NAME

pburg − variable cost code-generator generator

SYNOPSIS

pburg [*option*] ... [[*input*] *output*]

DESCRIPTION

pburg reads a BURG specification from *input* and writes a pattern-matching code generator to *output*.

If *input* is '−' or is omitted, *pburg* reads the standard input;

If *output* is '−', *pburg* writes to the standard output.

If *output* is omitted, *pburg* writes to a file named *yyselect.c*

pburg accepts specifications that conform to the following EBNF grammar.

Terminals are enclosed in single quotes or are given in uppercase, all other symbols are nonterminals or English phrases, {X} denotes zero or more instances of X, and [X] denotes an optional X.

```
spec:     '%{' config '%}' { decl } '%%' { rule } [ '%%' C code ]

decl:      '%start' nonterm
           '%term' { ID '=' init }
           '%include' '"' filename '"'

init:     INT
          ''' CHAR '''

rule:     nonterm ':' tree [ cost ] [ '{' C code '}' ]

tree:     term '(' tree ',' tree ')'
          term '(' tree ')'
          term
          nonterm

nonterm:  ID

cost:     INT
          ID
```

Specifications are structurally similar to *yacc*'s. Text between '%{' and '%}' is called the configuration section; there may be several such segments. All are concatenated and copied verbatim into the head of the output. Text after the second '%%', if any, is also copied verbatim into the output, at the end.

Specifications consist of declarations, a '%%' separator, and rules.

Input is line-oriented; each declaration and rule must appear on a separate line, and declarations must begin in column 1.

Declarations declare terminals — the operators in subject trees — and associate a unique, positive external symbol number with each one.

Nonterminals are declared by their presence on the left side of rules. The `%start` declaration optionally declares a nonterminal as the start symbol.

In the grammar above, `term` and `nonterm` denote identifiers that are terminals and nonterminals.

Rules define tree patterns in a fully parenthesized prefix form. Every nonterminal denotes a tree.

Each operator has a fixed arity, which is inferred from the rules in which it is used.

A chain rule is a rule whose pattern is another nonterminal.

If no start symbol is declared, the nonterminal defined by the first rule is used.

Each rule ends with a cost that computes the cost of matching that rule; omitted costs default to zero. Costs of chain rules must be non-negative constants or function names.

If the cost is defined as function, the function is called with a node (see `NODEPTR_TYPE` below) and the function should return a non-negative integer cost.

The configuration section configures the output for the trees being parsed and the client's environment.

As shown, this section must define `NODEPTR_TYPE` to be a visible typedef symbol for a pointer to a node in the subject tree.

The labeller invokes

`OP_LABEL(p)`,

`LEFT_CHILD(p)`, and

`RIGHT_CHILD(p)`

to read the operator and children from the node pointed to by `p`.

If the configuration section defines these operations as macros, they are implemented in-line; otherwise, they must be implemented as functions.

The matcher computes and stores a single integral state in each node of the subject tree. The configuration section must define a macro

`STATE_LABEL(p)`

to access the state field of the node pointed to by `p`. It must be large enough to hold a pointer, and a macro is required because it is used as an lvalue.

The selector, `int yyselect(NODEPTR_TYPE tree)`, consists of a labeling pass, using the `yylabel` rotine, and an emiting pass, using the `yyreduce` rotine. When a successful labeling is obtained the selected code is emmited and the `yyselect` rotine returns 0. Otherwise, if no complete labeling can be achieved, the `yyselect` rotine returns 1, and does not invoque the `yyreduce` rotine.

When an internal error occurs, the ‘PANIC’ rotine is called. A default version, called ‘yypanic’, is supplied. The macro ‘PANIC’ can be defined in the declaration section as the name of a user supplied panic rotine. The supplied rotine should have the same signature: `(YYCONST char *rot, YYCONST char *msg, int val)`.

OPTIONS

−p *prefix*

−p*prefix*

Use

prefix

as the disambiquating prefix for visible names and fields.

The default is ‘yy’.

−T

Arrange for a debug rotine to be called at each successful match. A cost of 32767, or graeter, represents the infinite cost of no previous match.

The macro ‘TRACE’ can be defined in the declaration section as the name of a user supplied trace rotine. The supplied rotine should have the same signature: `(NODEPTR_TYPE p, int eruleno, int cost, int bestcost)`.

−m

Generate a maximal-munch selector. These generators are faster than the default generators since the tree is transversed only once. However, the generator does not accept rules that differ only in the non-terminals of the selection pattern. At each step, starting at the root node, the algorithm selects the matching rule that has the highest number of tree nodes, i.e. the highest number of terminals in the selection pattern. Rules can be excluded if they return a value graeter or equal to MAX_COST or 32767. An extended maximal-munch selector is generated if the `-m` is used twice, `-m -m`, allowing more complex grammars, but with possibly higher matching time..

−A

Parse all `#define` declarations in a `%include` file. Otherwise, the parsing of the `%include` file ends at the first non `#define` instruction.

−n

The outputed parser does not include the original line numbers of the user C code in the input file. It can be used for a more extensive debugging of the generated parser’s behaviour.

−J

Generate a `Java` selector. By default, the generated public class is named `Selector` and the file `Selector.java`. The syntactic tree is of a `Tree` class and must provide an empty constructor that sets a negative `label` and a `void state(Object st)` method to set the given object, as well as the query methods `Object state()`, `public int label()`, `public Tree left()` and `public Tree right()` that return the `label`, the `state` and the left and right tree branches, respectively. The command line options `-Jclass=`, `-Jfinal`, `-Jextends=`, `-Jthrows=` and `-Jtree=` allow the overriding of the generated class name, whether it is a final class, if it extends from an existing class, any exceptions that the embeded code (from the source file .brg) might throw, or the name of the class used as a syntactic tree, respectively.

−v

Print version and exit.

ERRORS

Null tree

when a NULL node is accessed. The tree has a NULL branch. When the declared arity of the tree pattern is higher than actual number of branches in the tree, the missing branches can be interpreted as NULL nodes.

Null kids in

when there are NULL nodes in the computed kids of a pattern. The kids of a pattern are a sequential list of all terminals and nonterminals of a pattern, in the that they appear in the pattern (left to right).

Bad terminal

when the label of a node does not match any of labels in the patterns. Some tree branch was tagged with an undefined label.

Bad goal nonterminal

when an invalod nonterminal is used as a goal nonterminal.

Bad rule number

when the internaly assigned rule number is invalid.

SEE ALSO

lcc(1)

C. W. Fraser and D. R. Hanson, *A*Retargetable*C*Compiler:*Design*and*Implementation*, Benjamin/Cummings, Redwood City, CA, 1995, ISBN 0-8053-1670-1. Chapter 14.

C. W. Fraser, D. R. Hanson and T. A. Proebsting, ‘Engineering a simple, efficient code generator generator,’ *ACM Letters on Programming Languages and Systems* **1**, 3 (Sep. 1992), 213-226.

BUGS

Mail bug reports along with the shortest input that exposes them to reis.santos@tecnico.ulisboa.pt.