

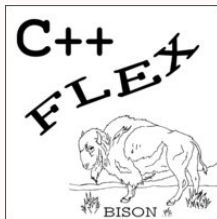
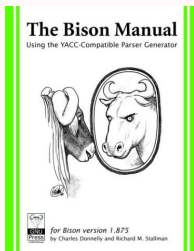


The Bison Parser Generator

Part I

October 9, 2017

Brian A. Malloy



Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 1 of 28

Go Back

Full Screen

Quit



1. Parsing Overview

- Parsing methods partition into two approaches:
 1. **Top-down**: Find a derivation of a string, starting the start symbol, and performing a top-down expansion of the formal grammar rules.
 2. Examples of Top-down: Table driven LL, and recursive descent
 3. **Bottom-up**: parser starts with the input, and attempts to rewrite it to the start symbol
 4. Examples of bottom-up: SLR, LALR, IELR, LR, GLR
 5. Bison is LALR(1)

Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 2 of 28

Go Back

Full Screen

Quit



Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts:...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 3 of 28

Go Back

Full Screen

Quit

2. Why Understand Bison?

- We want to build an interpreter for Python:
⇒ we need a parser
- Developing a parser is a challenging task, made even more so if you don't understand how the parser generator works.
- Understanding Bison can help:
 - To insert semantic actions
 - To remove conflicts,
 - to insert error recovery routines,
 - to develop a new grammar.



Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 4 of 28

Go Back

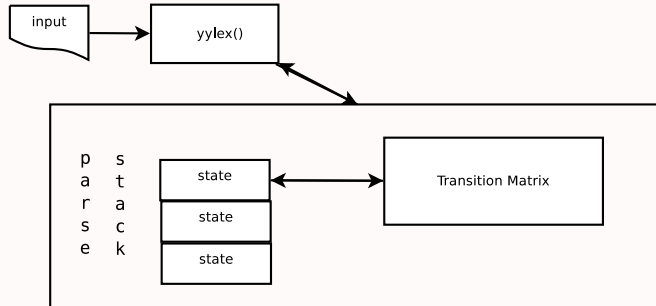
Full Screen

Quit

3. Bison Overview

- Bison performs a bottom-up parse, with 1 symbol look-ahead.
- bison consists of:
 1. A stack (PDA) with current states,
 2. a transition matrix to derive a new state,
 3. a table of user-defined actions, and
 4. an interpreter to choreograph the actions.
- All of this is packaged in `yyparse()`, a function that calls `yylex` to get the next token.

3.1. Visualizing Bison



Slide 5 of 28

Go Back

Full Screen

Quit



Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 6 of 28

Go Back

Full Screen

Quit

3.2. Shift/Reduce Parsing: **shift**

- Bison looks for rules that might match the tokens seen so far.
- Bison creates a set of states, each of which reflects a possible position in one or more partially parsed rules
- Each time bison “reads” a token that doesn’t complete a rule it pushes the token onto the stack and goes to a new state.
- This action is called a **shift**.



Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts:...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 7 of 28

Go Back

Full Screen

Quit

3.3. Shift/reduce parsing: **reduce**

- When the symbols at the top of the stack match the right hand side (RHS) of a rule, it pops the RHS symbols off the stack, pushes the LHS onto the stack, and switches to a new state.
- When Bison reduces a rule it can execute user code associated with the rule.



Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 8 of 28

Go Back

Full Screen

Quit

3.4. Five Operations of `yyparse()`

1. `accept`: this happens only once
2. `shift` `<state>`
3. `reduce` `<formulation number>`
4. `goto` `<new state>`
5. `error`: for tokens that shouldn't appear in the current state



Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts:...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 9 of 28

Go Back

Full Screen

Quit

4. The Pointer Model

- Shows a pointer moving through grammar
- The pointer starts at the **start** rule

```
%token A B C
%%
start: ↑ A B C
```

- If A and B are read, move pointer:

```
%token A B C
%%
start: A B ↑ C
```



Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce

4.1. More than one Pointer

- Example, assume we read A, B:

```
%%  
start: x | y  
x: A B ↑ C D  
y: A B ↑ E F
```

- Pointers can disappear in two ways:
 1. A rule doesn't work; Eg next token C

```
%%  
start: x | y  
x: A B C ↑ D  
y: A B E F
```

2. Pointers merge into a common subrule



Slide 10 of 28

Go Back

Full Screen

Quit



- For merge example, assume we read A

```
start: x | y
x: A ↑ B z R
y: A ↑ B z S
z: C D
```

- After A B C, there is only one pointer:

```
start: x | y
x: A B z R
y: A B z S
z: C ↑ D
```

- After A B C D, there are two pointers

```
start: x | y
x: A B z ↑ R
y: A B z ↑ S
```

Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 11 of 28

Go Back

Full Screen

Quit



Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 12 of 28

Go Back

Full Screen

Quit

4.2. Reductions

- If there is only one pointer at the end of a rule, we reduce. E.g., assume an A is read:

```
%%  
start: x | y  
x: A ↑  
y: B
```

- After A, there is only one pointer in x, and rule x is reduced.
- Similarly, if B is read, there is only one pointer in y, and rule y is reduced.



5. Bison Conflicts: Two kinds

- Reduce/reduce conflict: two pointers at the end of two rules.

```
%%  
start: x | y  
x: A ↑  
y: A ↑
```

- Never a conflict if only one arrow:

```
%%  
start: x | y  
x: z R  
y: z S  
z: A B ↑
```

- After z is reduced, there are two pointers.
Why?

- Shift/reduce conflict: One pointer wants to shift, the other pointer wants to reduce:

```
%%  
start: x | y  
x: A ↑ R  
y: A ↑
```



Slide 14 of 28

Go Back

Full Screen

Quit

6. Parse Tree Helps



Parsing Overview

Why Understand . . .

Bison Overview

The Pointer Model

Bison Conflicts: . . .

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 15 of 28

Go Back

Full Screen

Quit



Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 16 of 28

Go Back

Full Screen

Quit

7. Bison & Ambiguity

- In the face of ambiguity, bison will make a choice, unless the user specifies otherwise.
- The user can set the precedence, refactor the grammar to remove conflicts, or suppress conflict warnings using `%expect n`
- Default action for Shift/Reduce conflict: **shift**.
- Default action for Reduce/Reduce conflict: choose the rule that appears first in the grammar.



8. Parser States

- You can generate `name.output` by supplying the `-v` flag to bison (`-v` \Rightarrow verbose)
- `name.output` is a description of the state machine generated by bison, including:
 - Listing of the grammar rules
 - Rules that are never used
 - Summary of the conflicts
 - All of the parser **states**

Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts:...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 17 of 28

Go Back

Full Screen

Quit



Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts:...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 18 of 28

Go Back

Full Screen

Quit

8.1. States in name.output

- For each state, bison lists
 - The rules and positions that correspond to the state,
 - the shifts and reductions the parser will make when it reads various tokens in the state,
 - and the state it will switch to after a reduction produces a non-terminal in that state,
 - the conflicts in each state.



Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts:...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 19 of 28

Go Back

Full Screen

Quit

8.2. What are States

- Each state corresponds to a unique combination of possible pointers in the bison grammar.
- Bison assigns a number to each state:

State 0 conflicts: 1 shift/reduce

- Numbers can differ across bison implementations; the actual number isn't significant.



9. Tracing reduce/reduce

```
start: a Y | b Y;  
a:    X ;  
b:    X ;
```

- We will trace bison as it parses the above ambiguous grammar
- assume the input is `X Y <eof>`
- bison always starts at `state 0`, `shift 0`; i.e., push `0` onto the stack.

Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 20 of 28

Go Back

Full Screen

Quit



Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts:...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 21 of 28

Go Back

Full Screen

Quit

- (1) Starting parse
- (2) Entering state 0
- (3) Reading a token: -(end of buffer or a NUL)
- (4) X
- (5) -accepting rule at line 10 ("X")
- (6) Next token is token X ()
- (7) Shifting token X ()
- (8) Entering state 1

- (1) Bison announces the start of the parse
- (2) Bison always begins in state 0
- (3) Bison is about to read a token
- (4) reads X
- (5) This line is debug info from Flex
- (8) Bison enters **state 1**, stack is:
(1, x) (0, -)



Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts:...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 22 of 28

Go Back

Full Screen

Quit

state 1

3 a: X ↑ (arrow shows where we are before next token)

4 b: X ↑

Y reduce using rule 3 (a)

Y [reduce using rule 4 (a)]

\$default reduce using rule 3 (a)

(1) Reducing stack by rule 3 (line 12):

(2) \$1 = token X ()

(3) -> \$\$ = nterm a ()

(4) Stack now 0

(5) Entering state 3

- (1) line 12 is in parse.y
- (3) nterm a is non-terminal a
- (5) stack is: (3, a) (0, -)



Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 23 of 28

Go Back

Full Screen

Quit

- (1) Reading a token: –accepting rule at line 12 (“
- (2) ”)
- (3) –(end of buffer or a NUL)
- (4) Y
- (5) –accepting rule at line 11 (“Y”)
- (6) Next token is token Y (
- (7) Shifting token Y (
- (8) Entering state 6

- (1), (3), (5) is from Flex
- (7) and (8) push (6, Y) onto stack
- stack is:
 - (6, Y)
 - (3, a)
 - (0, -)



Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 24 of 28

Go Back

Full Screen

Quit

state 6

1 start: a Y ↑

\$default reduce using rule 1 (start)

(1) Reducing stack by rule 1 (line 9):

(2) \$1 = nterm a ()

(3) \$2 = token Y ()

(4) -> \$\$ = nterm start ()

(5) Stack now 0

(6) Entering state 2

- recall that stack is: (6, Y) (3, a) (0, -)
- (1) reduce by rule 1 pops Y and a off stack
- \$default & (4) pushes start onto stack
- (5) says stack is state 0, start: goto state 2



Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 25 of 28

Go Back

Full Screen

Quit

state 0

0 \$accept: ↑ start \$end

...

start: goto state 2

- Above, on **start** goto state 2
- stack is:
(2, start),
(0, -)



Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 26 of 28

Go Back

Full Screen

Quit

state 2

0 \$accept: ↑ start \$end

\$end shift, and goto state 5

- Bison now reads \$eof, shifts it onto the stack, and goes to state 5
- stack is:
 - (5, \$end)
 - (2, start)
 - (0, -)

- (1) -EOF (start condition 0)
- (2) Now at end of input.
- (3) Shifting token \$end ()
- (4) Entering state 5

- Recall, stack is:
(5, \$end)
(2, start)
(0, -)

- (1) Entering state 5
- (2) Stack now 0 2 5
- (3) Cleanup: popping token \$end ()
- (4) Cleanup: popping nterm start ()
- (5) Syntactically correct.



Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts:...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 27 of 28

Go Back

Full Screen

Quit



Parsing Overview

Why Understand...

Bison Overview

The Pointer Model

Bison Conflicts:...

Parse Tree Helps

Bison & Ambiguity

Parser States

Tracing reduce/reduce



Slide 28 of 28

Go Back

Full Screen

Quit

9.1. Some notes on reduce/reduce

- The arrow (bison uses a dot) shows where we are in the grammar before reading the next token.
- For **reduce/reduce** conflicts, the two arrows are always at the end of the rules.
- In a conflict, the rule not used is shown in brackets
- In **state 1**, bison chose to reduce to **a** by **rule 3** because with **reduce/reduce** conflicts it chooses the earliest one in the grammar.