CS 565, Jens Palsberg Feb 21, 2000

Homework 3: Java Structural Subtyping

A grammar for a list of BabyJava interfaces together with a list of subtype queries, in JavaCC notation, is on the CS 565 webpage (interface.jj).

A subtype query is of the form

```
A <= B ?
```

where `A` and `B` are names of interfaces. This query has the answer Yes if `A` is a subtype of `B` in the *structural* subtyping order for Java described below, and No otherwise.

Use JTB/GJ and JavaCC and write in Java one or more visitors which decides the subtype queries.

Your main file should be called `DecideSubtyping.java`, and if `P` contains an input file then

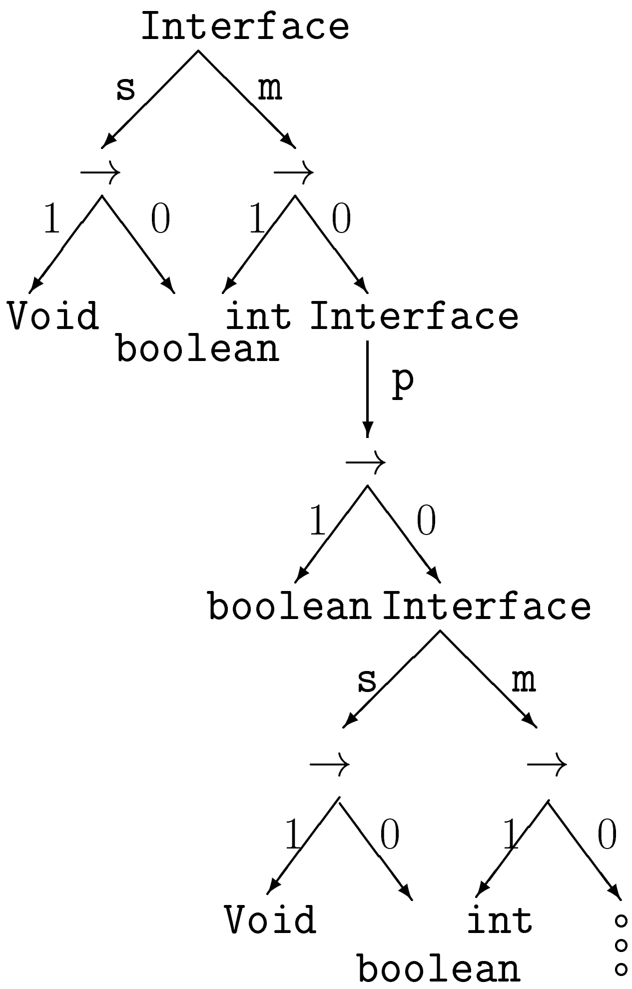
```
java DecideSubtyping < P
```

should print a sequence of `Yes` and `No`, each on a new line.

In this Homework, we view a BabyJava interface as an infinite, regular tree which is obtained by replacing occurrences of interface names by their definition.

```
interface I {
  void s(boolean a);
  int m(J a);
}

interface J {
  boolean p(I a);
}
```



A path is in the set

$$(Identifier\ \{0,1\})^*$$

where the identifiers are method names, 0 denotes "argument type", and 1 denotes "result type". We use id to range over $Identifier$.

Labels are from the set $\Sigma = \{\texttt{int}, \texttt{boolean}, \texttt{Void}, \rightarrow, \texttt{Interface}\}$.

An interface can now be represented as a partial function

$$t : (Identifier\ \{0,1\})^* \ \rightarrow\ \Sigma$$

with domain $\mathcal{D}(t)$ satisfying

- $\mathcal{D}(t)$ is nonempty and prefix-closed;
- if $t(\alpha) = \rightarrow$, then $\alpha 0, \alpha 1 \in \mathcal{D}(t)$;
- if $\alpha 0 \in \mathcal{D}(t)$, then $t(\alpha 0) \in \{\texttt{int}, \texttt{boolean}, \texttt{Void}, \texttt{Interface}\}$;
- if $\alpha 1 \in \mathcal{D}(t)$, then $t(\alpha 1) \in \{\texttt{int}, \texttt{boolean}, \texttt{Void}, \texttt{Interface}\}$;
- if $\alpha\ id \in \mathcal{D}(t)$, then $t(\alpha\ id) = \rightarrow$.

We can now define a structural subtyping order of BabyJava interfaces by combining the ideas from Lecture Note 3, p.16 with the idea that a interface with a certain number of fields should be a subtype of an interface with fewer fields.

For example, we want to have

```
A <= B ?  // No
B <= A ?  // Yes
C <= D ?  // No
D <= C ?  // Yes

interface A {
  A m(int a);
}

interface B {
  B m(int a);
  void p(boolean a);
}

interface C {
  int q(B a);
}

interface D {
  int q(A a);
  void r(int a);
}
```

We leave it to the reader to work out the details of the definition of structural subtyping.