Type inference

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming Concepts using Java

Week 8

Type declarations vs type inference

- Java insists that all variables are declared in advance, with type information

```
public class Employee {...}

public class Manager extends Employee {...}

Employee e;

Manager m;
```

Type declarations vs type inference

- Java insists that all variables are declared in advance, with type information
- The compiler can then check whether the program is well-typed

```
public class Employee {...}

public class Manager extends Employee {...}

Employee e;

Manager m;

...

m = new Manager(...);
e = m;  // Allowed by subtyping
```

Type declarations vs type inference

- Java insists that all variables are declared in advance, with type information
- The compiler can then check whether the program is well-typed
- An alternative approach is to do type inference

```
public class Employee {...}

public class Manager extends Employee {...}

Employee e;

Manager m;

...

m = new Manager(...);
e = m;  // Allowed by subtyping
```

Type declarations vs type inference

- Java insists that all variables are declared in advance, with type information
- The compiler can then check whether the program is well-typed
- An alternative approach is to do type inference
- Derive type information from context. For instance, `s` should be `String`

```
s = "Hello, " + "world";
```

```
public class Employee {...}

public class Manager extends Employee {...}

Employee e;

Manager m;

...

m = new Manager(...);
e = m;  // Allowed by subtyping
```

Type declarations vs type inference

- Java insists that all variables are declared in advance, with type information
- The compiler can then check whether the program is well-typed
- An alternative approach is to do type inference
- Derive type information from context. For instance, `s` should be `String`

```
s = "Hello, " + "world";
```

- Propagate type information: now `t` is also `String`

```
t = s + 5;
```

```
public class Employee {...}

public class Manager extends Employee {...}

Employee e;

Manager m;

...

m = new Manager(...);
e = m;  // Allowed by subtyping
```

Type inference

- Assume code is well-typed, derive most general types
  - Use information from constants to determine type

    ```
    s = "Hello, " + "world";
    ```

  - Propagate type information based on already inferred types

    ```
    t = s + 5;
    ```

Type inference

- Assume code is well-typed, derive most general types
  - Use information from constants to determine type

    ```
    s = "Hello, " + "world";
    ```

  - Propagate type information based on already inferred types

    ```
    t = s + 5;
    ```

- More ambitious?

Type inference

- Assume code is well-typed, derive most general types
  - Use information from constants to determine type

    ```
    s = "Hello, " + "world";
    ```

  - Propagate type information based on already inferred types

    ```
    t = s + 5;
    ```

- More ambitious?
  - If `x.bonus()` is legal, `x` must be `Manager` rather than `Employee`

```
public class Employee {...}

public class Manager extends Employee {
   ...
   public double bonus (...) {...}
}

...

public static f(Employee x){
  ...
  double d = x.bonus(...);
    // x must be a Manager?
  ...
}
```

Type inference

- Assume code is well-typed, derive most general types
  - Use information from constants to determine type

    ```
    s = "Hello, " + "world";
    ```

  - Propagate type information based on already inferred types

    ```
    t = s + 5;
    ```

- More ambitious?
  - If `x.bonus()` is legal, `x` must be `Manager` rather than `Employee`
- Keep track of and validate type obligations

```
public class Employee {...}

public class Manager extends Employee {
   ...
   public double bonus (...) {...}
}

...

public static f(Employee x){
  ...
  double d = x.bonus(...);
    // x must be a Manager?
  ...
}
```

Type inference

- Assume program is type-safe, derive most general types compatible with code
  - Use information from constants to determine type
  - Propagate type information based on already inferred types

```
public class Employee {...}

public class Manager extends Employee {
    ...
    public double bonus (...) {...}
}

...

public static f(Employee x){
   ...
   double d = x.bonus(...);
     // x must be a Manager?
   ...
}
```

Type inference

- Assume program is type-safe, derive most general types compatible with code
  - Use information from constants to determine type
  - Propagate type information based on already inferred types
- Typing judgements should ideally be made at compile-time, not at run-time
  - Static analysis of code

```
public class Employee {...}

public class Manager extends Employee {
    ...
    public double bonus (...) {...}
}

...

public static f(Employee x){
   ...
   double d = x.bonus(...);
     // x must be a Manager?
   ...
}
```

Type inference

- Assume program is type-safe, derive most general types compatible with code
  - Use information from constants to determine type
  - Propagate type information based on already inferred types
- Typing judgements should ideally be made at compile-time, not at run-time
  - Static analysis of code
- Balance flexibility with algorithmic tractability

```
public class Employee {...}

public class Manager extends Employee {
    ...
    public double bonus (...) {...}
}

...

public static f(Employee x){
   ...
   double d = x.bonus(...);
     // x must be a Manager?
   ...
}
```

Type inference in Java

- Java allows limited type inference
  - Only for local variables in functions
  - Not for instance variables of a class

Type inference in Java

- Java allows limited type inference
  - Only for local variables in functions
  - Not for instance variables of a class
- Use generic `var` to declare variables
  - Must be initialized when declared
  - Type is inferred from initial value

```
var b = false;  // boolean

var s = "Hello, world";  // String
```

Type inference in Java

- Java allows limited type inference
  - Only for local variables in functions
  - Not for instance variables of a class
- Use generic `var` to declare variables
  - Must be initialized when declared
  - Type is inferred from initial value
- Be careful about format for numeric constants

```
var b = false;  // boolean

var s = "Hello, world";  // String

var d = 2.0;  // double

var f = 3.141f; // float
```

Type inference in Java

- Java allows limited type inference
  - Only for local variables in functions
  - Not for instance variables of a class
- Use generic `var` to declare variables
  - Must be initialized when declared
  - Type is inferred from initial value
- Be careful about format for numeric constants
- For classes, infer most constrained type
  - `e` is inferred to be `Manager`
  - `Manager` extends `Employee`
  - If `e` should be `Employee`, declare explicitly

```
var b = false;  // boolean

var s = "Hello, world";  // String

var d = 2.0;  // double

var f = 3.141f; // float

var e = new Manager(...);  // Manager
```

Summary

- Automatic type inference can avoid redundancy in declarations

```
Manager m = new Manager(...);
```

- Assuming the program is type-safe, derive most general types compatible with the code
  - Compiler can infer type from expressions used to assign values
  - Inferred type information can be propagated
- Challenge is to do this statically, at compile-time
- Java allows limited type inference
  - Only local variables that are initialized when they are declared