]

  \[
  \frac{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}}
  \]

  \[
  \frac{\Gamma \vdash e_1 + e_2 : \text{int}}{\Gamma \vdash e_1 + e_2 : \text{int}}
  \]

  \[
  \frac{\Gamma \vdash x = e.m(e_1, \ldots, e_n) : \text{ok}}{\Gamma \vdash x = e.m(e_1, \ldots, e_n) : \text{ok}}
  \]
Java-Like Language: Types

• Types: int, any class name

\[
\frac{?}{\Gamma \vdash e.f : \tau}
\]

• \(e\) is an object of a class \(C\)
• \(C\) has a field \(f\) of type \(\tau\)

class A extends Object {
    int x;
    ...
}
objA.x : int
Java-Like Language: Types

• Types: int, any class name

\[ \Gamma \vdash e : C \quad (\text{fields}(\Gamma, C) = \cdots, \tau f) \]

\[ \Gamma \vdash e.f : \tau \]

class A extends Object {
    int x;
    ...
}

objA.x : int
Java-Like Language: Types

• Types: int, any class name

\[
\Gamma \vdash \text{new } C(e_1, \ldots e_n) : \tau
\]

class A extends Object {
  int x;
  A(int x){ super(); this.x = x; }
}

new A(5) : ?
Java-Like Language: Types

• Types: int, any class name

\[
? \quad \Gamma \vdash \text{new } C(e_1, ... e_n) : C
\]

• \( e_1, ... e_n \) have the right types

• The arguments to the constructor for \( C \) are exactly the fields of \( C \)

```java
class A extends Object {
    int x;
    A(int x){ super(); this.x = x; }
}
```

new A(5) : A
Java-Like Language: Types

• Types: int, any class name

\[
\text{fields}(\Gamma, C) = \tau_1 \, f_1, \ldots, \tau_n \, f_n \\
\Gamma \vdash e_1 : \tau_1 \quad \ldots \quad \Gamma \vdash e_n : \tau_n \\
\Gamma \vdash \text{new} \ C(e_1, \ldots, e_n) : C
\]

• \(e_1, \ldots, e_n\) have the right types

```java
class A extends Object {
    int x;
    A(int x){ super(); this.x = x; }
}
new A(5) : A
```
Java-Like Language: Types

• Types: int, any class name

\[ \Gamma \vdash x = e. m(e_1, ... e_n); : \text{ ok} \]

• \( e \) is an object of a class \( C \)
• \( C \) has a method \( m \)
• args and return of \( m \) are well typed

```java
class A extends Object {
    int x;
    int getX() { return this.x; }
}
```

\( y = \) objA.getx();
Java-Like Language: Types

• Types: int, any class name

\[ \Gamma \vdash e : C \quad (\text{methods}(\Gamma, C) = \ldots, \tau \ m(\tau_1 \ x_1, \ldots, \tau_n \ x_n)) \]

\[ \Gamma \vdash e_1 : \tau_1 \quad \ldots \quad \Gamma \vdash e_n : \tau_n \quad \Gamma(x) = \tau \]

\[ \Gamma \vdash x = e.\ m(e_1, \ldots e_n);: \text{ok} \]

• \( e \) is an object of a class \( C \)
• \( C \) has a method \( m \)
• args and return of \( m \) are well typed

```java
class A extends Object {
    int x;
    int getx() { return this.x; }
}
```
Questions
Java-Like Language: Declarations

\[
\text{decls}(\Gamma) \cup \{ x_1 : \tau_1, \ldots, x_n : \tau_n, \_\_\_ret : \tau \} \vdash c : \text{ok}
\]
\[
\Gamma \vdash \tau \ m(\tau_1 \ x_1, \ldots, \tau_n \ x_n)\{c\} : \text{ok}
\]

• A method declaration is typed like a function declaration
Java-Like Language: Declarations

\[
\text{decls}(\Gamma) \uplus \{ \text{this} : C, x_1 : \tau_1, \ldots, x_n : \tau_n, _\text{ret} : \tau \} \vdash c : \text{ok}
\]

\[
\Gamma, C \vdash \tau \ m(\tau_1 \ x_1, \ldots, \tau_n \ x_n)\{\ c\ \} : \text{ok}
\]

• A method declaration is typed like a function declaration
• Except that it is called on an object
Java-Like Language: Declarations

\[
\text{decls}(\Gamma) \cup \{\text{this} : C, x_1 : \tau_1, \ldots, x_n : \tau_n, \_\_\text{ret} : \tau\} \vdash c : \text{ok} \\
\Gamma, C \vdash \tau m(\tau_1 x_1, \ldots, \tau_n x_n)\{ c \} : \text{ok}
\]

• A method declaration is typed like a function declaration
• Except that it is called on an object
• And it might be overriding a superclass’s method
Java-Like Language: Overriding

class A extends Object {
    int x;
    A(int x){
        super(); this.x = x;
    }
    int getval(){
        return this.x;
    }
}

class B extends A {
    int y;
    B(int x, int y){
        super(x); this.y = y;
    }
    int getval(){
        return this.x + this.y;
    }
}
Java-Like Language: Declarations

\[
\text{decls}(\Gamma) \cup \{ \text{this} : C, x_1 : \tau_1, \ldots, x_n : \tau_n, \_\text{ret} : \tau \} \vdash c : \text{ok} \\
(\Gamma(C) = \text{class} C \text{ extends } D \{ \ldots \}) \\
\left( \text{override-safe}(\tau m(\tau_1 x_1, \ldots, \tau_n x_n), \Gamma, D) \right) \\
\Gamma, C \vdash \tau m(\tau_1 x_1, \ldots, \tau_n x_n)\{ c \} : \text{ok}
\]

- A method declaration is typed like a function declaration
- Except that it is called on an object
- And it might be overriding a superclass’s method
Java-Like Language: Declarations

\[
\text{decls}(\Gamma) \uplus \{\text{this} : C, x_1 : \tau_1, \ldots, x_n : \tau_n, \_\text{ret} : \tau\} \vdash c : \text{ok}
\]

\[
(\Gamma(C) = \text{class } C \text{ extends } D \{ ... \})
\]

\[
(\text{override safe}(\tau \, m(\tau_1 \, x_1, \ldots, \tau_n \, x_n), \Gamma, D))
\]

\[
\Gamma, C \vdash \tau \, m(\tau_1 \, x_1, \ldots, \tau_n \, x_n)\{ c \} : \text{ok}
\]

• \text{override safe}(\tau \, m(\tau_1 \, x_1, \ldots, \tau_n \, x_n), \Gamma, D)\) means that if a method named \(m\) is in methods(\(\Gamma, D\)), its type signature must be exactly \(\tau(\tau_1, \ldots, \tau_n)\)
Java-Like Language: Declarations

\[
\begin{align*}
\text{fields}(\Gamma, D) &= \tau_1 x_1, \ldots, \tau_m x_m \\
\Gamma(C) &= \text{class } C \text{ extends } D \{ \tau_{(m+1)} x_{m+1}; \ldots; \tau_n x_n; \ldots \} \\
\Gamma, C \vdash C(\tau_1 x_1, \ldots, \tau_n x_n)\{ \text{super}(x_1, \ldots, x_m); \\
& \text{this.} x_{m+1} = x_{m+1}; \ldots \text{this.} x_n = x_n; \} : \text{ok}
\end{align*}
\]

\[
\begin{align*}
\Gamma, C \vdash K : \text{ok} & \quad \Gamma, C \vdash M_1 : \text{ok} \quad \ldots \Gamma, C \vdash M_j : \text{ok} \\
\Gamma \vdash \text{class } C \text{ extends } D \{ \tau_1 f_1; \ldots; \tau_n f_n; K M_1 \ldots M_j \} : \text{ok}
\end{align*}
\]

• A class declaration is well typed if all its methods are well typed and its constructor is correctly defined
Questions
Object-Oriented Programming

• An object is a kind of value

• Objects have fields (object-specific variables) and methods (object-specific functions)

• Different objects may provide the same method but have different code for it (dynamic dispatch)

➢ We can only access a field/method by going through its object

• An object may belong to a class, which describes a list of fields and methods the object contains

• Classes may have subclasses, which extend them with more fields and methods
Java-Like Language: Access Modifiers

\[
\Gamma \vdash e : C \\
(\text{fields}(\Gamma, C) = \cdots, \tau f) \\
\Gamma \vdash e.f : \tau
\]

```java
class B extends A {
    private int y;
    public int gety() {
        return this.y;
    }
    public int addy(B b) {
        this.y = this.y + b.y;
    }
}
```

B b1 = new B(1, 2);
b1.y = 5; // error
Java-Like Language: Access Modifiers

\[ \Gamma \vdash e : C \]
\[ (\text{fields}(\Gamma, C) = \ldots, \text{public } \tau f) \]
\[ \Gamma \vdash e.f : \tau \]

\[ \Gamma, C \vdash e : C \]
\[ (\text{fields}(\Gamma, C) = \ldots, \text{private } \tau f) \]
\[ \Gamma, C \vdash e.f : \tau \]

class B extends A {
    private int y;
    
    public int gety() {
        return this.y;
    }
    
    public int addy(B b) {
        this.y = this.y + b.y;
    }
}
Java-Like Language: Access Modifiers

\[
\begin{align*}
\Gamma, D &\vdash e : C \\
\text{(fields} \Gamma, C) &= \cdots, \text{public } \tau f) \\
\Gamma, D &\vdash e.f : \tau \\
\Gamma, C &\vdash e : C \\
\text{(fields} \Gamma, C) &= \cdots, \text{private } \tau f) \\
\Gamma, C &\vdash e.f : \tau \\
decls(\Gamma) \cup \{\text{this} : C, x_1 : \tau_1, \ldots\} &\vdash c : \text{ok} \\
\Gamma, C &\vdash \tau m(\tau_1 x_1, \ldots)\{ c \} : \text{ok}
\end{align*}
\]

class B extends A {
  private int y;
  
  public int gety() {
    return this.y;
  }
  
  public int addy(B b) {
    this.y = this.y + b.y;
  }
}
Java-Like Language: Access Modifiers

\[
\begin{align*}
\Gamma, D \vdash e : C \\
& \text{(fields}(\Gamma, C) = \ldots, \text{public } \tau \ f) \\
\Gamma, D \vdash e. f : \tau \\
\end{align*}
\]

\[
\begin{align*}
\Gamma, C \vdash e : C \\
& \text{(fields}(\Gamma, C) = \ldots, \text{private } \tau \ f) \\
\Gamma, C \vdash e. f : \tau \\
\end{align*}
\]

decls(\Gamma) \cup \{\text{this} : C, x_1 : \tau_1, \ldots\}, C \vdash c : \text{ok}
\]

class B extends A {
    private int y;
    public int gety(){
        return this.y;
    }
    public int addy(B b){
        this.y = this.y + b.y;
    }
}