中山大學 SUN YAT-SEN UNIVERSITY 计算机学院（软件学院） SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Compilation Principle
编 译 原 理

第8讲：语法分析(5)

张献伟

xianweiz.github.io

DCS290, 3/21/2023

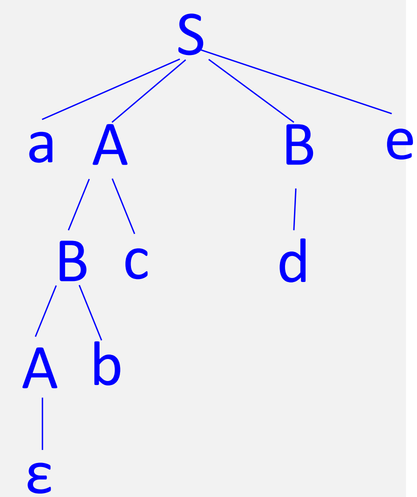
Quiz Questions

- Q1: Grammar G: S → aABe, A → Bc | ε, B → Ab | d
  give a rightmost derivation of abcde?

  S => aABe => aAde => aBcde => aAbcde => abcde

- Q2: Plot the parse tree of Q1?

  See right.



- Q3: Can G be parsed by RD-backtrack? Why?

  No. A => Bc => Abc, indirect left-recursive.

- Q4: Grammar G_2: S →A| F, A → id = exp, F→id(exp). Can G_2 be parsed by predictive parser with one lookahead? Why?

  No. A and F have a common prefix 'id'.

- Q5: Fix G_2 to make it LL(1)?

  S → id S', S' → =exp | (exp)

Cheating/Plagiarism[学术不端]

- Unauthorized use of information
  - Borrowing code: by copying, retyping, looking at a file
  - Describing: verbal desc. of code from one person to another
  - Searching the Web for solutions, discussions, tutorials, blogs …
  - Reusing your code from a previous semester …
  - …
- Unauthorized supplying of information
  - Providing copy: Giving a copy of a file to someone
  - Providing access …
  - …
- Collaborations beyond high-level, strategic advice
  - Anything more than block diagram or a few words
  - …

Why a Big Deal?

- This material is best learned by doing
  - Even though that can, at times, be difficult and frustrating
  - Starting with a copy of a program and then tweaking it is very different from writing from scratch
    - Planning, designing, organizing a program are important skills
- We are the gateway to other system courses
  - Want to make sure everyone completing the course has mastered the material
- Industry appreciates the value of this course
  - We want to make sure anyone claiming to have taken the course is prepared for the real world
- Working in teams and collaboration is an important skill
  - But only if team members have solid foundations
  - This course is about foundations, not teamwork

Cheating: Consequences[后果]

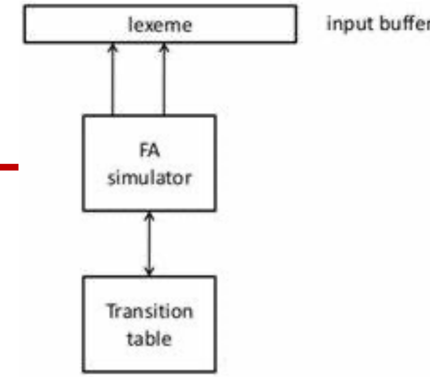
- Penalty for cheating:
  - Best case: -100% for assignment
    - ***You would be better off to turn in nothing***
  - Worst case: Removal from course with failing grade
    - This is the default
  - University-level involvement (from notification to serious things)
  - Loss of respect by you, the instructors and your colleagues
  - ***If you do cheat – come clean asap!***

- Detection of cheating:
  - We have sophisticated tools for detecting code plagiarism
  - In Fall 2015, 20 students were caught cheating and failed the course.
    - Some were **expelled** from the University
  - In January 2016, 11 students were penalized for cheating violations that occurred as far back as Spring 2014.
  - In May 2019, we gave an AIV to a student who took the course in Fall 2018 for unauthorized coaching of a Spring 2019 student. His grade was changed retroactively.

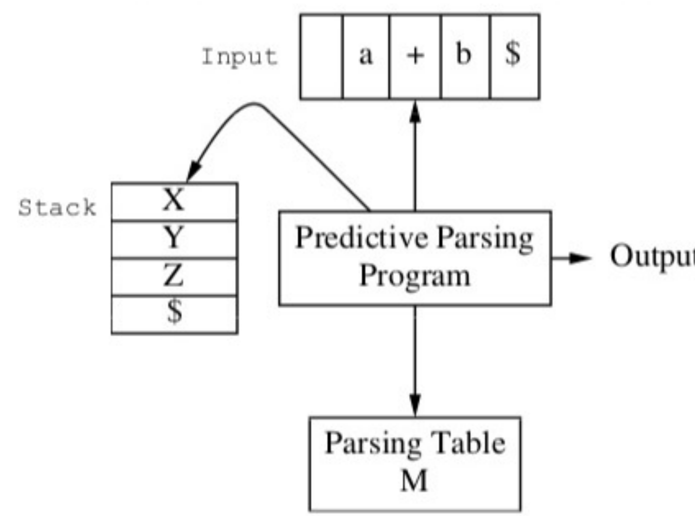
- Don't do it!
  - Manage your time carefully
  - Ask the staff for help when you get stuck
  - We will help you! We will give you extensions! We want you to succeed.

CMU15213

LL(1) Parser[非递归]



- Table-driven parser[表驱动]: amenable to automatic code generation (just like lexers)
  - **Input buffer**: contains the string to be parsed, followed by $
  - **Stack**: holds unmatched portion of derivation string, $ marks the stack end
  - **Parse table** M[A, b]: an entry containing rule "A→…" or error
  - **Parser driver** (a.k.a., predictive parsing program): next action based on <stack top, current token?>
    - Reject on reaching error state
    - Accept on end of input & empty stack



A stack records frontier of parse tree
- Non-terminals that have yet to be expanded
- Terminals that have yet to be matched against the input
- Top of stack = leftmost pending terminal or non-terminal

?: The current token is treated as lookahead token.

LL(1) Parse Table: Example

table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

E → TE'
E' → +E | ε
T → intT' | (E)
T' → *T | ε

- Implementation with 2D parse table

LL(1) Parse Table: Example

table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

E → TE'
E' → +E | ε
T → intT' | (E)
T' → *T | ε

- Implementation with 2D parse table

LL(1) Parse Table: Example

table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

E → TE'
E'→ +E | ε
T → intT' | (E)
T'→ *T | ε

- Implementation with 2D parse table
  - **First column** lists all non-terminals in the grammar
    - I.e., leftmost non-terminal in derivation

LL(1) Parse Table: Example

table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

E → TE'
E' → +E | ε
T → intT' | (E)
T' → *T | ε

- Implementation with 2D parse table
  - **First column** lists all <u>non-terminals</u> in the grammar
    - I.e., leftmost non-terminal in derivation

中山大學 SUN YAT-SEN UNIVERSITY

NSCC GZ

LL(1) Parse Table: Example

table	int	*	+	(	)	$
E	E → TE’			E → TE’		
E’			E’ → +E		E’ → ε	E’ → ε
T	T → int T’			T → (E)		
T’		T’ → *T	T’ → ε		T’ → ε	T’ → ε

E → TE’
E’→ +E | ε
T → intT’ | (E)
T’→ *T | ε

- Implementation with 2D parse table
  - **First column** lists all non-terminals in the grammar
    - I.e., leftmost non-terminal in derivation

LL(1) Parse Table: Example

table	int	*	+	(	)	$
E	E → TE’			E → TE’		
E’			E’ → +E		E’ → ε	E’ → ε
T	T → int T’			T → (E)		
T’		T’ → *T	T’ → ε		T’ → ε	T’ → ε

E → TE’
E’→ +E | ε
T → intT’ | (E)
T’→ *T | ε

- Implementation with 2D parse table
  - **First column** lists all non-terminals in the grammar
    - I.e., leftmost non-terminal in derivation
  - **First row** lists all possible terminals in the grammar and $
    - I.e., next input token

LL(1) Parse Table: Example

table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

E → TE'
E' → +E | ε
T → intT' | (E)
T' → *T | ε

- Implementation with 2D parse table
  - **First column** lists all non-terminals in the grammar
    - I.e., leftmost non-terminal in derivation
  - **First row** lists all possible terminals in the grammar and $
    - I.e., next input token

LL(1) Parse Table: Example

table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

E → TE'
E' → +E | ε
T → intT' | (E)
T' → *T | ε

- Implementation with 2D parse table
  - **First column** lists all non-terminals in the grammar
    - I.e., leftmost non-terminal in derivation
  - **First row** lists all possible terminals in the grammar and $
    - I.e., next input token

LL(1) Parse Table: Example

table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

E → TE'
E' → +E | ε
T → intT' | (E)
T' → *T | ε

- Implementation with 2D parse table
  - **First column** lists all non-terminals in the grammar
    - I.e., leftmost non-terminal in derivation
  - **First row** lists all possible terminals in the grammar and $
    - I.e., next input token
  - A **table entry** contains one production
    - One action for each <non-terminal, input> combination
    - It "predicts" the correct action based on one lookahead
    - No backtracking required

中山大學 SUN YAT-SEN UNIVERSITY

NSCC GZ

LL(1) Parse Table: Example

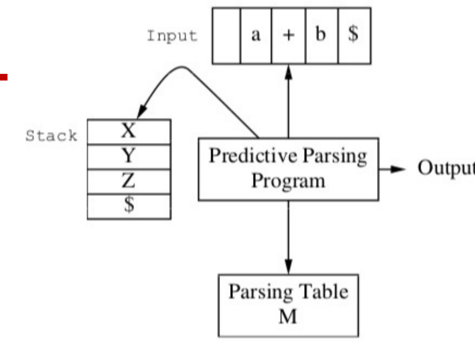
table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

E → TE'
E' → +E | ε
T → intT' | (E)
T' → *T | ε

- Implementation with 2D parse table
  - **First column** lists all non-terminals in the grammar
    - I.e., leftmost non-terminal in derivation
  - **First row** lists all possible terminals in the grammar and $
    - I.e., next input token
  - A **table entry** contains one production
    - One action for each <non-terminal, input> combination
    - It "predicts" the correct action based on one lookahead
    - No backtracking required

中山大學 SUN YAT-SEN UNIVERSITY

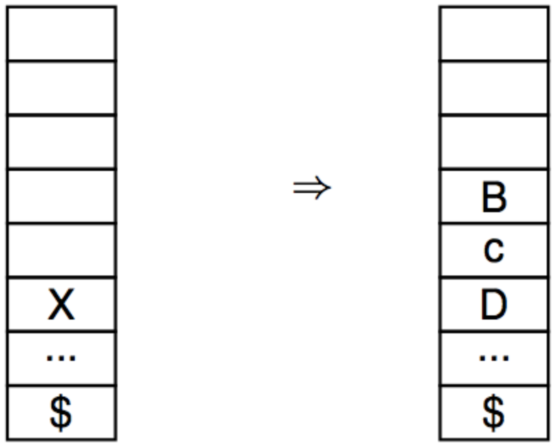
LL(1) Parsing Algorithm[算法]



- Initial state[初始态]
  - **Input** tape: input tokens followed by '$'
  - **Stack**: start symbol followed by '$' at bottom
- General idea[总体思路]: repeat one of two actions
  - **Expand** symbol at top of stack by applying a production
  - **Match** terminal symbol at top of stack with input token
- Step-by-step[每步操作] parsing based on <X, a>
  - X: symbol at the top of the stack
  - a: current input token
    - If X ∈ T, then[终结符-比较]
      - If X == a == $, parser halts with "success"
      - If X == a != $, successful match, pop X from stack and advance input head
      - If X != a, parser halts and input is rejected
    - If X ∈ N, then[非终结符-展开]
      - If M[X,a] == 'X→RHS", pop X and push RHS to stack
      - If M[X,a] == empty, parser halts and input is rejected

Push RHS in Reverse Order[逆序入栈]

- For <X, a>
  - X: symbol at the top of the stack
  - a: current input token
- If M[X,a] = "X → BcD"



逆序入栈：最左符号需要被最先展开或比较（即，最左推导），因此需在靠近栈顶位置

- Performs the <u>leftmost derivation</u>: α X β ⇒ α BcD β
  - α: string that has already been matched with input
  - β: string yet to be matched, corresponding to the ... above

Apply LL(1) Parsing to Grammar[应用]

- Consider the grammar

  E → T+E|T

  T → int*T | int | (E)

  - Left recursion? **NO!**
  - Left factoring? **YES.** E → T+E|T, T → int*T | int

- After rewriting grammar, we have

  E → TE’

  E’→ +E | ε

  T → intT’ | (E)

  T’→ *T | ε

Use the Parse Table

- To recognize "int * int"

E → TE'
E' → +E | ε
T → intT' | (E)
T' → *T | ε

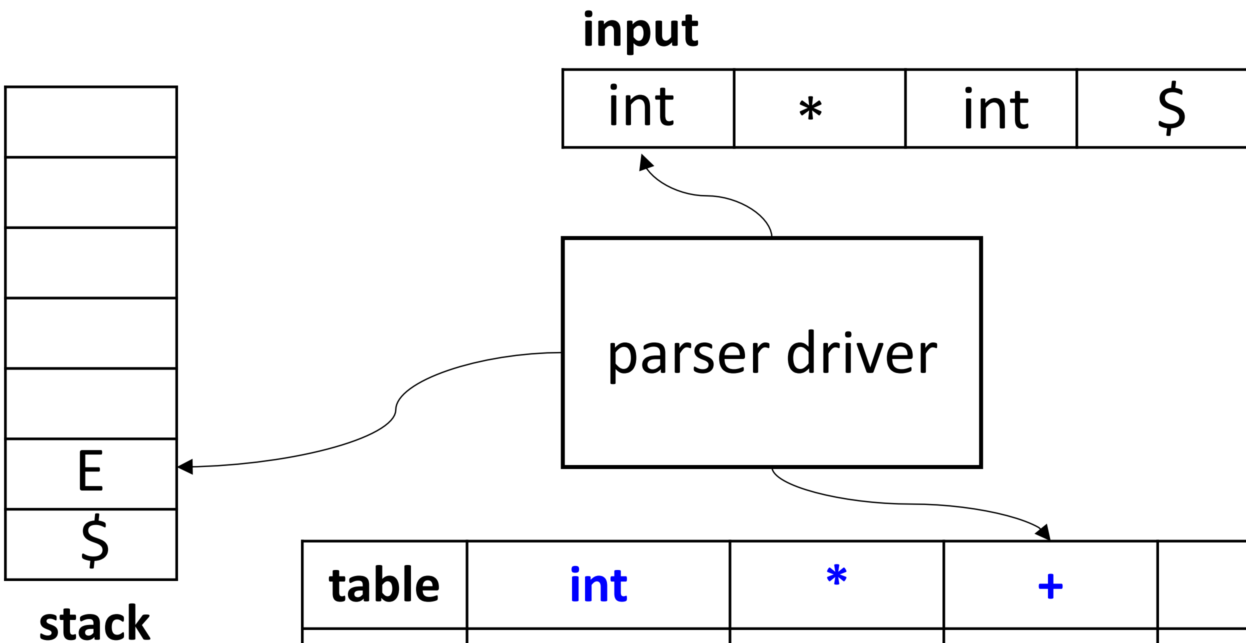


table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

Use the Parse Table

- To recognize "int * int"

E → TE'
E' → +E | ε
T → intT' | (E)
T' → *T | ε

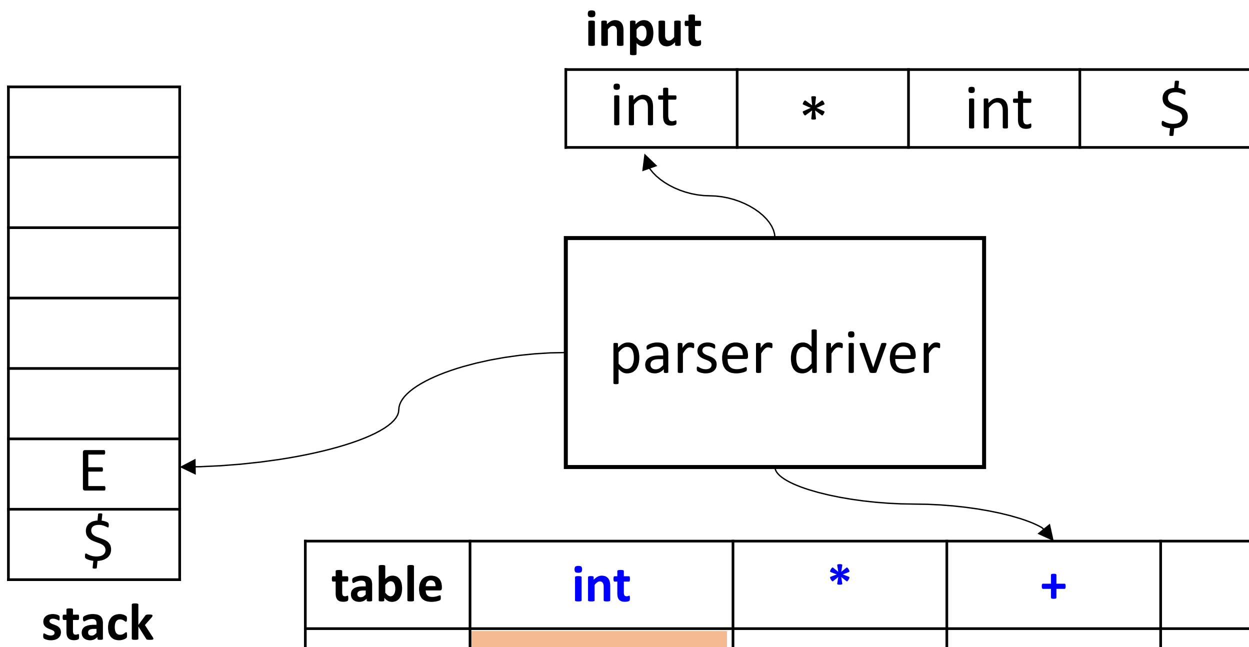


table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

Use the Parse Table

- To recognize "int * int"

E → TE'
E' → +E | ε
T → intT' | (E)
T' → *T | ε

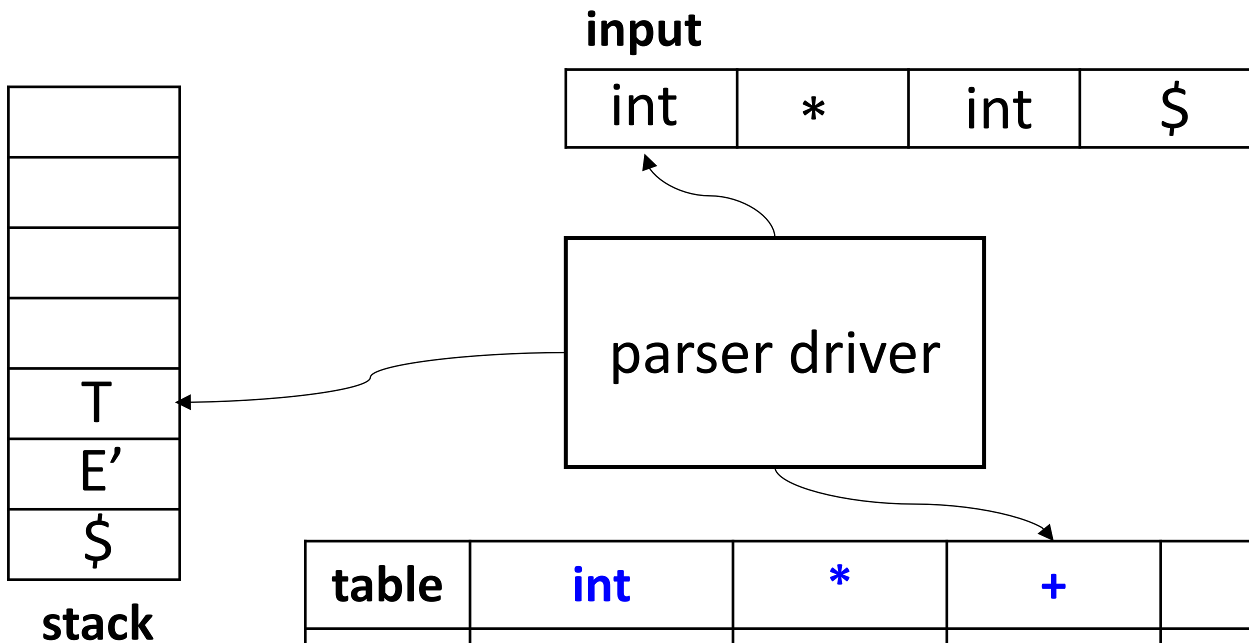


table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

Use the Parse Table

- To recognize "int * int"

E → TE'
E' → +E | ε
T → intT' | (E)
T' → *T | ε

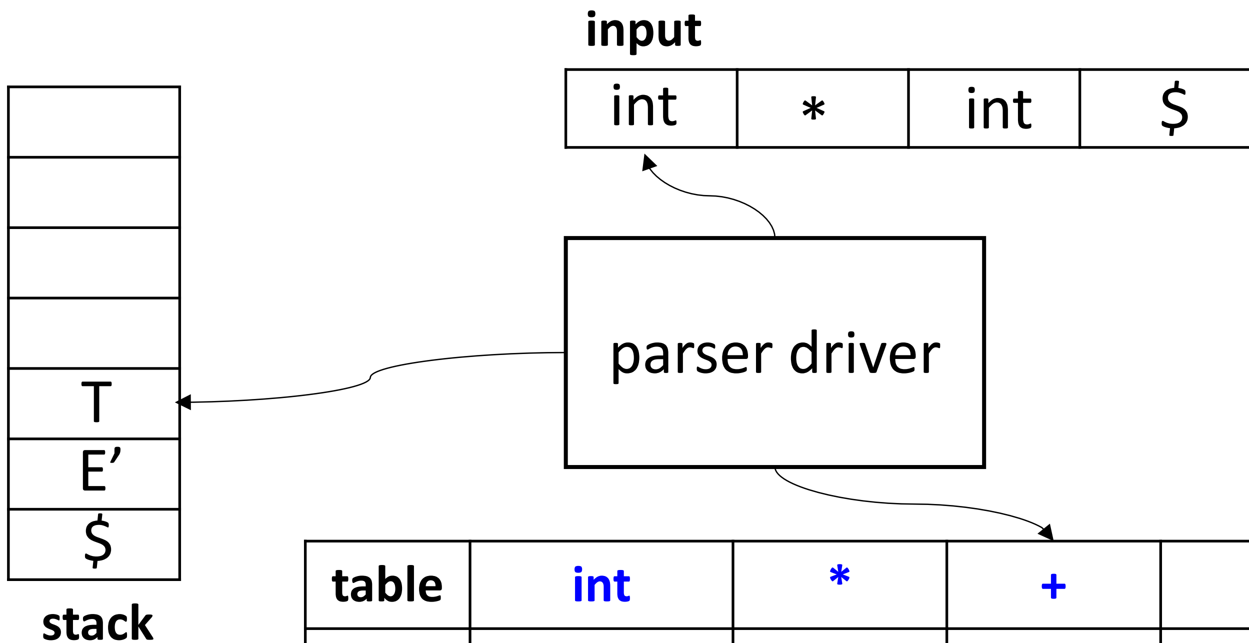


table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

Use the Parse Table

- To recognize "int * int"

E → TE'
E'→ +E | ε
T → intT' | (E)
T'→ *T | ε

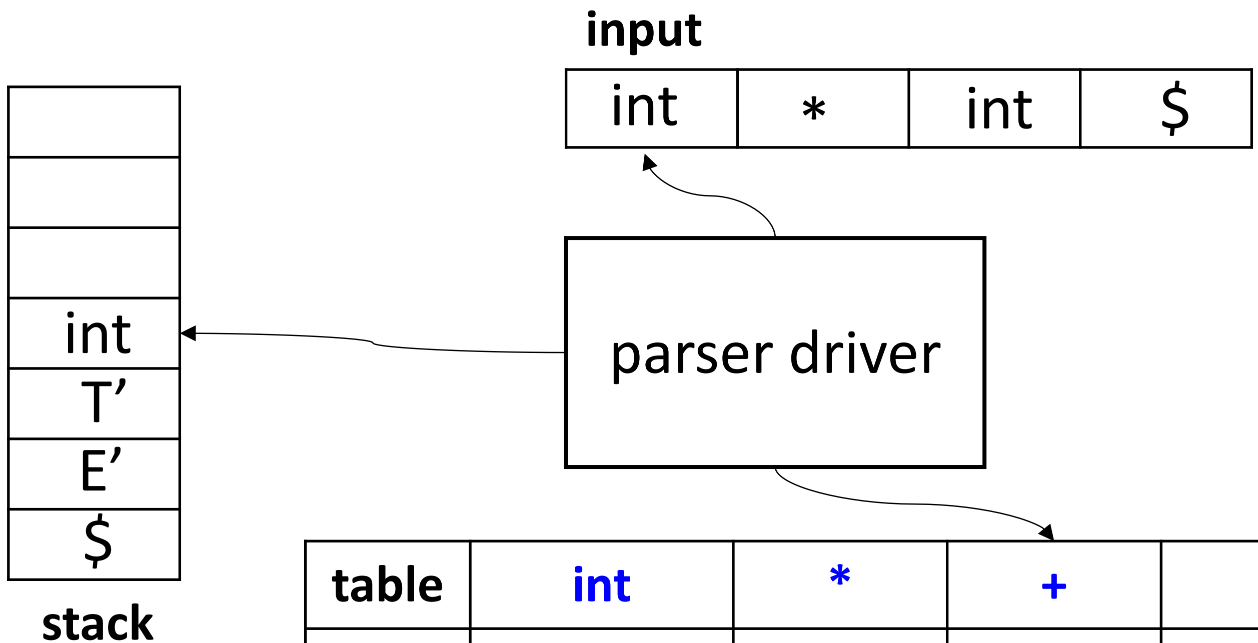


table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

Use the Parse Table

- To recognize "int * int"

E → TE'
E' → +E | ε
T → intT' | (E)
T' → *T | ε

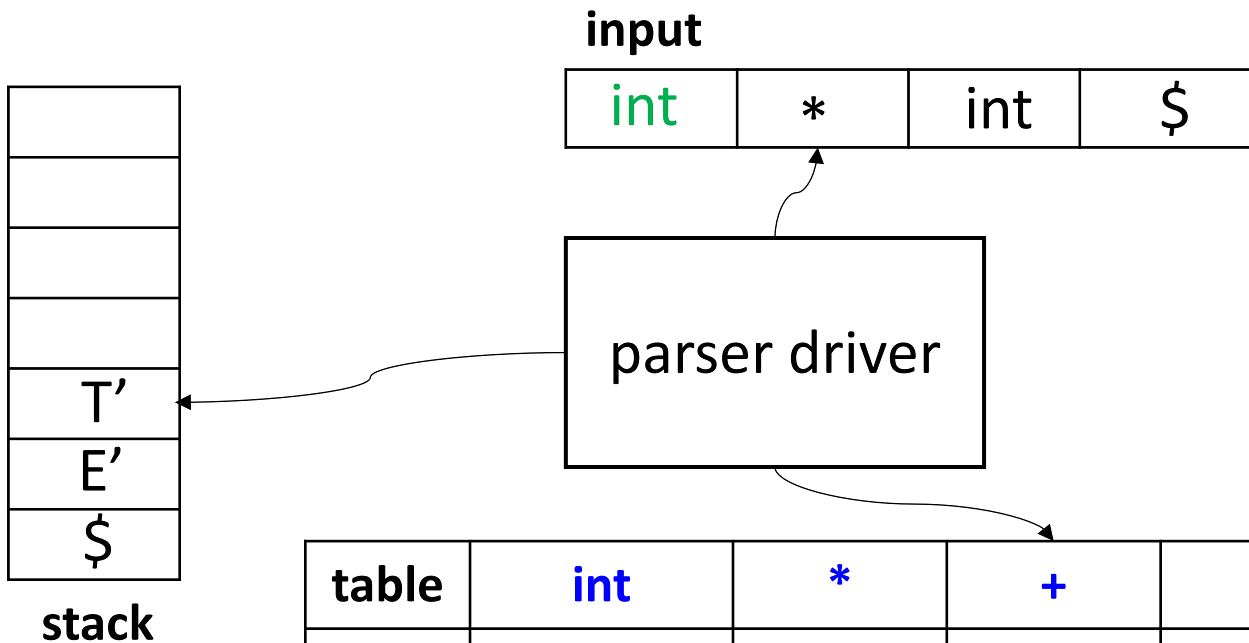


table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

Use the Parse Table

- To recognize "int * int"

E → TE'
E' → +E | ε
T → intT' | (E)
T' → *T | ε

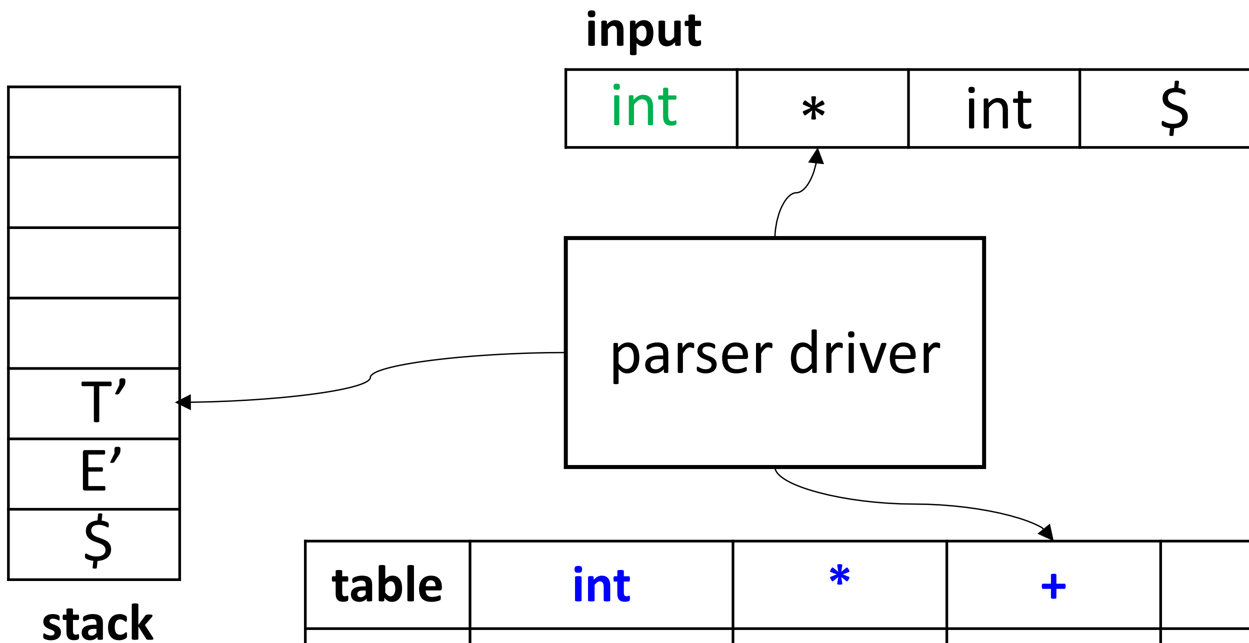


table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

Use the Parse Table

- To recognize “int * int”

E → TE’
E’→ +E | ε
T → intT’ | (E)
T’→ *T | ε

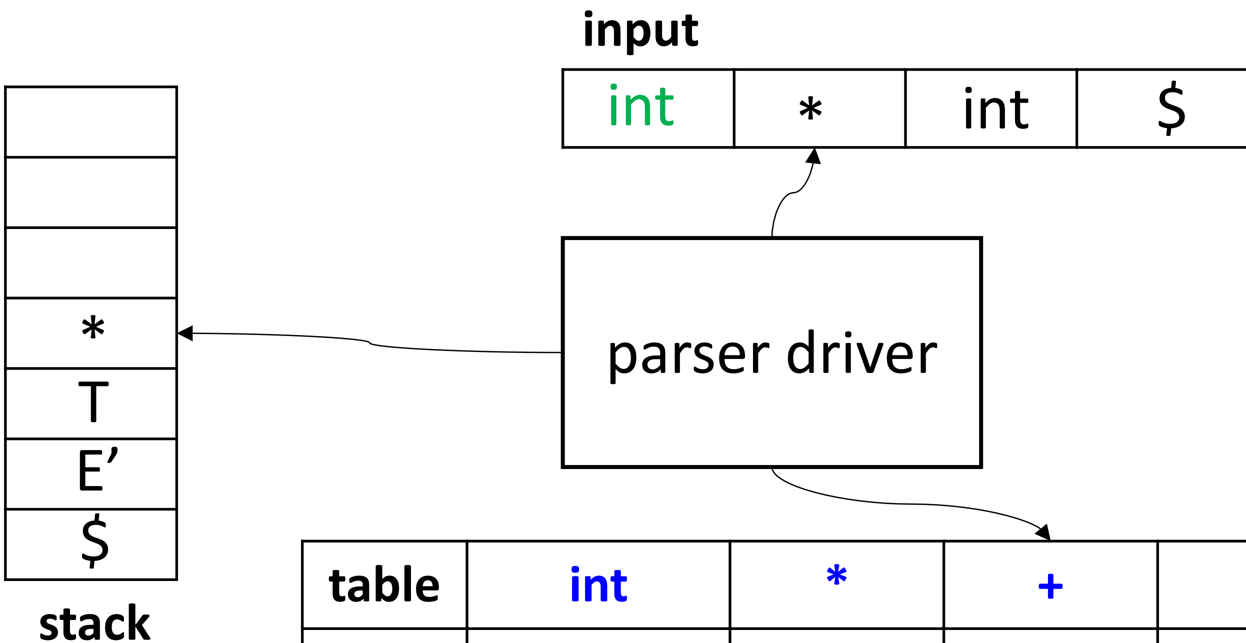


table	int	*	+	(	)	$
E	E → TE’			E → TE’		
E’			E’ → +E		E’ → ε	E’ → ε
T	T → int T’			T → (E)		
T’		T’ → *T	T’ → ε		T’ → ε	T’ → ε

Use the Parse Table

- To recognize "int * int"

E → TE'
E' → +E | ε
T → intT' | (E)
T' → *T | ε

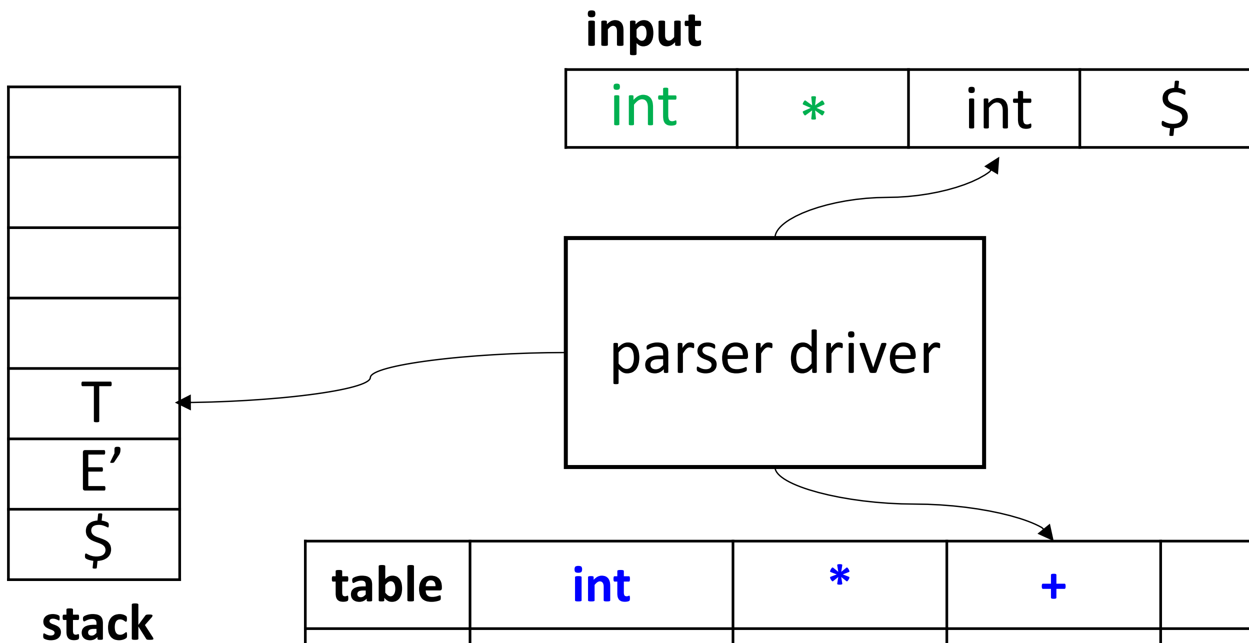


table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

Use the Parse Table

- To recognize "int * int"

E → TE'
E' → +E | ε
T → intT' | (E)
T' → *T | ε

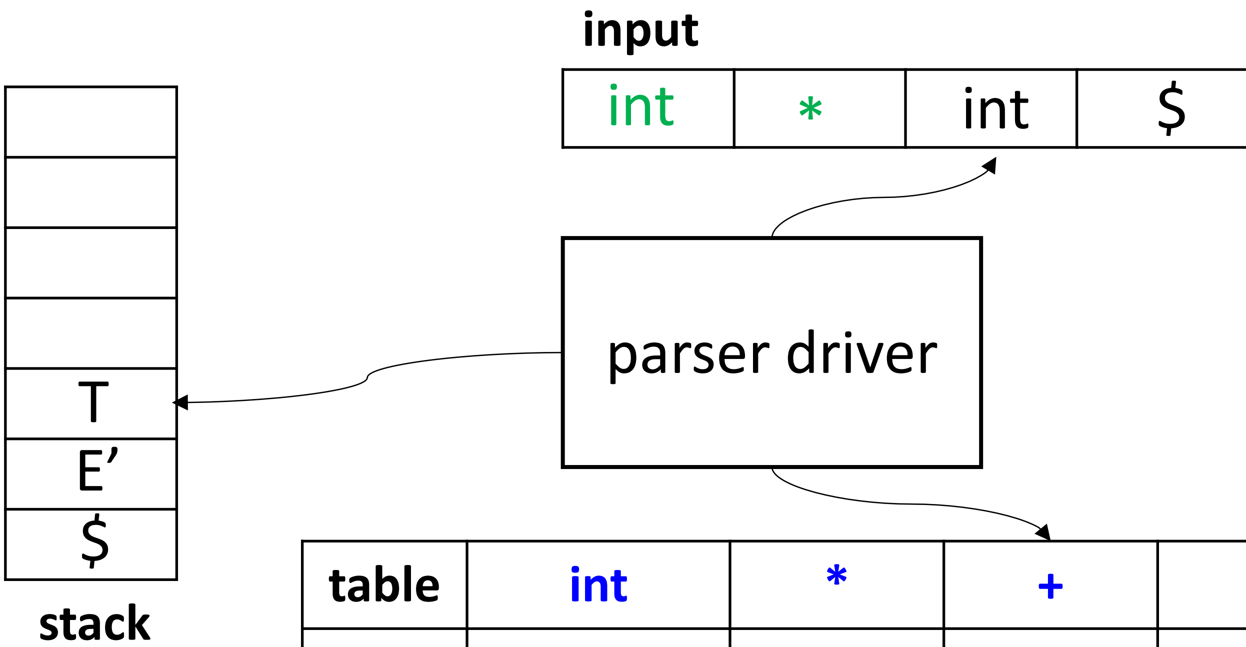


table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

Use the Parse Table

- To recognize “int * int”

E → TE’

E’→ +E | ε

T → intT’ | (E)

T’→ *T | ε

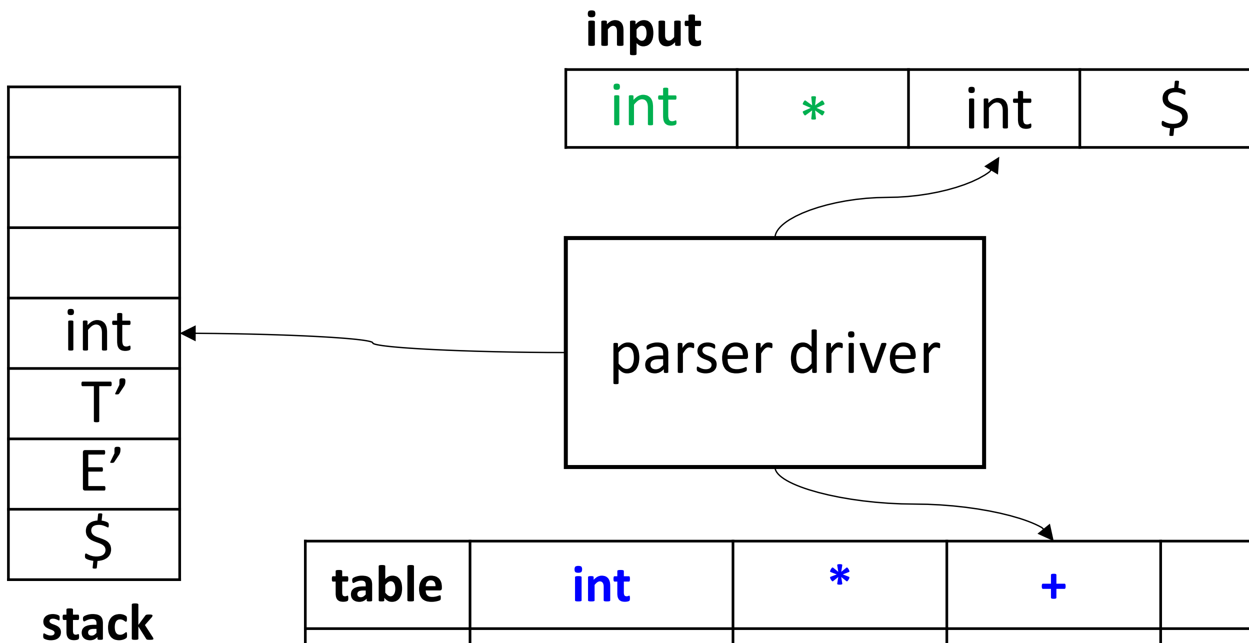


table	int	*	+	(	)	$
E	E → TE’			E → TE’		
E’			E’ → +E		E’ → ε	E’ → ε
T	T → int T’			T → (E)		
T’		T’ → *T	T’ → ε		T’ → ε	T’ → ε

Use the Parse Table

- To recognize "int * int"

E → TE'
E' → +E | ε
T → intT' | (E)
T' → *T | ε

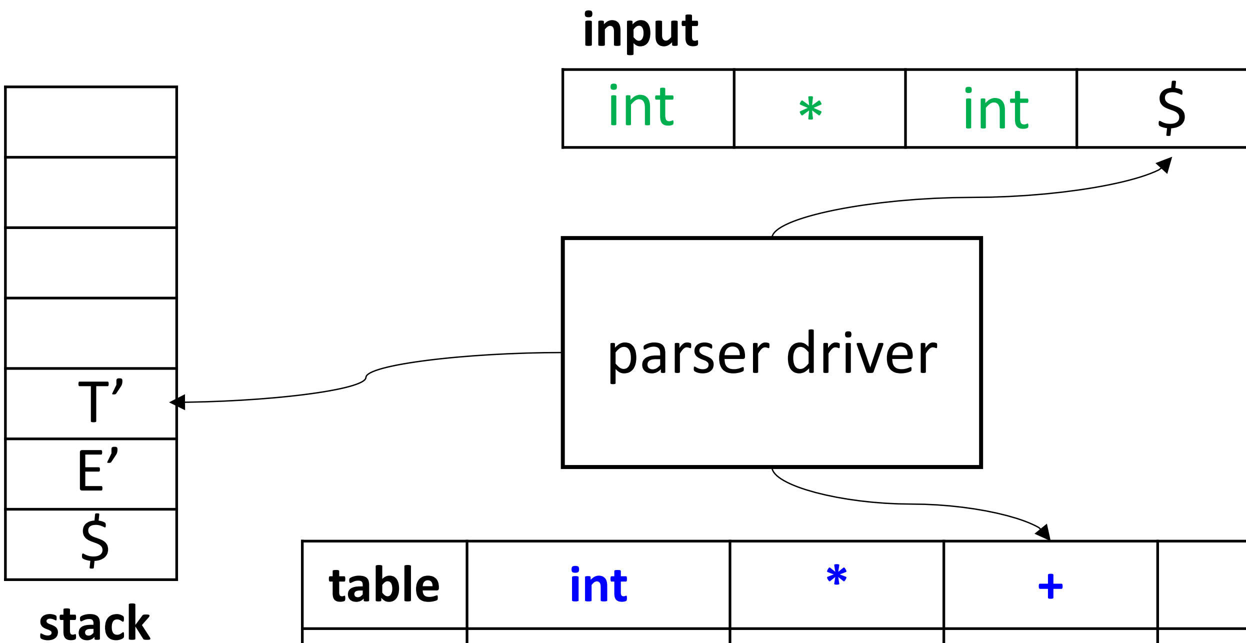
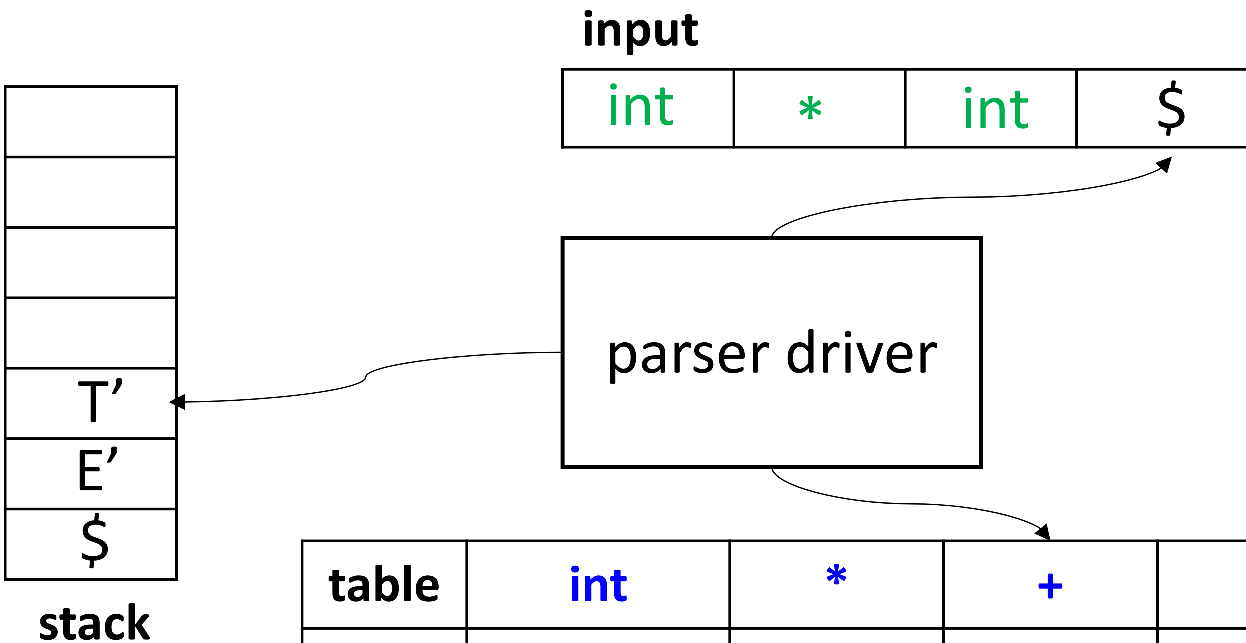


table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

Use the Parse Table

- To recognize "int * int"



E → TE'
E' → +E | ε
T → intT' | (E)
T' → *T | ε

table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

Use the Parse Table

- To recognize “int * int”

E → TE’
E’→ +E | ε
T → intT’ | (E)
T’→ *T | ε

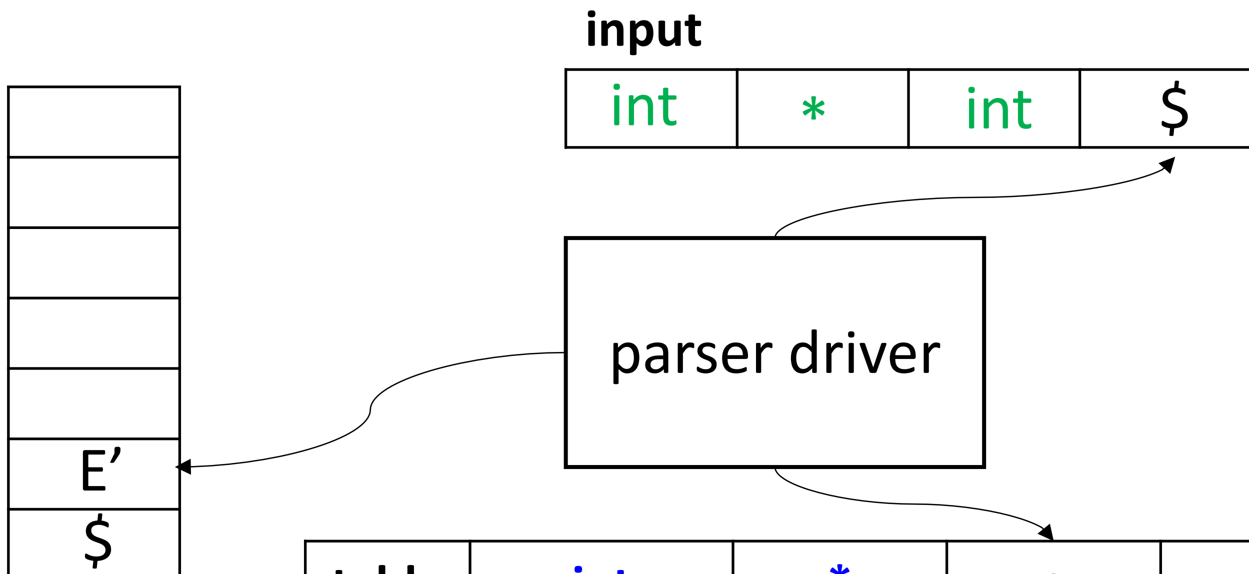


table	int	*	+	(	)	$
E	E → TE’			E → TE’		
E’			E’ → +E		E’ → ε	E’ → ε
T	T → int T’			T → (E)		
T’		T’ → *T	T’ → ε		T’ → ε	T’ → ε

Use the Parse Table

- To recognize "int * int"

E → TE'
E'→ +E | ε
T → intT' | (E)
T'→ *T | ε

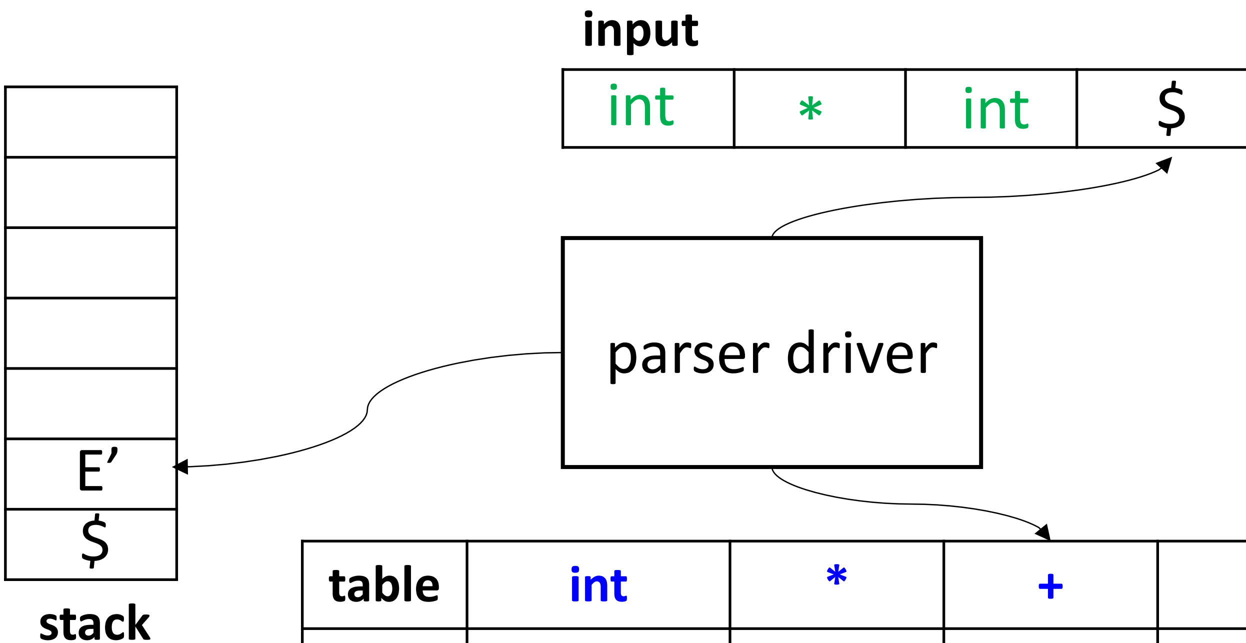


table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

Use the Parse Table

- To recognize "int * int"

E → TE'
E' → +E | ε
T → intT' | (E)
T' → *T | ε

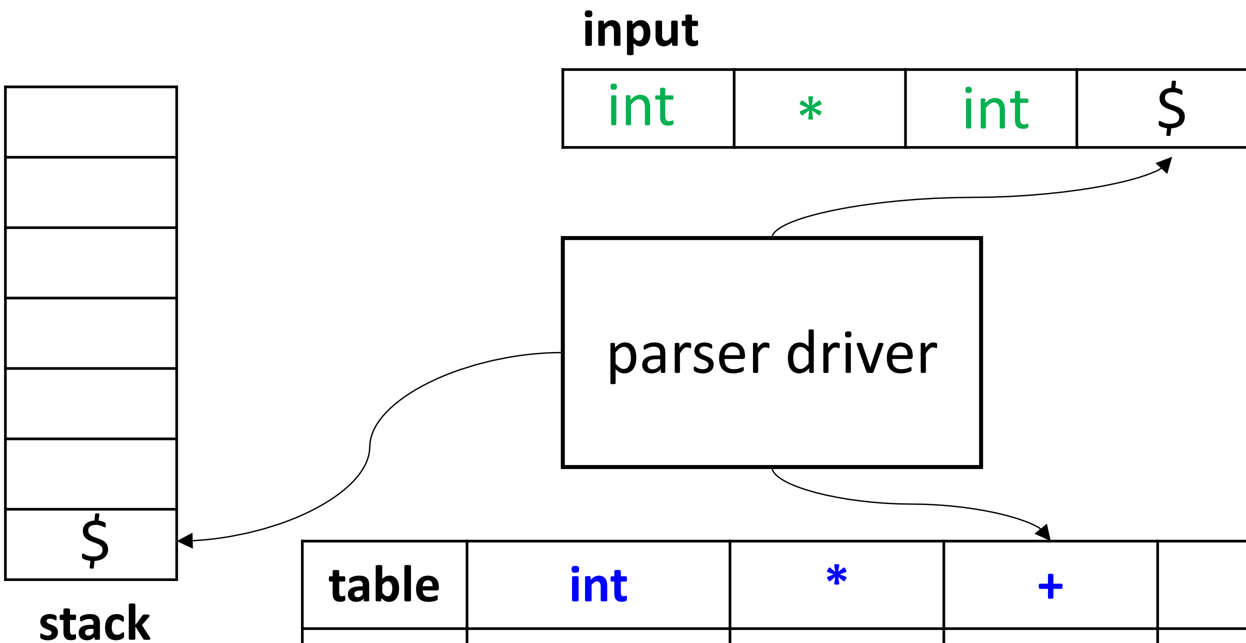
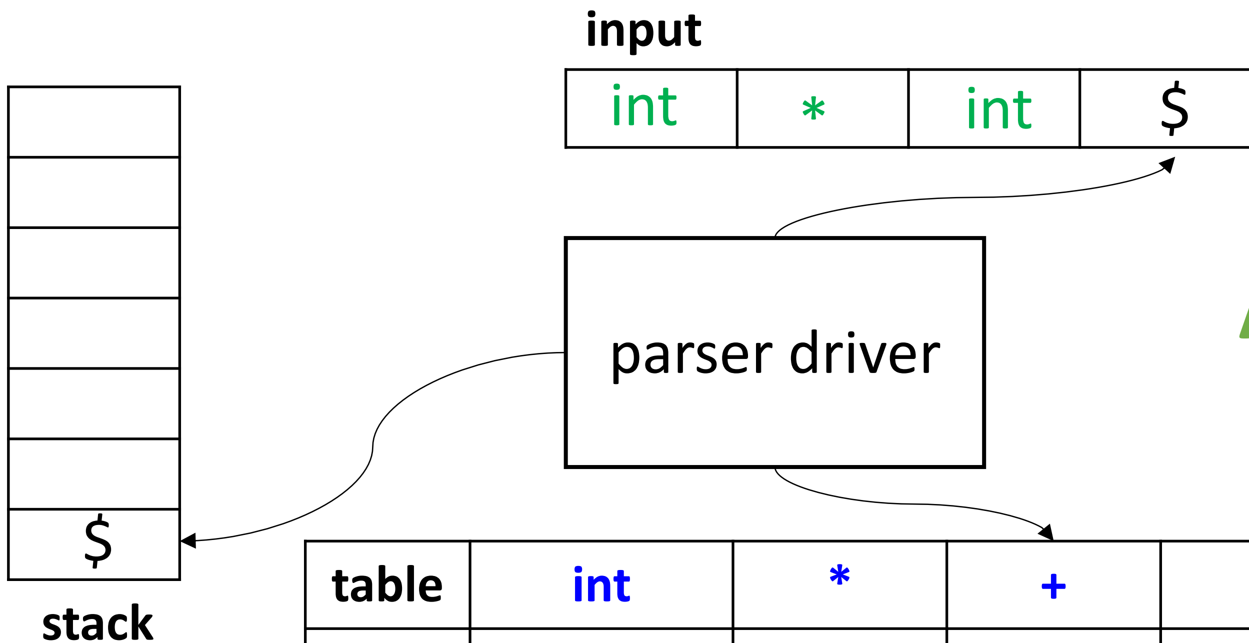


table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

Use the Parse Table

- To recognize "int * int"

E → TE'
E'→ +E | ε
T → intT' | (E)
T'→ *T | ε



ACCEPT!

table	int	*	+	(	)	$
E	E → TE'			E → TE'		
E'			E' → +E		E' → ε	E' → ε
T	T → int T'			T → (E)		
T'		T' → *T	T' → ε		T' → ε	T' → ε

Recognize Sequence[解析过程]

Matched	Stack (unmatched)	Input	Action
	E $	int * int $	E → TE’
	T E’ $	int * int $	T → int T’
int	int T’ E’ $	int * int $	match
int	T’ E’ $	* int $	T’ → *T
int	* T E’ $	* int $	match
int *	T E’ $	int $	T → int T’
int *	int T’ E’ $	int $	match
int * int	T’ E’ $	$	T’ → ε
int * int	E’ $	$	E’ → ε
int * int	$	$	Halt-accept

E → TE’
E’→ +E | ε
T → intT’ | (E)
T’→ *T | ε

Input: int * int

- ‘Matched + Stack’ constructs the sentential form[句型]
- Actions correspond to productions in leftmost derivation