LL(1) Error Recovery

Motivation

- Sometimes, input is not syntactically correct
- Several options:
  - Give up immediately with error message
  - Try to guess at what user meant; insert/change tokens; continue
  - Discard some input and/or stack and resume

Give Up

- Pro: Easy
- Con: Not very useful
- Con: Not helpful if we're doing IntelliSense style parsing

Guess

- Pro: If we guess right, we can fix the program for the user!
  - They'll love us!
- Con: If the user doesn't know what they want, how the heck can *we* know?

Re-Sync

- Pro: This might be doable
- Con: We have to actually do it.

Panic Mode

- When parser hits an error, it enters *panic mode*
- "Hits an error" means either:
  - We have terminal on top of stack, but next terminal of input doesn't match
  - We have nonterminal on top of stack, but it can't expand to anything beginning with next terminal of input (i.e., table[stack][input] is empty)

Heuristic

- Goal: Throw out tokens (and maybe stack symbols) until we reach a point where we can resynchronize
- We can handle this two possible ways

Method 1

- This way is less flexible, but is simpler
- Decide on a set of *synchronizing tokens*
- Ex: For C/C++/Java/C#/Javascript: We might choose semicolon as our synchronizing token
- When we hit an error:
  - Discard tokens until one of our synchronizing tokens appears
  - Pop stack until we see a matching synchronizing token
- Then resume the parse

Method 2

- Second method is a bit more complex, but potentially more flexible

```
if error:
    while 1:
        if no more input:
            halt
        elif next token in follow[stack top]:
            stack.pop()
            break
        else:
            discard input token
```

Rationale

- If we've run out of input, we really can't do anything else except report where the early EOF was
- If next token's type is in follow[stack top]:
  - We're guessing user meant to finish up a nonterminal, but they left some things out
  - So, pop the stack. We're now resynchronized
- In any other case, discard an input token and try again

Analysis

- Can we end up in an infinite loop?
  - No
  - We always either pop stack or discard a token
  - We never push anything to stack
  - No way to keep doing either of these forever

Code

- We could add error entries to parse table:

```
...build table as before...
for all nonterminals N:
    for T in union(terminals,set("$")):
        if table[N][T] is empty:
            if T in follow[N]:
                table[N][T] = "pop stack"
            else:
                table[N][T] = "discard token"
```

Code

- Or just extend the parse algorithm:

```
panicking = False
while stack not empty and tokens remain:
    if panicking:
        if next token in follow[stack top]:
            stack.pop(); panicking = False
        else:
            discard next token
    elif stack top is terminal:
        if next token symbol == stack top symbol:
            consume token; pop stack
        else:
            panicking = True
    else:
        if next token symbol in table[stack top symbol]:
            stack.pop()
            for s in reversed(table[stack top][next token]):
                stack.push( s )
        else:
            panicking = True
```

Example 1: Table

	$	ADDOP	ELSE	EQ	ID	IF	LBR	LP	MULOP	NUM	RBR	RP	SEMI	WHILE
S	λ	•	•	•	stmt SEMI S	stmt SEMI S	•	•	•	•	λ	•	•	•
a-o-f	•	•	•	•	ID a-o-f'	•	•	•	•	•	•	•	•	•
a-o-f'	•	•	•	EQ e	•	•	•	LP a-o-f''	•	•	•	•	•	•
a-o-f''	•	•	•	•	e RP	•	•	e RP	•	e RP	•	RP	•	•
cond	•	•	•	•	•	IF LP e RP LBR S RBR cond'	•	•	•	•	•	•	•	•
cond'	•	•	ELSE LBR S RBR	•	•	•	•	•	•	•	•	•	λ	•
e	•	•	•	•	t e'	•	•	t e'	•	t e'	•	•	•	•
e'	•	ADDOP t e'	•	•	•	•	λ	•	•	•	•	λ	λ	•
f	•	•	•	•	ID	•	•	LP e RP	•	NUM	•	•	•	•
loop	•	•	•	•	•	•	•	•	•	•	•	•	•	WHILE e LBR S RBR
stmt	•	•	•	•	a-o-f	cond	•	•	•	•	•	•	•	•
t	•	•	•	•	f t'	•	•	f t'	•	f t'	•	•	•	•
t'	•	λ	•	•	•	•	λ	•	MULOP f t'	•	•	λ	λ	•

Example 1

- Input:
  x = ; y = 42 z=99 ; w = 0

Example 1: Actions

Stack	Tokens	Action
$ SEMI **stmt**	**x = ; y = 42 z = 99 ; w = 0 $**	Expand S→stmt SEMI S
$ SEMI **a-o-f**	**x = ; y = 42 z = 99 ; w = 0 $**	Expand stmt→a-o-f
$ SEMI a-o-f' **ID**	**x = ; y = 42 z = 99 ; w = 0 $**	Expand a-o-f→ID a-o-f'
$ SEMI **a-o-f'**	**x = ; y = 42 z = 99 ; w = 0 $**	Match and pop
$ SEMI e **EQ**	**x = ; y = 42 z = 99 ; w = 0 $**	Expand a-o-f'→EQ e
$ SEMI **e**	**x = ; y = 42 z = 99 ; w = 0 $**	Match and pop
$ SEMI **e**	**x = ; y = 42 z = 99 ; w = 0 $**	Parse error detected
$ SEMI	**x = ; y = 42 z = 99 ; w = 0 $**	PANIC: Pop stack; resume
$	**x = ; y = 42 z = 99 ; w = 0 $**	Match and pop
$ SEMI **stmt**	**x = ; y = 42 z = 99 ; w = 0 $**	Expand S→stmt SEMI S
$ SEMI **a-o-f**	**x = ; y = 42 z = 99 ; w = 0 $**	Expand stmt→a-o-f
$ SEMI a-o-f' **ID**	**x = ; y = 42 z = 99 ; w = 0 $**	Expand a-o-f→ID a-o-f'
$ SEMI **a-o-f'**	**x = ; y = 42 z = 99 ; w = 0 $**	Match and pop
$ SEMI e **EQ**	**x = ; y = 42 z = 99 ; w = 0 $**	Expand a-o-f'→EQ e
$ SEMI **e**	**x = ; y = 42 z = 99 ; w = 0 $**	Match and pop
$ SEMI e' **t**	**x = ; y = 42 z = 99 ; w = 0 $**	Expand e→t e'
$ SEMI e' t' **f**	**x = ; y = 42 z = 99 ; w = 0 $**	Expand t→f t'
$ SEMI e' t' **NUM**	**x = ; y = 42 z = 99 ; w = 0 $**	Expand f→NUM
$ SEMI e' **t'**	**x = ; y = 42 z = 99 ; w = 0 $**	Match and pop
$ SEMI e' **t'**	**x = ; y = 42 z = 99 ; w = 0 $**	Parse error detected
$ SEMI e' **t'**	**x = ; y = 42 z = 99 ; w = 0 $**	PANIC: Discard token
$ SEMI e' **t'**	**x = ; y = 42 z = 99 ; w = 0 $**	PANIC: Discard token
$ SEMI e' **t'**	**x = ; y = 42 z = 99 ; w = 0 $**	PANIC: Discard token
$ SEMI **e'**	**x = ; y = 42 z = 99 ; w = 0 $**	PANIC: Pop stack; resume
$ SEMI	**x = ; y = 42 z = 99 ; w = 0 $**	Expand e'→λ
$	**x = ; y = 42 z = 99 ; w = 0 $**	Match and pop
$ SEMI **stmt**	**x = ; y = 42 z = 99 ; w = 0 $**	Expand S→stmt SEMI S
$ SEMI **a-o-f**	**x = ; y = 42 z = 99 ; w = 0 $**	Expand stmt→a-o-f
$ SEMI a-o-f' **ID**	**x = ; y = 42 z = 99 ; w = 0 $**	Expand a-o-f→ID a-o-f'
$ SEMI **a-o-f'**	**x = ; y = 42 z = 99 ; w = 0 $**	Match and pop
$ SEMI e **EQ**	**x = ; y = 42 z = 99 ; w = 0 $**	Expand a-o-f'→EQ e
$ SEMI **e**	**x = ; y = 42 z = 99 ; w = 0 $**	Match and pop
$ SEMI e' **t**	**x = ; y = 42 z = 99 ; w = 0 $**	Expand e→t e'
$ SEMI e' t' **f**	**x = ; y = 42 z = 99 ; w = 0 $**	Expand t→f t'
$ SEMI e' t' **NUM**	**x = ; y = 42 z = 99 ; w = 0 $**	Expand f→NUM
$ SEMI e' **t'**	**x = ; y = 42 z = 99 ; w = 0 $**	Match and pop
$ SEMI e' **t'**	**x = ; y = 42 z = 99 ; w = 0 $**	Parse error detected
$ SEMI e' **t'**	**x = ; y = 42 z = 99 ; w = 0 $**	PANIC: Discard token
$ SEMI e' **t'**	**x = ; y = 42 z = 99 ; w = 0 $**	Error: Early EOF

Example 2

- Input:

```
if( foo ) {
    if( bar ) {
        x=42;
    y=12;
    z=100;
}
```

Example 2

Stack	Tokens	Action
S SEMI stmt	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand S→stmt SEMI S
S SEMI cond	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand stmt→cond
S SEMI cond' RBR S LBR RP e LP IF	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand cond→IF LP e RP LBR S RBR cond'
S SEMI cond' RBR S LBR RP e LP	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S LBR RP e	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S LBR RP e' t	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand e→t e'
S SEMI cond' RBR S LBR RP e' t' f	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand t→f t'
S SEMI cond' RBR S LBR RP e' t' ID	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand f→ID
S SEMI cond' RBR S LBR RP e' t'	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S LBR RP e'	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand t'→λ
S SEMI cond' RBR S LBR RP	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand e'→λ
S SEMI cond' RBR S LBR	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI stmt	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand S→stmt SEMI S
S SEMI cond' RBR S SEMI cond	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand stmt→cond
S SEMI cond' RBR S SEMI cond' RBR S LBR RP e LP IF	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand cond→IF LP e RP LBR S RBR cond'
S SEMI cond' RBR S SEMI cond' RBR S LBR RP e LP	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR S LBR RP e	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR S LBR RP e' t	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand e→t e'
S SEMI cond' RBR S SEMI cond' RBR S LBR RP e' t' f	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand t→f t'
S SEMI cond' RBR S SEMI cond' RBR S LBR RP e' t' ID	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand f→ID
S SEMI cond' RBR S SEMI cond' RBR S LBR RP e' t'	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR S LBR RP e'	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand t'→λ
S SEMI cond' RBR S SEMI cond' RBR S LBR RP	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand e'→λ
S SEMI cond' RBR S SEMI cond' RBR S LBR	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR S	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR S SEMI stmt	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand S→stmt SEMI S
S SEMI cond' RBR S SEMI cond' RBR S SEMI a-o-f	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand stmt→a-o-f
S SEMI cond' RBR S SEMI cond' RBR S SEMI a-o-f' ID	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand a-o-f→ID a-o-f'
S SEMI cond' RBR S SEMI cond' RBR S SEMI a-o-f'	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR S SEMI e EQ	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand a-o-f'→EQ e
S SEMI cond' RBR S SEMI cond' RBR S SEMI e	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR S SEMI e' t	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand e→t e'
S SEMI cond' RBR S SEMI cond' RBR S SEMI e' t' f	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand t→f t'
S SEMI cond' RBR S SEMI cond' RBR S SEMI e' t' NUM	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand f→NUM
S SEMI cond' RBR S SEMI cond' RBR S SEMI e' t'	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR S SEMI e'	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand t'→λ
S SEMI cond' RBR S SEMI cond' RBR S SEMI	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand e'→λ
S SEMI cond' RBR S SEMI cond' RBR S	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR S SEMI stmt	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand S→stmt SEMI S
S SEMI cond' RBR S SEMI cond' RBR S SEMI a-o-f	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand stmt→a-o-f
S SEMI cond' RBR S SEMI cond' RBR S SEMI a-o-f' ID	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand a-o-f→ID a-o-f'
S SEMI cond' RBR S SEMI cond' RBR S SEMI a-o-f'	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR S SEMI e EQ	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand a-o-f'→EQ e
S SEMI cond' RBR S SEMI cond' RBR S SEMI e	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR S SEMI e' t	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand e→t e'
S SEMI cond' RBR S SEMI cond' RBR S SEMI e' t' f	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand t→f t'
S SEMI cond' RBR S SEMI cond' RBR S SEMI e' t' NUM	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand f→NUM
S SEMI cond' RBR S SEMI cond' RBR S SEMI e' t'	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR S SEMI e'	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand t'→λ
S SEMI cond' RBR S SEMI cond' RBR S SEMI	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand e'→λ
S SEMI cond' RBR S SEMI cond' RBR S	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR S SEMI stmt	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand S→stmt SEMI S
S SEMI cond' RBR S SEMI cond' RBR S SEMI a-o-f	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand stmt→a-o-f
S SEMI cond' RBR S SEMI cond' RBR S SEMI a-o-f' ID	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand a-o-f→ID a-o-f'
S SEMI cond' RBR S SEMI cond' RBR S SEMI a-o-f'	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR S SEMI e EQ	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand a-o-f'→EQ e
S SEMI cond' RBR S SEMI cond' RBR S SEMI e	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR S SEMI e' t	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand e→t e'
S SEMI cond' RBR S SEMI cond' RBR S SEMI e' t' f	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand t→f t'
S SEMI cond' RBR S SEMI cond' RBR S SEMI e' t' NUM	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand f→NUM
S SEMI cond' RBR S SEMI cond' RBR S SEMI e' t'	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR S SEMI e'	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand t'→λ
S SEMI cond' RBR S SEMI cond' RBR S SEMI	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand e'→λ
S SEMI cond' RBR S SEMI cond' RBR S	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Expand S→λ
S SEMI cond' RBR S SEMI cond'	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond'	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Parse error detected
S SEMI cond' RBR S SEMI cond'	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	PANIC: Discard token
S SEMI cond' RBR S SEMI cond'	if (foo) { if (bar) { x = 42 ; y = 12 ; z = 100 ; } $	Error: Early EOF

Example 3

- Input:

```
if( foo ) {
    if( bar )
        x=42;
    }
    y=12;
    z=100;
}
```

Example 3

Stack	Tokens	Action
S SEMI **stmt**	**if** (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Expand S→stmt SEMI S
S SEMI **cond**	**if** (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Expand stmt→cond
S SEMI cond' RBR S LBR RP *e* LP **IF**	**if** (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Expand cond→IF LP e RP LBR S RBR cond'
S SEMI cond' RBR S LBR RP *e* **LP**	if **(** foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S LBR RP *e* **e**	if (**foo**) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S LBR RP *e'* **t**	if (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Expand e→t e'
S SEMI cond' RBR S LBR RP *e'* *t'* **f**	if (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Expand t→f t'
S SEMI cond' RBR S LBR RP *e'* *t'* **ID**	if (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Expand f→ID
S SEMI cond' RBR S LBR RP *e'* **t'**	if (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S LBR RP **e'**	if (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Expand t'→λ
S SEMI cond' RBR S LBR **RP**	if (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Expand e'→λ
S SEMI cond' RBR S **LBR**	if (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR **S**	if (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI **stmt**	if (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Expand S→stmt SEMI S
S SEMI cond' RBR S SEMI **cond**	if (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Expand stmt→cond
S SEMI cond' RBR S SEMI cond' RBR S LBR RP *e* LP **IF**	if (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Expand cond→IF LP e RP LBR S RBR cond'
S SEMI cond' RBR S SEMI cond' RBR S LBR RP *e* **LP**	if (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR S LBR RP *e* **e**	if (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR S LBR RP *e'* **t**	if (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Expand e→t e'
S SEMI cond' RBR S SEMI cond' RBR S LBR RP *e'* *t'* **f**	if (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Expand t→f t'
S SEMI cond' RBR S SEMI cond' RBR S LBR RP *e'* *t'* **ID**	if (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Expand f→ID
S SEMI cond' RBR S SEMI cond' RBR S LBR RP *e'* **t'**	if (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR S LBR RP **e'**	if (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Expand t'→λ
S SEMI cond' RBR S SEMI cond' RBR S LBR **RP**	if (foo) { **if** (bar) x = 42 ; } y = 12 ; z = 100 ; } $	Expand e'→λ
S SEMI cond' RBR S SEMI cond' RBR S **LBR**	if (foo) { **if** (bar) **x** = 42 ; } y = 12 ; z = 100 ; } $	Match and pop
S SEMI cond' RBR S SEMI cond' RBR **S** **LBR**	if (foo) { **if** (bar) **x** = 42 ; } y = 12 ; z = 100 ; } $	Error detected: Token mismatch
S SEMI cond' RBR S SEMI cond' RBR **S** **LBR**	if (foo) { **if** (bar) **x** = 42 ; } y = 12 ; z = 100 ; } $	Error detected: Token mismatch
S SEMI cond' RBR S SEMI cond' RBR **S** **LBR**	if (foo) { **if** (bar) **x** = 42 ; } y = 12 ; z = 100 ; } $	Error: Trailing tokens

Sources

- K. Louden. *Compiler Construction: Principles and Practice*. PWS Publishing.
- Bob Nystrom. Crafting Interpreters. http://craftinginterpreters.com/parsing-expressions.html#syntax-errors

Created using LaTeX.

Main font: Gentium Book Basic, by Victor Gaultney. See http://software.sil.org/gentium/
Monospace font: Source Code Pro, by Paul D. Hunt. See https://fonts.google.com/specimen/Source+Code+Pro and http://sourceforge.net/adobe
Icons by Ulisse Perusin, Steven Garrity, Lapo Calamandrei, Ryan Collier, Rodney Dawes, Andreas Nilsson, Tuomas Kuosmanen, Garrett LeSage, and Jakub Steiner. See http://tango-project.org