Generating Shift-Reduce (LR(0)) Parsing Tables

or "A Fresh Start" for parsing...

#	Rules
1	$S \rightarrow A\,B\,\$$
2	$S \rightarrow B\,C\,\$$
3	$A \rightarrow A\,x$
4	$A \rightarrow x$
5	$B \rightarrow y\,A\,B$
6	$B \rightarrow h$
7	$C \rightarrow x\,C\,y$
8	$C \rightarrow p$

We are going to construct a graph from the language grammar, we begin with its **starting goal** and create it's **fresh starts**:

$$S \rightarrow \bullet\, A\,B\,\$ \qquad S \rightarrow \bullet\, B\,C\,\$$$

A production rule with a • "bookmark" or **progess marker** is called an ITEM.

An ITEM with the • at the beginning of the RHS is called a FRESH START.

We use the • to track where we are in the completion of a **production rule**, the graph we are generating will track where we are in the **nesting** of production rules.

Generating Shift-Reduce (LR(0)) Parsing Tables

#	Rules
1	$S \rightarrow A\,B\,\$$
2	$S \rightarrow B\,C\,\$$
3	$A \rightarrow A\,x$
4	$A \rightarrow x$
5	$B \rightarrow y\,A\,B$
6	$B \rightarrow h$
7	$C \rightarrow x\,C\,y$
8	$C \rightarrow p$

Now we execute *Closure* on the starting goal's fresh starts:

1. First, we think of maintaining ITEM SETS, so we actually have

   Item Set 0: $\{\ S \rightarrow \bullet\, A\,B\,\$,\ S \rightarrow \bullet\, B\,C\,\$\ \}$

2. Now, for every non-terminal to the right of $\bullet$, add a fresh start **for all of that non-terminal's production rules**. In this case, we need to add fresh starts for A and B.

Generating Shift-Reduce (LR(0)) Parsing Tables

#	Rules
1	$S \rightarrow A\ B\ \$$
2	$S \rightarrow B\ C\ \$$
3	$A \rightarrow A\ x$
4	$A \rightarrow x$
5	$B \rightarrow y\ A\ B$
6	$B \rightarrow h$
7	$C \rightarrow x\ C\ y$
8	$C \rightarrow p$

And it is right about now that I'll represent **items sets** as tables in these slides — the name still holds! These are still proper mathematical **sets**!

ITEM SET 0					
S	$\rightarrow$	$\bullet$	A	B	$\$$
S	$\rightarrow$	$\bullet$	B	C	$\$$
A	$\rightarrow$	$\bullet$	A	x	
A	$\rightarrow$	$\bullet$	x		
B	$\rightarrow$	$\bullet$	y	A	B
B	$\rightarrow$	$\bullet$	h		

In general, this is called the *Closure* procedure, and we do it until our item set has no new items added.

In this case, we had only "one round" of closure. . .

Generating Shift-Reduce (LR(0)) Parsing Tables

ITEM SET 0				
S	$\rightarrow$	$\bullet$ A	B	\$
S	$\rightarrow$	$\bullet$ B	C	\$
A	$\rightarrow$	$\bullet$ A	x	
A	$\rightarrow$	$\bullet$ x		
B	$\rightarrow$	$\bullet$ y	A	B
B	$\rightarrow$	$\bullet$ h		

The next step is to execute the *GoTo* procedure **on this item set** for each **grammar symbol in the language**, $\mathcal{X} \in N \bigcup \Sigma_{\$}$:

In this example, specifically for non-terminal A:

i. Copy all the items with A to the right of $\bullet$ to a **new item set**.
(In this case we'll call it ITEM SET 4)

ii. In the new item set, move the $\bullet$ forward one grammar symbol past A

ITEM SET 4				
S	$\rightarrow$	A $\bullet$	B	\$
A	$\rightarrow$	A $\bullet$	x	

Incidentally, these items are called the new item set's **KERNEL**.

Generating Shift-Reduce (LR(0)) Parsing Tables

The next step is to execute the *GoTo* procedure **on this item set** for each **grammar symbol in the language**, $\mathcal{X} \in N \bigcup \Sigma_{\$}$:

In this example, specifically for non-terminal A:

i. Copy all the items with A to the right of • to a **new item set**.

ii. In the new item set, move the • forward one grammar symbol past A

iii. Execute *Closure* on this new item set.

ITEM SET 0

$S \rightarrow \bullet\ A\ B\ \$$
$S \rightarrow \bullet\ B\ C\ \$$
$A \rightarrow \bullet\ A\ x$
$A \rightarrow \bullet\ x$
$B \rightarrow \bullet\ y\ A\ B$
$B \rightarrow \bullet\ h$

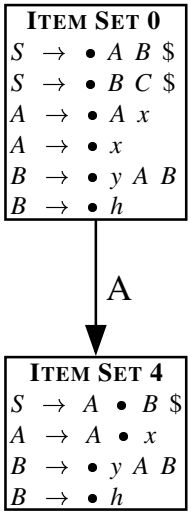
ITEM SET 4

$S \rightarrow A \bullet B\ \$$
$A \rightarrow A \bullet x$

ITEM SET 4

$S \rightarrow A \bullet B\ \$$
$A \rightarrow A \bullet x$
$B \rightarrow \bullet\ y\ A\ B$
$B \rightarrow \bullet\ h$

Generating Shift-Reduce (LR(0)) Parsing Tables



The next step is to execute the *GoTo* procedure **on this item set** for each **grammar symbol in the language**, $\mathcal{X} \in N \bigcup \Sigma_{\$}$:

In this example, specifically for non-terminal A:

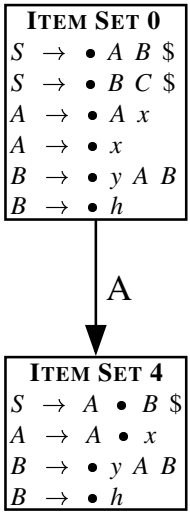
i. Copy all the items with A to the right of • to a **new item set**.

ii. In the new item set, move the • forward one grammar symbol past A

iii. Execute *Closure* on this new item set.

iv. In a graph, connect Item Set 0 to this new item set with an edge labeled with A.

Generating Shift-Reduce (LR(0)) Parsing Tables



We continue in this fashion for each grammar symbol. Some grammar symbols (C) won't generate any kernels because they aren't to the right of • in our **Item Set 0**.
The process is the same for all symbols:

For a symbol $\mathcal{X}$:

i. Copy all the items with $\mathcal{X}$ to the right of • to a **new item set**.

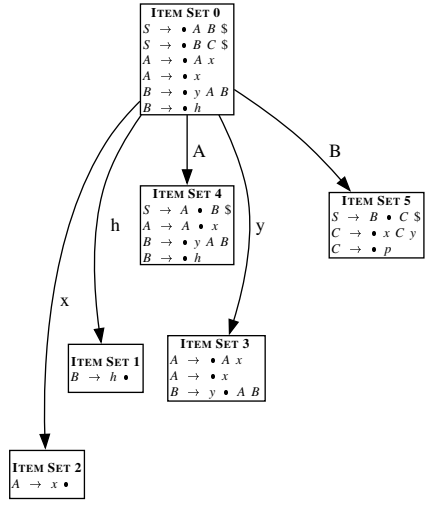
ii. In the new item set, move the • forward one grammar symbol.

iii. Execute *Closure* on this new item set.

iv. In a graph, connect Item Set 0 to this new item set with an edge labeled with $\mathcal{X}$.

Item Sets Generated by **Item Set 0**

When we have generated all the new items sets from “progressing” **Item Set 0**, we have a graph like this. . .

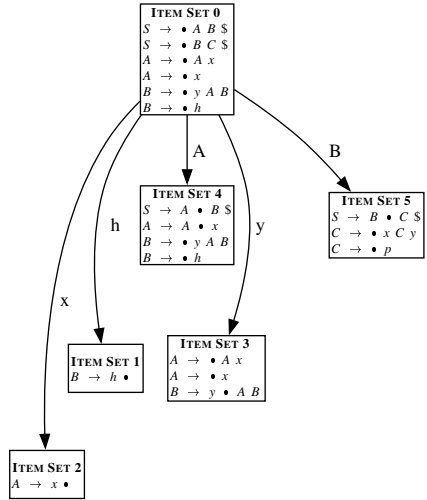


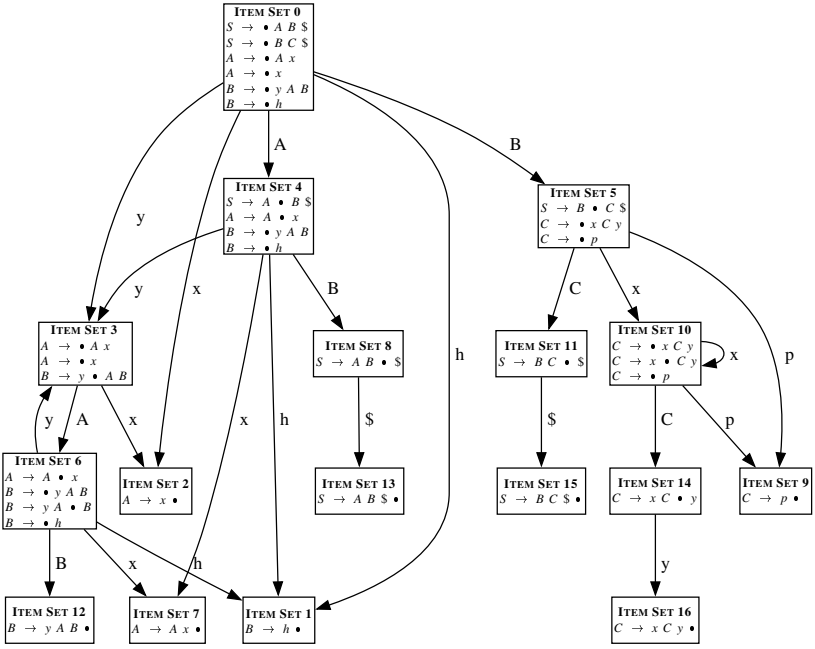
Item Sets Generated by **Item Set 0**

When we have generated all the new items sets from “progressing” **Item Set 0**, we have a graph like this. . .

Now (and you probably saw this coming) — for each of these new items sets, repeat the same process, generating **more item sets and enlarging our graph**.

Do this until no new item sets are added to the graph.



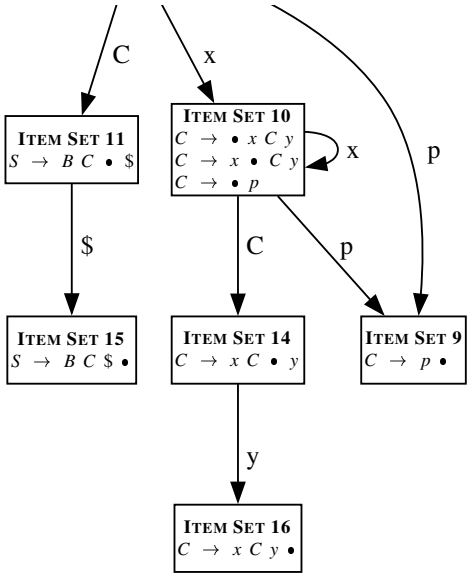


Beautimous!

The sort of thing only a CS person could love.

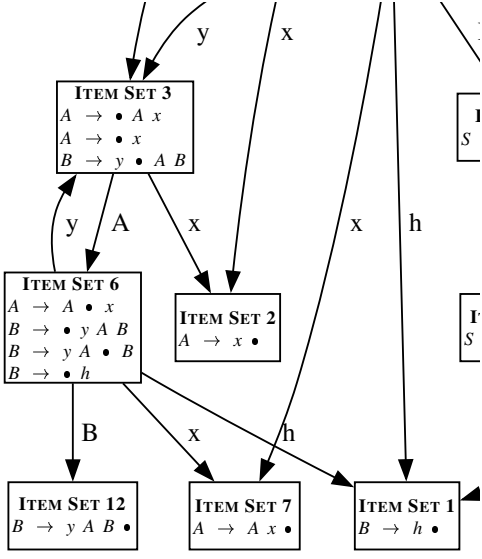
Now we just have to turn this graph (called a **characteristic finite state machine**, or CFSM) into a shift-reduce table. . .

CFSM Observations



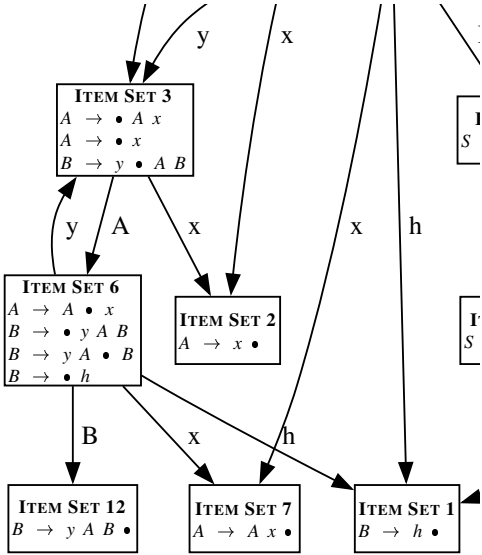
Notice that for **Item Set 10**, the graph has a self-incident edge for grammar symbol x — which item generated this edge?

CFSM Observations



Item Set 3 & 6 creates a slightly larger cycle on y — what is the **kernel** (or "generating items") for **Item Set 3**?

CFSM Observations



The **kernel** for **Item Set 3** is

$$B \rightarrow y \bullet A\ B$$

How do we know?

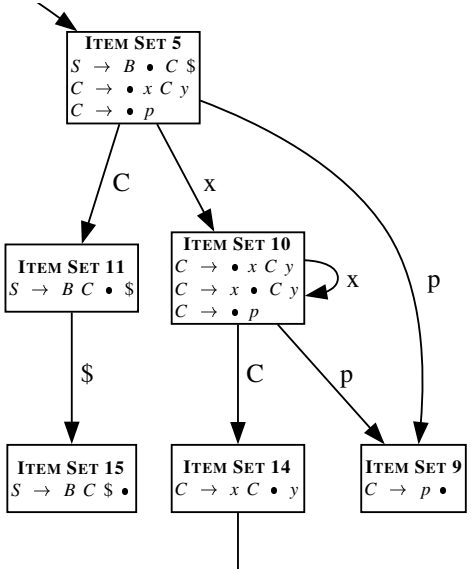
a. The two rules with A on the LHS are **fresh starts**, and aside from **Item Set 0**, fresh starts are not kernels of item sets — they are the result of *Closure* on the kernel.

b. Kernels are identified by **following a grammar symbol** from one item set to another — kernels have the incident grammar symbol **to the left** of their • marker.

(Again, except for Item Set 0 which has no incident edges.)

From CFSM to LR (shift-reduce) Parsing Table

Item Sets ≡ Parsing State ≡ LR Table Row

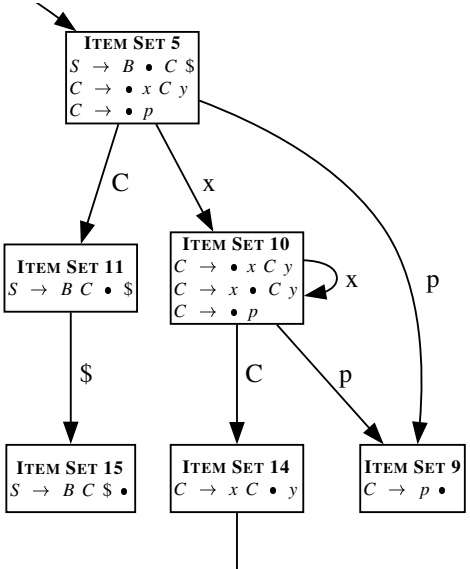


Let's look how **Item Set 10** informs the construction of LR table row 10:

- When we are in state 10 and see a C at the front of the deque
  shift the C and go to state 14: `sh-14`.
- When we are in state 10 and see a p at the front of the deque
  shift the p and go to state 9: `sh-9`.
- When we are in state 10 and see an x at the front of the deque
  shift the x and go to state 10: `sh-10`.
- We don't care about the incident x edge from **Item Set 5**, this is a `sh-10` action in table row 5!

From CFSM to LR (shift-reduce) Parsing Table

Item Sets ≡ Parsing State ≡ LR Table Row

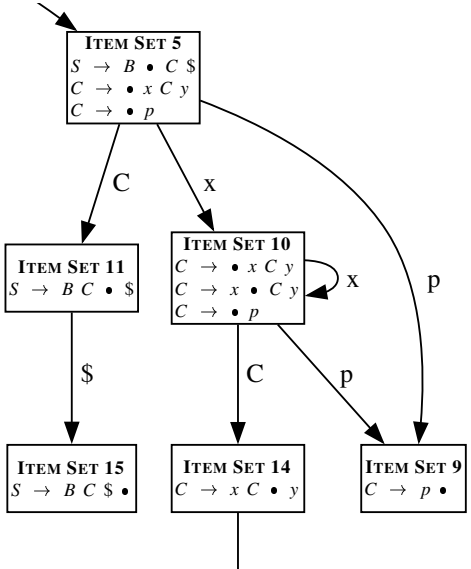


What type of action should occur for **Item Sets 9 & 15**?

#	Rules
1	$S \rightarrow A\ B\ \$$
2	$S \rightarrow B\ C\ \$$
3	$A \rightarrow A\ x$
4	$A \rightarrow x$
5	$B \rightarrow y\ A\ B$
6	$B \rightarrow h$
7	$C \rightarrow x\ C\ y$
8	$C \rightarrow p$

From CFSM to LR (shift-reduce) Parsing Table

Item Sets ≡ Parsing State ≡ LR Table Row



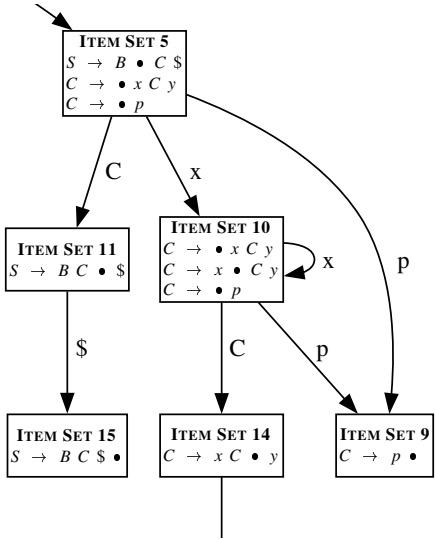
What type of action should occur for **Item Sets 9 & 15**?

#	Rules
1	$S \rightarrow A\,B\,\$$
2	$S \rightarrow B\,C\,\$$
3	$A \rightarrow A\,x$
4	$A \rightarrow x$
5	$B \rightarrow y\,A\,B$
6	$B \rightarrow h$
7	$C \rightarrow x\,C\,y$
8	$C \rightarrow p$

- **Item Set 9** says we should `Reduce-8`
- **Item Set 15** says we should `Reduce-2`

From CFSM to LR (shift-reduce) Parsing Table

Item Sets ≡ Parsing State ≡ LR Table Row



	h	*p*	*x*	*y*	$	*A*	*B*	*C*
0	sh-1		sh-2	sh-3		sh-4	sh-5	
1	Reduce 6							
2	Reduce 4							
3			sh-2			sh-6		
4	sh-1		sh-7	sh-3			sh-8	
5		sh-9	sh-10					sh-11
6	sh-1		sh-7	sh-3			sh-12	
7	Reduce 3							
8					sh-13			
9	Reduce 8							
10		sh-9	sh-10					sh-14
11					sh-15			
12	Reduce 5							
13	Reduce 1							
14				sh-16				
15	Reduce 2							
16	Reduce 7							

Conflicts in LR(0) Item Sets

All this seems pretty nice: we form a **graph of items sets (the CFSM)**

- the graph gives us only one decision to make during parsing (which item set $\equiv$ **parsing state**) to go to when we consume a grammar symbol from the input deque;
- and item sets help us track progressions through multiple production rules (iow: multiple possible parse tree results) when a grammar symbol is consumed.

Unfortunately, these **LR(0)** tables are prone to breakage for even the most simplist of languages — these are called table **conflicts**. (You may have noticed the example grammar was lacking any λ production rules . . .)

Conflicts in LR(0) Item Sets

All this seems pretty nice: we form a **graph of items sets (the CFSM)**

- the graph gives us only one decision to make during parsing (which item set $\equiv$ **parsing state**) to go to when we consume a grammar symbol from the input deque;
- and item sets help us track progressions through multiple production rules (iow: multiple possible parse tree results) when a grammar symbol is consumed.

Unfortunately, these **LR(0)** tables are prone to breakage for even the most simplist of languages — these are called table **conflicts**. (You may have noticed the example grammar was lacking any λ production rules . . .)

Before we get to ***conflict resolution*** though, we should ask what does the 0 in LR(0) mean?

Conflicts in LR(0) Item Sets

All this seems pretty nice: we form a **graph of items sets (the CFSM)**

- the graph gives us only one decision to make during parsing (which item set $\equiv$ **parsing state**) to go to when we consume a grammar symbol from the input deque;
- and item sets help us track progressions through multiple production rules (iow: multiple possible parse tree results) when a grammar symbol is consumed.

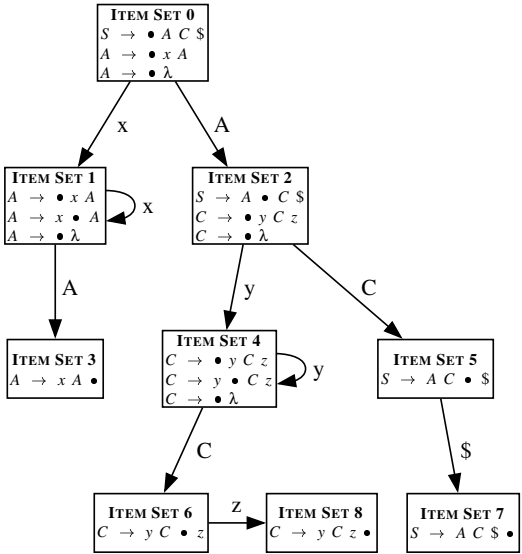
Unfortunately, these **LR(0)** tables are prone to breakage for even the most simplist of languages — these are called table **conflicts**. (You may have noticed the example grammar was lacking any λ production rules . . .)

Before we get to ***conflict resolution*** though, we should ask what does the 0 in LR(0) mean?

We haven't used $First$ or $Follow$ sets in our item set and CFSM construction, unlike predict set calculations that form LL(1) parsing tables.

If we had, we would have been building an LR(1) shift-reduce table and we would be consuming a lot more memory while we did it.

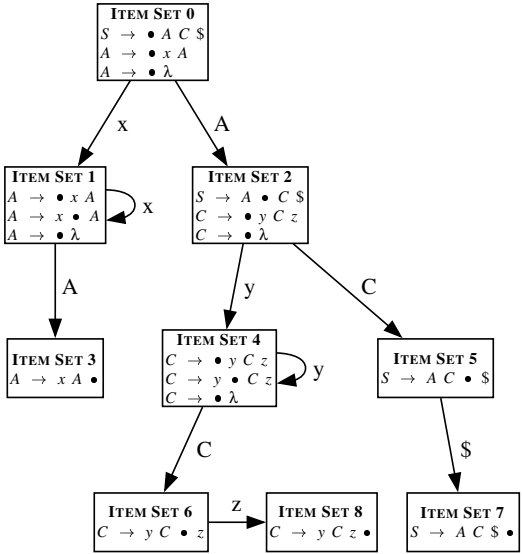
λ-Rules in LR(0) Graphs



	x	y	z	$	A	C
0	⋆					
1	⋆					
2	⋆					
3	Reduce 2					
4	⋆					
5				sh-7		
6			sh-8			
7	Reduce 1					
8	Reduce 4					

- Notice that there are no λ-edges in the CFSM
  (makes sense, since the edges are generated by $\mathcal{X} \in N \bigcup \Sigma_{\$}$)

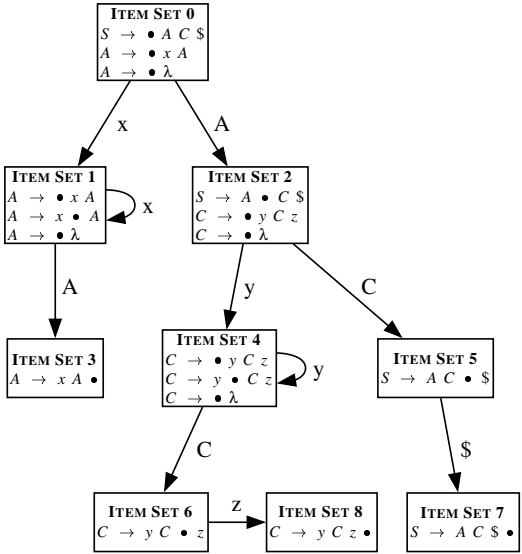
λ-Rules in LR(0) Graphs



	x	y	z	$	A	C
0	⋆					
1	⋆					
2	⋆					
3	Reduce 2					
4	⋆					
5				sh-7		
6			sh-8			
7	Reduce 1					
8	Reduce 4					

- If there are no λ-edges, how did we get λ-items in items sets?

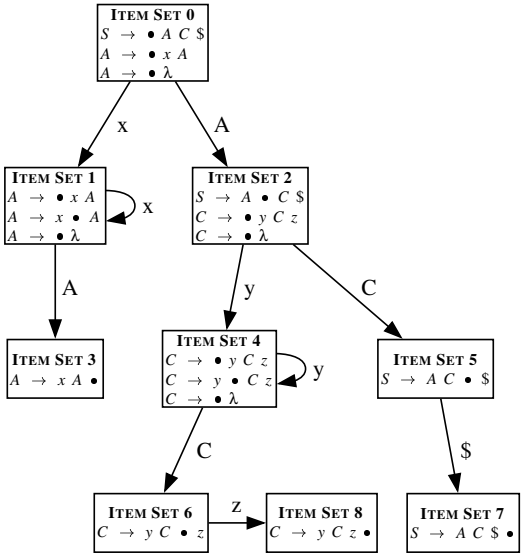
λ-Rules in LR(0) Graphs



	x	y	z	$	A	C
0	⋆					
1	⋆					
2	⋆					
3	Reduce 2					
4	⋆					
5				sh-7		
6			sh-8			
7	Reduce 1					
8	Reduce 4					

- If there are no λ-edges, how did we get λ-items in items sets?
  Fresh Starts!
- Which items generated the λ-item fresh starts in item sets 1 & 4?

λ-Rules in LR(0) Graphs



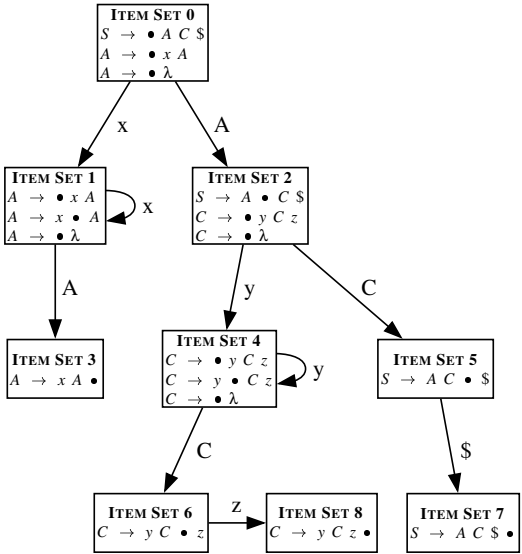
λ items are in the **completed** state without the need for transitions to other item sets.

Like any other completed item, they require a `Reduce` rule for the parsing state (≡ item set) in the LR parsing table...

- If a $X \rightarrow \lambda$ item is added to item sets due to the process of **closure**...
  ...there must be a shift transition edge for X to another item set as well.

But LR parsing won't let us do both reduce **and** shift at the same time — it is a **deterministic parsing algorithm!**

λ-Rules in LR(0) Graphs

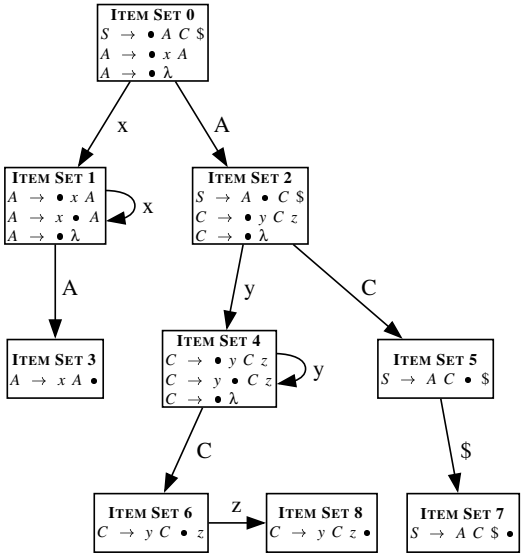


Grammars with λ-rules **always create conflicts** for LR(0) tables.

How do we work around this? By building an **SLR(1)** table from the CFSM instead of an LR(0) table.

SLR(1) tables install `Reduce` rules only for the columns of $\Sigma_{\$}$ in the $Follow(\mathcal{X})$ of a $\mathcal{X} \rightarrow \bullet\, \lambda$ or $\mathcal{X} \rightarrow \pi\, \beta\, \bullet$ completed item.

λ-Rules in LR(0) Graphs



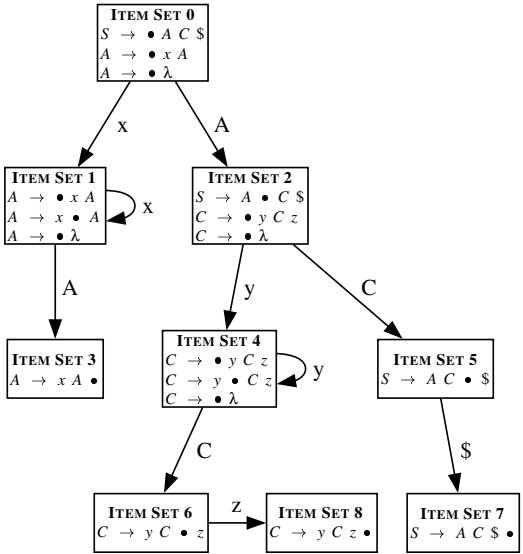
(Incomplete SLR(1) Table)

	x	*y*	*z*	$	*A*	*C*
0	sh-1	r-3		r-3	sh-2	
1	sh-1	r-3		r-3	sh-3	
2		sh-4				sh-5
3		r-2		r-2		
4		sh-4				sh-6
5				sh-7		
6			sh-8			
7	Reduce 1					
8			r-4	r-4		

$C \rightarrow \lambda$ is rule 5 in this grammar;

What **item sets** should have `r-5` actions in the SLR(1) table?

λ-Rules in LR(0) Graphs

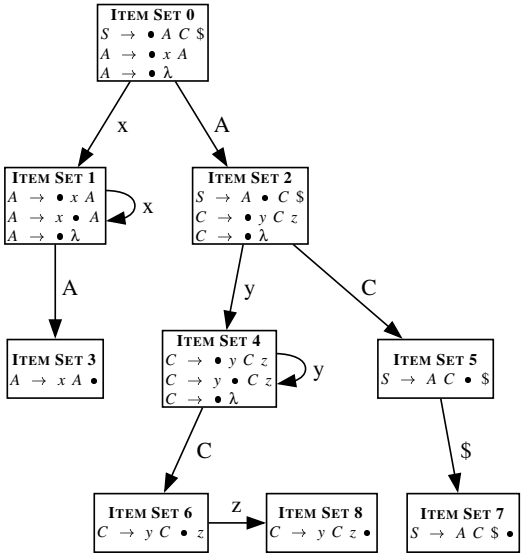


(Incomplete SLR(1) Table)

	x	*y*	*z*	$	*A*	*C*
0	sh-1	r-3		r-3	sh-2	
1	sh-1	r-3		r-3	sh-3	
2		sh-4				sh-5
3		r-2		r-2		
4		sh-4				sh-6
5				sh-7		
6			sh-8			
7	Reduce 1					
8			r-4	r-4		

What rule number is $A \rightarrow \lambda$?

λ-Rules in LR(0) Graphs



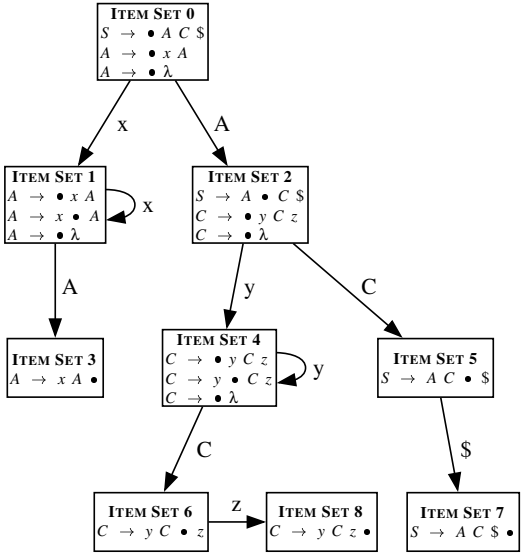
(Incomplete SLR(1) Table)

	x	*y*	*z*	$	*A*	*C*
0	sh-1	r-3		r-3	sh-2	
1	sh-1	r-3		r-3	sh-3	
2		sh-4				sh-5
3		r-2		r-2		
4		sh-4				sh-6
5				sh-7		
6			sh-8			
7	Reduce 1					
8			r-4	r-4		

What rule number is $A \rightarrow \lambda$?

$A \rightarrow \bullet \lambda$ occurs only in Item Sets 0 & 1, and it is the only completed item in these item sets. So all the `r-?` entries in SLR table rows 0 & 1 are for $A \rightarrow \lambda$.

λ-Rules in LR(0) Graphs

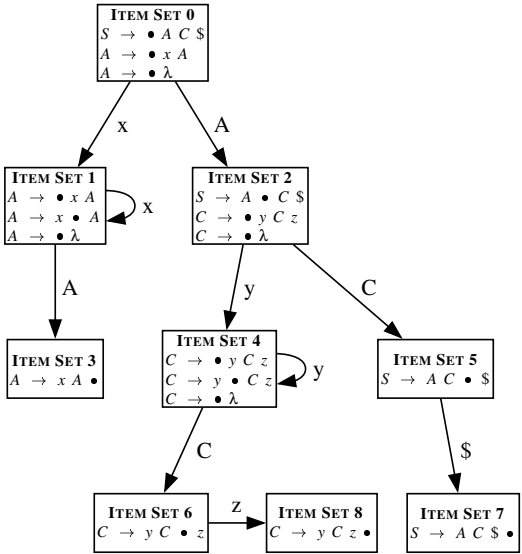


(Incomplete SLR(1) Table)

	x	*y*	*z*	$	*A*	*C*
0	sh-1	r-3		r-3	sh-2	
1	sh-1	r-3		r-3	sh-3	
2		sh-4				sh-5
3		r-2		r-2		
4		sh-4				sh-6
5				sh-7		
6			sh-8			
7	Reduce 1					
8			r-4	r-4		

Can you tell what $followSet(A)$ is **without calculating any** $firstSet(\cdot)$**s**?

λ-Rules in LR(0) Graphs



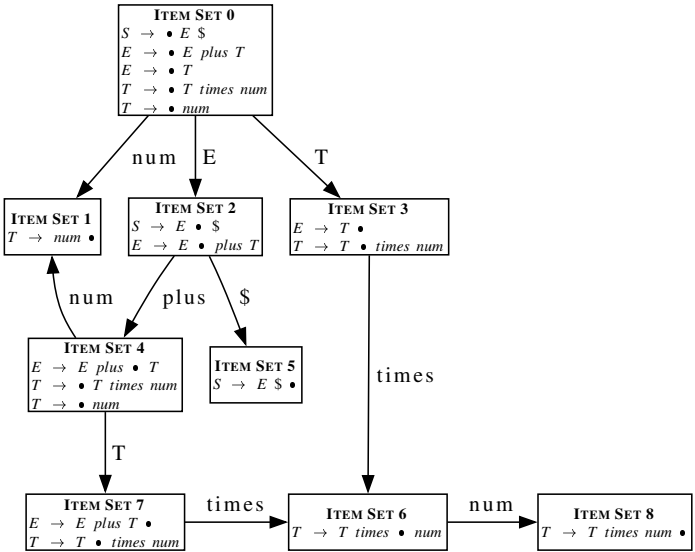
(Incomplete SLR(1) Table)

	x	y	z	$\$$	A	C
0	sh-1	r-3		r-3	sh-2	
1	sh-1	r-3		r-3	sh-3	
2		sh-4				sh-5
3		r-2		r-2		
4		sh-4				sh-6
5				sh-7		
6			sh-8			
7	Reduce 1					
8			r-4	r-4		

Can you tell what $followSet(A)$ is **without calculating any** $firstSet(\cdot)$**s?**

The `r-3` entries in row 0 & 1 occur only for terminals in the follow set of A. So

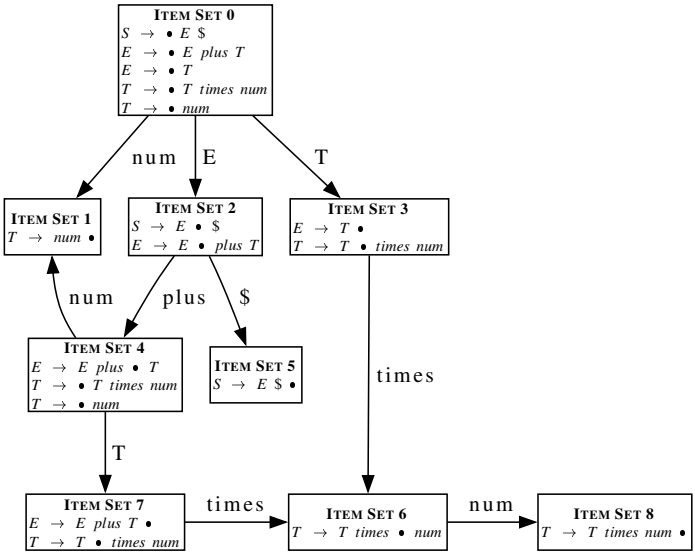
$$followSet(A) = \{y, \$\}$$



Shift-Reduce Conflicts without λ-Rules

	num	*plus*	*times*	$	*E*	*T*
0	sh-1				sh-2	sh-3
1	Reduce 5					
2		sh-4		sh-5		
3	⋆					
4	sh-1					sh-7
5	Reduce 1					
6	sh-8					
7	⋆					
8	Reduce 4					

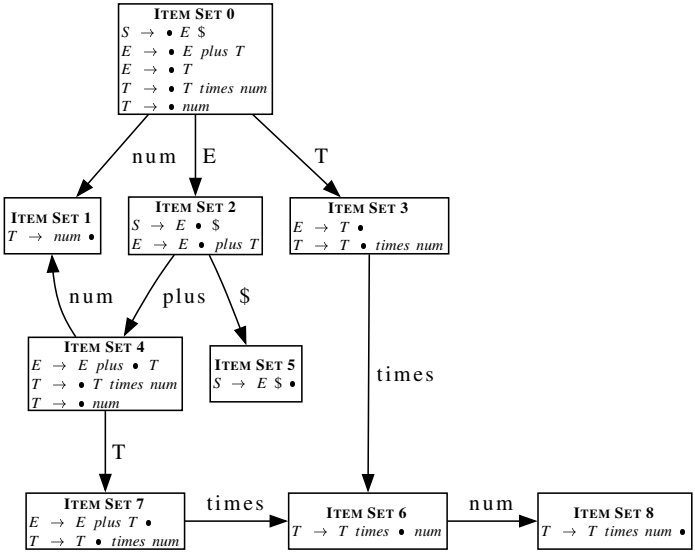
Items sets 3 & 7 say we should both **shift** past a grammar symbol and **reduce by a production rule** (because an item's • has no more grammar symbols to its right).



Shift-Reduce Conflicts without λ-Rules

	num	*plus*	*times*	$	*E*	*T*
0	sh-1				sh-2	sh-3
1	Reduce 5					
2		sh-4		sh-5		
3	⋆					
4	sh-1					sh-7
5	Reduce 1					
6	sh-8					
7	⋆					
8	Reduce 4					

The **SLR** table construction method resolves this by inserting reduction actions into the shift-reduce table for **only the appropriate** *Follow* **symbols**...

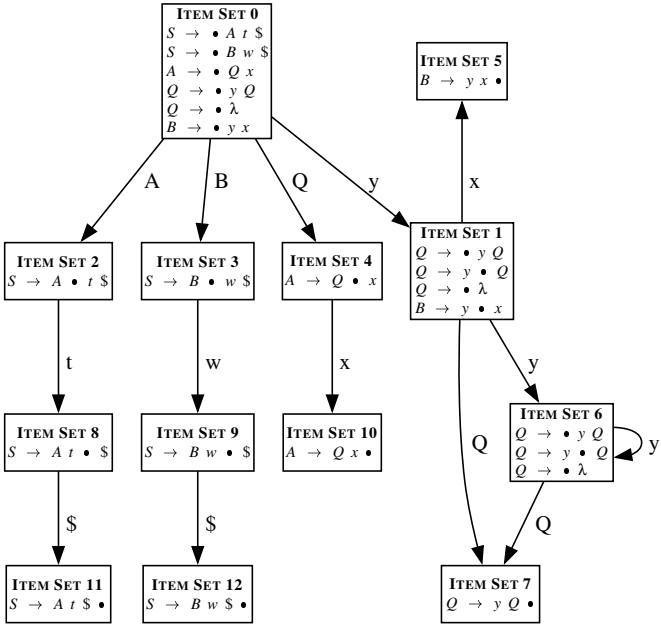


Shift-Reduce Conflicts without λ-Rules

	num	*plus*	*times*	$	*E*	*T*
0	sh-1				sh-2	sh-3
1	Reduce 5					
2		sh-4		sh-5		
3	⋆					
4	sh-1					sh-7
5	Reduce 1					
6	sh-8					
7	⋆					
8	Reduce 4					

For item sets 3 and 7,
$Follow(E) = \{\$, plus\}$

Which is fortuitously **disjoint** from the set of grammar symbols to the right of • in each item set: $\{\, times \,\}$.

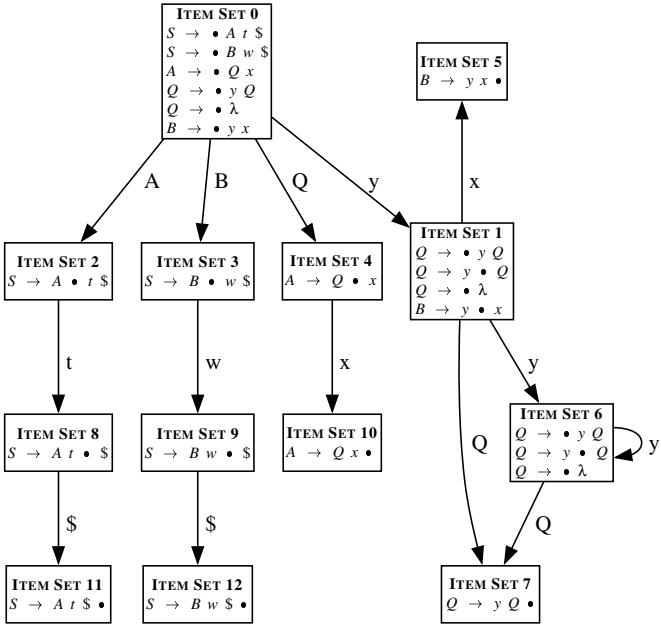


(More) Shift-Reduce Conflicts in SLR(1) Table Generation

	t	*w*	*x*	*y*	$	*A*	*B*	*Q*
0			r-5	sh-1		sh-2	sh-3	sh-4
1			⋆	sh-6				sh-7
2	sh-8							
3		sh-9						
4			sh-10					
5		r-6						
6			r-5	sh-6				sh-7
7			r-4					
8					sh-11			
9					sh-12			
10	r-3							
11	Reduce 1							
12	Reduce 2							

A **shift-reduce** conflict in Item Set 1, why?

(hint: the entire grammar happens to be shown in Item Set 0)



(More) Shift-Reduce Conflicts in SLR(1) Table Generation

	t	*w*	*x*	*y*	$	*A*	*B*	*Q*
0			r-5	sh-1		sh-2	sh-3	sh-4
1			⋆	sh-6				sh-7
2	sh-8							
3		sh-9						
4			sh-10					
5		r-6						
6			r-5	sh-6				sh-7
7			r-4					
8					sh-11			
9					sh-12			
10	r-3							
11	Reduce 1							
12	Reduce 2							

$Follow(Q) = \{ x \} \ldots$

Generating an SLR(1) table doesn't help since the $Follow(Q)$ shares terminals with the set of **shifted grammar symbols**.

Rearranging Grammars to Avoid LR Conflicts

Sometimes the reasons for conflicts can be reasoned out within the grammar, or a grammar subset.

#	Rules
1	$S \rightarrow A\ t\ \$$
2	$S \rightarrow B\ w\ \$$
3	$A \rightarrow Q\ x$
4	$Q \rightarrow y\ Q$
5	$Q \rightarrow \lambda$
6	$B \rightarrow y\ x$

In the most recent example (grammar to the left), the input $y\ x$ can begin a sentential form for the goal rules $S \rightarrow A\ t\ \$$ or $S \rightarrow B\ w\ \$$. The x terminal in item set 1 wants to "trigger" both a $Q \rightarrow \lambda$ reduction as well as complete the item for $B \rightarrow y \bullet x$.

We could fix this conflict by using a look a head of $k = 2$ for the follow sets. The $Q \rightarrow \bullet\ \lambda$ would be triggered only on sequences

$$y\ y \quad y\ x \quad \text{or} \quad x\ t$$

and $B \rightarrow y \bullet x$ would only be completed on token sequence $x\ w$.

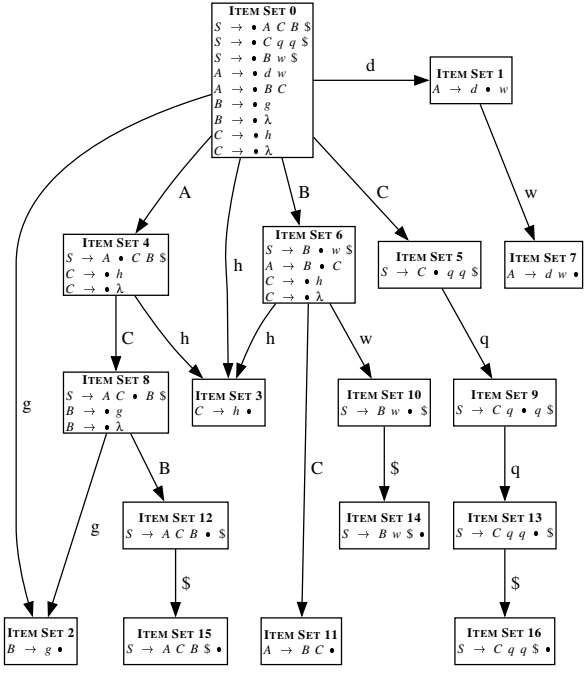
Rearranging Grammars to Avoid LR Conflicts

Sometimes the reasons for conflicts can be reasoned out within the grammar, or a grammar subset.

But increasing k is a brute force solution and for large languages can greatly expand the SLR(1) table size in memory. A more elegant approach requires just a little bit of language analysis to realize we could easily rewrite this grammer to accept the same language $(yxw|y\star xt)$

#	Rules
1	$S \rightarrow x\,t\,\$$
2	$S \rightarrow Q\,t\,\$$
3	$S \rightarrow B\,w\,\$$
4	$Q \rightarrow y\,Q$
5	$Q \rightarrow y\,x$
6	$B \rightarrow y\,x$

	t	w	x	y	$\$$	Q	B
0			sh-1	sh-2		sh-3	sh-4
1	sh-5						
2			sh-6	sh-7		sh-8	
3	sh-9						
4		sh-10					
5					sh-11		
6	r-5	r-6					
7			sh-12	sh-7		sh-8	
8	r-4						
9					sh-13		
10					sh-14		
11	Reduce 1						
12	r-5						
13	Reduce 2						
14	Reduce 3						

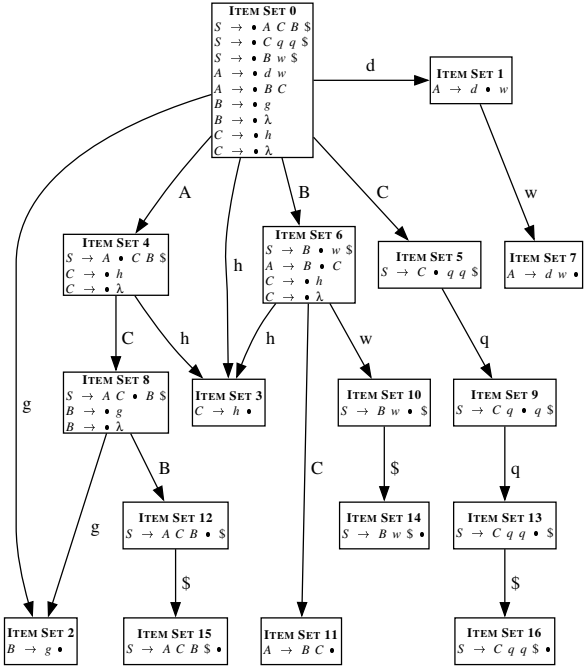


Reduce-Reduce Conflicts in SLR(1) Tables

	d	g	h	q	w	$\$$	A	C	B
0	sh-1	⋆	⋆	r-9	r-7	⋆	sh-4	sh-5	sh-6
1					sh-7				
2		r-6	r-6		r-6	r-6			
3		r-8	r-8	r-8		r-8			
4		r-9	⋆	r-9		r-9		sh-8	
5				sh-9					
6		r-9	⋆	r-9	sh-10	r-9		sh-11	
7		r-4	r-4			r-4			
8		⋆	r-7		r-7	r-7			sh-12
9				sh-13					
10						sh-14			
11		r-5	r-5			r-5			
12						sh-15			
13						sh-16			
14	Reduce 3								
15	Reduce 1								
16	Reduce 2								

Multiple **reduce-reduce** conflicts in Item Set 0, why?

$$Follow(A) = \{\ \$,\ g,\ h\ \}$$
$$Follow(B) = \{\ \$,\ w,\ g,\ h\ \}$$
$$Follow(C) = \{\ \$,\ q,\ g,\ h\ \}$$



Reduce-Reduce Conflicts in SLR(1) Tables

	d	g	h	q	w	$\$$	A	C	B
0	sh-1	⋆	⋆	r-9	r-7	⋆	sh-4	sh-5	sh-6
1					sh-7				
2		r-6	r-6		r-6	r-6			
3		r-8	r-8	r-8		r-8			
4		r-9	⋆	r-9		r-9		sh-8	
5				sh-9					
6		r-9	⋆	r-9	sh-10	r-9		sh-11	
7		r-4	r-4			r-4			
8		⋆	r-7		r-7	r-7			sh-12
9				sh-13					
10						sh-14			
11		r-5	r-5			r-5			
12						sh-15			
13						sh-16			
14	Reduce 3								
15	Reduce 1								
16	Reduce 2								

Because the λ-items for B and C share $\{ \$, g, h \}$ in their follow sets.

$Follow(A) = \{ \$, g, h \}$

$Follow(B) = \{ \$, w, g, h \}$

$Follow(C) = \{ \$, q, g, h \}$

#	Rules
1	$S \rightarrow d\ w\ h\ g\ \$$
2	$S \rightarrow d\ w\ g\ \$$
3	$S \rightarrow d\ w\ h\ \$$
4	$S \rightarrow d\ w\ \$$
5	$S \rightarrow g\ h\ h\ g\ \$$
6	$S \rightarrow g\ h\ g\ \$$
7	$S \rightarrow g\ h\ h\ \$$
8	$S \rightarrow g\ h\ \$$
9	$S \rightarrow g\ h\ g\ \$$
10	$S \rightarrow g\ g\ \$$
11	$S \rightarrow g\ h\ \$$
12	$S \rightarrow g\ \$$
13	$S \rightarrow h\ h\ g\ \$$
14	$S \rightarrow h\ g\ \$$
15	$S \rightarrow h\ h\ \$$
16	$S \rightarrow h\ \$$
17	$S \rightarrow h\ g\ \$$
18	$S \rightarrow g\ \$$
19	$S \rightarrow h\ \$$
20	$S \rightarrow \$$
21	$S \rightarrow h\ q\ q\ \$$
22	$S \rightarrow q\ q\ \$$
23	$S \rightarrow g\ w\ \$$
24	$S \rightarrow w\ \$$

	d	*g*	*h*	*q*	*w*	$
0	sh-1	sh-2	sh-3	sh-4	sh-5	sh-6
1					sh-7	
2		sh-8	sh-9		sh-10	sh-11
3		sh-12	sh-13	sh-14		sh-15
4				sh-16		
5						sh-17
6	Reduce 20					
7		sh-18	sh-19			sh-20
8						sh-21
9		sh-22	sh-23			sh-24
10						sh-25
11						
12						sh-26
13		sh-27				sh-28
14				sh-29		
15						
16						sh-30
17	Reduce 24					
18						sh-31
19		sh-32				sh-33
20	Reduce 4					
21	Reduce 10					
22						sh-34
23		sh-35				sh-36
24						
25	Reduce 23					
26						
27						sh-37
28	Reduce 15					
29						sh-38
30	Reduce 22					
31	Reduce 2					
32						sh-39
33	Reduce 3					
34						
35						sh-40
36	Reduce 7					
37	Reduce 13					
38	Reduce 21					
39	Reduce 1					
40	Reduce 5					

Reduce-Reduce Conflicts in LR(0) Item Sets

Sometimes the reasons for conflicts can be reasoned out within the grammar, or subset of the grammar.

In this case, simple inspection of the (previous slide's) grammar reveals that it has **no recursive rules**, it's a **finite language** and we can generate an LR(0) compatible grammar for it by brute-force.

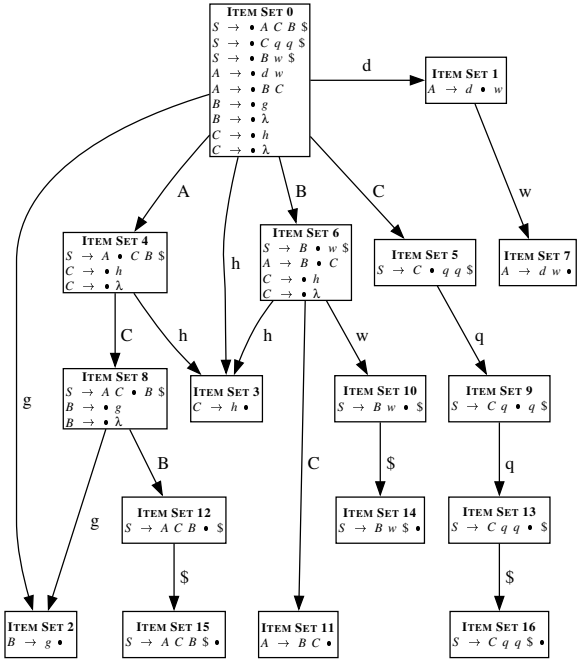
#	Rules
1	$S \rightarrow d\ w\ h\ g\ \$$
2	$S \rightarrow d\ w\ g\ \$$
3	$S \rightarrow d\ w\ h\ \$$
4	$S \rightarrow d\ w\ \$$
5	$S \rightarrow g\ h\ h\ g\ \$$
6	$S \rightarrow g\ h\ g\ \$$
7	$S \rightarrow g\ h\ h\ \$$
8	$S \rightarrow g\ h\ \$$
9	$S \rightarrow g\ h\ g\ \$$
10	$S \rightarrow g\ g\ \$$
11	$S \rightarrow g\ h\ \$$
12	$S \rightarrow g\ \$$
13	$S \rightarrow h\ h\ g\ \$$
14	$S \rightarrow h\ g\ \$$
15	$S \rightarrow h\ h\ \$$
16	$S \rightarrow h\ \$$
17	$S \rightarrow h\ g\ \$$
18	$S \rightarrow g\ \$$
19	$S \rightarrow h\ \$$
20	$S \rightarrow \$$
21	$S \rightarrow h\ q\ q\ \$$
22	$S \rightarrow q\ q\ \$$
23	$S \rightarrow g\ w\ \$$
24	$S \rightarrow w\ \$$

	d	g	h	q	w	$\$$
0	sh-1	sh-2	sh-3	sh-4	sh-5	sh-6
1					sh-7	
2		sh-8	sh-9		sh-10	sh-11
3		sh-12	sh-13	sh-14		sh-15
4				sh-16		
5						sh-17
6	Reduce 20					
7		sh-18	sh-19			sh-20
8						sh-21
9		sh-22	sh-23			sh-24
10						sh-25
11						
12						sh-26
13		sh-27				sh-28
14				sh-29		
15						
16						sh-30
17	Reduce 24					
18						sh-31
19		sh-32				sh-33
20	Reduce 4					
21	Reduce 10					
22						sh-34
23		sh-35				sh-36
24						
25	Reduce 23					
26						
27						sh-37
28	Reduce 15					
29						sh-38
30	Reduce 22					
31	Reduce 2					
32						sh-39
33	Reduce 3					
34						
35						sh-40
36	Reduce 7					
37	Reduce 13					
38	Reduce 21					
39	Reduce 1					
40	Reduce 5					

Yes! That's been two **Regular Languages** that failed SLR(1) table construction due to their grammar representations.

SLR(1), LALR(1) and LR(1) **are not silver bullets** for language design.

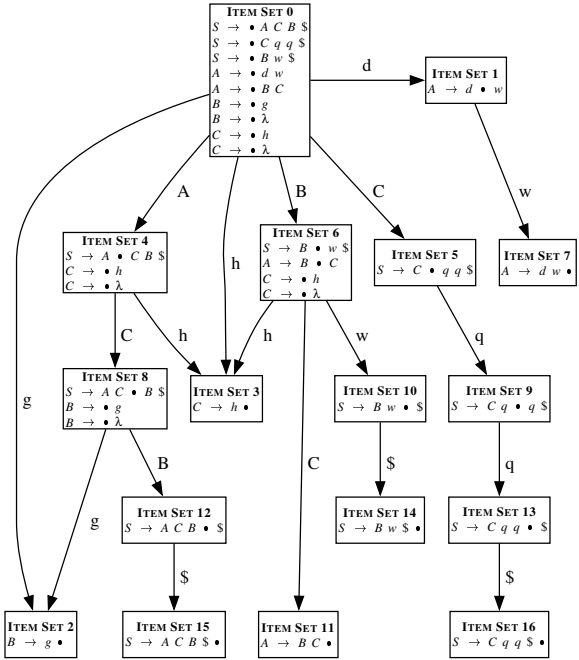
Developing a **non-conflicting grammar definition** for a particular language is a skill the language-writer-to-be must practice and develop.



Shift-Shift Conflicts?

	d	*g*	*h*	*q*	*w*	$	*A*	*C*	*B*
0	sh-1	⋆	⋆	r-9	r-7	⋆	sh-4	sh-5	sh-6
1					sh-7				
2		r-6	r-6		r-6	r-6			
3		r-8	r-8	r-8		r-8			
4		r-9	⋆	r-9		r-9		sh-8	
5				sh-9					
6		r-9	⋆	r-9	sh-10	r-9		sh-11	
7		r-4	r-4			r-4			
8		⋆	r-7		r-7	r-7			sh-12
9				sh-13					
10						sh-14			
11		r-5	r-5			r-5			
12						sh-15			
13						sh-16			
14	Reduce 3								
15	Reduce 1								
16	Reduce 2								

Could the remaining conflicts in **item sets 4, 6 and 8** be **shift-shift** conflicts?



Shift-Shift Conflicts?

	d	*g*	*h*	*q*	*w*	$	*A*	*C*	*B*
0	sh-1	⋆	⋆	r-9	r-7	⋆	sh-4	sh-5	sh-6
1					sh-7				
2		r-6	r-6		r-6	r-6			
3		r-8	r-8	r-8		r-8			
4		r-9	⋆	r-9		r-9		sh-8	
5				sh-9					
6		r-9	⋆	r-9	sh-10	r-9		sh-11	
7		r-4	r-4			r-4			
8		⋆	r-7		r-7	r-7			sh-12
9				sh-13					
10						sh-14			
11		r-5	r-5			r-5			
12						sh-15			
13						sh-16			
14	Reduce 3								
15	Reduce 1								
16	Reduce 2								

No: shift-shift conflicts don't exist, by nature of the construction algorithm for the CFSM. All items with grammar symbol X on the RHS of • form the **KERNEL** of (the same) new item set.

RL, LL(1), LR(0) Graphs, SLR(1) Table Generation, oh my

We've now looked at **regular languages** (NFAs $\equiv$ DFAs $\equiv$ REs), **context free grammars**, their analysis and their parsing using **recursive descent** (LL(1) tables), LR(0) graph construction (**item sets** and the **CFSM**), and **shift-reduce** LR-parsing tables generated directly from the CFSM (**LR-zero languages**) and by incorporating $k = 1$ **follow sets** to resolve conflicts (**SLR(1) tables**). These LR(0) graph based parsing tables are used with the **"knitting algorithm"** for rightmost derived parses.

CFGs are great for confirming a grammar (language) is **unambiguous**, and they have enough structure for common programming idioms (matched parenthesis, arithmetic expressions and operators, the **dangling bracket language**).

RL, LL(1), LR(0) Graphs, SLR(1) Table Generation, oh my

The last two examples of SLR(1) failing grammars[1] show that pedantic and sometimes tedious construction of a language grammar is essential to the construction of parsing tables.

We'll pause our discussion of CFGs and syntax verification ("parsing") now — it is time we move on to some of the most fundamental tasks of compilers: **abstract syntax tree construction**, **semantic analysis**, and the generation of **machine code**.

The hope is that we'll have enough time at the end of the semester to "circle back" and discuss a slightly better LR table construction method (**LALR(1)** with **propagation graphs**) as well as the truely canonical LR(1) grammar analysis.

[1] Which were what, really? Languages that could have been matched by a DFA! Geesh.