Review Chapter 10

parse configuration file of Linear programming

Speaker: Lung-Sheng Chien

Application 1: configuration file of Linear Programming

Objective: read configuration file, extract coefficient of vector ***c***, **b** and matrix ***A***, then output ***c***, ***b***, ***A***

configure.txt

```
1
2  // minimize z = C' *x
3  <objective>
4    1*x1 + 0.5*x2 + 1.0*x4
5  </objective>
6
7  // subject to Ax <= b
8  // x >= 0 is implicit
9  <constraint>
10   -2*x1 + 2.*x2 <= 5.0
11   3*x2 - 1*x5 >= 7
12   6*x2 + 3.14*x1 = 6
13 </constraint>
14
```

$$\min z = c^T x$$

$$\text{subject to } Ax \le b, x \ge 0$$

token

<objective> <constraint>

</objective> </constraint>

C++-comment

x1 x2 x4 x5

integer real number

\+ - * >= <= =

Assumption 1: separate **sign** from integer and real

Assumption 2: format is coeff * var

Assumption 3: coeff is a number, not an expression

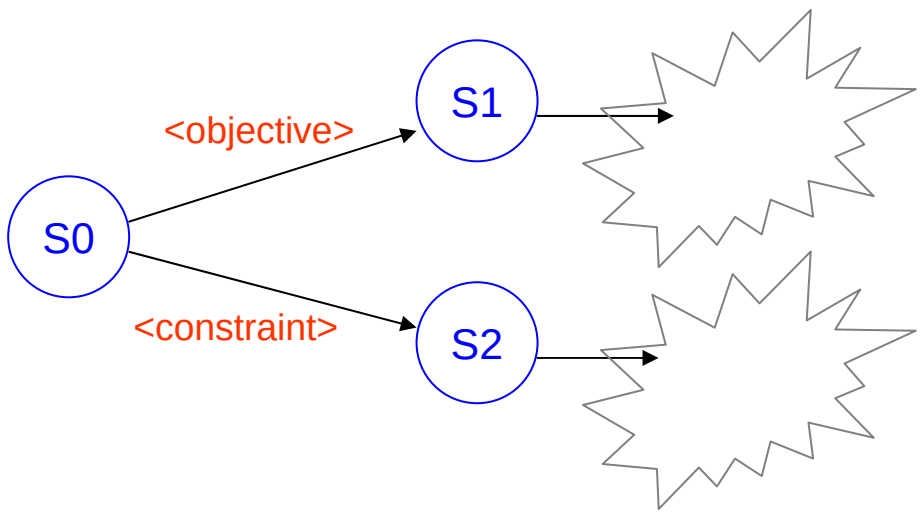
Exercise

- Complete input file for *flex* (add rule to deal with C++-comment) and test the scanner for different cases.

- Depict state transition diagram to collect information from configuration file and construct vector ***c***, ***b*** and matrix ***A***

configure.txt

```
 1
 2 // minimize z = C' *x
 3 <objective>
 4   1*x1 + 0.5*x2 + 1.0*x4
 5 </objective>
 6
 7 // subject to Ax <= b
 8 // x >= 0 is implicit
 9 <constraint>
10   -2*x1 + 2.*x2 <= 5.0
11   3*x2 - 1*x5 >= 7
12   6*x2 + 3.14*x1 = 6
13 </constraint>
14
```



Two-steps solver

- Step 1: create symbol table and find number of equation >= (*numOfGE*), number of equation <= (*numOfLE*) and number of equation = (*numOfEQ*).
  m = (# of equation >=) + (# of equation <=) + (# of equation =)
  n = (# of variables) + (# of equation >=) + (# of equation <=)

- Step 2: find cost vector ***c*** and
  find coefficient of equation >=, <= and =

c

x_1	x_2	x_4	x_5	s_1	sp_1
1	0.5	1.0	0	0	0

$s_1 : slack$

$sp_1 : surplus$

```
1
2  // minimize z = C' *x
3  <objective>
4    1*x1 + 0.5*x2 + 1.0*x4
5  </objective>
6
7  // subject to Ax <= b
8  // x >= 0 is implicit
9  <constraint>
10   -2*x1 + 2.*x2 <= 5.0
11   3*x2 - 1*x5 >= 7
12   6*x2 + 3.14*x1 = 6
13 </constraint>
14
```

Symbol table

0	useless
1	x1
2	x2
3	x4
4	x5

numOfGE = 1

numOfLE = 1

numOfEQ = 1

m = 3, n = 6

symbol.h

```
/*  record symbole(variable) in configuration file of LP
 *  we use array (from position 1) to store all symbols
 *  for each symbol, we have natural index (index of array), say
 *      table[2] = cat    ==>  map "cat" to x2
 */

#ifndef SYMBOL_H
#define SYMBOL_H

#define MAX_TABLE_SIZE  1024

typedef struct symbolTable{
    char *table[MAX_TABLE_SIZE+1] ;
    int  size ; // size of table
}symbolTable ;

typedef symbolTable* symbolTablePtr ;

// duplicate string s
char*  dup_string( char* s ) ;

// allocate an empty symbol table
symbolTable*  symTable_alloc( void ) ;

// deallocate all string in symbol table
void  symTable_dealloc( symbolTable* table ) ;

// return:  j  if table[j] is symbol "s"
//          0  otherwise
int   symTable_lookup( symbolTable* table, char *s ) ;

// insert symbol "s" to symbol table, neglect repeated symbol
int   symTable_insert( symbolTable* table, char *s ) ;

// show configuration of symbol table
void  symTable_display( FILE* fp, symbolTable* table ) ;

#endif // SYMBOL_H
```

we use string array “table” to record symbol (variable), you can use linked-list to implement. Moreover in this example, we hardcode maximum size of table, you can relax it

Methods for symbol table

symbol.cpp

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>
#include "symbol.h"

char*  dup_string( char* s )
{
  assert( s ) ;
  int len = strlen( s ) ;
  char *t ;
  t = (char*) malloc(sizeof(char)*(len+1)) ;
  assert( t ) ;
  strcpy( t, s ) ;
  return t ;
}

symbolTable*  symTable_alloc( void )
{
  symbolTable* table ;
  table = (symbolTable*)
         malloc(sizeof(symbolTable)) ;
  assert( table ) ;

  table->size = 0 ; // empty table
  return table ;
}
void symTable_dealloc(symbolTable* table)
{
  assert( table ) ;
  int i ;
  for( i=1 ; i <= table->size ; i++ ){
     free( table->table[i] ) ;
  }
}
```

```
int symTable_lookup(symbolTable* table, char *s )
{
  assert( table ) ;
  int i ;
  for( i=1 ; i <= table->size ; i++ ){
    if ( 0 == strcmp(table->table[i], s) ){
        return i ;
    }
  }
  return 0 ;
}

int symTable_insert( symbolTable* table, char *s )
{
  assert( table ) ;
  if ( 0 < symTable_lookup(table, s) ){
    printf("Warning: symbol %s is already in symbol table\n", s );
    return 0 ;
  }
// check if symbol table is overflow or not
  if ( MAX_TABLE_SIZE == table->size ){
    printf("Error: symbol table is full, no insertion\n");
    printf("   please increase macro MAX_TABLE_SIZE \n");
    exit(1) ;
  }
  table->size++ ;
  table->table[ table->size ] = dup_string( s ) ;
  return 1 ;
}
void  symTable_display( FILE* fp, symbolTable* table )
{
    int i ;
    fprintf(fp,"Total number of symbol = %d\n", table->size );
    for( i = 1 ; i <= table->size ; i++ ){
        fprintf(fp,"table[%d] = %s\n", i, table->table[i]);
    }
}
```

Linear search, O(n), bad

Question: How to improve lookup

- Function *symTable_lookup* uses linear search (*O(n)*) to check repeated element. This means that to construct cost vector and constraint matrix, we need $O(n^2)$, can you reach search limit $O(\log n)$ by using binary search?
  Hint: we may use a binary tree

```
int symTable_lookup(symbolTable* table, char *s )
{
  assert( table ) ;
  int i ;
  for( i=1 ; i <= table->size ; i++ ){
    if ( 0 == strcmp(table->table[i], s) ){
        return i ;
    }
  }
  return 0 ;
}
```

First step: extract symbol and number of equations

buildLP.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "y.tab.h"
#include "symbol.h"

extern "C" {
  extern FILE* yyin ;
  extern char *yytext ;
  int yylex( void ) ;
}
```

```
1
2 // minimize z = C' *x
3 <objective>
4   1*x1 + 0.5*x2 + 1.0*x4
5 </objective>
6
7 // subject to Ax <= b
8 // x >= 0 is implicit
9 <constraint>
10   -2*x1 + 2.*x2 <= 5.0
11    3*x2 - 1*x5 >= 7
12   6*x2 + 3.14*x1 = 6
13 </constraint>
14
```

```
void  extractSymbol( char* fname, symbolTablePtr *t,
        int *numOfGE, int *numOfLE, int *numOfEQ )
{
    int   token ;

    *t = symTable_alloc() ; // empty symbol table
    *numOfGE = *numOfLE = *numOfEQ = 0 ;

    yyin = fopen( fname, "r") ;
    assert( yyin ) ;
    while( token = yylex() ){
        switch( token ){
        case IDENTIFIER_ :
            symTable_insert( *t, yytext ) ;
            break ;
        case GE_ :
            (*numOfGE)++ ;
            break ;
        case LE_ :
            (*numOfLE)++ ;
            break ;
        case '=' :
            (*numOfEQ)++ ;
            break ;
        default:
            break ;
        }// switch(token)
    }// for each token

    fclose( yyin );
}
```

```
3*x2 - 1*x5 >= 7
-2*x1 + 2.*x2 <= 5.0
6*x2 + 3.14*x1 = 6
```

main.cpp

```
#include <stdio.h>
#include <string.h>
#include "symbol.h"
#include "buildLP.h"

#define  DEFAULT_INPUT_FILE  "configure.txt"

int main(int argc, char* argv[])
{
    char  inFile[ 128 ] ; // input file name
    symbolTablePtr  tablePtr ; // symbol table
    int numOfGE, numOfLE, numOfEQ ;

    ++argv ;
    --argc ; // skip over command

#if defined(_WIN32) || defined(__WIN32__)
    strcpy( inFile, DEFAULT_INPUT_FILE ) ;
#else
    if ( 0 < argc ){
        strcpy( inFile, argv[0] ) ;
    }else{
        printf("Error: please specify input file\n");
        exit(1) ;
    }
#endif

// step 1: setup symbol table and find number of equation <= , >= and =
    extractSymbol( inFile, &tablePtr, &numOfGE, &numOfLE, &numOfEQ ) ;
    symTable_display( stdout, tablePtr ) ;

    printf("number of >= equations = %d\n", numOfGE );
    printf("number of <= equations = %d\n", numOfLE );
    printf("number of  = equations = %d\n", numOfEQ );

    return 0 ;
}
```

```
Warning: symbol x1 is already in symbol table
Warning: symbol x2 is already in symbol table
Warning: symbol x2 is already in symbol table
Warning: symbol x2 is already in symbol table
Warning: symbol x1 is already in symbol table
Total number of symbol = 4
table[1] = x1
table[2] = x2
table[3] = x4
table[4] = x5
number of >= equations = 1
number of <= equations = 1
number of  = equations = 1
Press any key to continue
```

configure.txt

```
 1
 2 // minimize z = C' *x
 3 <objective>
 4   1*x1 + 0.5*x2 + 1.0*x4
 5 </objective>
 6
 7 // subject to Ax <= b
 8 // x >= 0 is implicit
 9 <constraint>
10   -2*x1 + 2.*x2 <= 5.0
11   3*x2 - 1*x5 >= 7
12   6*x2 + 3.14*x1 = 6
13 </constraint>
14
```

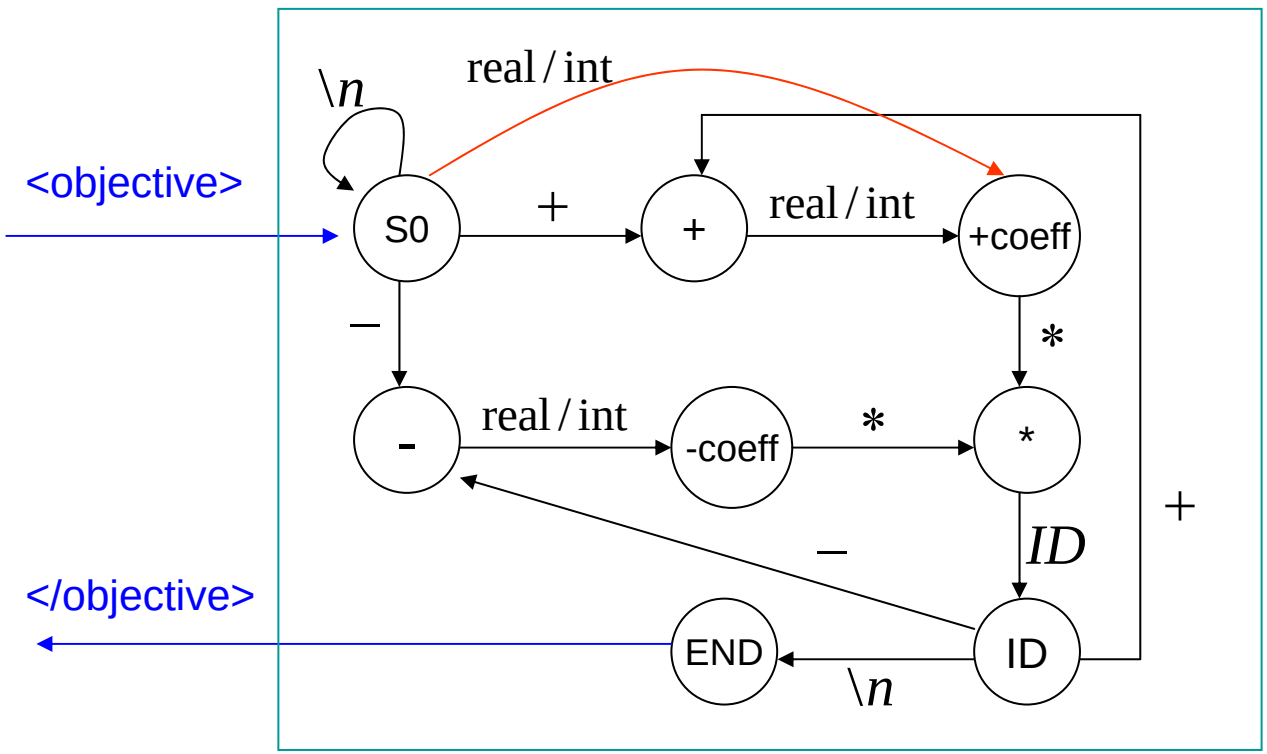
Extract cost vector

objective:extract $c_1x_1 + c_2x_2 + \cdots + c_nx_n$

configure.txt

```
1
2 // minimize z = C' *x
3 <objective>
4   1*x1 + 0.5*x2 + 1.0*x4
5 </objective>
6
7 // subject to Ax <= b
8 // x >= 0 is implicit
9 <constraint>
10   -2*x1 + 2.*x2 <= 5.0
11   3*x2 - 1*x5 >= 7
12   6*x2 + 3.14*x1 = 6
13 </constraint>
14
```

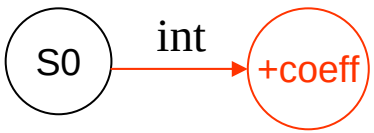
Flow chart (finite state machine)



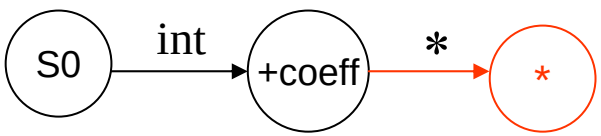
pack into a function

State sequence [1]

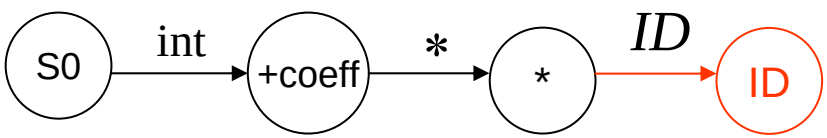
1	*	x1		+		0.5	*	x2		+		1.0	*	x4

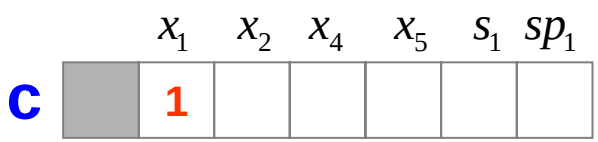


1	*	x1		+		0.5	*	x2		+		1.0	*	x4



1	*	x1		+		0.5	*	x2		+		1.0	*	x4





configure.txt

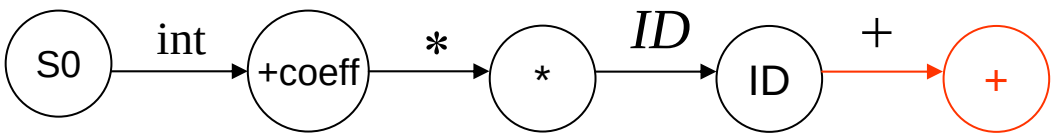
```
1
2 // minimize z = C' *x
3 <objective>
4   1*x1 + 0.5*x2 + 1.0*x4
5 </objective>
6
7 // subject to Ax <= b
8 // x >= 0 is implicit
9 <constraint>
10   -2*x1 + 2.*x2 <= 5.0
11   3*x2 - 1*x5 >= 7
12   6*x2 + 3.14*x1 = 6
13 </constraint>
14
```

Symbol table

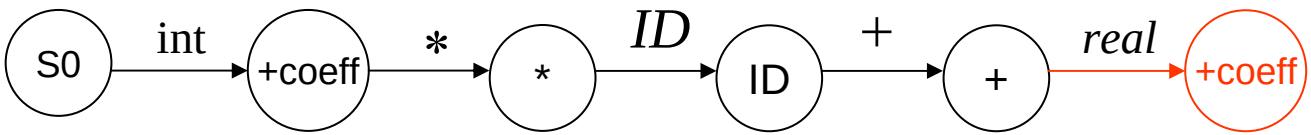
0	useless
1	x1
2	x2
3	x4
4	x5

State sequence [2]

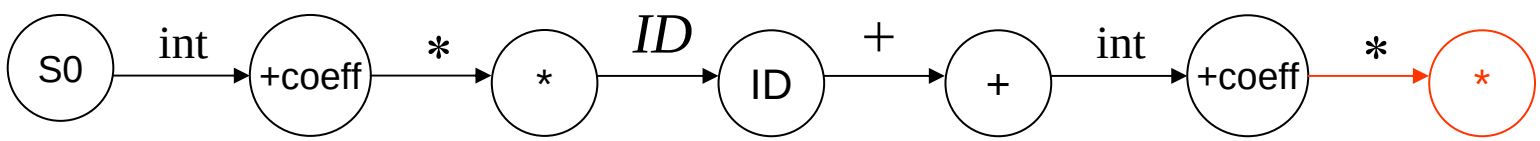
1	*	x1		+		0.5	*	x2		+		1.0	*	x4



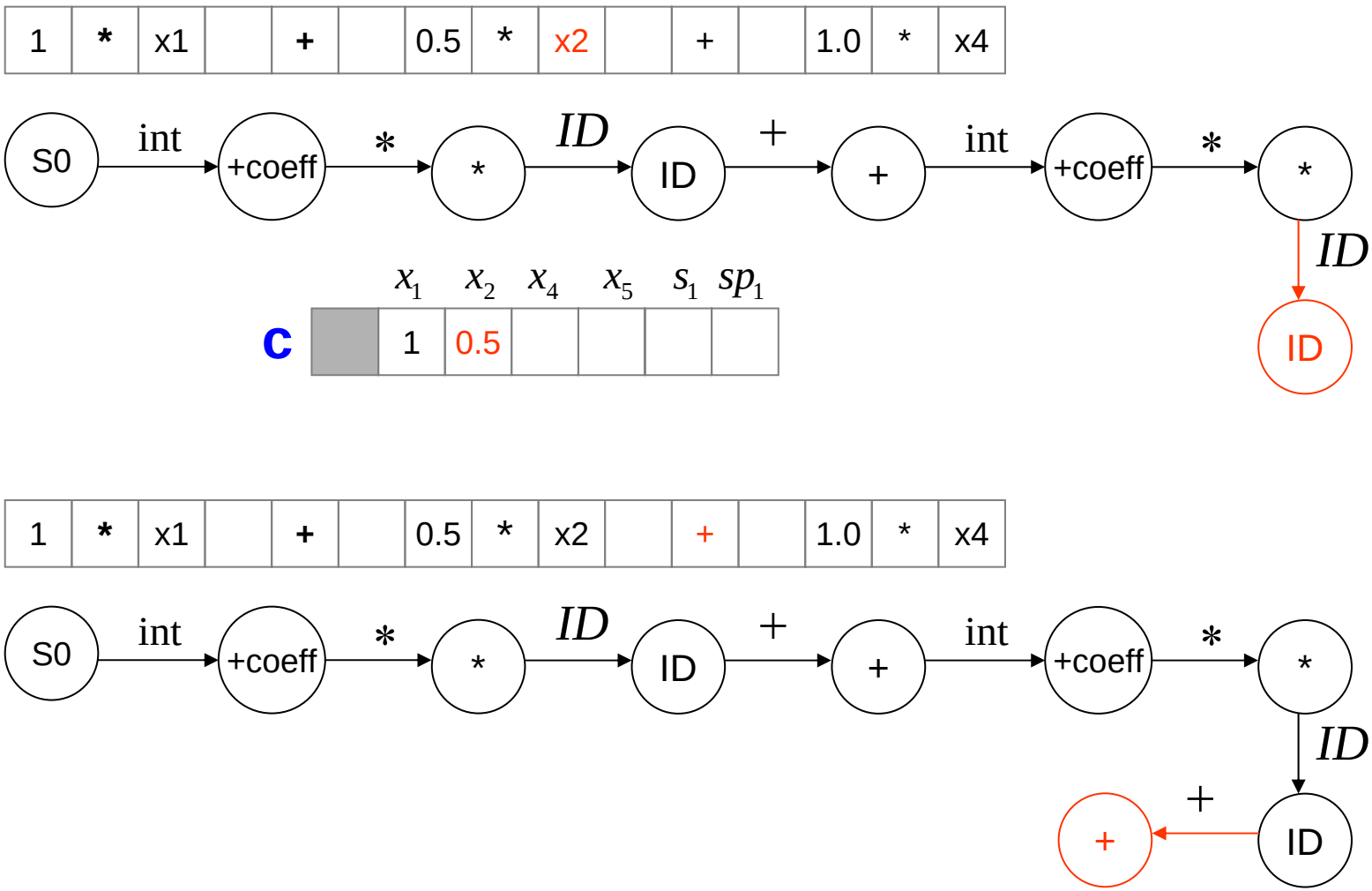
1	*	x1		+		0.5	*	x2		+		1.0	*	x4



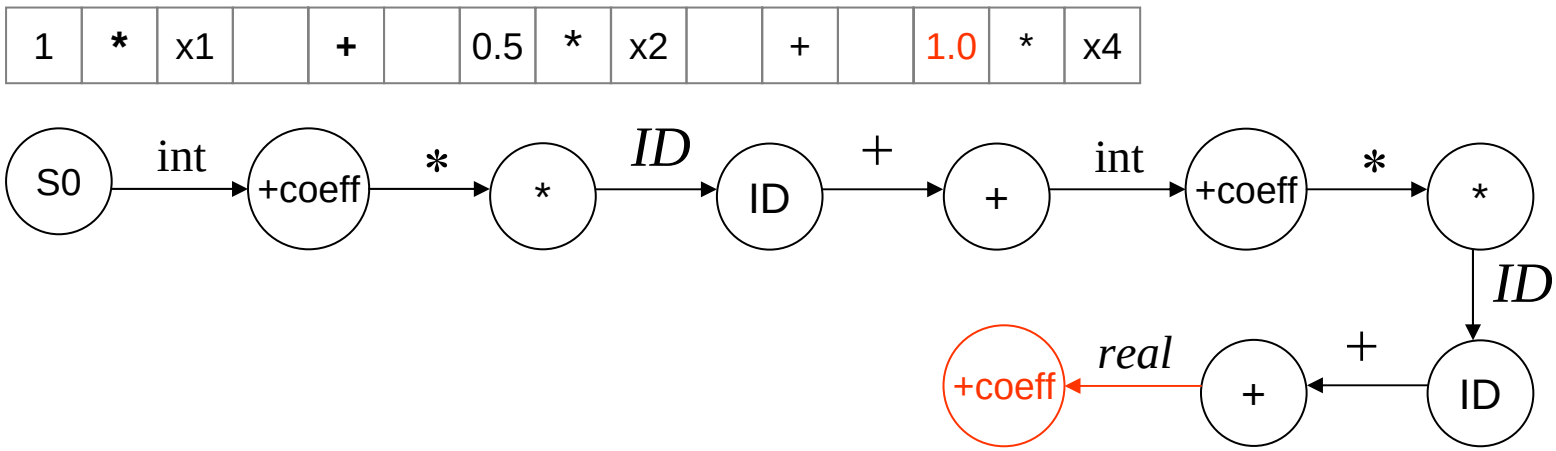
1	*	x1		+		0.5	*	x2		+		1.0	*	x4

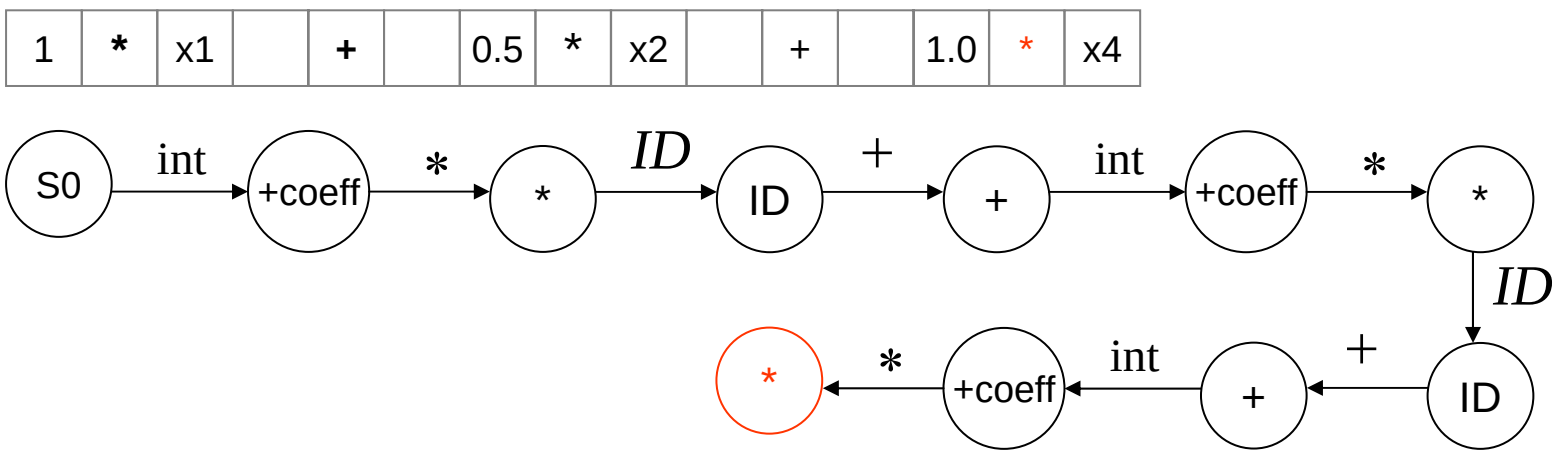


State sequence [3]

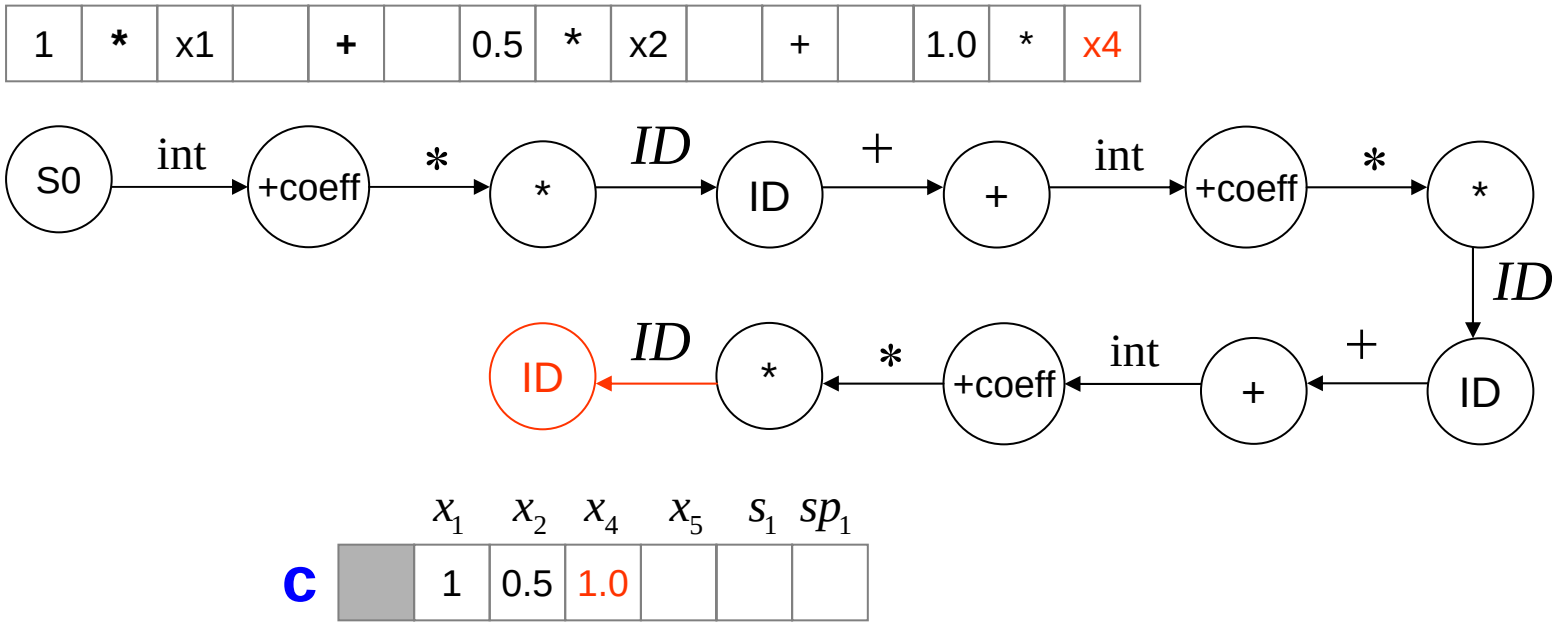


State sequence [4]





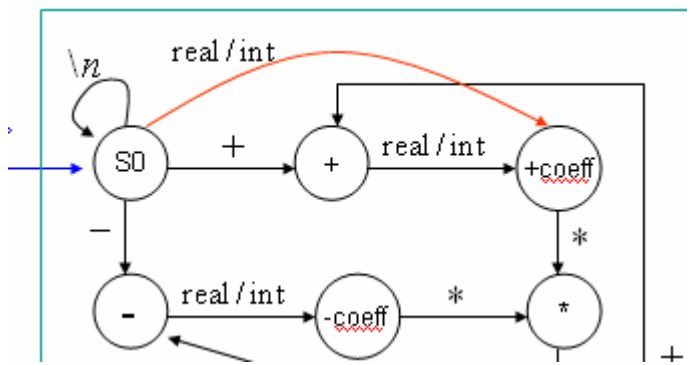
State sequence [5]



buildLP.cpp

Extract cost vector: implementation [1]

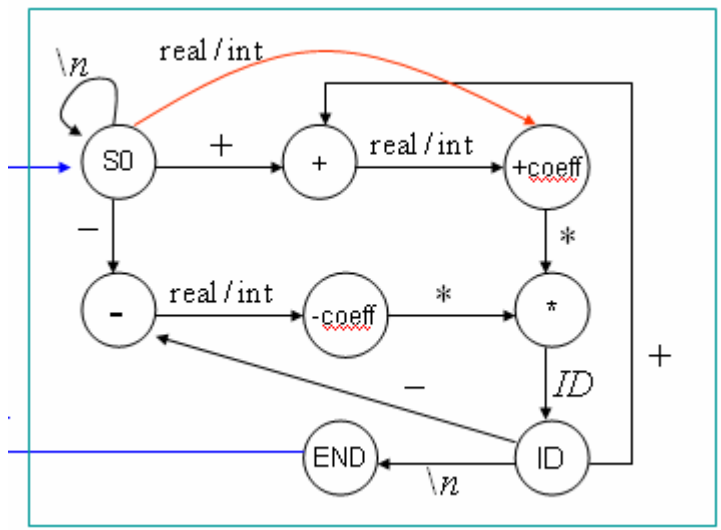
```
void  findCostVector(symbolTable* table, double *c )
{
    enum stateVar { INIT, PLUS, MINUS, PLUS_COEFF,
           MINUS_COEFF, STAR, ID, TERMINATE, TRAP } ;
    stateVar state = INIT ;
    int token ;
    double coeff ;
    int  var_index;

    while( 1 ){
        if ( TERMINATE == state ){ break ; }
        switch(state){
        case INIT :
            token = yylex() ;
            if ( '+' == token ){
                state = PLUS ;
            } else if ( '-' == token ){
                state = MINUS ;
            }else if ((INTEGER_ == token) ||
                      (REAL_ == token)   ){
                state = PLUS_COEFF ;
            }else if ( '\n' == token ){
                state = INIT ;
            }else{
                state = TRAP ;
            }
            break ;

        case PLUS :
            token = yylex() ;
            if ( (INTEGER_ == token) ||
                 (REAL_ == token)   ){
                state = PLUS_COEFF ;
            }else{
                state = TRAP ;
            }
            break ;
```



Extract cost vector: implementation [2]

buildLP.cpp

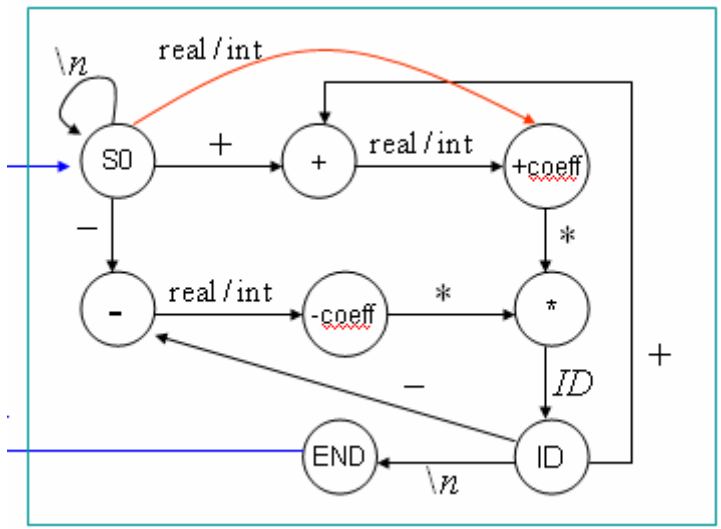
```
        case MINUS :
            token = yylex() ;
            if ( (INTEGER_ == token) ||
                 (REAL_ == token)   ){
                state = MINUS_COEFF ;
            }else{
                state = TRAP ;
            }
            break ;
        case PLUS_COEFF :
            coeff = atof(yytext) ;
            token = yylex() ;
            if ( '*' == token ){
                state = STAR ;
            }else{
                state = TRAP ;
            }
            break ;
        case MINUS_COEFF :
            coeff = -atof(yytext) ;
            token = yylex() ;
            if ( '*' == token ){
                state = STAR ;
            }else{
                state = TRAP ;
            }
            break ;
```



Extract cost vector: implementation [3]

buildLP.cpp

```
        case STAR :
            token = yylex() ;
            if (IDENTIFIER_ == token){
                state = ID ;
            }else{
                state = TRAP ;
            }
            break ;
        case ID :
            var_index = symTable_lookup( table, yytext ) ;
            assert( var_index ) ;
            c[var_index] = coeff ;
            token = yylex() ;
            if ( '+' == token ){
                state = PLUS ;
            } else if ( '-' == token ){
                state = MINUS ;
            }else if ('\n' == token ){
                state = TERMINATE ;
            }else{
                state = TRAP ;
            }
            break ;
        case TRAP :
            printf("Error: wrong format\n");
            exit(1) ;
            break ;
        default:
            printf("Error: wrong state = %d\n", state);
            exit(1) ;
            break;
        }
    }// for each token
}
```

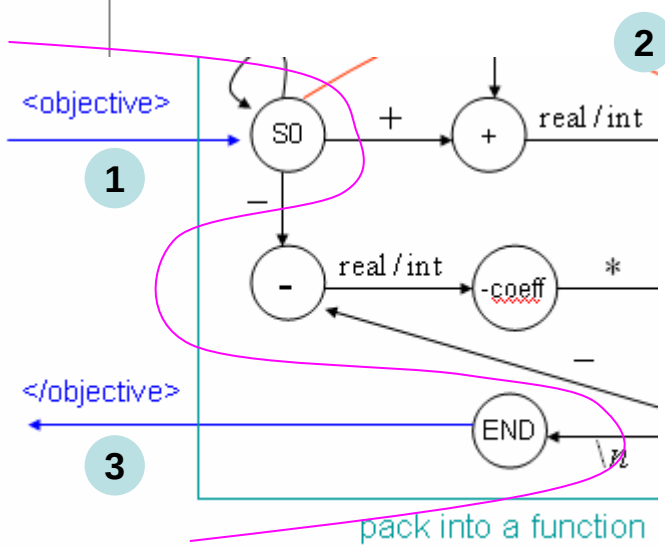


Extract cost vector: implementation [4]

buildLP.cpp

```
void   verifyToken( int target )
{
    int token ;
    while ( token = yylex() ){
        if ( '\n' == token){
            continue ;
        }else if ( target == token ){
            return ;
        }else{
            printf("Error: wrong format, token = %d", token);
            exit(1) ;
        }
    }// for each token
}

void  buildLP( char* fname, int m, int n, symbolTable* table, double *c,
               double **GElist, double **LElist, double **EQlist  )
{
    yyin = fopen( fname, "r") ;
    assert( yyin ) ;

    verifyToken( OBJECTIVE_ ) ;
    findCostVector( table, c ) ;
    verifyToken( END_OBJECTIVE_ ) ;

    fclose( yyin ) ;
}
```

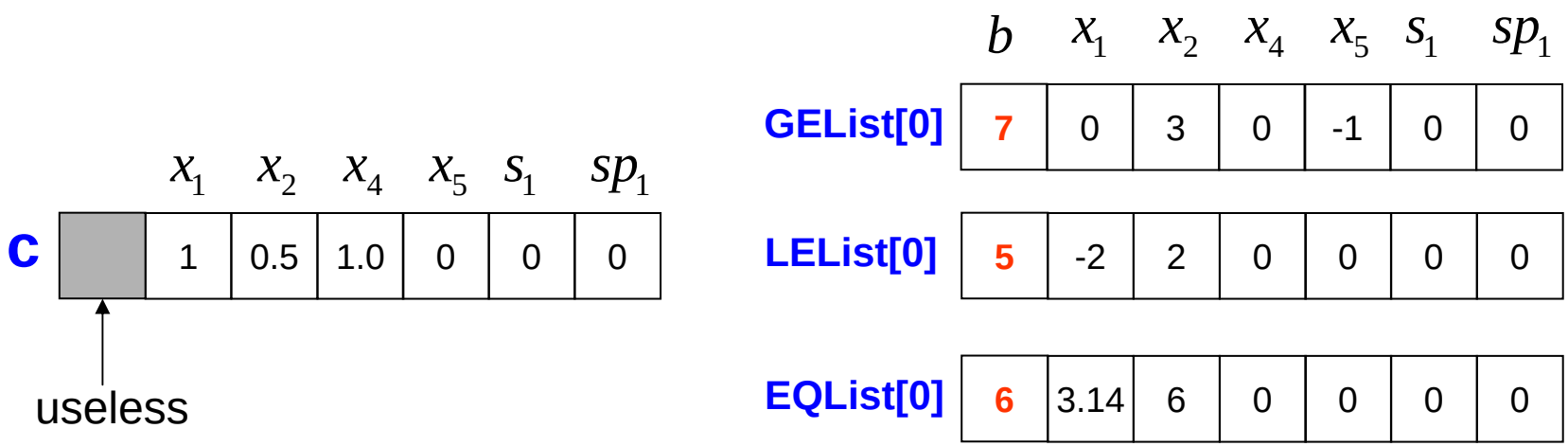
1
2
3



Allocate vector c and constraint matrix [1]

- Index of array in C-language starts from zero, however index of vector starts from 1, hence ***c[0]*** is useless.

- We record Ax =b, Ax<=b and Ax >=b respectively.
  Ax = b : use ***double** EQlist*** to represent A
  Ax >=b : use ***double** GElist*** to represent A
  Ax <=b : use ***double** LElist*** to represent A
  we record right hand side vector b in *EQlist[i][0], GElist[i][0]* and *LElist[i][0]* respectively.



Allocate vector c and constraint matrix [2]

```
void  alloc_system_eq( int numVar, int numOfGE, int numOfLE, int numOfEQ, int *m, int *n,
            double **c, double ***GElist, double ***LElist, double ***EQlist )
{
    int i ;

    *n = numVar + numOfGE + numOfLE ;
    *m = numOfGE + numOfLE + numOfEQ ;

    *c = (double*) malloc(sizeof(double)*(*n+1) ) ;
    assert( *c ) ;
    memset( *c, 0, sizeof(double)*(*n+1) ) ;

    if ( 0 < numOfGE ){
        *GElist = (double**) malloc(sizeof(double*)*numOfGE ) ;
        assert( *GElist ) ;
        for( i = 0 ; i < numOfGE ; i++){
            (*GElist)[i] = (double*) malloc(sizeof(double)*(*n+1) ) ;
            assert( (*GElist)[i] ) ;
            memset( (*GElist)[i], 0, sizeof(double)*(*n+1) ) ;
        }
    }else{
        *GElist = NULL ;
    }
    if ( 0 < numOfLE ){
        *LElist = (double**) malloc(sizeof(double*)*numOfLE ) ;
        assert( *LElist ) ;
        for( i = 0 ; i < numOfLE ; i++){
            (*LElist)[i] = (double*) malloc(sizeof(double)*(*n+1) ) ;
            assert( (*LElist)[i] ) ;
            memset( (*LElist)[i], 0, sizeof(double)*(*n+1) ) ;
        }
    }else{
        *LElist = NULL ;
    }
```

Two level allocation

•
•
•

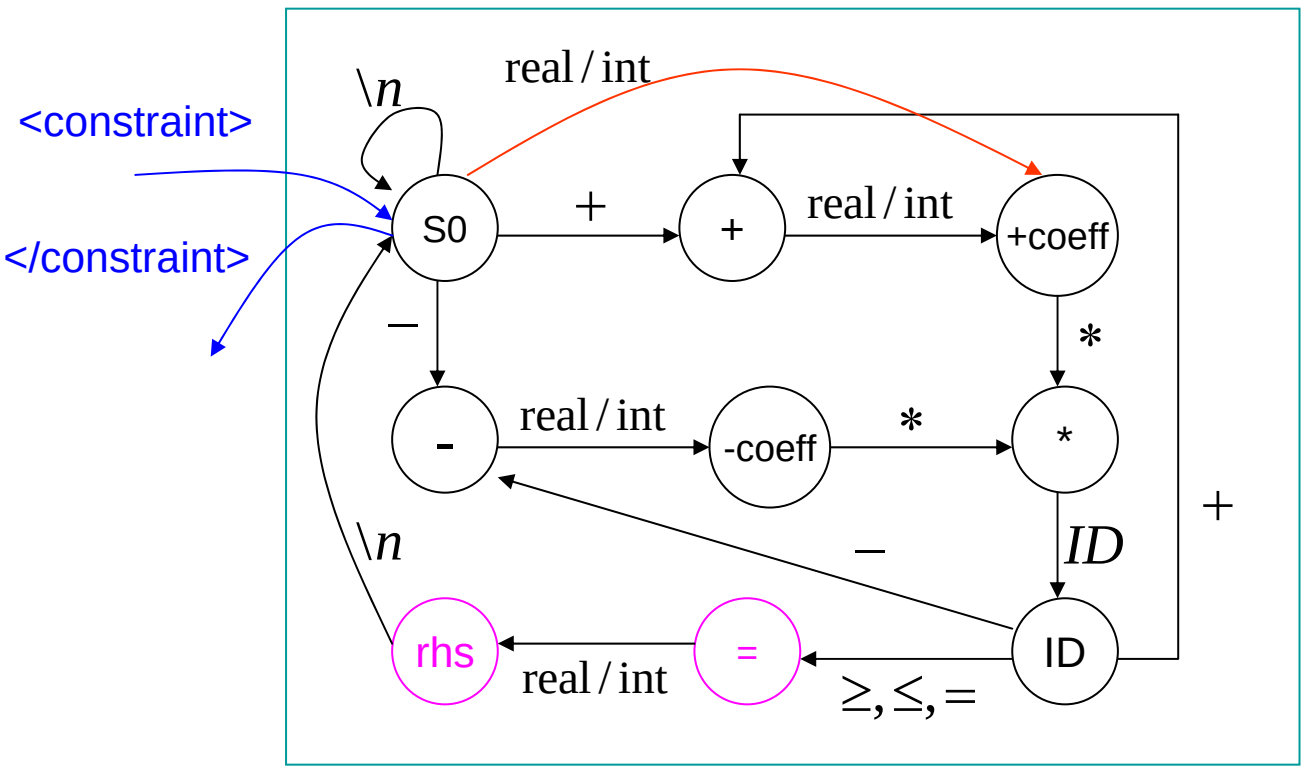
Extract constraint matrix [1]

objective:extract $Ax \geq b, Ax \leq b, Ax = b$

configure.txt

```
1
2 // minimize z = C' *x
3 <objective>
4   1*x1 + 0.5*x2 + 1.0*x4
5 </objective>
6
7 // subject to Ax <= b
8 // x >= 0 is implicit
9 <constraint>
10   -2*x1 + 2.*x2 <= 5.0
11   3*x2 - 1*x5 >= 7
12   6*x2 + 3.14*x1 = 6
13 </constraint>
14
```

Flow chart (finite state machine)



pack into a function

Implementation note

- We don’t know which equation the coefficient belongs until token >=, <= or = is extracted. Hence we need a temperary array, called *temp* to store coeffient read in *+coeff* or *–coeff* state and right hand side value read in *rhs* state, also a *flag* (旗標) to distinguish what kind of equation we encounter.

- In *+coeff* or *–coeff* state, we record coefficient we read

- In *ID* state, we lookup index of variable in symbol table and set coefficent to array *temp* in proper location.

- In *rhs* state, we set right hand side value to *temp[0]* and copy whole array temp to *GElist*, *LElist* or *EQList*

Add slack and surplus variable

```
1
2 // minimize z = C' *x
3 <objective>
4   1*x1 + 0.5*x2 + 1.0*x4
5 </objective>
6
7 // subject to Ax <= b
8 // x >= 0 is implicit
9 <constraint>
10   -2*x1 + 2.*x2 <= 5.0
11   3*x2 - 1*x5 >= 7
12   6*x2 + 3.14*x1 = 6
13 </constraint>
14
```

x_1	x_2	x_4	x_5	s_1	sp_1		
-2	2	0	0	0	0	$\leq$	5
0	3	0	-1	0	0	$\geq$	7
3.14	6	0	0	0	0	$=$	6

slack variable →

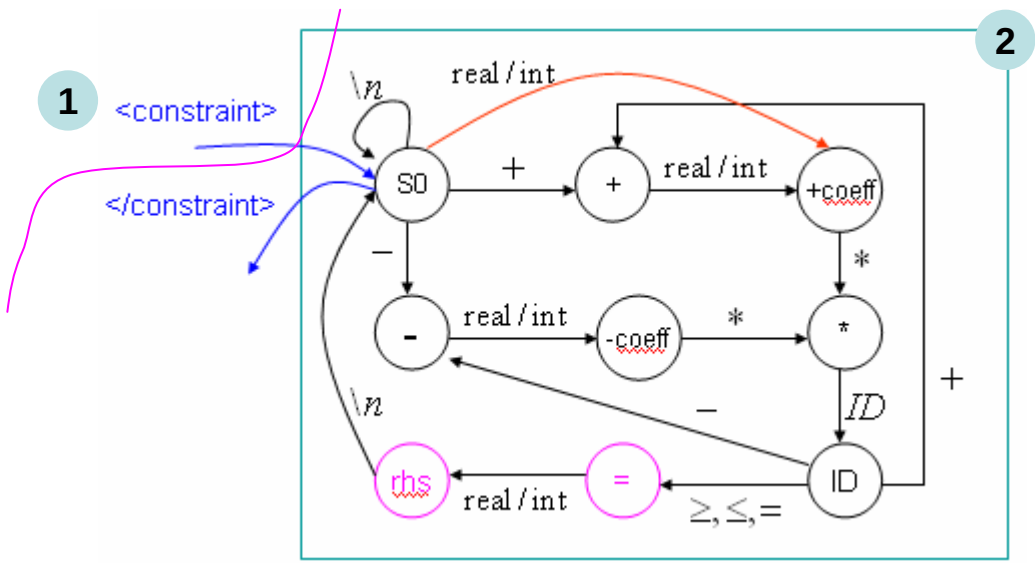
surplus variable →

x_1	x_2	x_4	x_5	s_1	sp_1		
-2	2	0	0	***1***	0	$=$	5
0	3	0	-1	0	***-1***	$=$	7

Extract cost vector and constraint matrix

buildLP.cpp

```
void  buildLP( char* fname, int m, int n, symbolTable* table, double *c,
               double **GElist, double **LElist, double **EQlist  )
{
    yyin = fopen( fname, "r") ;
    assert( yyin ) ;

        verifyToken( OBJECTIVE_ ) ;
    findCostVector( table, c ) ;
        verifyToken( END_OBJECTIVE_ ) ;

    1
        verifyToken( CONSTRAINT_ ) ;
    2 findConstraint(table, m, n, GElist, LElist, EQlist ) ;
        // END_CONSTRAINT_  has been verified in "findConstraint"

    fclose( yyin ) ;
}
```



main.cpp

```
#include <stdio.h>
#include <string.h>
#include "symbol.h"
#include "buildLP.h"

#define  DEFAULT_INPUT_FILE  "configure.txt"

int main(int argc, char* argv[])
{
    char  inFile[ 128 ] ; // input file name
    symbolTablePtr  tablePtr ; // symbol table
    int numOfGE, numOfLE, numOfEQ ;

    ++argv ;
    --argc ; // skip over command

#if defined(_WIN32) || defined(__WIN32__)
    strcpy( inFile, DEFAULT_INPUT_FILE ) ;
#else
    if ( 0 < argc ){
        strcpy( inFile, argv[0] ) ;
    }else{
        printf("Error: please specify input file\n");
        exit(1) ;
    }
#endif

// step 1: setup symbol table and find number of <= , >= and = equation
    extractSymbol( inFile, &tablePtr, &numOfGE, &numOfLE, &numOfEQ ) ;
    symTable_display( stdout, tablePtr ) ;
    printf("number of >= equations = %d\n", numOfGE );
    printf("number of <= equations = %d\n", numOfLE );
    printf("number of  = equations = %d\n", numOfEQ );
```

```
// step 2: allocate cost vector and constraint matrix
    int m, n ;
    double *c ;
    double **GElist ;
    double **LElist ;
    double **EQlist ;
    alloc_system_eq( tablePtr->size, numOfGE, numOfLE, numOfEQ,
            &m, &n,
            &c, &GElist, &LElist, &EQlist ) ;

    printf("dimension of LP (m,n) = (%d,%d)\n", m, n );

// step 3: build up constraint matrix
    buildLP( inFile, m, n, tablePtr, c, GElist, LElist, EQlist ) ;
    int i, j ;
// show cost vecotr c
    printf("c =  ");
    for(i=1 ; i <=n ; i++ ){
        printf("%5.2f  ", c[i] );
    }
    printf("\n");

// show constraint Ax = b
    printf("configuration of <= equation after adding slack var\n");
    for(i=0 ; i < numOfLE ; i++ ){
        for(j=1; j <= n ; j++){
            printf("%5.2f  ", LElist[i][j] );
        }
        printf(" = %5.2f\n", LElist[i][0] );
    }
    printf("configuration of >= equation after adding surplus var\n");
    for(i=0 ; i < numOfGE ; i++ ){
        for(j=1; j <= n ; j++){
            printf("%5.2f  ", GElist[i][j] );
        }
        printf(" = %5.2f\n", GElist[i][0] );
    }
    printf("configuration of = equation\n");
    for(i=0 ; i < numOfEQ ; i++ ){
        for(j=1; j <= n ; j++){
            printf("%5.2f  ", EQlist[i][j] );
        }
        printf(" = %5.2f\n", EQlist[i][0] );
    }
    return 0 ;
}
```

```
Warning: symbol x1 is already in symbol table
Warning: symbol x2 is already in symbol table
Warning: symbol x2 is already in symbol table
Warning: symbol x2 is already in symbol table
Warning: symbol x1 is already in symbol table
Total number of symbol = 4
table[1] = x1
table[2] = x2
table[3] = x4
table[4] = x5
number of >= equations = 1
number of <= equations = 1
number of  = equations = 1
dimension of LP (m,n) = (3,6)
c =   1.00   0.50   1.00   0.00   0.00   0.00
configuration of <= equation after adding slack var
-2.00   2.00   0.00   0.00   1.00   0.00    =  5.00
configuration of >= equation after adding surplus var
 0.00   3.00   0.00  -1.00   0.00  -1.00    =  7.00
configuration of = equation
 3.14   6.00   0.00   0.00   0.00   0.00    =  6.00
Press any key to continue_
```

Exercise 1: lack coefficient

- If we regard *x1* as *1*x1*, can you modify finite state machine to accept this new rule?

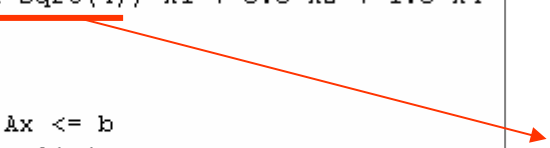
```
 1
 2 // minimize z = C' *x
 3 <objective>
 4
 5   1*x1 + 0.5*x2 + 1.0*x4
 6
 7 </objective>
 8
 9 // subject to Ax <= b
10 // x >= 0 is implicit
11 <constraint>
12   -2*x1 + 2.*x2 <= 5.0
13    3*x2 - 1*x5 >= 7
14   6*x2 + 3.14*x1 = 6
15 </constraint>
16
```

```
 1
 2 // minimize z = C' *x
 3 <objective>
 4
 5   x1 + 0.5*x2 + 1.0*x4
 6
 7 </objective>
 8
 9 // subject to Ax <= b
10 // x >= 0 is implicit
11 <constraint>
12   -2*x1 + 2.*x2 <= 5.0
13    3*x2 - 1*x5 >= 7
14   6*x2 + 3.14*x1 = 6
15 </constraint>
16
```

Exercise 2: expression evaluation

- In this work, we assume coefficient is a number, *NOT* an expression. If we remove this assumption, say that coefficient can be an expression. How to deal with?
  Hint: think about three-step solver
  step 1: use ***RPN (Reverse Polish Notation)*** technique to compute expression to a number
  step 2: construct symbol table
  step 3: setup cost vector and constraint matrix

```
 1
 2 // minimize z = C' *x
 3 <objective>
 4
 5   (3.0/2 + 5.1-sqrt(4))*x1 + 0.5*x2 + 1.0*x4
 6
 7 </objective>
 8
 9 // subject to Ax <= b
10 // x >= 0 is implicit
11 <constraint>
12   -2*x1 + 2.*x2 <= 5.0
13    3*x2 - 1*x5 >= 7
14   6*x2 + 3.14*x1 = 6
15 </constraint>
16
```



expression

Exercise 3: macro substitution

- Usually, we like to use macro instead of number explicitly, for example, we may define *pi=3.1415926* and then use macro *pi* in coeffient computation. Two reasons for macro substitution
  1. save space: since pi is 2 characters but 3.1415926 is 9 characters
  2. save time: we may use *pi* several times, if we use 3.1415926 every time when we use *pi*, then it is clumsy.

```
 1
 2 #define   pi  3.1415926
 3
 4 // minimize z = C' *x
 5 <objective>
 6
 7   ( pi /2 + 5.1-sqrt(4))*x1 + 0.5*x2 + 1.0*x4
 8
 9 </objective>
10
11 // subject to Ax <= b
12 // x >= 0 is implicit
13 <constraint>
14   -2*x1 + 2.*x2 <= 5.0
15    3*x2 - 1*x5 >= 7
16   6*x2 + 3.14*x1 = 6
17 </constraint>
18
```

Macro definition