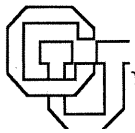
A Scanner-Parser Project
(Preliminary Report)

Michael Main

CU-CS-CT001-97



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

A Scanner-Parser Project

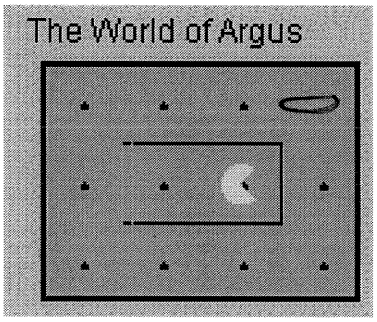
(Preliminary Report)

Michael Main
(main@cs.colorado.edu)
Computer Science Department
University of Colorado at Boulder

Research Reports on Curriculum and Teaching
CU-CS-CT001-97
December 27, 1997

Abstract. This report describes a scanner-parser project that is appropriate for about two weeks of an undergraduate class in principles of programming languages. The project builds a complete program verifier for a small programming language that controls a dog in a finite-state world. The report will be expanded later this year to include the results of my Spring 1998 class.

A Scanner-Parser Project
(Preliminary Report)



Argus is a dog who lives in the small world shown here. He is the 3/4 circle in the inner rectangle. The open part of the circle is his mouth, which is always open because he is always happy. In the picture shown here, he is facing east, and he is inside his small house. There are a few other things in his world: a bone (which is the grey oval in the top right corner) and a door (which is the double vertical lines at the west end of Argus's house).

Argus is based on Richard Pattis's *Karel the Robot*. The motivation for his design is to provide a small programming language which can be used in an undergraduate programming languages class as the basis of a scanner-parser project. This report describes the Argus Programming Language and suggests three projects that can be carried out by undergraduate students. Further details on these projects will be given in a follow-up project in 1998. This report it currently available at http://www.cs.colorado.edu/~main/proglang/argus.html. When the expanded report is completed, it will be available at the same location.

Supplement for Programming Languages
by Michael Main

Argus Commands

Argus can obey five commands:

1. `turn` (which makes Argus turn 90 degrees, clockwise)
2. `forward` (which makes Argus move forward one spot--but don't do this if there's a wall or closed door in front of him!)
3. `bark` (which can only be done if the door is directly in front of Argus, and Argus is not holding his bone; barking changes the door from open-to-shut or from shut-to-open)
4. `pickup` (which makes Argus pickup his bone; this can only be done if Argus is on top of the bone and not holding it already)
5. `drop` (which makes Argus drop his bone; this can only be done if he is holding his bone)

If you try to do anything illegal, then Argus will crash and a big red X appears in his world.

The Argus Environment

I have written a small programming environment to let you control Argus. When you run the environment, you can control Argus directly by pushing buttons for turn, forward, bark, pickup and drop. If you crash the system, then you can press reset to get things going again.

You can also write a program to control Argus, using the Argus Programming Language described below. You can save Argus Programs to a disk file with the save button, or you can load a file from the disk with the load button. To run a program, click the run button. While a program is running, you can make it run faster or slower with the speed bar. You can also pause the execution or stop the execution with two other buttons. If you want to allow recursion in your programs, then the "Allow Recursion" box must be checked on.

To run the Argus Environment on the undergrad machines at the University of Colorado: ~main/bin/argus (After starting the environment, you may need to grab a corner and resize it so that everything is visible.)

To run the Argus Environment on your home machine (which must have jdk1.1 or higher installed):

- Download the file
  ftp://ftp.cs.colorado.edu/pub/techreports/main/java/Argus.java
  to any directory on your machine
- Change to the directory where you downloaded Argus.java and compile it with: javac Argus.java
- From this same directory, execute Argus with the command: java Argus

The Argus Programming Language

Here is a description of the Argus Programming Language. Keywords and symbols are written in boldface. Programs may also contain comments that start with // and continue to the end of a line.

`procedure`
An Argus Program is a series of one or more procedures. One of the procedures must be called "main" and this procedure will be executed when the program is run. The format of a procedure is:

```
procedure name
{
   a sequence of statements
}
```

The "name" may be any non-keyword starting with a letter and followed by a sequence of letters and digits.

`statement`
A statement may be an assert-statement, an if-statement, an if-else-statement, a while-statement, a procedure-call, a block-statement, or one of these five basic statements:

```
turn;
forward;
bark;
pickup;
drop;
```

`assert-statement`
An assert-statement has the form:

```
assert(boolean-expression);
```

if-statement
An if-statement has this form, where the statement may be a block-statement if you like:

```
if (boolean-expression) statement
```

`if-else-statement`
An if-else-statement has this form, where each statements may be a block-statement if you like:

```
if (boolean-expression) statement else statement
```

`while-statement`
A while-statement has this form, where the statement may be a block-statement if you like:

```
while (boolean-expression) statement
```

`procedure-call`
A procedure-call is the name of one of your procedures followed by a semicolon.

`block-statement`
A block-statement is an opening curly bracket, followed by a sequence of zero or more statements, followed by a closing curly bracket.

boolean-expression
A boolean-expression is the same as a C++ boolean expression, formed from the operations &&(and), ||(or), !(not) and any of these basic predicates:

```
atBone
atDoor
atX1
atX2
atX3
atX4
atY1
atY2
atY3
hasBone
true
false
facingNorth
facingEast
facingSouth
facingWest
isInside
isClear
```

Most of these names are self-explanatory, except perhaps `isClear`. The `isClear` predicate is true if Argus can move forward without running into a wall or door.

PROJECTS FOR PROGRAMMING LANGUAGES CLASSES

Scanner Project

Use lex or flex to write a lexical analyzer for the Argus language. The scanner should ignore separators and comments. The list of token numbers is available in http://www.cs.colorado.edu/~main/proglang/argus.tab.h

Simple Parser Project

Use yacc or bison to write a parser for the Argus language. The parser should use the lexical analyzer from the previous project to read an Argus program. The parser prints a message indicating whether the input is a legal Argus program, or whether a parse error is found.

More Interesting Parser Project

This is the project that my students will be doing in Spring 1998.

Use yacc or bison to write a parser for the Argus language. As the program is parsed, each statement and each procedure has a state transition matrix attached to it. For the matrixes, you may use the ArgusMatrix class argusmat.h and argusmat.cxx (in http://www.cs.colorado.edu/~main/proglang). Whenever a procedure is parsed, the corresponding transition matrix should be stored in a symbol table, indexed by the procedure's name. You may use the SymbolTable class from symtab.h and symtab.cxx (in the same location on the web).

After you have parsed an entire program, you should check to see that there is a procedure called "main". If a main procedure exists, then verify that its transition matrix does not have any rows with all false values. (Such a row indicates that when the program starts in that state, then the program will crash or go into an infinite loop).

In effect, you are building a complete program verifier for Argus programs. The same technique can be used to verify that a program is correct for any programming language that controls a small finite-state device (up to several thousand states).