Regex as Easy as /(abc)?/

Daniel Colson

/^.bc/

```
/^.bc/
.match?("abcdefg")
```

```
/^.bc/
.match?("abcdefg")
```

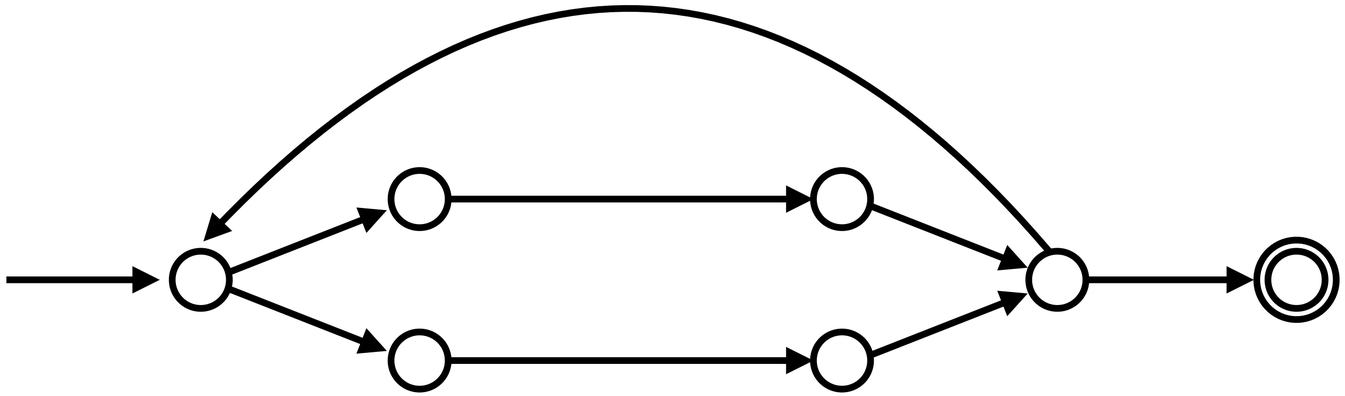
```
/^.bc([de]*f)+g|h/
```

```
/^.bc([de]*f)+g|h/
.match?("abcdefg")
```

```
/^.bc([de]*f)+g|h/
.match?("abcdefg")
```

```
/^.bc([de]*f)+g|h/
```

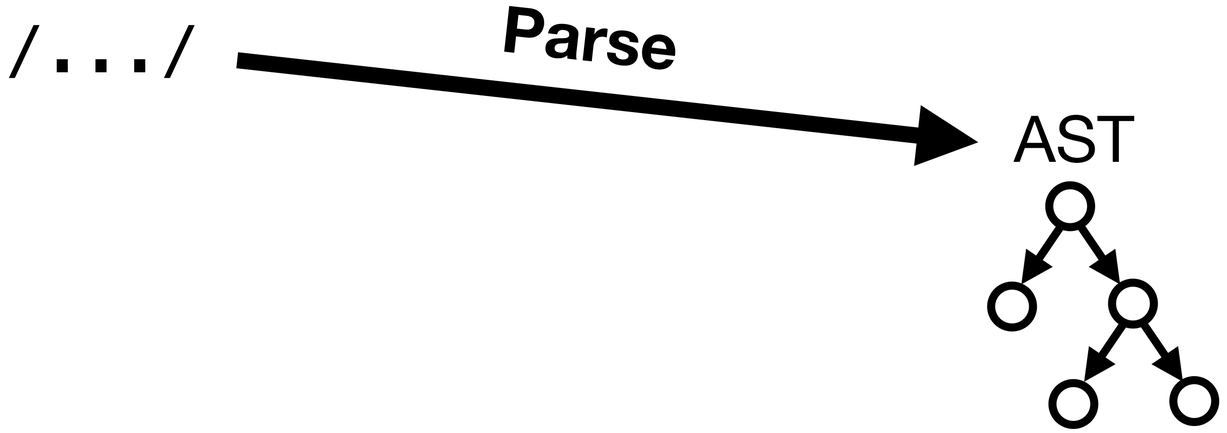
/^.bc([de]*f)+g|h/

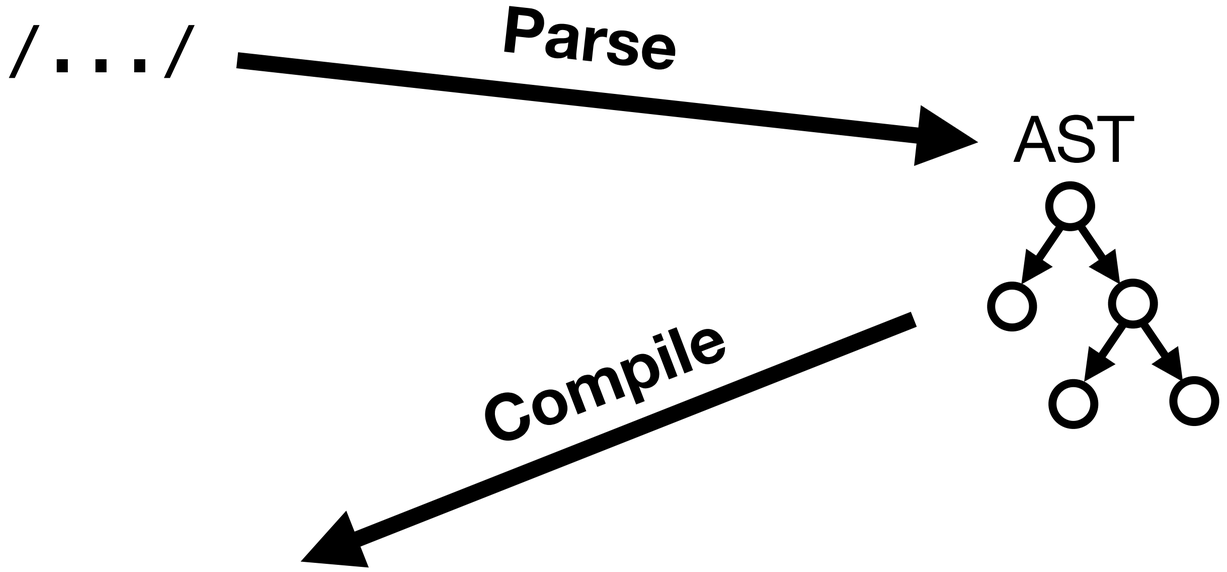


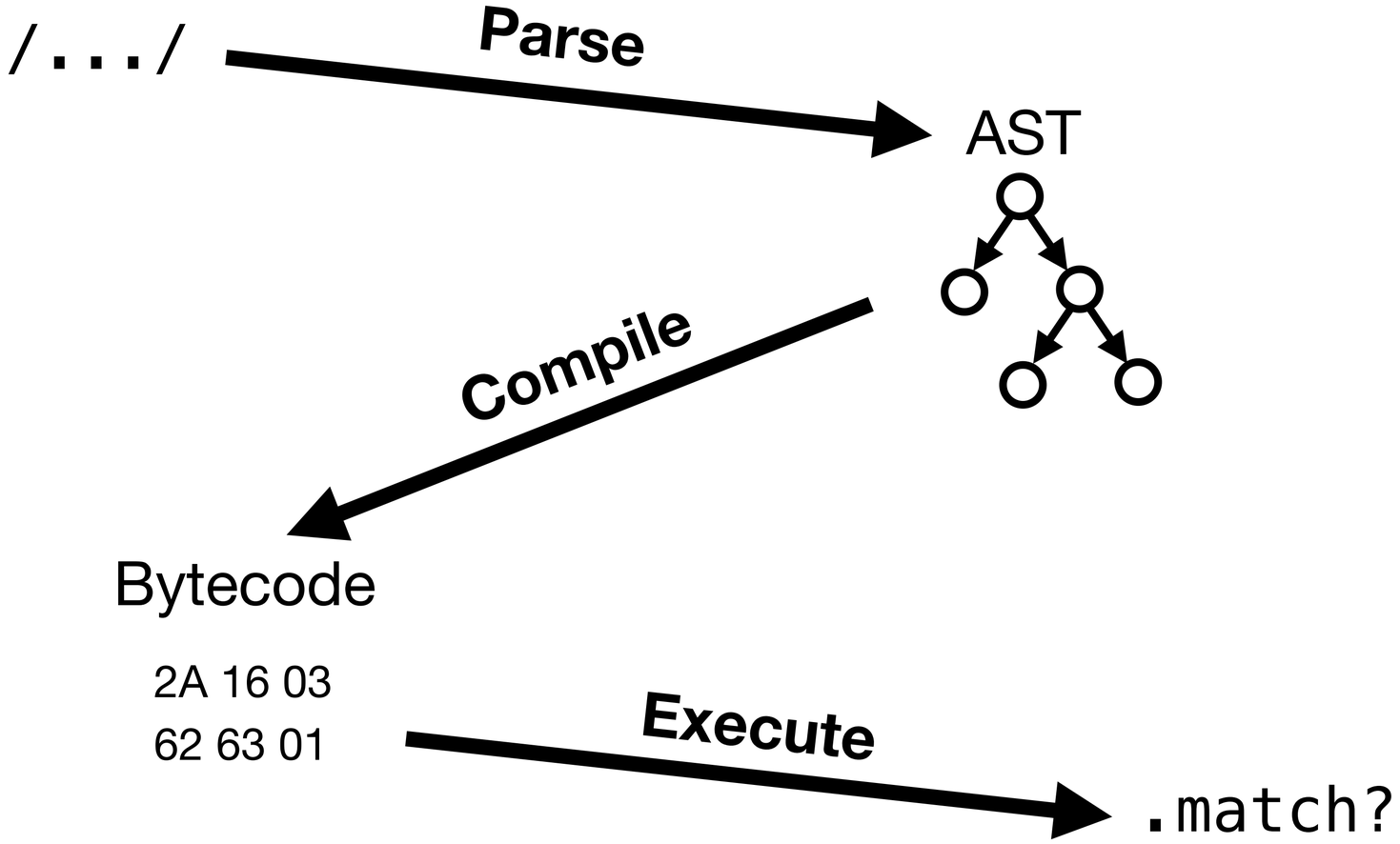
/^.bc([de]*f)+g|h/

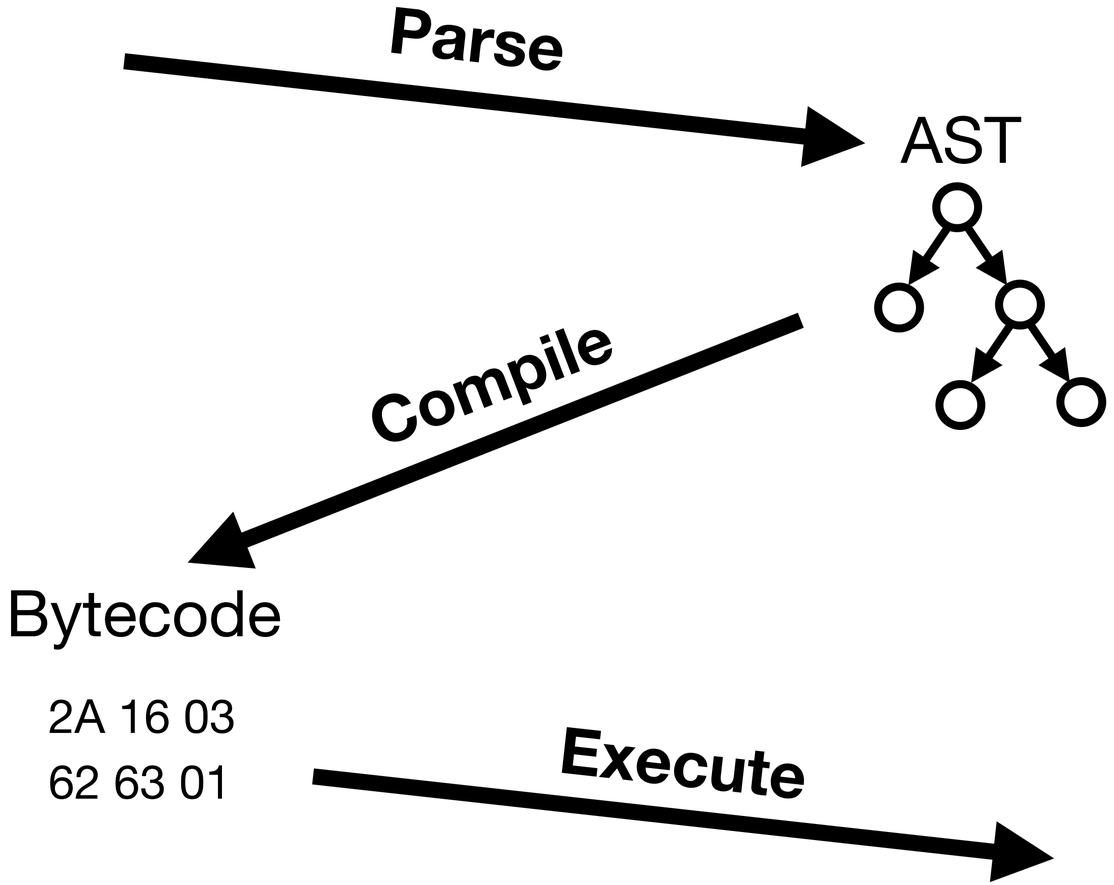
```
3E 52 00 00 00 2A 16 03 62 63 3D 06 00 00 00 41 3B 00
00 00 67 36 01 00 50 41 26 00 00 00 66 10 00 00 00 00
00 00 00 00 00 00 00 00 30 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 3D D4 FF FF FF 51 02 66
39 01 00 3D BF FF FF FF 02 67 3D 02 00 00 00 02 68 01
```

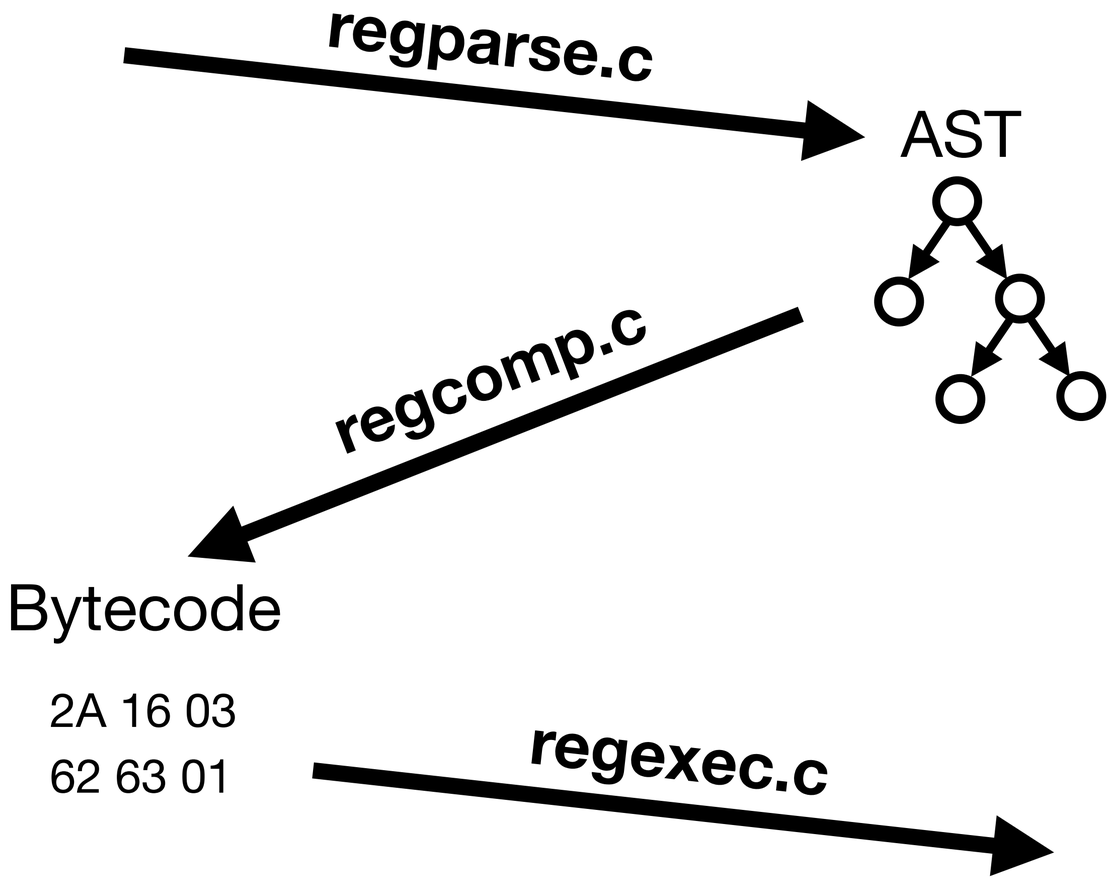
```
/^.bc([de]*f)+g|h/
```


/ . . . /
Parse
AST


/ . . . /
Parse
AST
Compile
Bytecode
2A 16 03
62 63 01


/.../
Parse
AST
Compile
Bytecode
2A 16 03
62 63 01
Execute
.match?

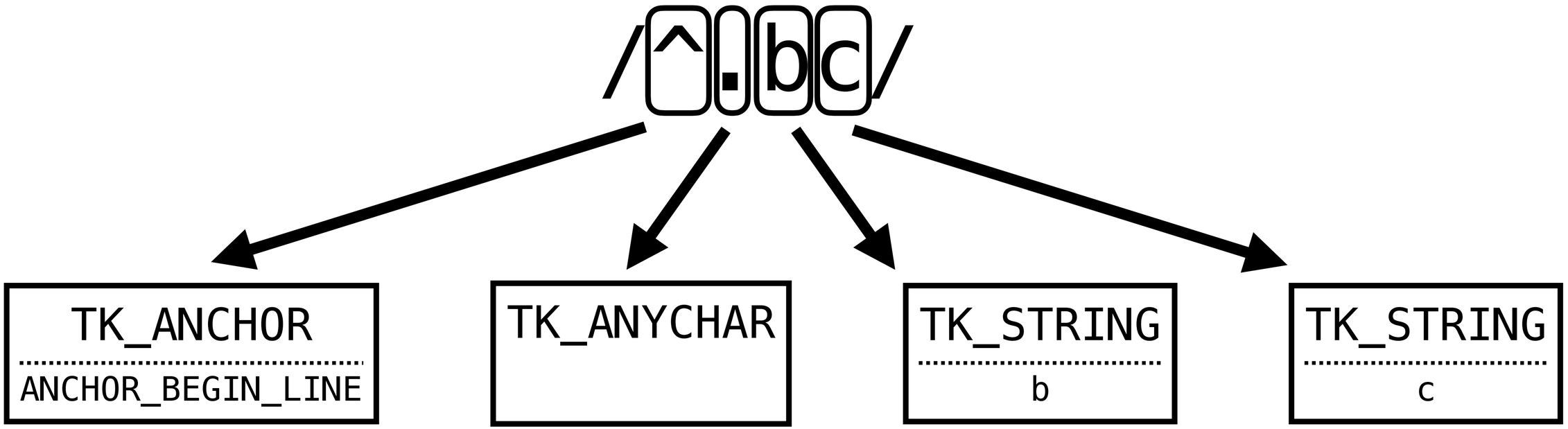

Parse
AST
Compile
Bytecode
2A 16 03
62 63 01
Execute


regparse.c
AST
regcomp.c
Bytecode
2A 16 03
62 63 01
regexec.c

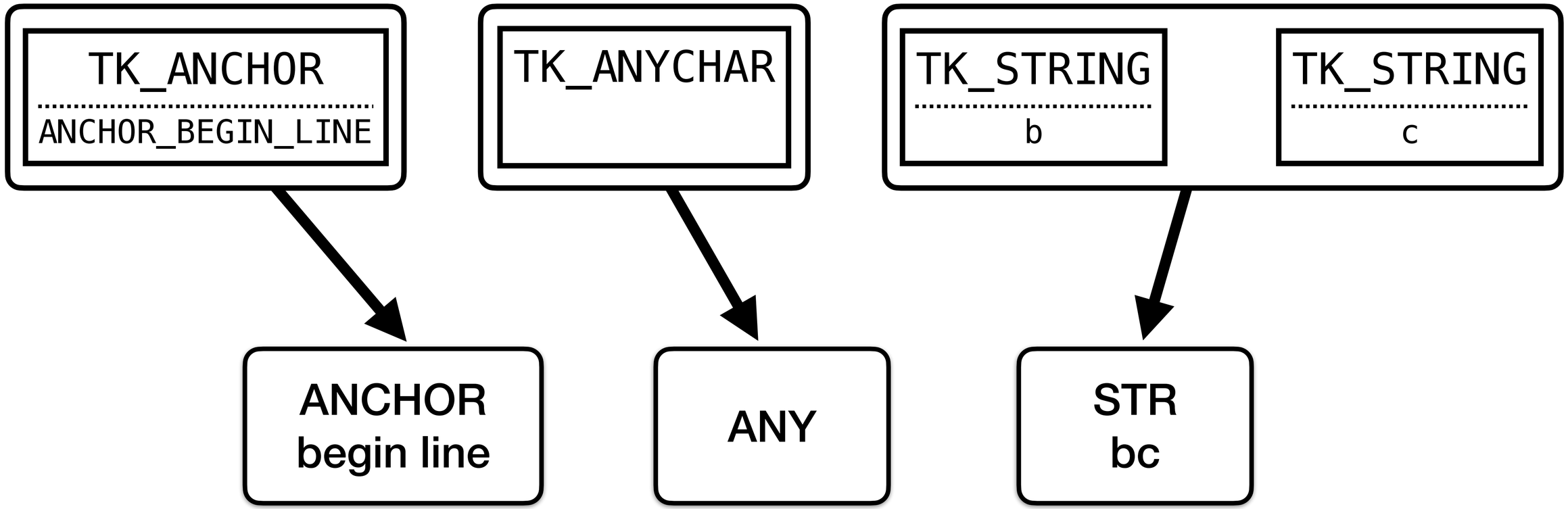
Parse

```
/^.bc/
```

Parse


/^.bc/
TK_ANCHOR
ANCHOR_BEGIN_LINE
TK_ANYCHAR
TK_STRING
b
TK_STRING
c

Parse


TK_ANCHOR
ANCHOR_BEGIN_LINE
TK_ANYCHAR
TK_STRING
b
TK_STRING
c
ANCHOR
begin line
ANY
STR
bc

Parse - fetch_token( )

```
switch (c) {
  case '^':


  case '.':
```

Parse - fetch_token( )

```
switch (c) {
  case '^':
    tok->type = TK_ANCHOR;

  case '.':
    tok->type = TK_ANYCHAR;
```

Parse - fetch_token( )

```
switch (c) {
  case '^':
    tok->type = TK_ANCHOR;
    tok->u.anchor.subtype = ANCHOR_BEGIN_LINE;
  case '.':
    tok->type = TK_ANYCHAR;
```

Parse - parse_exp( )

Parse - parse_exp( )

```
switch (tok->type) {
  case TK_ANCHOR:

  case TK_ANYCHAR:

  case TK_STRING:
```

Parse - parse_exp( )

```
switch (tok->type) {
  case TK_ANCHOR:
    return node_new_anchor(tok->u.anchor.subtype);
  case TK_ANYCHAR:

  case TK_STRING:
```

Parse - parse_exp( )

```
switch (tok->type) {
  case TK_ANCHOR:
    return node_new_anchor(tok->u.anchor.subtype);
  case TK_ANYCHAR:
    return node_new_anychar();
  case TK_STRING:
```

Parse - parse_exp( )

```
case tok.type
  when :anchor
    return AnchorNode.new(tok.subtype)
  when :anychar
    return AnycharNode.new()
  when :string
```

Parse - parse_exp( )

```
switch (tok->type) {
  case TK_STRING:
```

Parse - parse_exp( )

```
switch (tok->type) {
  case TK_STRING:
    node = node_new_str(...);
```

Parse - parse_exp( )

```
switch (tok->type) {
  case TK_STRING:
    node = node_new_str(...);

    while (1) {
      tok = fetch_token(...);
```

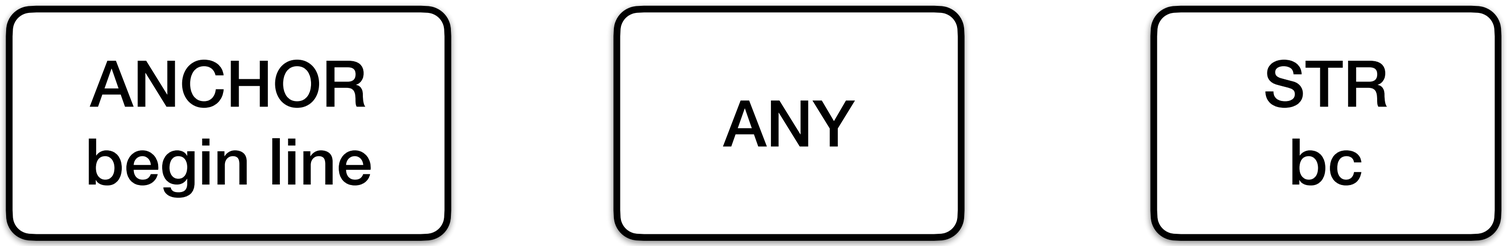
Parse - parse_exp( )

```
switch (tok->type) {
  case TK_STRING:
    node = node_new_str(...);

    while (1) {
      tok = fetch_token(...);
      if (tok->type == TK_STRING) {
         node_str_cat(node, ...);
```

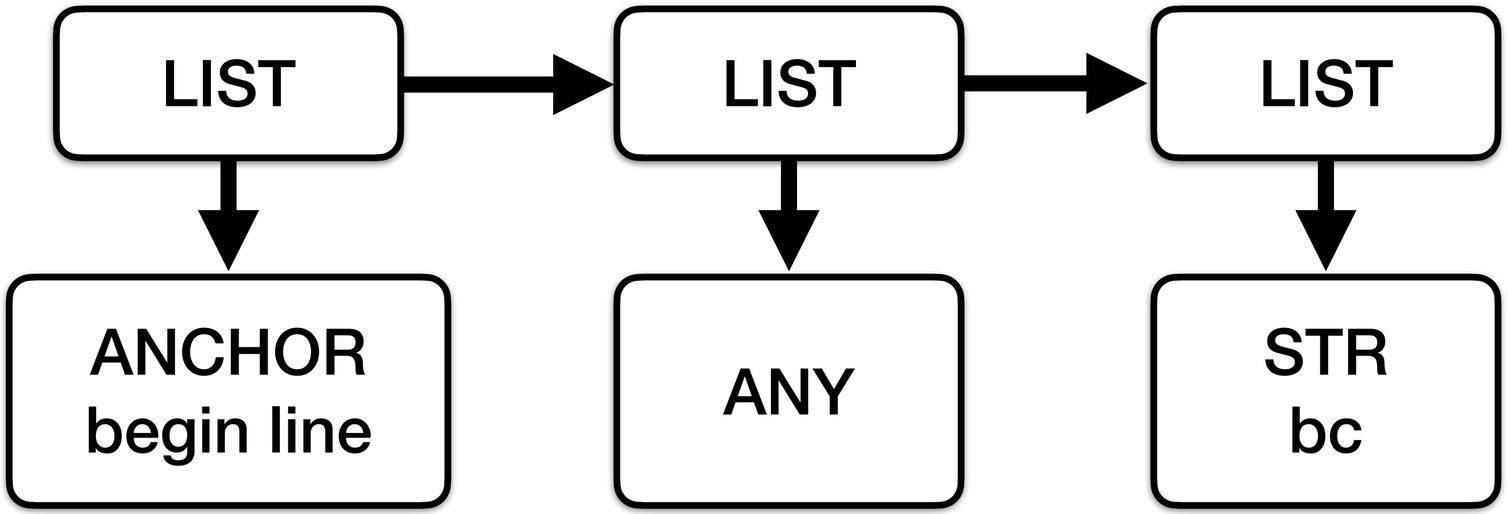
Parse

```
/^.bc/
```



Parse

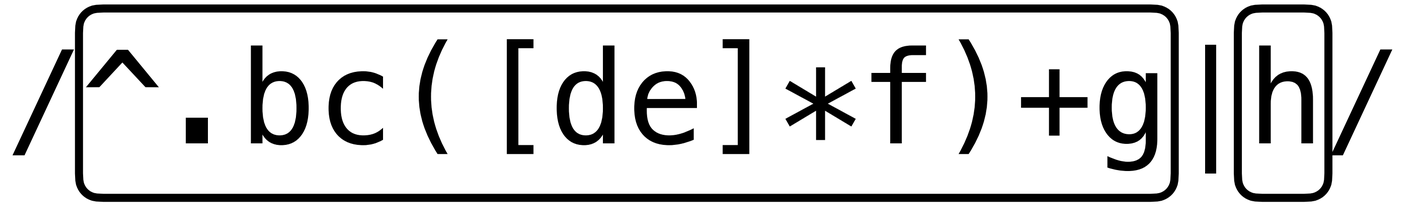
/^.bc/



Parse

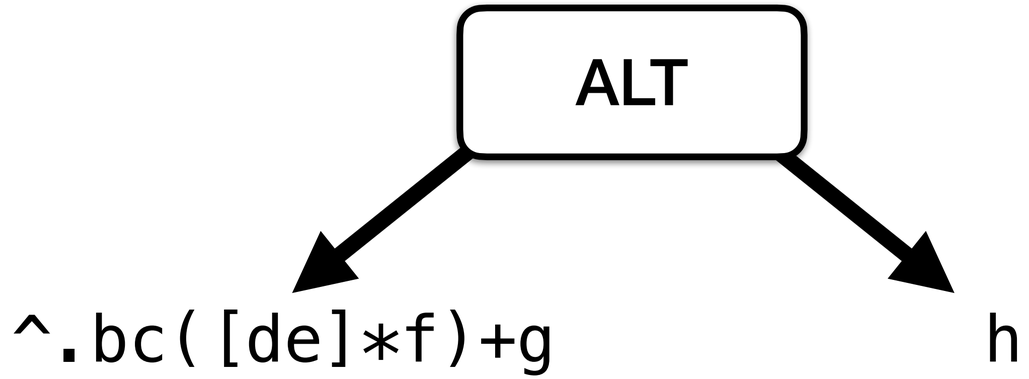
```
/^.bc([de]*f)+g|h/
```

Parse

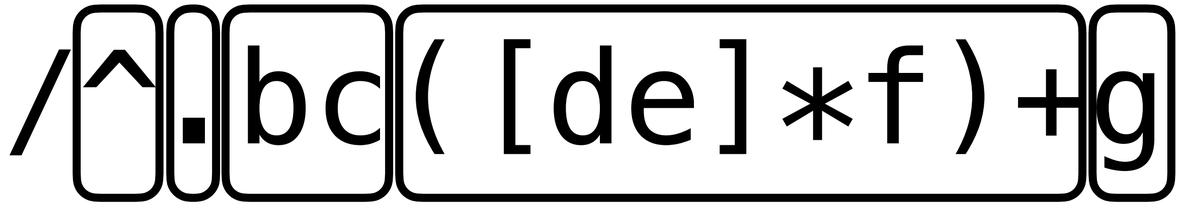


Parse

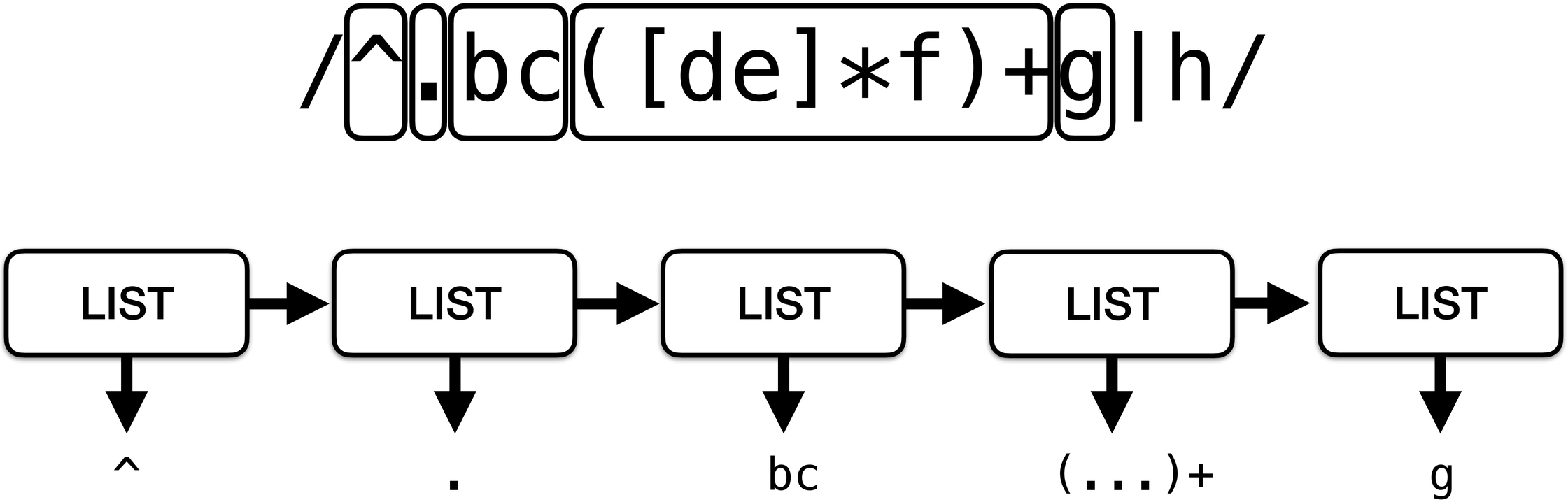
/^.bc([de]*f)+g|h/



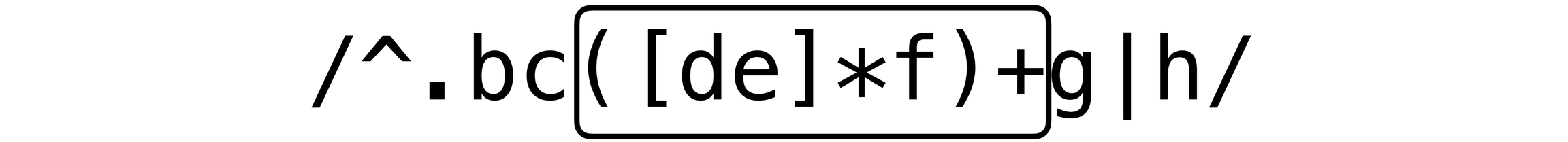
Parse



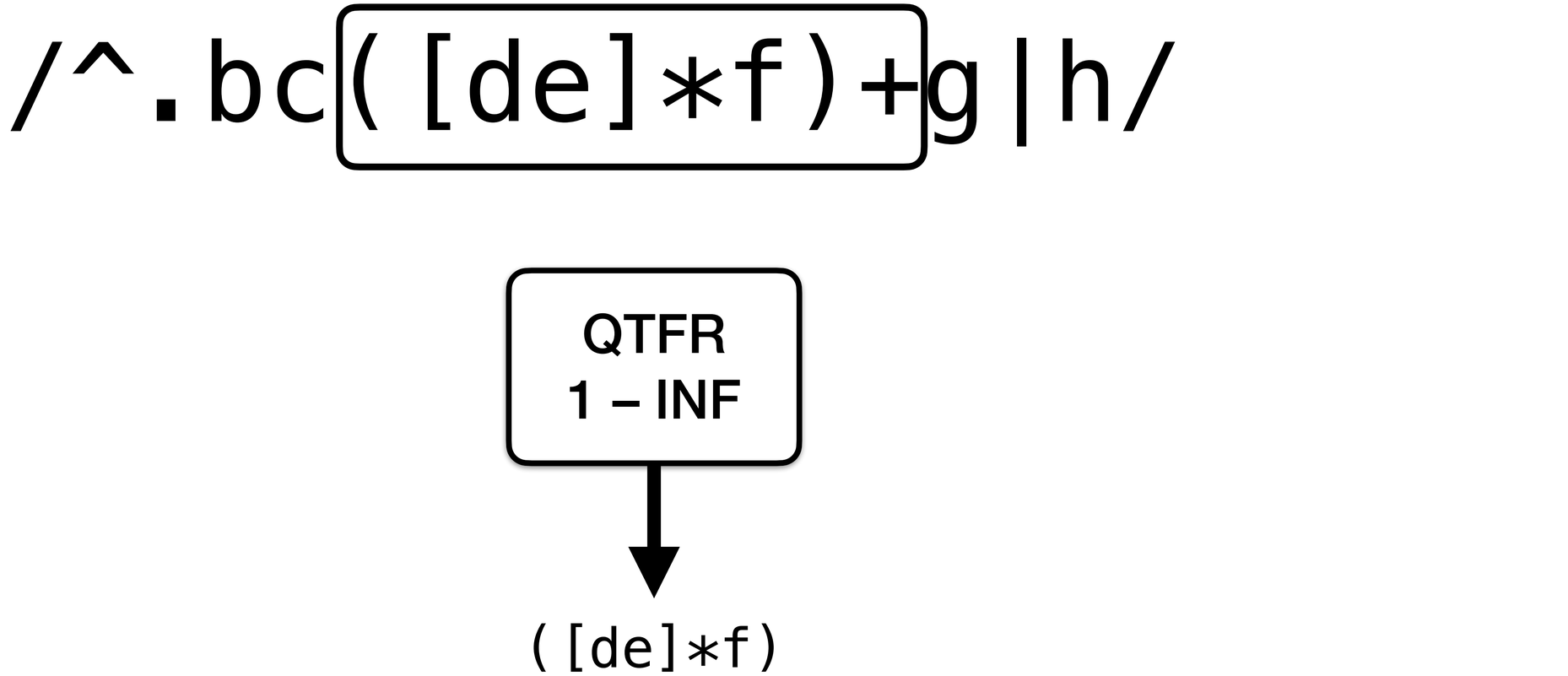
Parse



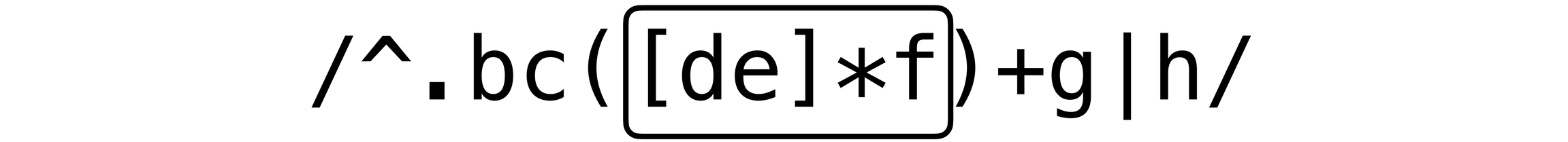
Parse



Parse

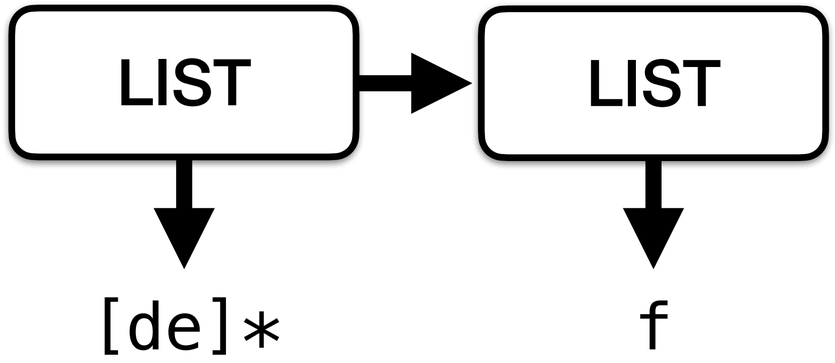


Parse

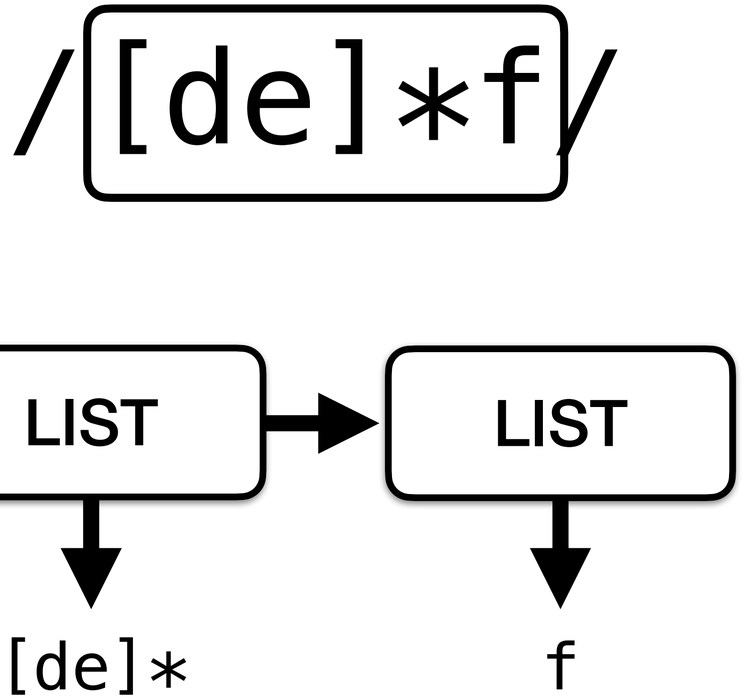


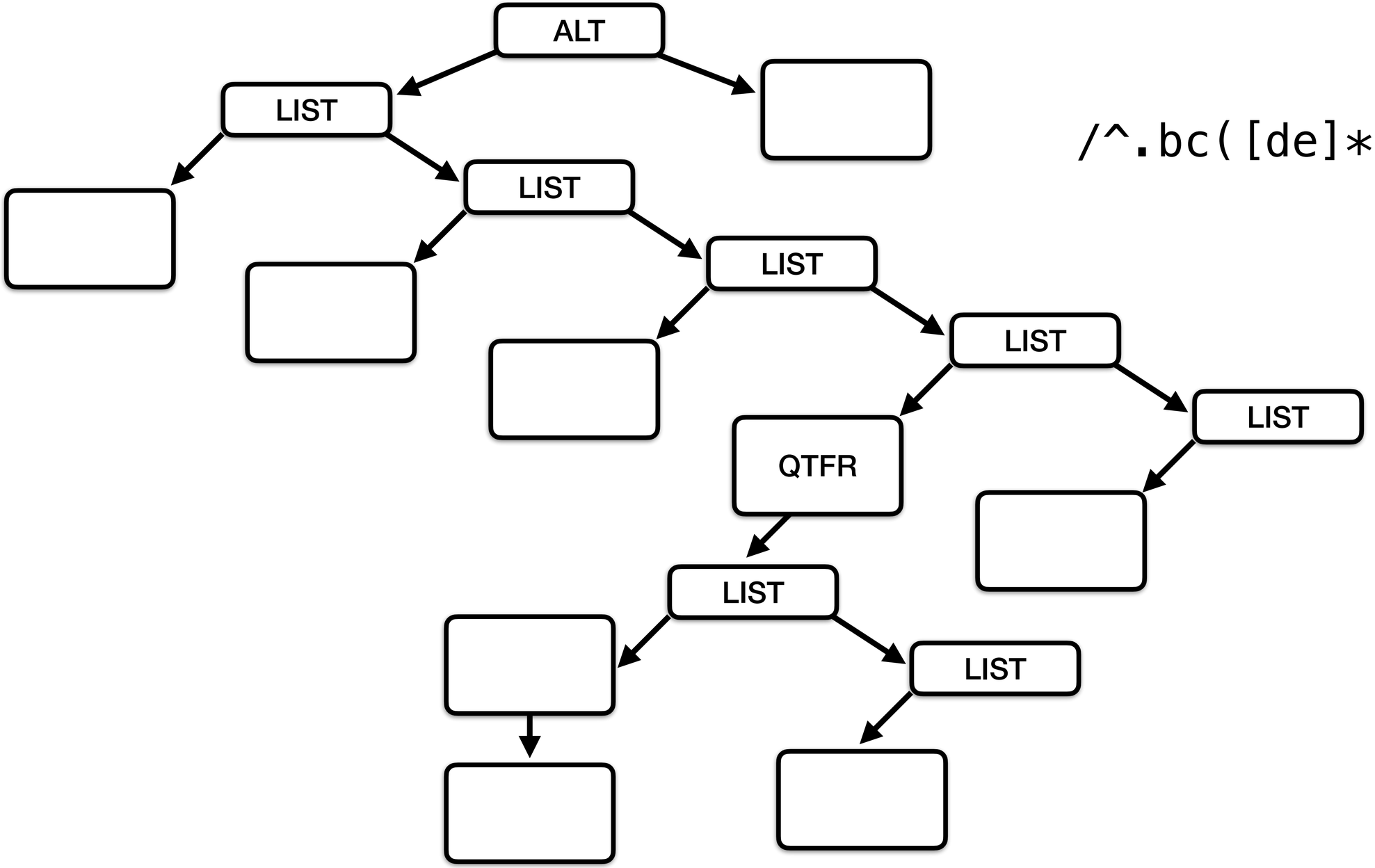
Parse

/^.bc([de]*f)+g|h/

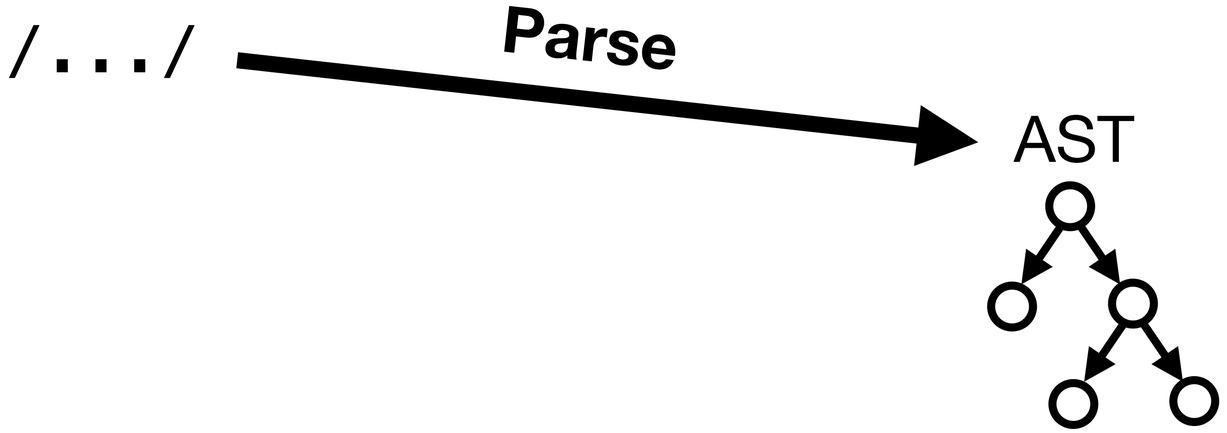


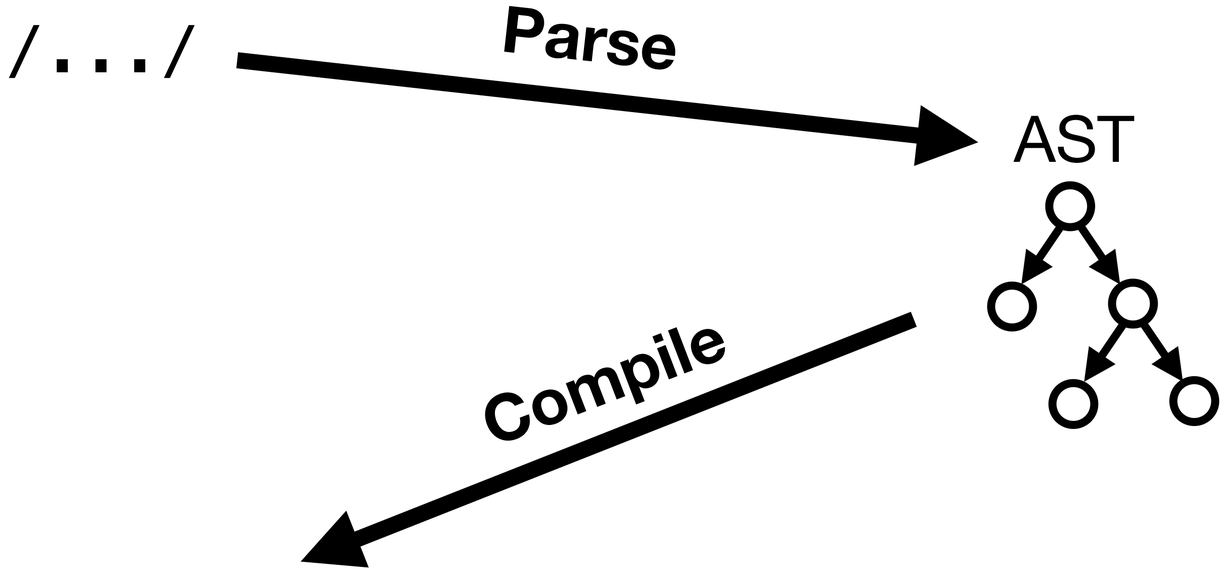
Parse






```
/^.bc([de]*f)+g|h/
```


/ . . . /
Parse
AST


/ . . . /
Parse
AST
Compile
Bytecode
2A 16 03
62 63 01

Compile

```
/^.bc/
```

```
2A 16 03 62 63 01
```

Compile

/^.bc/

```
2A
16
03 62 63
01
```

Compile

/^.bc/

```
2A                      OP_BEGIN_LINE
16
03 62 63
01
```

Compile

```
/^.bc/
```

```
2A                  OP_BEGIN_LINE
16                  OP_ANYCHAR
03 62 63
01
```

Compile

/^.bc/

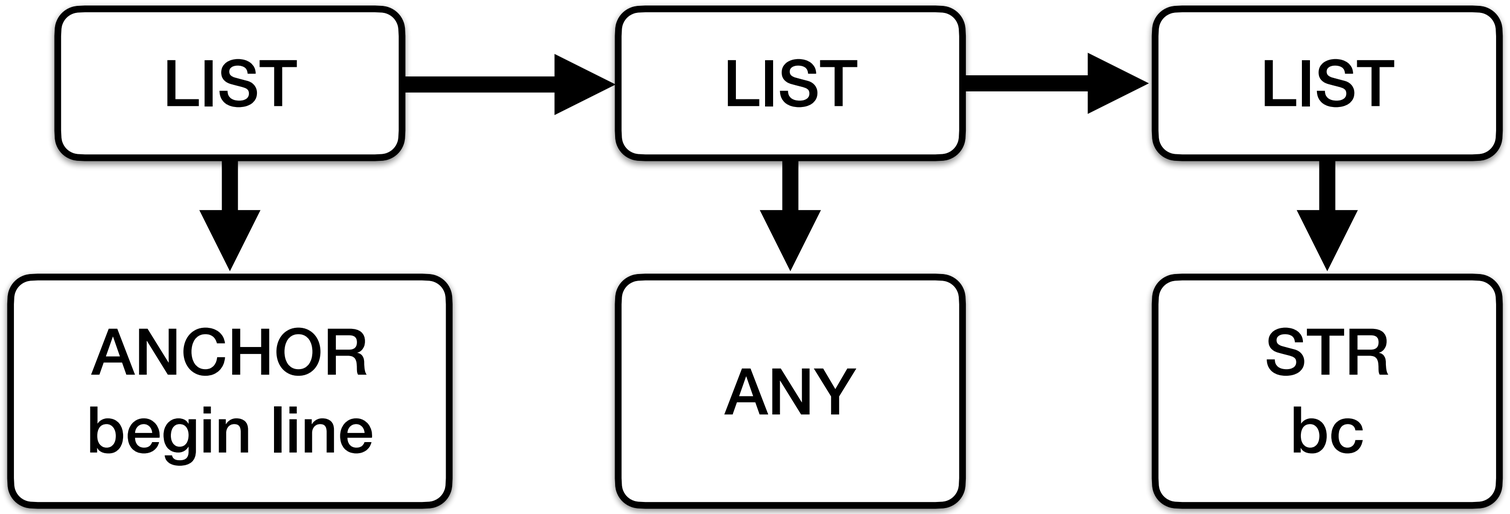
```
2A                  OP_BEGIN_LINE
16                  OP_ANYCHAR
03 62 63            OP_EXACT2       bc
01
```

Compile

/^.bc/

```
2A                  OP_BEGIN_LINE
16                  OP_ANYCHAR
03 62 63            OP_EXACT2       bc
01                  OP_END
```

Compile




```
2A                  OP_BEGIN_LINE
16                  OP_ANYCHAR
03 62 63            OP_EXACT2       bc
01                  OP_END
```

Compile - compile_tree( )

```
switch (node->type) {
```

Compile - compile_tree( )

```
switch (node->type) {
  case NT_LIST:
```

Compile - compile_tree( )

```
switch (node->type) {
  case NT_LIST:
    do {

      node = TAIL(node);
    } while (IS_NOT_NULL(node));
```

Compile - compile_tree( )

```
switch (node->type) {
  case NT_LIST:
    do {
      compile_tree(HEAD(node));
      node = TAIL(node);
    } while (IS_NOT_NULL(node));
```

Compile - compile_tree( )

```
switch (node->type) {
  case NT_ANCHOR:
```

Compile - compile_tree( )

```
switch (node->type) {
  case NT_ANCHOR:
    switch (node->subtype) {
      case ANCHOR_BEGIN_LINE:
```

Compile - compile_tree( )

```
switch (node->type) {
  case NT_ANCHOR:
    switch (node->subtype) {
      case ANCHOR_BEGIN_LINE:
        add_opcode(reg, OP_BEGIN_LINE);
```

Compiled bytecode: **2A**

Compile - compile_tree( )

```
switch (node->type) {
  case NT_CANY:
    add_opcode(reg, OP_ANYCHAR);
```

Compiled bytecode: 2A **16**

Compile - compile_tree( )

```
switch (node->type) {
  case NT_STRING:
    op = select_str_opcode(len);
    add_opcode(reg, op);
```

Compiled bytecode: `2A 16 **03**`

Compile - compile_tree( )

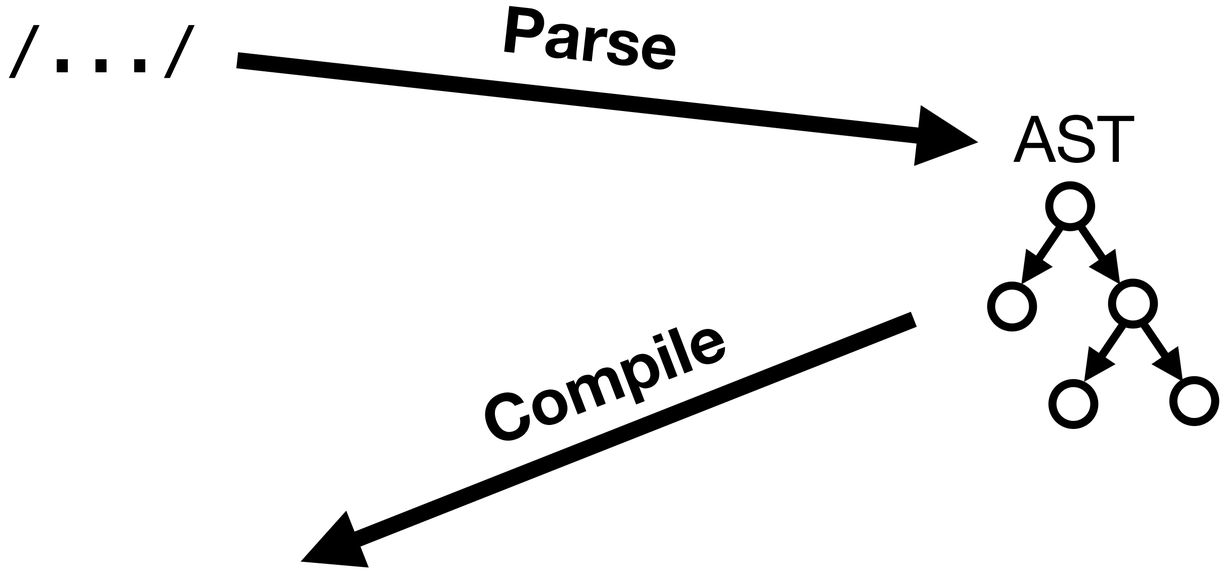
```
switch (node->type) {
  case NT_STRING:
    op = select_str_opcode(len);
    add_opcode(reg, op);
    if (IS_NEED_STR_LEN(op))
      add_length(reg, len);
```

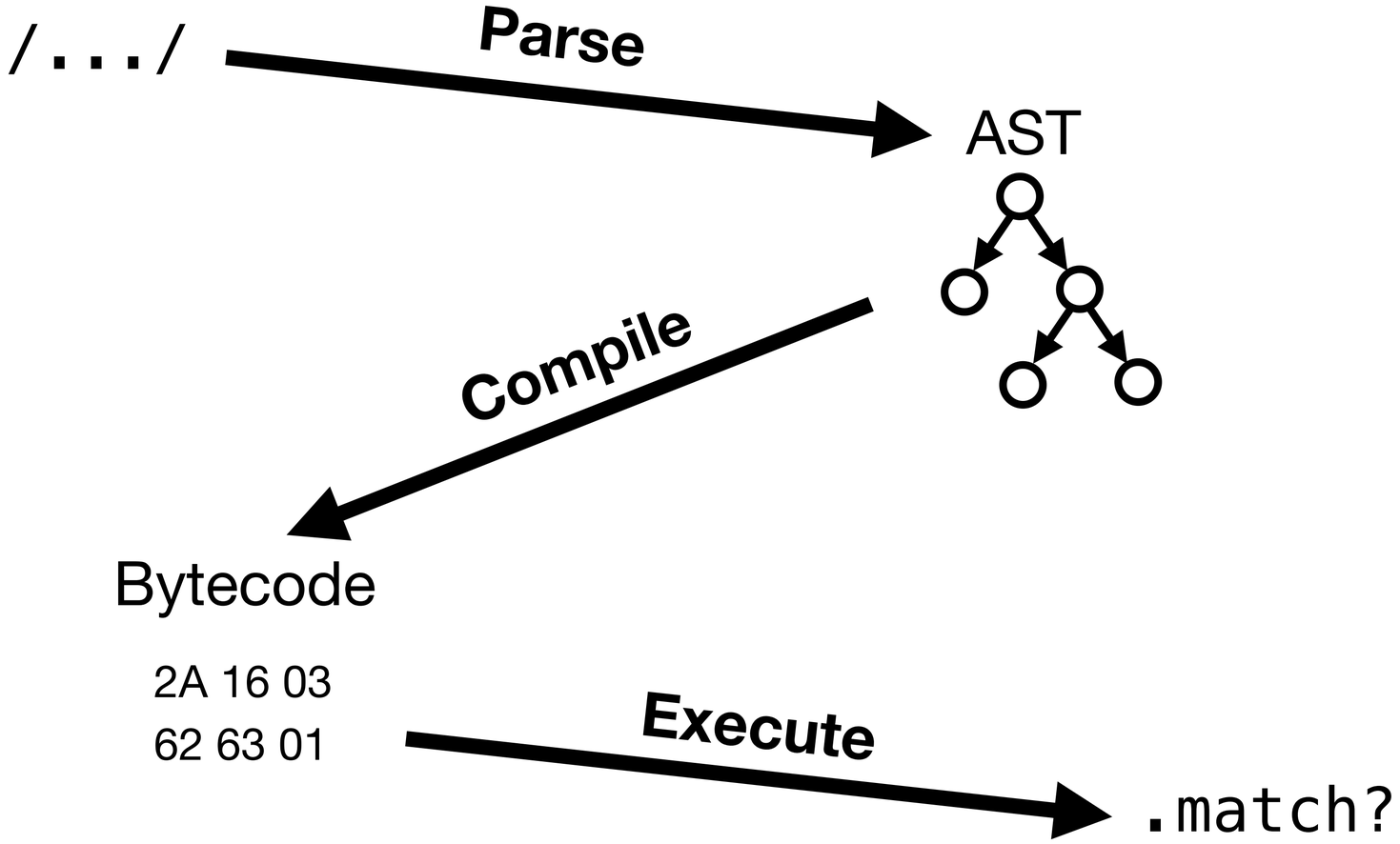
Compiled bytecode: `2A 16 03`

Compile - compile_tree( )

```
switch (node->type) {
  case NT_STRING:
    op = select_str_opcode(len);
    add_opcode(reg, op);
    if (IS_NEED_STR_LEN(op))
      add_length(reg, len);
    add_bytes(reg, node->str, len)
```

Compiled bytecode: `2A 16 03 62 63`


/ . . . /
Parse
AST
Compile
Bytecode
2A 16 03
62 63 01


/.../
Parse
AST
Compile
Bytecode
2A 16 03
62 63 01
Execute
.match?

Execute

```
/^.bc/
.match?("abcdefg")
```

Execute - match_at( )

```
while (1) {
  switch (*p++) {
    case OP_BEGIN_LINE:

    case OP_ANYCHAR:

    case OP_EXACT2:
```

Execute - match_at( )

```
2A 16 03 62 63 01                    a b c d e f g
```

Execute - match_at( )

Execute - match_at( )

Execute - match_at( )


```
switch (*p++) {
  case OP_BEGIN_LINE:
```

Execute - match_at( )


```
switch (*p++) {
  case OP_BEGIN_LINE:
```

Execute - match_at( )


```
switch (*p++) {
  case OP_BEGIN_LINE:
    if (ON_LINE_BEGIN(s)) break;
```

Execute - match_at( )


```
switch (*p++) {
  case OP_BEGIN_LINE:
    if (ON_LINE_BEGIN(s)) break;
    goto fail;
```

Execute - match_at( )


```
switch (*p++) {
  case OP_ANYCHAR:
```

Execute - match_at( )


```
switch (*p++) {
  case OP_ANYCHAR:
    DATA_ENSURE(1);
    s += 1;
```

Execute - match_at( )


```
switch (*p++) {
  case OP_ANYCHAR:
    DATA_ENSURE(1);
    s += 1;
```

Execute - match_at( )


```
switch (*p++) {
  case OP_EXACT2:
```

Execute - match_at( )


```
switch (*p++) {
  case OP_EXACT2:
    if (*p != *s) goto fail;
    p++; s++;
```

Execute - match_at( )


```
switch (*p++) {
  case OP_EXACT2:
    if (*p != *s) goto fail;
    p++; s++;
```

Execute - match_at( )

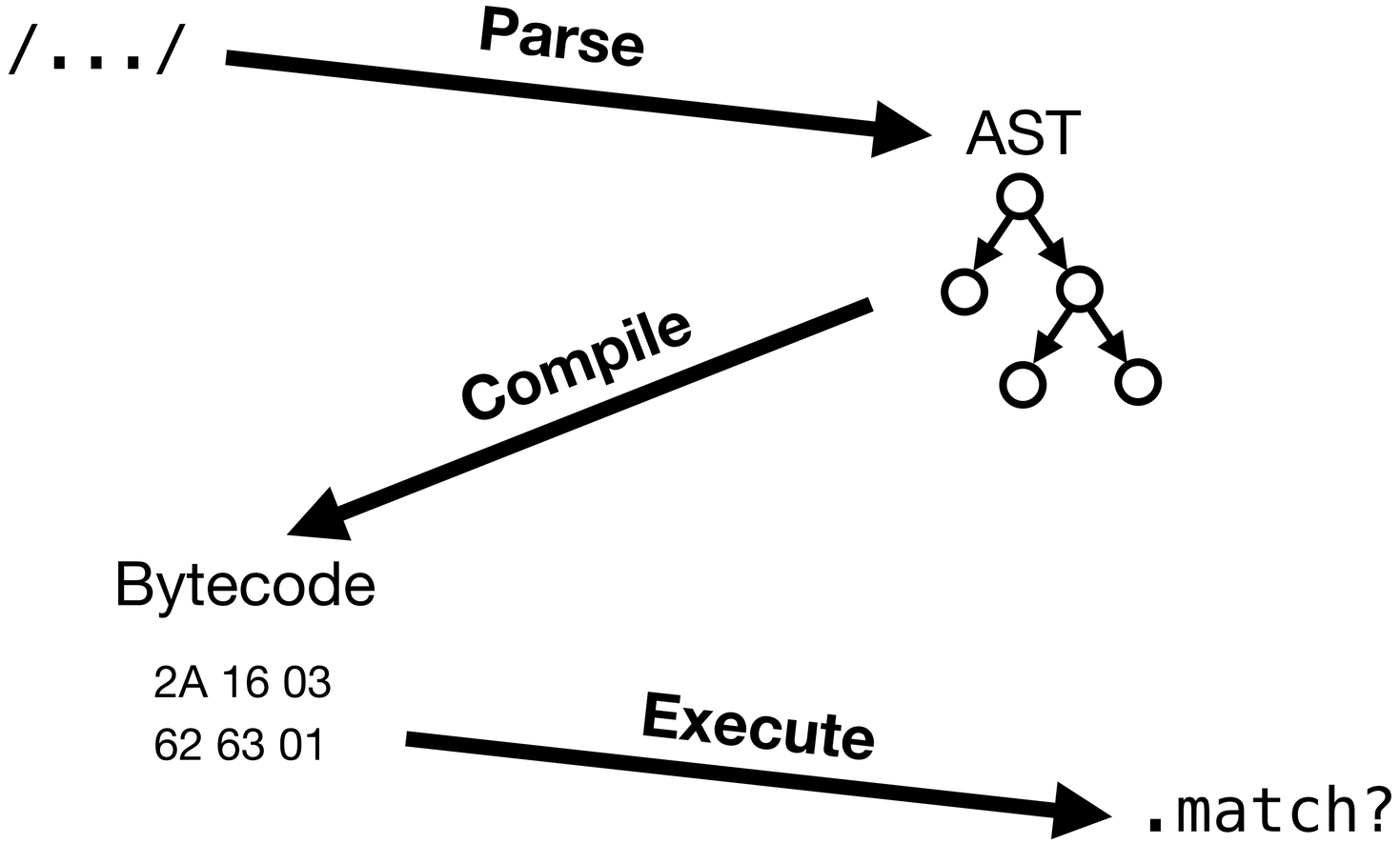

```
switch (*p++) {
  case OP_EXACT2:
    if (*p != *s) goto fail;
    p++; s++;
    if (*p != *s) goto fail;
    p++; s++;
```

Execute - match_at( )


```
switch (*p++) {
  case OP_EXACT2:
    if (*p != *s) goto fail;
    p++; s++;
    if (*p != *s) goto fail;
    p++; s++;
```

Execute - match_at( )


```
switch (*p++) {
  case OP_END:
    goto finish;
```


/.../
Parse
AST
Compile
Bytecode
2A 16 03
62 63 01
Execute
.match?

Backtracking

```
/\w*.c/
```

Backtracking

/\w*.c/

```
3E 06 00 00 00
22
3D F5 FF FF FF
16
02 63
01
```

Backtracking

/\w*.c/

```
3E 06 00 00 00
22
3D F5 FF FF FF
16                  OP_ANYCHAR
02 63               OP_EXACT1       c
01                  OP_END
```

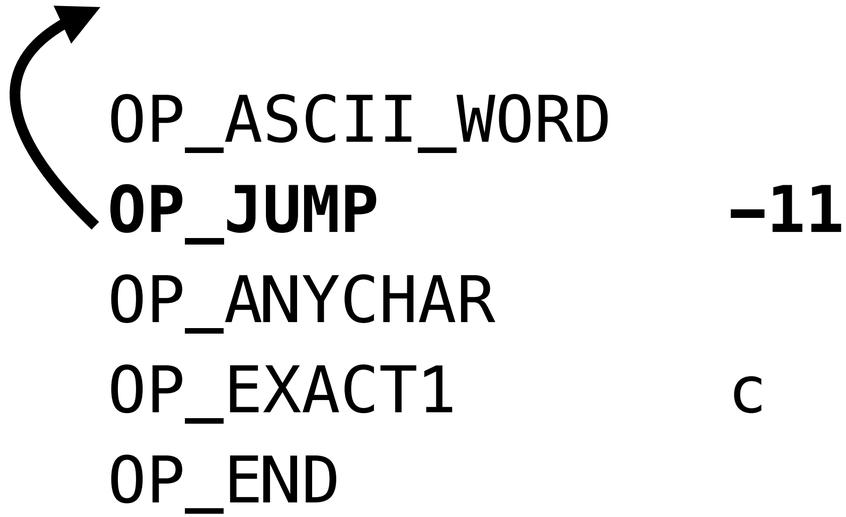
Backtracking

/\w*.c/

```
3E 06 00 00 00
22                  OP_ASCII_WORD
3D F5 FF FF FF
16                  OP_ANYCHAR
02 63               OP_EXACT1       c
01                  OP_END
```

Backtracking

/\w*.c/



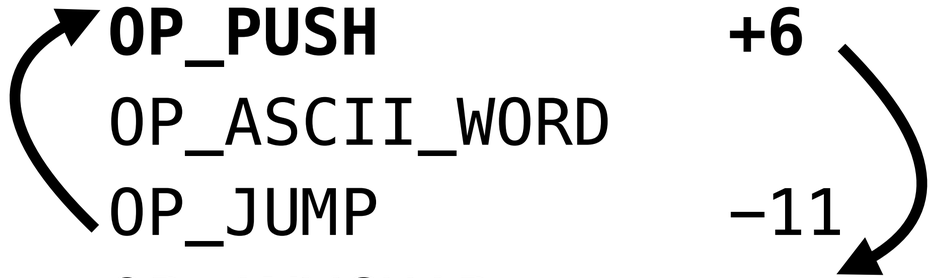
```
3E 06 00 00 00
22
3D F5 FF FF FF
16
02 63
01
```

```
OP_ASCII_WORD
OP_JUMP          -11
OP_ANYCHAR
OP_EXACT1        c
OP_END
```

Backtracking

/\w*.c/

```
3E 06 00 00 00    OP_PUSH           +6
22                OP_ASCII_WORD
3D F5 FF FF FF    OP_JUMP           -11
16                OP_ANYCHAR
02 63             OP_EXACT1         c
01                OP_END
```



Backtracking

```
/\w*.c/
.match?("abc")
```

Backtracking


```
switch (*p++) {
  case OP_PUSH:
```

Backtracking


```
switch (*p++) {
  case OP_PUSH:
    int reladdr = *(int *)p;
```

Backtracking


```
switch (*p++) {
  case OP_PUSH:
    int reladdr = *(int *)p;  // 6
```

Backtracking


```
switch (*p++) {
  case OP_PUSH:
    int reladdr = *(int *)p;  // 6
    p += sizeof(int);
```

Backtracking


```
switch (*p++) {
  case OP_PUSH:
    int reladdr = *(int *)p;  // 6
    p += sizeof(int);
    STACK_PUSH(p + reladdr, s);
```

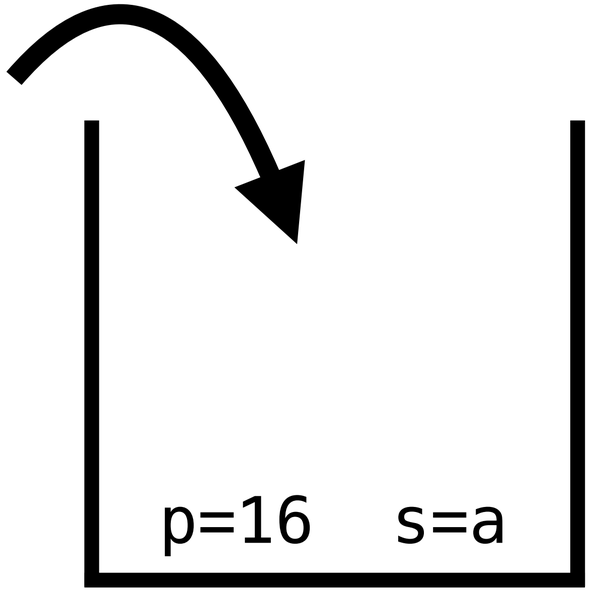
Backtracking


```
switch (*p++) {
  case OP_PUSH:
    int reladdr = *(int *)p;  // 6
    p += sizeof(int);
    STACK_PUSH(p + reladdr, s);
```

Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
                  ^                             ^
                  p                             s
```

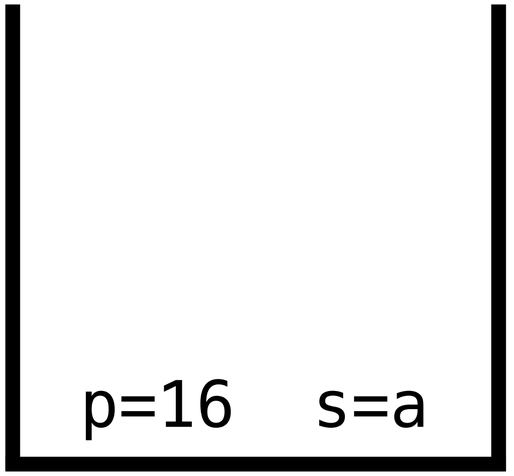
```
switch (*p++) {
  case OP_PUSH:
    int reladdr = *(int *)p;  // 6
    p += sizeof(int);
    STACK_PUSH(p + reladdr, s);
```



Backtracking

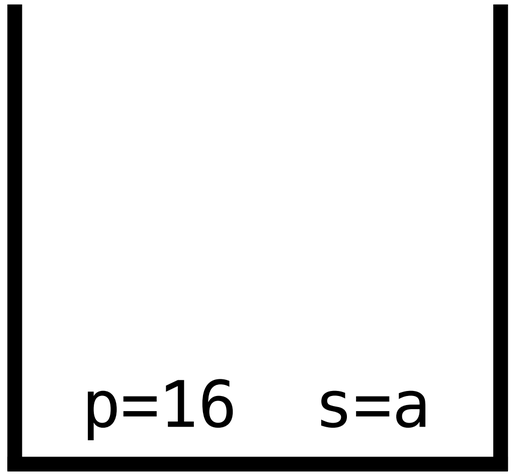
```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
               ↑                                ↑
               p                                s
```

```
switch (*p++) {
  case OP_ASCII_WORD:
```



Backtracking

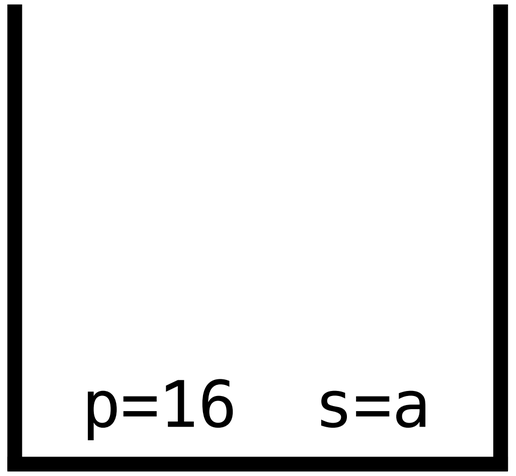

```
switch (*p++) {
  case OP_ASCII_WORD:
    DATA_ENSURE(1);
    if (!IS_ASCII_WORD(s)) goto fail;
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
                  ↑                             ↑
                  p                             s
```

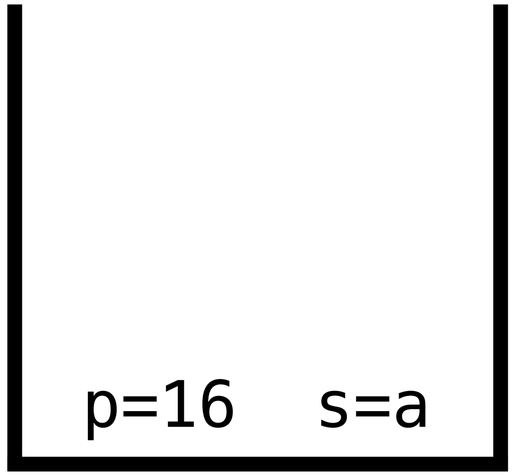
```
switch (*p++) {
  case OP_ASCII_WORD:
    DATA_ENSURE(1);
    if (!IS_ASCII_WORD(s)) goto fail;
    s += 1;
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
                  ↑                             ↑
                  p                             s
```

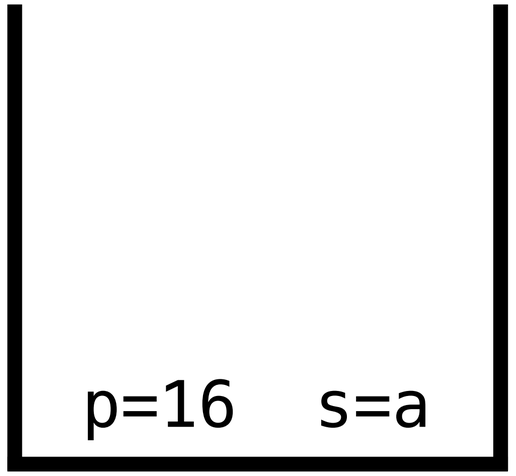
```
switch (*p++) {
  case OP_JUMP:
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
                     ↑                          ↑
                     p                          s
```

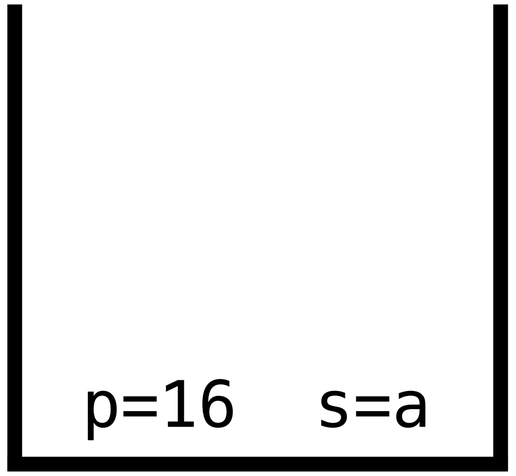
```
switch (*p++) {
  case OP_JUMP:
    int reladdr = *(int *)p;
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
                     ↑                          ↑
                     p                          s
```

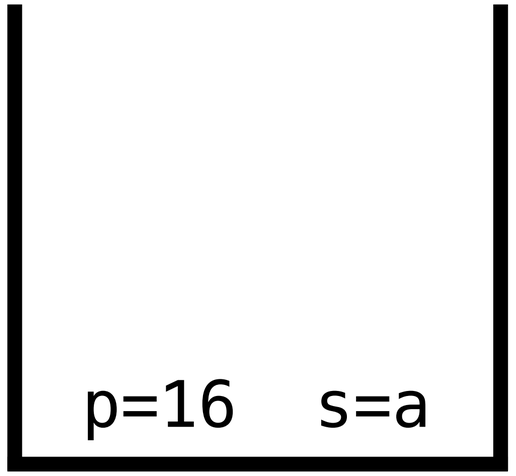
```
switch (*p++) {
  case OP_JUMP:
    int reladdr = *(int *)p;  // -11
```



Backtracking

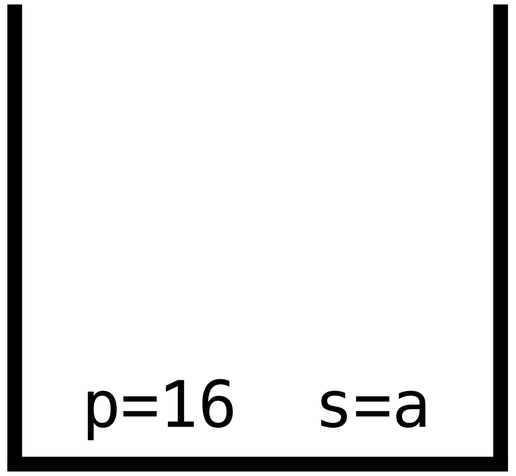
```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
                                 ↑              ↑
                                 p              s
```

```
switch (*p++) {
  case OP_JUMP:
    int reladdr = *(int *)p;  // -11
    p += sizeof(int);
```



Backtracking

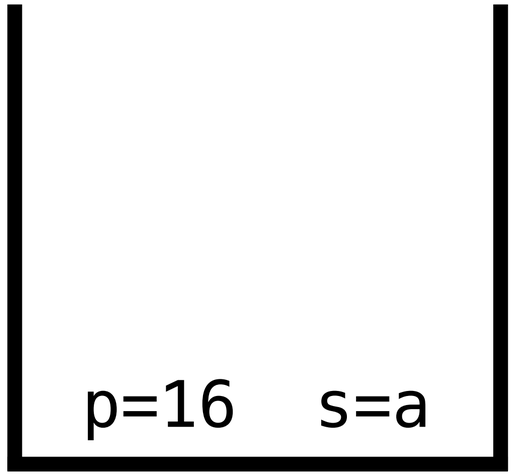

```
switch (*p++) {
  case OP_JUMP:
    int reladdr = *(int *)p;  // -11
    p += sizeof(int);
    p += reladdr;
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
 ↑                                               ↑
 p                                               s
```

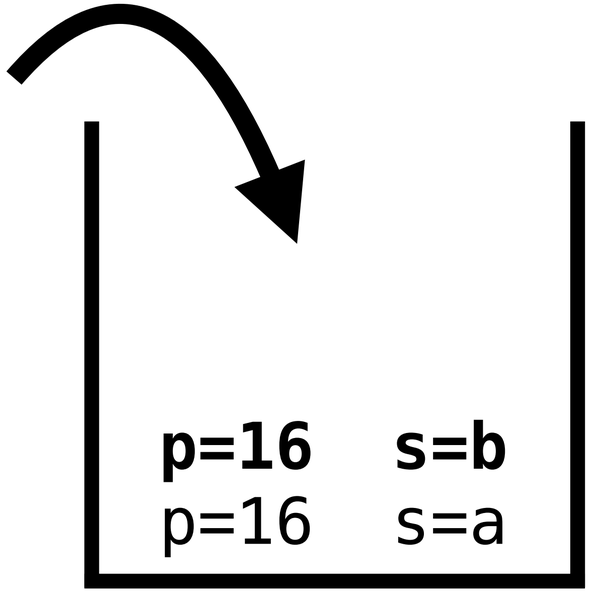
```
switch (*p++) {
  case OP_PUSH:
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
 ↑                                               ↑
 p                                               s
```

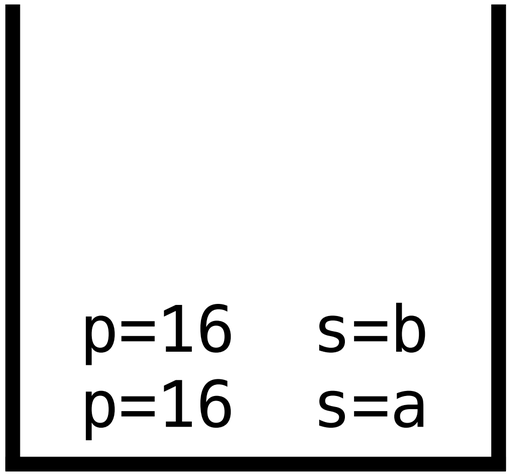
```
switch (*p++) {
  case OP_PUSH:
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
               ↑                                   ↑
               p                                   s
```

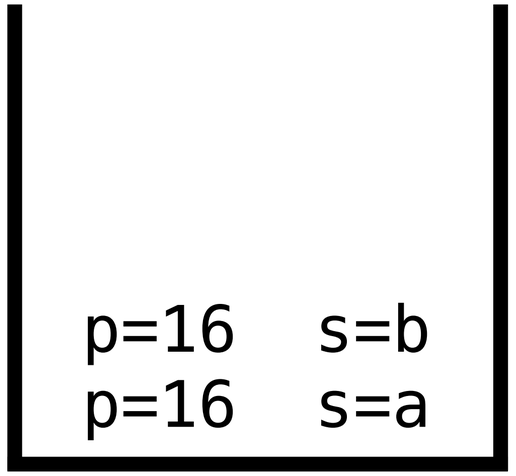
```
switch (*p++) {
  case OP_ASCII_WORD:
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
                  ^                             ^
                  p                             s
```

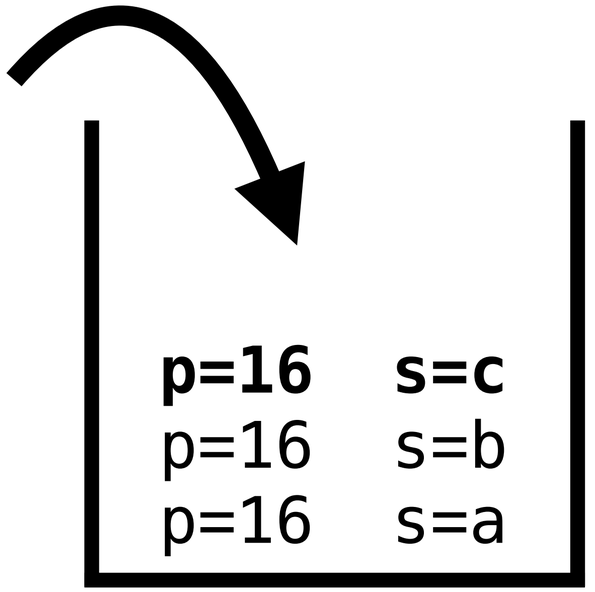
```
switch (*p++) {
  case OP_JUMP:
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
 ↑                                              ↑
 p                                              s
```

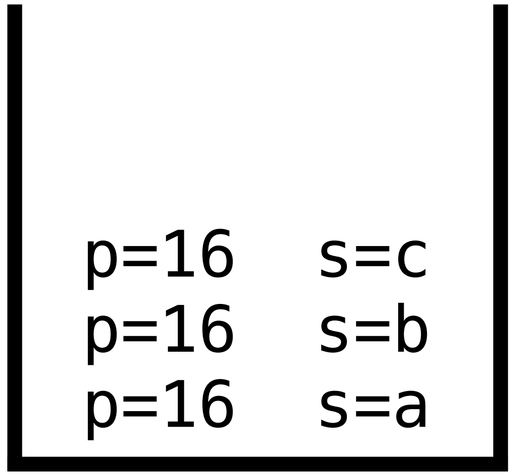
```
switch (*p++) {
  case OP_PUSH:
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
               ↑                                ↑
               p                                s
```

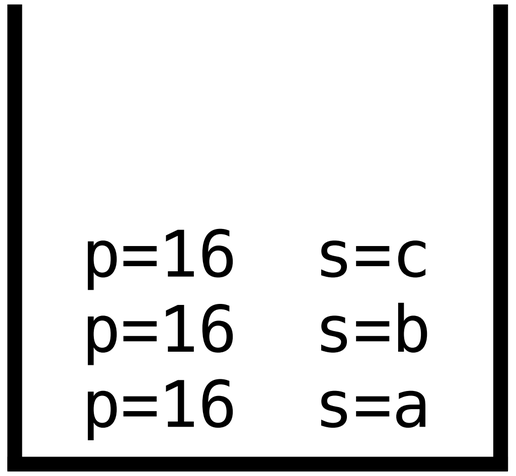
```
switch (*p++) {
  case OP_ASCII_WORD:
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
                  ↑                             ↑
                  p                             s
```

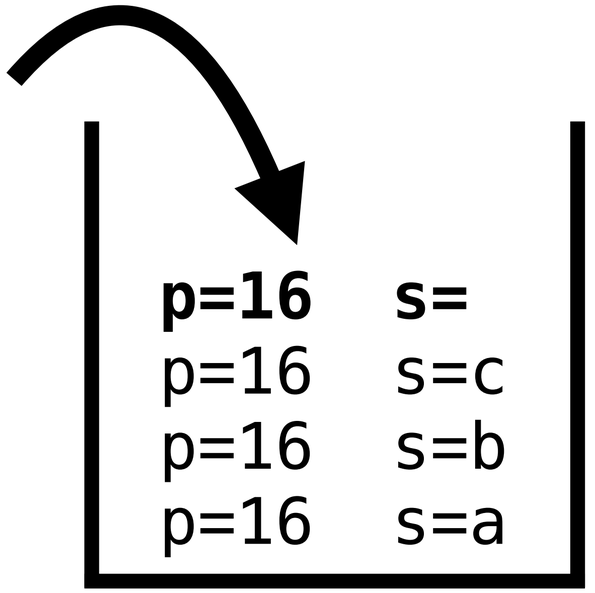
```
switch (*p++) {
  case OP_JUMP:
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
 ↑                                              ↑
 p                                              s
```

```
switch (*p++) {
  case OP_PUSH:
```



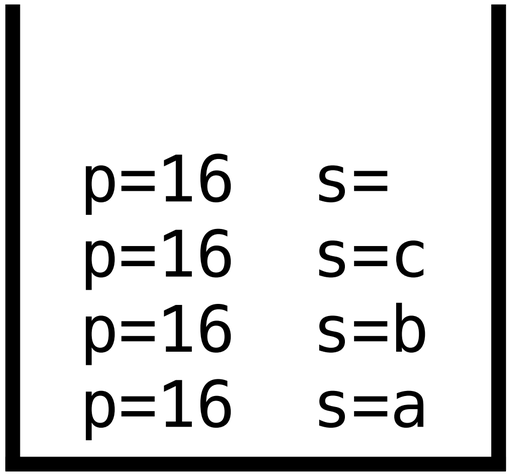
Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
```

p

s

```
switch (*p++) {
  case OP_ASCII_WORD:
```



Backtracking

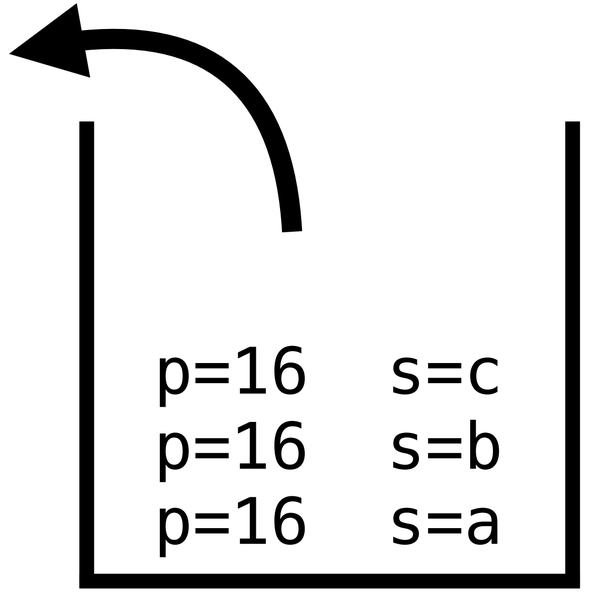

```
switch (*p++) {
  case OP_ASCII_WORD:
    DATA_ENSURE(1);
```

```
p=16  s=
p=16  s=c
p=16  s=b
p=16  s=a
```

Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
                                 ↑              ↑
                                 p              s
```

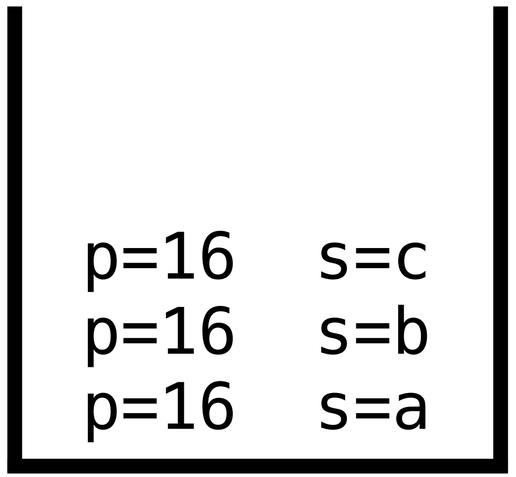
```
switch (*p++) {
  case OP_ANYCHAR:
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
                                   ↑            ↑
                                   p            s
```

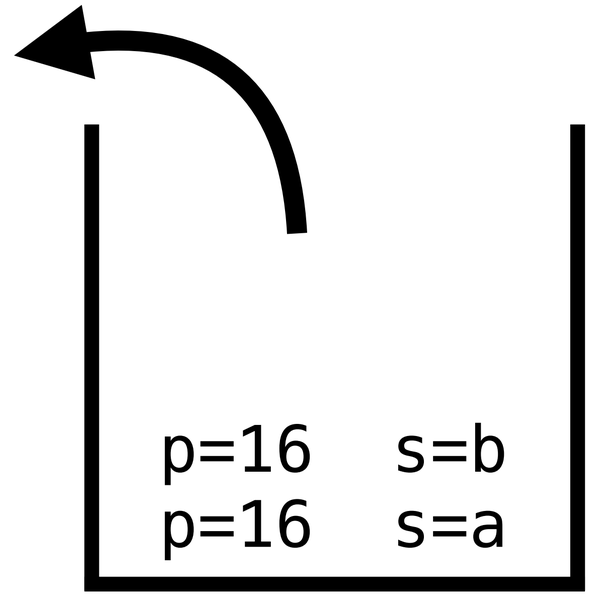
```
switch (*p++) {
  case OP_ANYCHAR:
    DATA_ENSURE(1);
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
                                 ↑              ↑
                                 p              s
```

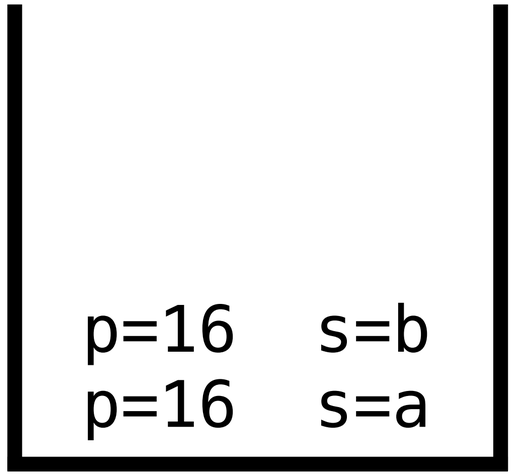
```
switch (*p++) {
  case OP_ANYCHAR:
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
                                   ↑            ↑
                                   p            s
```

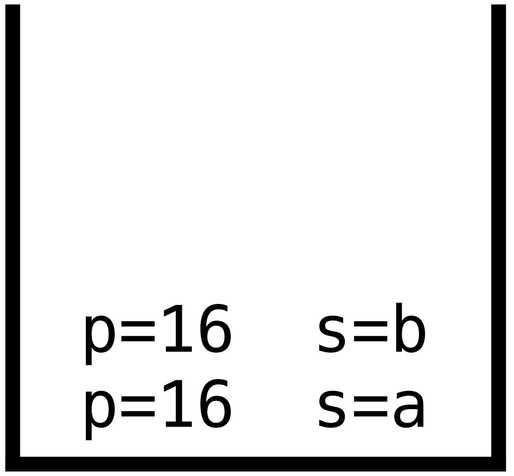
```
switch (*p++) {
  case OP_ANYCHAR:
    DATA_ENSURE(1);
    s += 1;
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
                                   ↑            ↑
                                   p            s
```

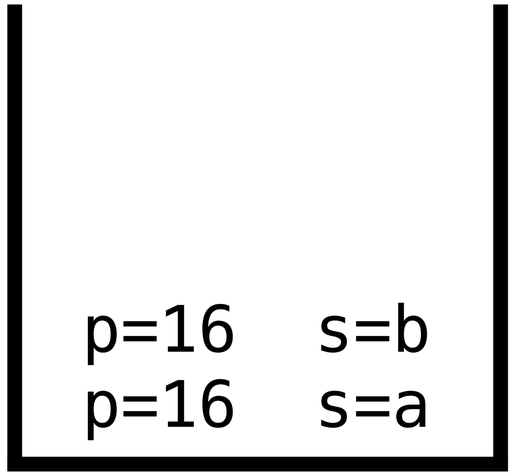
```
switch (*p++) {
  case OP_EXACT1:
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
                                   ↑            ↑
                                   p            s
```

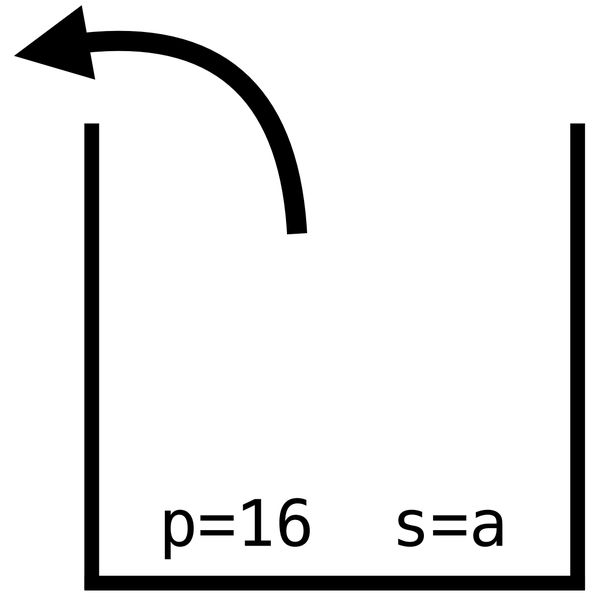
```
switch (*p++) {
  case OP_EXACT1:
    if (*p != *s) goto fail;
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
                                 ↑              ↑
                                 p              s
```

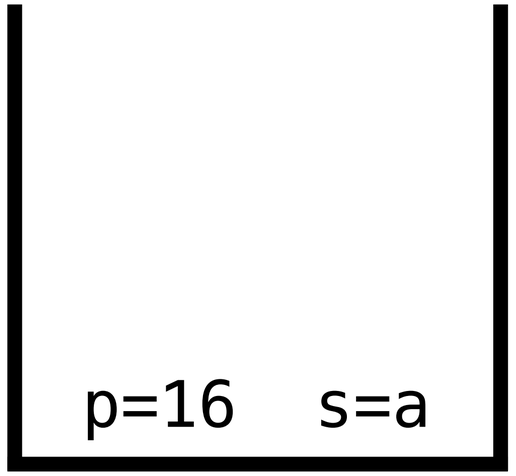
```
switch (*p++) {
  case OP_ANYCHAR:
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
                                   ↑            ↑
                                   p            s
```

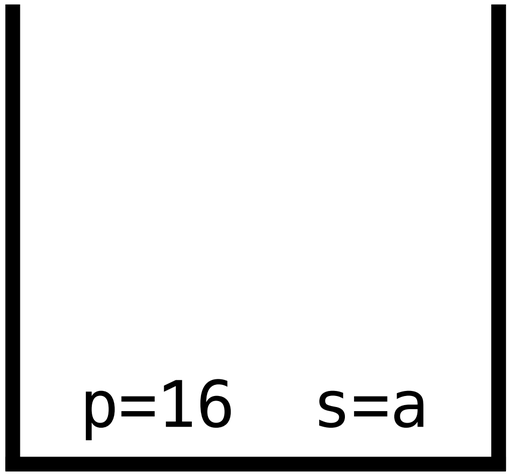
```
switch (*p++) {
  case OP_ANYCHAR:
    DATA_ENSURE(1);
    s += 1;
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
                                   ↑           ↑
                                   p           s
```

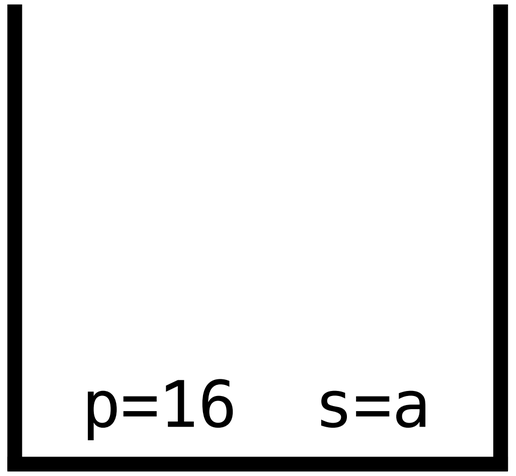
```
switch (*p++) {
  case OP_EXACT1:
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
                                    ↑           ↑
                                    p           s
```

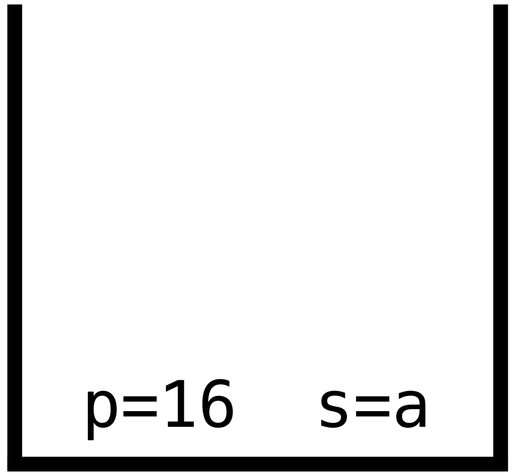
```
switch (*p++) {
  case OP_EXACT1:
    if (*p != *s) goto fail;
    p++; s++;
```



Backtracking

```
3E 06 00 00 00 22 3D F5 FF FF FF 16 02 63 01    a b c
                                                p       s
```

```
switch (*p++) {
  case OP_END:
    goto finish;
```



/^.bc([de]*f)+g|h/

```
3E 52 00 00 00 2A 16 03 62 63 3D 06 00 00 00 41 3B 00
00 00 67 36 01 00 50 41 26 00 00 00 66 10 00 00 00 00
00 00 00 00 00 00 00 00 30 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 3D D4 FF FF FF 51 02 66
39 01 00 3D BF FF FF FF 02 67 3D 02 00 00 00 02 68 01
```

```
3E 52 00 00 00
2A
16
03 62 63
3D 06 00 00 00
41 3B 00 00 00 67
36 01 00
50
41 26 00 00 00 66
10 [bitset]
3D D4 FF FF FF
51
02 66
39 01 00
3D BF FF FF FF
02 67
3D 02 00 00 00
02 68
01
```

```
/^.bc([de]*f)+g|h/
```

/^.bc([de]*f)+g|h/

```
3E 52 00 00 00
2A                          OP_BEGIN_LINE
16                          OP_ANYCHAR
03 62 63                    OP_EXACT2           bc
3D 06 00 00 00
41 3B 00 00 00 67
36 01 00
50
41 26 00 00 00 66
10 [bitset]
3D D4 FF FF FF
51
02 66                       OP_EXACT1           f
39 01 00
3D BF FF FF FF
02 67                       OP_EXACT1           g
3D 02 00 00 00
02 68                       OP_EXACT1           h
01                          OP_END
```

/^.bc([de]*f)+g|h/

```
3E 52 00 00 00
2A                          OP_BEGIN_LINE
16                          OP_ANYCHAR
03 62 63                    OP_EXACT2           bc
3D 06 00 00 00
41 3B 00 00 00 67
36 01 00
50
41 26 00 00 00 66
10 [bitset]                 OP_CCLASS           [bitset with d,e]
3D D4 FF FF FF
51
02 66                       OP_EXACT1           f
39 01 00
3D BF FF FF FF
02 67                       OP_EXACT1           g
3D 02 00 00 00
02 68                       OP_EXACT1           h
01                          OP_END
```

```
3E 52 00 00 00          OP_PUSH                 82
2A                      OP_BEGIN_LINE
16                      OP_ANYCHAR
03 62 63                OP_EXACT2               bc
3D 06 00 00 00          OP_JUMP                 6
41 3B 00 00 00 67       OP_PUSH_IF_PEEK_NEXT    59, g
36 01 00
50
41 26 00 00 00 66       OP_PUSH_IF_PEEK_NEXT    38, f
10 [bitset]             OP_CCLASS               [bitset with d,e]
3D D4 FF FF FF          OP_JUMP                 -44
51
02 66                   OP_EXACT1               f
39 01 00
3D BF FF FF FF          OP_JUMP                 -65
02 67                   OP_EXACT1               g
3D 02 00 00 00          OP_JUMP                 2
02 68                   OP_EXACT1               h
01                      OP_END
```

/^.bc([de]*f)+g|h/

```
3E 52 00 00 00                OP_PUSH                 82
2A                            OP_BEGIN_LINE
16                            OP_ANYCHAR
03 62 63                      OP_EXACT2               bc
3D 06 00 00 00                OP_JUMP                 6
41 3B 00 00 00 67             OP_PUSH_IF_PEEK_NEXT    59, g
36 01 00
50                            OP_PUSH_STOP_BT
41 26 00 00 00 66             OP_PUSH_IF_PEEK_NEXT    38, f
10 [bitset]                   OP_CCLASS               [bitset with d,e]
3D D4 FF FF FF                OP_JUMP                 -44
51                            OP_POP_STOP_BT
02 66                         OP_EXACT1               f
39 01 00
3D BF FF FF FF                OP_JUMP                 -65
02 67                         OP_EXACT1               g
3D 02 00 00 00                OP_JUMP                 2
02 68                         OP_EXACT1               h
01                            OP_END
```

/^.bc([de]*f)+g|h/

```
3E 52 00 00 00          OP_PUSH                 82
2A                      OP_BEGIN_LINE
16                      OP_ANYCHAR
03 62 63                OP_EXACT2               bc
3D 06 00 00 00          OP_JUMP                 6
41 3B 00 00 00 67       OP_PUSH_IF_PEEK_NEXT    59, g
36 01 00                OP_MEM_START_PUSH       1
50                      OP_PUSH_STOP_BT
41 26 00 00 00 66       OP_PUSH_IF_PEEK_NEXT    38, f
10 [bitset]             OP_CCLASS               [bitset with d,e]
3D D4 FF FF FF          OP_JUMP                 -44
51                      OP_POP_STOP_BT
02 66                   OP_EXACT1               f
39 01 00                OP_MEM_END              1
3D BF FF FF FF          OP_JUMP                 -65
02 67                   OP_EXACT1               g
3D 02 00 00 00          OP_JUMP                 2
02 68                   OP_EXACT1               h
01                      OP_END
```

/^.bc([de]*f)+g|h/

Regex as Easy as

3E 0A 00 00 00 36 01 00 04 61 62 63 39 01 00 01

Daniel Colson

Thank you!

Regex Engine

- hparker.xyz/ruby-regex-engine/
- rubyevents.org/talks/make-your-own-regex-engine

Ruby VM

- rubyevents.org/talks/a-rails-developer-s-guide-to-the-ruby-vm
- patshaughnessy.net/ruby-under-a-microscope