CMPS104A: Bison Tutorial

Christopher Schuster

University of California, Santa Cruz

November 7, 2013

Outline

S-Expressions

()

S-Expressions

```
()
(a)
```

S-Expressions

```
()
(a)
(a b)
```

S-Expressions

```
()
(a)
(a b)
```

```
(a (b))
```

S-Expressions

```
()
(a)
(a b)
```

```
(a (b))
(a (b (c)))
```

S-Expressions

```
()
(a)
(a b)
```

```
(a (b))
(a (b (c)))
(a (b (c) ()))
```

S-Expressions Grammar

S-Expressions Grammar

atom $\rightarrow$ ‘a’ | ‘b’ | ‘c’

S-Expressions Grammar

atom $\rightarrow$ ‘a’ | ‘b’ | ‘c’
list $\rightarrow$ ‘(’ [list | atom]... ‘)’

.

S-Expressions Grammar

atom → ‘a’ | ‘b’ | ‘c’
list → ‘(’ [list | atom]... ‘)’

Example: `(a (b (c) ()))`

.

S-Expressions Grammar

atom → ‘a’ | ‘b’ | ‘c’
list → ‘(’ [list | atom]... ‘)’

Example: `(a (b (c) ()))`

( )

list

.

S-Expressions Grammar

atom → ‘a’ | ‘b’ | ‘c’
list → ‘(’ [list | atom]... ‘)’

Example: `(a (b (c) ()))`

`(a )`

list

|

‘a’

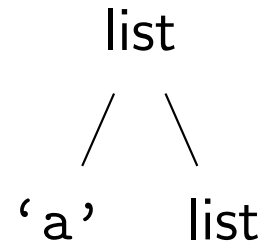
.

S-Expressions Grammar

atom → ‘a’ | ‘b’ | ‘c’
list → ‘(’ [list | atom]... ‘)’

Example: `(a (b (c) ()))`

`(a (  ))`



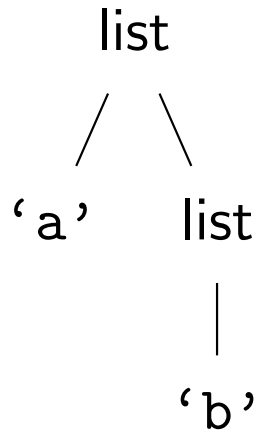
.

S-Expressions Grammar

atom → ‘a’ | ‘b’ | ‘c’
list → ‘(’ [list | atom]... ‘)’

Example: `(a (b (c) ()))`

`(a (b ))`



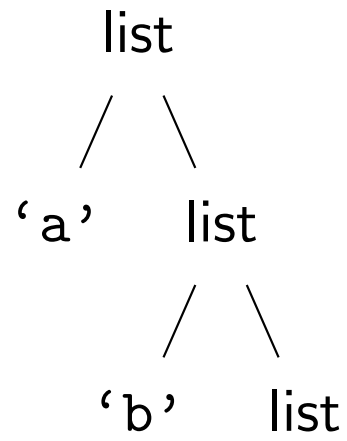
.

S-Expressions Grammar

atom → ‘a’ | ‘b’ | ‘c’
list → ‘(’ [list | atom]... ‘)’

Example: `(a (b (c) ()))`

`(a (b ( )  ))`



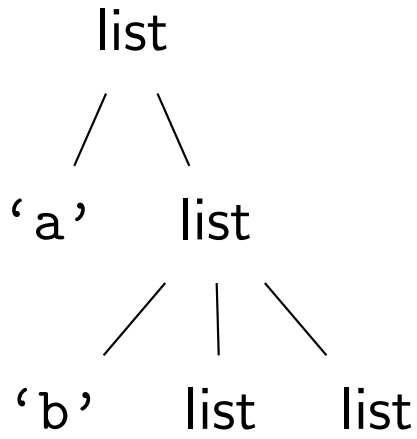
.

S-Expressions Grammar

atom → ‘a’ | ‘b’ | ‘c’
list → ‘(’ [list | atom]... ‘)’

Example: `(a (b (c) ()))`

`(a (b ( ) ()))`



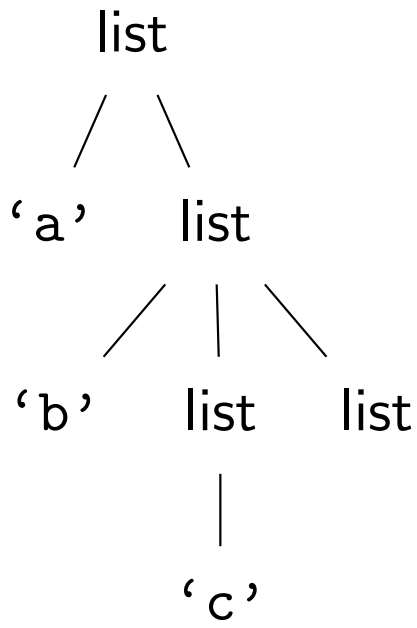
.

S-Expressions Grammar

atom → ‘a’ | ‘b’ | ‘c’
list → ‘(’ [list | atom]... ‘)’

Example: `(a (b (c) ()))`

`(a (b (c) ()))`



.

Parsing with bison

atom $\rightarrow$ ‘a’ | ‘b’ | ‘c’
list $\rightarrow$ ‘(’ [list | atom]... ‘)’

Parsing with bison

atom → ‘a’ | ‘b’ | ‘c’
list → ‘(’ [list | atom]... ‘)’

```
atom : 'a' | 'b' | 'c'
     ;
```

Parsing with bison

atom → ‘a’ | ‘b’ | ‘c’
list → ‘(’ [list | atom]... ‘)’

```
atom : 'a' | 'b' | 'c'
     ;
list : '(' members ')'
     ;
```

Parsing with bison

atom → ‘a’ | ‘b’ | ‘c’
list → ‘(’ [list | atom]... ‘)’

```
atom : 'a' | 'b' | 'c'
     ;
list : '(' members ')'
     ;
members : members list
        | members atom
        |
        ;
```

Understanding states `bison --report=state`

```
list : '(' members ')'
members : members list
        | members atom
        |
atom : 'a' | 'b' | 'c'
```

Understanding states `bison --report=state`

```
list : '(' members ')'
members : members list
        | members atom
        |
atom : 'a' | 'b' | 'c'
```

state 0

```
list : . '(' members ')'
```

Understanding states `bison --report=state`

```
list : '(' members ')'
members : members list
        | members atom
        |
atom : 'a' | 'b' | 'c'
```

state 0

```
list : . '(' members ')'
```

'(' → shift, and go to state 1

Understanding states `bison --report=state`

state 1

Understanding states `bison --report=state`

state 1

```
list :  '(' .  members ')'
```

Understanding states `bison --report=state`

state 1

```
list :   '(' .   members ')'
members :   .   members list
        |   .   members atom
        |   .
```

Understanding states `bison --report=state`

state 1

```
list :   '(' .   members ')'
members :   .   members list
        |   .   members atom
        |   .
```

members → go to state 3

Understanding states `bison --report=state`

state 1

```
list :   '(' .   members ')'
members :   .   members list
        |   .   members atom
        |   .
```

members → go to state 3
$default → reduce using rule 4 (members)

Understanding states `bison --report=state`

state 3

```
list :  '(' members .  ')'
```

Understanding states `bison --report=state`

state 3

```
list :  '(' members .  ')'

members :  members .  list
```

Understanding states `bison --report=state`

state 3

```
list :  '(' members .  ')'
     |  .  '(' members ')'
members :  members .  list
```

Understanding states `bison --report=state`

state 3

```
list :  '(' members .  ')'
     |  .  '(' members ')'
members :  members .  list
        | members .  atom
```

Understanding states `bison --report=state`

state 3

```
list :  '(' members .  ')'
     |  .  '(' members ')'
members :  members .  list
        | members .  atom
atom :  .  'a'
     |  .  'b'
     |  .  'c'
```

Understanding states `bison --report=state`

state 3

```
list :   '(' members .   ')'
       |   .   '(' members ')'
members :   members .   list
          | members .   atom
atom :   .   'a'
       |   .   'b'
       |   .   'c'
```

'(' $\rightarrow$ shift, and go to state 1

Understanding states `bison --report=state`

state 3

```
list :   '(' members .   ')'
       |   .   '(' members ')'
members :   members .   list
          | members .   atom
atom :   .   'a'
       |   .   'b'
       |   .   'c'
```

'(' → shift, and go to state 1

')' → shift, and go to state 5

Understanding states `bison --report=state`

state 3

```
list :   '(' members .   ')'
       |   .   '(' members ')'
members :   members .   list
          | members .   atom
atom :   .   'a'
       |   .   'b'
       |   .   'c'
```

'(' → shift, and go to state 1
')' → shift, and go to state 5
'a' → shift, and go to state 6
'b' → shift, and go to state 7
'c' → shift, and go to state 8

Understanding states `bison --report=state`

state 3

```
list :   '(' members .   ')'
       |   .   '(' members ')'
members :   members .   list
          | members .   atom
atom :   .   'a'
       |   .   'b'
       |   .   'c'
```

'(' → shift, and go to state 1
')' → shift, and go to state 5
'a' → shift, and go to state 6
'b' → shift, and go to state 7
'c' → shift, and go to state 8

list → go to state 9

Understanding states `bison --report=state`

state 3

```
list :  '(' members .  ')'
     |  .  '(' members ')'
members :  members .  list
        | members .  atom
atom :  .  'a'
     |  .  'b'
     |  .  'c'
```

'(' → shift, and go to state 1
')' → shift, and go to state 5
'a' → shift, and go to state 6
'b' → shift, and go to state 7
'c' → shift, and go to state 8

list → go to state 9
atom → go to state 10

Shift/Reduce conflicts

```
E : E '+' E
  | '1'
```

Shift/Reduce conflicts

```
E : E '+' E
  | '1'
```

state 5

```
E : E . '+' E
```

Shift/Reduce conflicts

```
E : E '+' E
  | '1'
```

state 5

```
E : E . '+' E
  | E '+' E .
```

Shift/Reduce conflicts

```
E : E '+' E
  | '1'
```

state 5

```
E : E . '+' E
  |   E '+' E .
```

'+' $\rightarrow$ shift, and go to state 4

Shift/Reduce conflicts

```
E : E '+' E
  | '1'
```

state 5

```
E : E . '+' E
  |   E '+' E .
```

'+' $\rightarrow$ shift, and go to state 4

'+' $\rightarrow$ reduce, using rule 1

Shift/Reduce conflicts

```
E : E '+' E
  | '1'
```

state 5

```
E : E . '+' E
  |   E '+' E .
```

'+' → shift, and go to state 4

'+' → reduce, using rule 1

$default → reduce, using rule 1

Shift/Reduce conflicts

```
%left '+'
E : E '+' E
  | '1'
```

Shift/Reduce conflicts

```
%left '+'
E : E '+' E
  | '1'
```

state 5

```
E : E . '+' E
  | E '+' E .
```

Shift/Reduce conflicts

```
%left '+'
E : E '+' E
  | '1'
```

state 5

```
E : E . '+' E
  | E '+' E .
```

$default → reduce, using rule 1

Shift/Reduce conflicts

```
%right '+'
E : E '+' E
  | '1'
```

Shift/Reduce conflicts

```
%right '+'
E : E '+' E
  | '1'
```

state 5

```
E : E . '+' E
  |   E '+' E .
```

Shift/Reduce conflicts

```
%right '+'
E : E '+' E
  | '1'
```

state 5

```
E : E . '+' E
  |   E '+' E .
```

'+' $\rightarrow$ shift, and go to state 4
$default $\rightarrow$ reduce, using rule 1

Shift/Reduce conflicts

```
E : E '+' E
  | E '*' E
  | '1'
```

Shift/Reduce conflicts

```
E : E '+' E
  | E '*' E
  | '1'
```

state 6

```
E : E . '+' E
  |   E '+' E .
  |   E . '*' E
```

Shift/Reduce conflicts

```
E : E '+' E
  | E '*' E
  | '1'
```

state 6

```
E : E . '+' E
  |   E '+' E .
  |   E . '*' E
```

'+' → shift, and go to state 4
'*' → shift, and go to state 5
'*' → reduce, using rule 1
$default → reduce, using rule 1

Shift/Reduce conflicts

```
%left '+'
%left '*'
E : E '+' E
  | E '*' E
  | '1'
```

Shift/Reduce conflicts

```
%left '+'
%left '*'
E : E '+' E
  | E '*' E
  | '1'
```

state 6

```
E : E . '+' E
  |   E '+' E .
  |   E . '*' E
```

Shift/Reduce conflicts

```
%left '+'
%left '*'
E : E '+' E
  | E '*' E
  | '1'
```

state 6

```
E : E . '+' E
  |   E '+' E .
  |   E . '*' E
```

'*' → shift, and go to state 5
$default → reduce, using rule 1

Reduce/Reduce conflicts

```
list : '(' members ')'
members : members lists
        | members atoms
        |
atoms : atom 'a' | 'b' | 'c'
      |
lists : lists list
      |
```

Reduce/Reduce conflicts

```
list : '(' members ')'
members : members lists
        | members atoms
        |
atoms : atom 'a' | 'b' | 'c'
      |
lists : lists list
      |
```

state 3

```
list : . '(' members ')'
list : '(' members . ')'
members: members . lists
       | members . atoms
```

Reduce/Reduce conflicts

```
list : '(' members ')'
members : members lists
        | members atoms
        |
atoms : atom 'a' | 'b' | 'c'
      |
lists : lists list
      |
```

state 3

```
list : . '(' members ')'
list : '(' members . ')'
members: members . lists
       | members . atoms
```

')' → reduce using rule 6 (lists)
')' → reduce using rule 10 (atoms)

Parsing with bison

```
list     : '(' members ')'
         ;
members  : members list
         | members atom
         |
         ;
atom     : 'a' | 'b' | 'c'
         ;

Input:  (a (b (c) ()))
Output:
```

Parsing with bison: Hello, World!

```
list     : '(' members ')'    {printf("A list!\n");}
         ;
members  : members list
         | members atom
         |
         ;
atom     : 'a' | 'b' | 'c'
         ;

Input:  (a (b (c) ()))
Output: A list!
        A list!
        A list!
        A list!
```

Parsing with bison: Hello, World!

```
start    : list
         ;
list     : '(' members ')'    {printf("A list!\n");}
         ;
members  : members list
         | members atom
         |
         ;
atom     : 'a' | 'b' | 'c'
         ;

Input:  (a (b (c) ()))
Output: A list!
        A list!
        A list!
        A list!
```

Parsing with bison: Counting atoms

```
start    : list                  {printf("Atoms: %d\n", $1);}
         ;
list     : '(' members ')'       {$$ = $2; }
         ;
members  : members list          {$$ = $1 + $2; }
         | members atom          {$$ = $1 + 1; }
         |                       {$$ = 0; }
         ;
atom     : 'a' | 'b' | 'c'
         ;
```

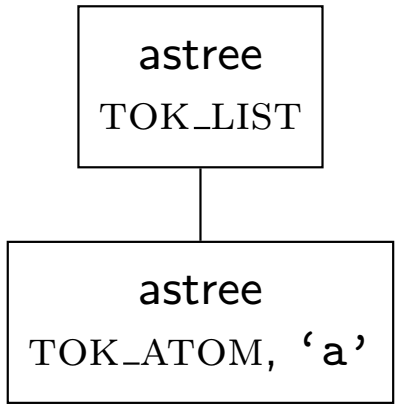
```
Input:  (a (b (c) ()))
Output: Atoms: 3
```

Creating trees with adopt()

```
astree* a = new_astree (TOK_ATOM, ..., 'a');
```

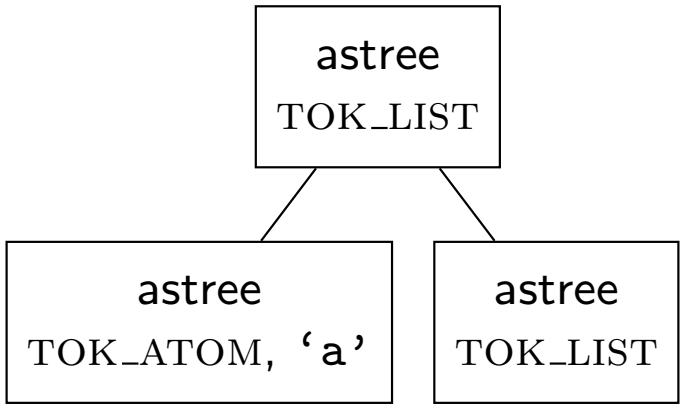
Creating trees with adopt()

```
astree* a = new_astree (TOK_ATOM, ..., 'a');
astree* x = new_astree (TOK_LIST, ...);
adopt1(x, a);
```



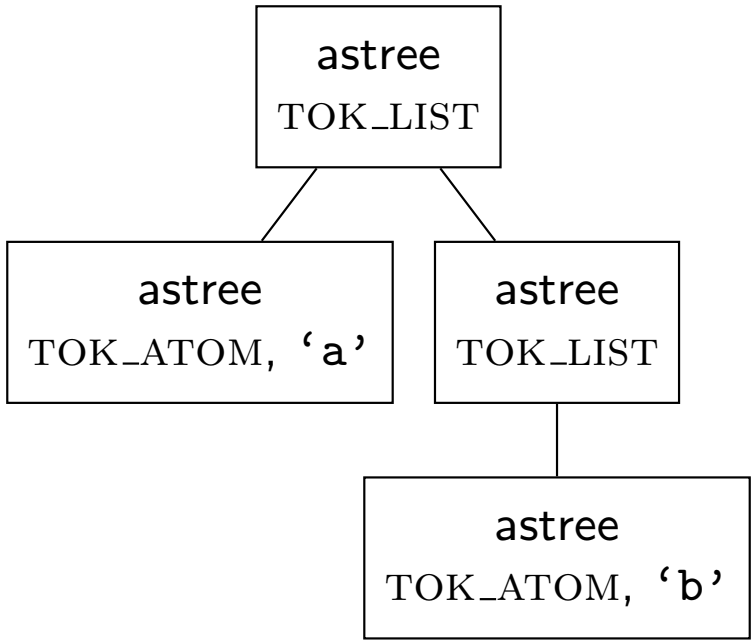
Creating trees with adopt()

```
astree* a = new_astree (TOK_ATOM, ..., 'a');
astree* x = new_astree (TOK_LIST, ...);
adopt1(x, a);
astree* y = new_astree (TOK_LIST, ...);
adopt1(x, y);
```



Creating trees with adopt()

```
astree* a = new_astree (TOK_ATOM, ..., 'a');
astree* x = new_astree (TOK_LIST, ...);
adopt1(x, a);
astree* y = new_astree (TOK_LIST, ...);
adopt1(x, y);
astree* b = new_astree (TOK_ATOM, ..., 'b');
adopt1(y, b);
```

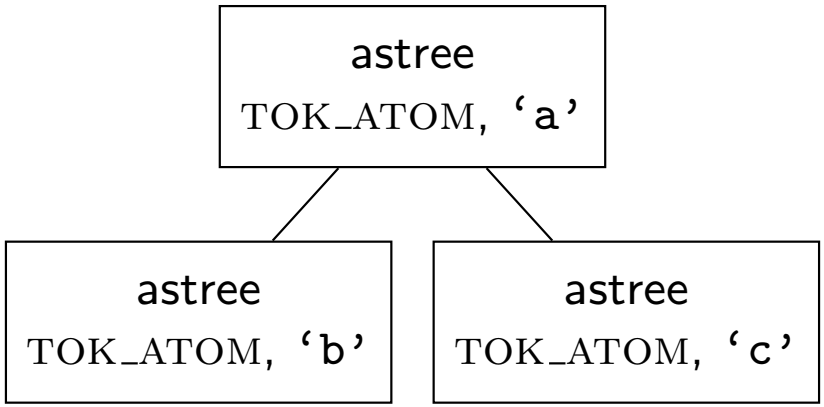


Creating trees with adopt()

```
astree* a = new_astree (TOK_ATOM, ..., 'a');
astree* b = new_astree (TOK_ATOM, ..., 'b');
astree* c = new_astree (TOK_ATOM, ..., 'c');
```

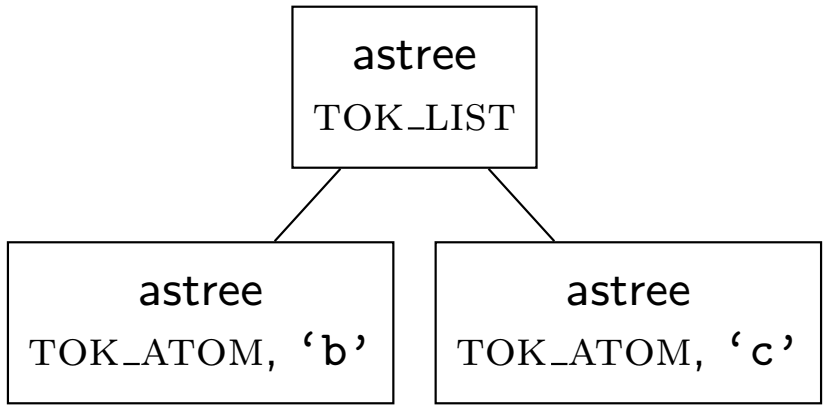
Creating trees with adopt()

```
astree* a = new_astree (TOK_ATOM, ..., 'a');
astree* b = new_astree (TOK_ATOM, ..., 'b');
astree* c = new_astree (TOK_ATOM, ..., 'c');
adopt2(a, b, c);
```



Creating trees with adopt()

```
astree* a = new_astree (TOK_ATOM, ..., 'a');
astree* b = new_astree (TOK_ATOM, ..., 'b');
astree* c = new_astree (TOK_ATOM, ..., 'c');
adopt1sym(a, b, TOK_LIST);
adopt1sym(a, c, TOK_LIST);
```



Parsing with bison: Creating a tree with adopt()

```
start    : list                 {dump($1,0);}
         ;
list     : '(' members ')'      {$$ = $2; }
         ;
members  : members list         {$$ = adopt1($1, $2); }
         | members atom         {$$ = adopt1($1, $2); }
         |                      {$$ = new_astree(TOK_LIST,""); }
         ;
atom     : 'a'                  {$$ = new_astree(TOK_ATOM,"a");}
         | 'b'                  {$$ = new_astree(TOK_ATOM,"b");}
         | 'c'                  {$$ = new_astree(TOK_ATOM,"c");}
         ;
```

Creating abstract syntax trees with classes

```
class Node {
public:
  virtual void dump(int depth);
};
```

Creating abstract syntax trees with classes

```
class Atom : public Node {
  char symbol;
public:
  Atom(char symbol) {
    this->symbol = symbol;
  }
  void dump(int depth) {
    printf ("%*satom (%c)\n", depth*2, "", this->symbol);
  }
};
```

Creating abstract syntax trees with classes

```
class List : public Node {
  std::vector<Node*> members;
public:
  List() {}
  Node* add(Node* n) {
    this->members.push_back(n);
    return this;
  }
  void dump(int depth) {
    printf ("%*slist\n", depth*2, "");
    for (int i = 0; i < this->members.size(); ++i) {
      this->members.at(i)->dump(depth+1);
    }
  }
};
```

Parsing with bison: Creating a tree with classes

```
start    : list                 {$1->dump(0);}
         ;
list     : '(' members ')'      {$$ = $2; }
         ;
members  : members list         {$$ = ((List*)$1)->add($2); }
         | members atom         {$$ = ((List*)$1)->add($2); }
         |                      {$$ = new List(); }
         ;
atom     : 'a'                  {$$ = new Atom('a');}
         | 'b'                  {$$ = new Atom('b');}
         | 'c'                  {$$ = new Atom('c');}
         ;
```

Resources

Bison manual

`http://gnu.org/software/bison/manual/bison.html`