Compilers Scribe Report

CLASS DATE: 7/10/13

Abhishek Pant (11CS30001) | CS31003 | October 7, 2013

Topics covered: SLR Parser

- Viable Prefixes
- Valid Items
- Limitations of SLR Parser

Viable Prefixes:

Consider the CFG:

E -> E + T | T

T -> T * F | F

F -> id

Let w = id * id (input string)

Here is the trace of the SLR parsing algorithm:

STACK	INPUT	ACTION
$	id * id	shift
$ id	* id	reduce
$ F	* id	reduce
$ T	* id	shift
$T *	id	shift
$T * id	$	reduce
$T * F	$	reduce

$ T	$	reduce
$ E	$	ACCEPT

We observe that at any point of time, the stack contents must be a prefix of a right sentential form.

However, not all prefixes of a right sentential form can appear on the stack.

For example, consider the rightmost derivation:

E -> T -> T * F -> T * id -> F * id -> **id *** id

Here, 'id *' is a prefix of a right sentential form. But it can never appear on the stack! This is because we will always reduce by F -> id before shifting '*'

Definition (viable prefix): The prefixes of right sentential forms that can appear on the stack of a shift-reduce parser are called viable prefixes.

By definition, a viable prefix is a prefix of a right sentential form that does not continue past the right end of the rightmost handle of that sentential form.

Example:

Let: S -> X_1 X_2 X_3 X_4

A -> X_1 X_2

Let w = X_1 X_2 X_3

SLR parse trace:

STACK	INPUT
$	X_1 X_2 X_3
$ X_1	X_2 X_3

$ X_1 X_2 X_3

$ A X_3

$ A X_3 $

.

.

.

As we see, X_1 X_2 X_3 will never appear on the stack. So, it is not a viable prefix.

Importance of Viable Prefixes:

The entire SLR parsing algorithm is based on the idea that the LR(0) automaton can recognize viable prefixes and reduce them appropriately.

Equivalently, this means that the set of viable prefixes for a given SLR (1) grammar is a regular language!

Valid Items:

Consider the item: A -> β_1 . β_2

Let S → α A β -> α β_1 β_2 B (rightmost derivation, sentential form)

Since the dot is between β_1 and β_2, α β_1 will be on top of the stack.

So, α β_1 is a viable prefix.

We say that A -> β_1 . β_2 is a ***valid item*** for the viable prefix α β_1.

Every viable prefix is associated with a valid item. In general, an item will be valid for many viable prefixes.

Viable prefixes and valid items enable us to make shift reduce decisions, I e if we know the valid items for a viable prefix, we can create our own rules about when to shift and reduce.

Example:

Let A -> β_1 . β_2 be a valid item for the viable prefix $\alpha\ \beta_1$.

Let w = $\alpha\ \beta_1\ \beta_2$ x

Let the current state be {A -> β_1 . β_2 , A -> β_1 . }

Consider the configuration:

STACK	INPUT
$ $\alpha\ \beta_1$	β_2 x $

Q. What action should we take, given the above information?

A. Case 1: β_2 != ϵ

Action: shift

Since the dot is in the middle, we know that we must shift.

Case 2: $\beta_2 = \epsilon$

Action: reduce A -> β_1

When the dot reaches the end, we know that we should try to reduce.

Finding Valid Items for a Viable Prefix:

If there exists a string (prefix) ɣ such that the path traversed by ɣ in the LR(0) automaton is from state S to state 'i', then the items in state 'i' are the set of valid items for the viable prefix ɣ.

Limitations of SLR Parser:

For an SLR parser, if we are in a state with the item A -> α ., then we reduce by A -> α iff the next input symbol ‘a’ belongs to Follow(A).

This is actually a weak rule and can lead to erroneous results and/or shift/reduce conflicts.

Example:

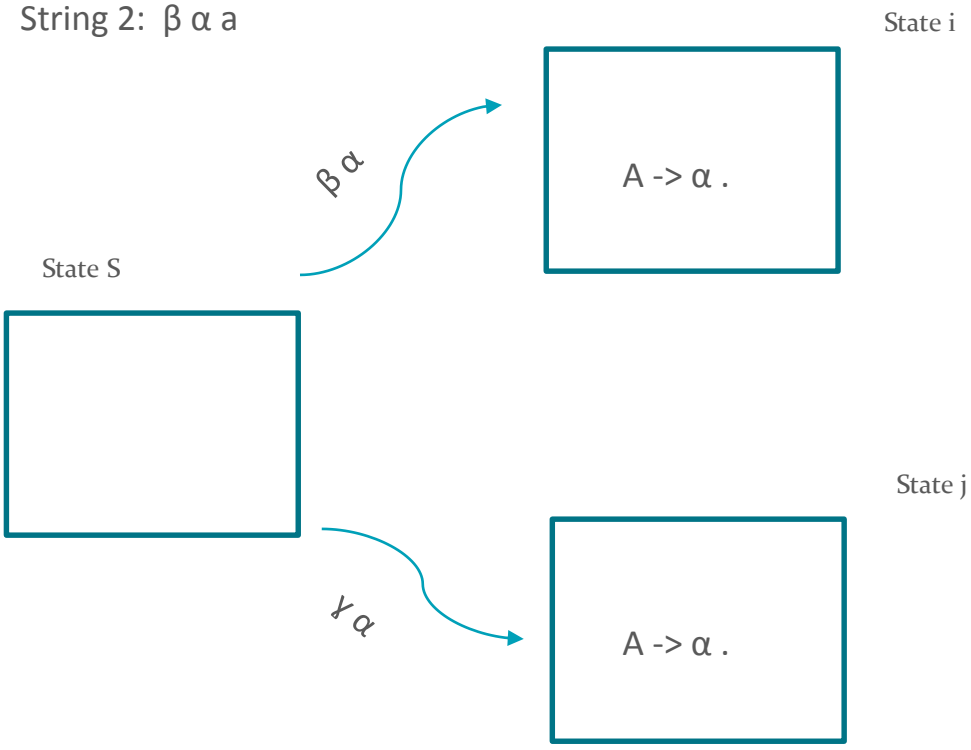
Consider A -> α

State i: A -> α .

State j: A -> α .

String 1: ɣ α a

String 2: β α a



Now suppose

S => ɣ A a -> ɣ α a

S =/=> β A a -> β α a

‘a’ belongs to Follow(A)

In both cases, since ‘a’ belongs to Follow(A), the parser will reduce by A -> α

However, this reduction is incorrect in the case of string 2, since we cannot reach the start state if we make this reduction, I e we are proceeding in the wrong direction!

This is a major limitation of the SLR parser.