

Java Applications

CS4000 Section 003

Term: Summer 2, 2002

Project #1: B to A-Prolog translator

Introduction

You are required to build a translator from (a simplified version of) action language B to A-Prolog. The translator must be able to invoke the A-Prolog interpreter (the `lparse+smodels` program suite) as a remote CORBA object.

The `lparse+smodels` suite is installed on the Suns of the CS Department. If you want to install it on a different computer, you will have to download the two programs from:

<http://www.tcs.hut.fi/Software/smodels/>

`lparse` can be installed by un-compressing the `lparse` tar file, and running “./configure”, “make” and “make install” as root from the `lparse` directory that is created when you un-compress the file. `smodels` can be installed by un-compressing the `smodels` tar file, and running “make” from the `smodels` directory that is created when you un-compress the file. After this, you will have to copy (as root) the `smodels` executable to `/usr/local/bin` or to another directory in your `PATH`. (Complete instructions can be found in the above web site.)

The use of Unix is **strongly** recommended for this project.

Description

The system is composed of three parts:

1. a GUI that allows users to type programs in language B;
2. a translator that converts the program in language B into an A-Prolog program;

3. a CORBA server for the lparse+smodels program suite that provides one method that invokes lparse+smodels and returns the duration of the execution of smodels.

The minimum requirement for the GUI is to provide a window containing a text area in which the program can be typed, a text box in which the user inserts the name of the program, and two buttons – one to perform the translation, the other to invoke lparse+smodels.

The translator must take as input the text inserted into the text area, analyze each statement, and create two files: file “XXX.b” will contain the program typed by the user; file “XXX.apl” will contain the translated program. “XXX” stands for the program name inserted by the user in the text box.

The translator must detect statements not matching the syntax of the input language, output the line number where the error was found, and abort the translation process (**without** terminating the program). The syntax of the input language is as follows:

- $causes(\langle string_1 \rangle, \langle string_2 \rangle, [\langle string_3 \rangle, \langle string_4 \rangle, \dots, \langle string_K \rangle])$.
- $caused(\langle string_1 \rangle, [\langle string_2 \rangle, \langle string_3 \rangle, \dots, \langle string_K \rangle])$.
- $impossible_if(\langle string_1 \rangle, [\langle string_2 \rangle, \langle string_3 \rangle, \dots, \langle string_K \rangle])$.
- $initially(\langle string_1 \rangle)$.
- $occurs(\langle string_1 \rangle, \langle string_2 \rangle)$.

$\langle string_X \rangle$ denotes any alpha-numeric string. The list in square brackets can contain *any* number of argument, including 0 arguments.

The translation of the statements is as follows:

- Statement *causes*:

$$\begin{aligned}
 h(\langle string_2 \rangle, T) \quad : - \quad & o(\langle string_1 \rangle, T - 1), \\
 & h(\langle string_3 \rangle, T - 1), \\
 & \dots \\
 & h(\langle string_K \rangle, T - 1).
 \end{aligned}$$

- Statement *caused*:

$$h(\langle string_1 \rangle, T) : - \begin{array}{l} h(\langle string_2 \rangle, T), \\ h(\langle string_3 \rangle, T), \\ \dots \\ h(\langle string_K \rangle, T). \end{array}$$

- Statement *impossible_if*:

$$: - \begin{array}{l} o(\langle string_1 \rangle, T), \\ h(\langle string_2 \rangle, T), \\ \dots \\ h(\langle string_K \rangle, T). \end{array}$$

- Statement *initially*:

$$h(\langle string_1 \rangle, 0).$$

- Statement *occurs*:

$$o(\langle string_1 \rangle, \langle string_2 \rangle).$$

The object file must also start with the following lines:

```
time(0..3).
#domain time(T).
```

The CORBA server must invoke the `lparse+smodels` program suite so that it reads the translated file, and computes the result. The server must then filter the output of the program suite, searching for a line beginning with “Duration:”, followed by a floating-point number. This number must be returned as the result of the invocation of the CORBA method.

In order to run the `lparse+smodels` program suite you can use the following technique:

- Assuming that you are using the Suns of the CS Department, create the following file, named “`smodels.sh`”, and give it executability permissions for all users:

```
#!/bin/bash
lparse $1 | smodels > $1.out
```

- From the CORBA server, execute the following command line:

smodels.sh XXX.apl

where “XXX.apl” is the name of the translated file. In order to run the above command line, you can adapt this fragment of Java code:

```
try
{ Process p;
...
// I assume that aplFile is a String containing the filename of the translated file
  p=Runtime.getRuntime().exec("smodels.sh "+aplFile);
  p.waitFor();
...
}
catch (IOException e)
{ System.out.println("IO Exception: "+e);
}
catch (InterruptedException e)
{ System.out.println("InterruptedException: "+e);
}
catch (SecurityException e)
{ System.out.println("SecurityException: "+e);
}
```

- you will find the output of lparse+smodels in file XXX.apl.out, where “XXX.apl” is the name of the translated file.

Tests

You should make sure that your project works by running it with the following B program:

causes(a1, f2, [f1]).
caused(f3, [f1, f2]).

initially(f1).
occurs(a1, 0).