Ex-no: 7 M. Rohith
07-11-2024 3122 21 5001 085

SSN COLLEGE OF ENGINEERING, KALAVAKKAM
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UCS2702 - Compiler Lab
Programming Assignment-7 Implementation of code optimization techniques

Consider the Three Address Code sequences and apply the following techniques to optimize the code.
Constant folding
Algebraic identities
Strength reduction
Dead code elimination

Input:
t1= 5*3
t2=a+0
t3=b*1
t4=b*d
t5=d2**
t6=b*d

Output:
t1=15
t3=b
t4=b*d
t5=b*b

Program code:

optimize.l

```
%{
#include "optimize.tab.h"
%}

digit    [0-9]+
variable [a-zA-Z][a-zA-Z0-9]*

%%

{digit}        { yylval.intval = atoi(yytext); return NUMBER; }
{variable}     { yylval.strval = strdup(yytext); return VARIABLE; }
"="            { return ASSIGN; }
"**"           { return POWER; }
"*"            { return MULTIPLY; }
"+"            { return ADD; }
\n             { return NEWLINE; }
[ \t]          { /* ignore whitespace */ }
```

```
.           { return yytext[0]; }

%%

int yywrap() {
   return 1;
}
```

optimize.y

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void yyerror(const char *s);
int yylex();

typedef struct {
   char *name;
   char *expression;
} entry;

entry symbol_table[100];
int symbol_table_index = 0;

int lookup(char *expr);
void add_expression(char *name, char *expr);
char *concatenate(char *a, char *op, char *b);
int is_constant(char *name);

%}

%union {
   int intval;
   char *strval;
}

%token <intval> NUMBER
%token <strval> VARIABLE
%token ASSIGN MULTIPLY ADD POWER NEWLINE

%left ADD
%left MULTIPLY
%right POWER

%type <strval> expression line

%%

input:
   | input line NEWLINE
```

```
    ;

line:
      VARIABLE ASSIGN expression {
          int index = lookup($3);
          if (index == -1) {
              // Expression not yet seen, so we add it
              add_expression($1, $3);
              printf("%s = %s\n", $1, $3);
          } else {
              // Expression already exists, skip duplicate
              printf("// Duplicate of %s; skipping %s\n", symbol_table[index].name, $1);
          }
      }
    ;

expression:
      NUMBER {
          char buffer[12];
          sprintf(buffer, "%d", $1);
          $$ = strdup(buffer);
      }
    | VARIABLE {
          $$ = strdup($1);
      }
    | expression MULTIPLY expression {
          if (is_constant($1) && is_constant($3)) {
              char buffer[12];
              sprintf(buffer, "%d", atoi($1) * atoi($3));
              $$ = strdup(buffer);
          } else if (strcmp($1, "1") == 0) $$ = $3;
          else if (strcmp($3, "1") == 0) $$ = $1;
          else {
              $$ = concatenate($1, "*", $3);
          }
      }
    | expression ADD expression {
          if (is_constant($1) && is_constant($3)) {
              char buffer[12];
              sprintf(buffer, "%d", atoi($1) + atoi($3));
              $$ = strdup(buffer);
          } else if (strcmp($1, "0") == 0) $$ = $3;
          else if (strcmp($3, "0") == 0) $$ = $1;
          else {
              $$ = concatenate($1, "+", $3);
          }
      }
    | expression POWER expression {
          if (strcmp($3, "2") == 0) $$ = concatenate($1, "*", $1);
          else $$ = concatenate($1, "**", $3);
      }
    ;
```

```
%%

int main() {
    yyparse();
    return 0;
}

void yyerror(const char *s) {

}

int lookup(char *expr) {
    for (int i = 0; i < symbol_table_index; i++) {
        if (strcmp(symbol_table[i].expression, expr) == 0) {
            return i;
        }
    }
    return -1;
}

void add_expression(char *name, char *expr) {
    symbol_table[symbol_table_index].name = strdup(name);
    symbol_table[symbol_table_index].expression = strdup(expr);
    symbol_table_index++;
}

int is_constant(char *name) {
    for (int i = 0; name[i] != '\0'; i++) {
        if (name[i] < '0' || name[i] > '9') return 0;
    }
    return 1;
}

char *concatenate(char *a, char *op, char *b) {
    char *result = malloc(strlen(a) + strlen(op) + strlen(b) + 1);
    sprintf(result, "%s%s%s", a, op, b);
    return result;
}
```

input.txt

```
t1= 10 * 10
t2= a + 0
t3= b * 1
t4= c * d
t5= c**2
t6= c * d
t7= c * d
t8= a + 0
```

input2.txt

t1=5*3
t2=a+0
t3=b*1
t4=b*d
t5=d**2
t6=b*d

output: