Earley Parser

Christopher Millar and Ekaterina Volkova

Seminar für Sprachwissenschaft

Universität Tübingen

January 2007

Earley Parser: Bottom-up parsers

In general, **breadth-first bottom-up** parsers are attractive since:

- they work on-line;
- can handle left-recursion;
- can be doctored to handle ε-rules.

Earley Parser: Bottom-up problem

Still the question remains:

How to curb their needless activity?

A method that will *restrict the fan-out* to reasonable proportions while still *retaining full generality* was developed by Earley .

Earley Parser: Basic Concept

Main problem: the **spurious reductions** can never derive from the **start symbol**.

Solution: give a method to **restrict the reductions** only to those that derive from the **start symbol**.

The resulting parser takes **at most n^3** units of time for input of length n **rather than C^n**.

Earley Parser: Definition

Earley's parser can also be described as a **breadth-first top-down** parser with **bottom-up recognition**, Still, we prefer to treat it as a **bottom-up method,** for it can handle left-recursion directly but needs special measures to handle ε-rules.

Earley Parser: Earley Item

An Earley item is an **item** with an indication of the **position** of the symbol at which the recognition of the recognized part started.

E->E•QF@3 ⟵ ***Position***

The sets of items contain exactly those items...

a) of which the part before the dot has been recognized so far **...and...**

b) are useful in reaching the start symbol.

Earley Parser: Methods

The Earley Parser uses methods called **Scanner, Completer** and **Predictor**.

- **Scanner** is like “shift”.
- **Completer** is like “reduce”.
- **Predictor** is unique to the Earley parser.

Earley Parser: Scanner

Scanner

Earley Parser: Completer

Completer

Earley Parser: Predictor

Predictor

Earley Parser: The Sigma

The Scanner, Completer and Predictor deal with **four sets of items** for each token in the input.

We'll refer to a token as **sigma@p** or as

$$\boldsymbol{\delta}_{\mathbf{p}}$$

Earley Parser: The Four Sets

sigma@p is surrounded by four sets:

- **itemset@p-1**
- **completed@p**
- **active@p**
- **predicted@p**

Earley Parser: itemset@p-1

itemset@p-1

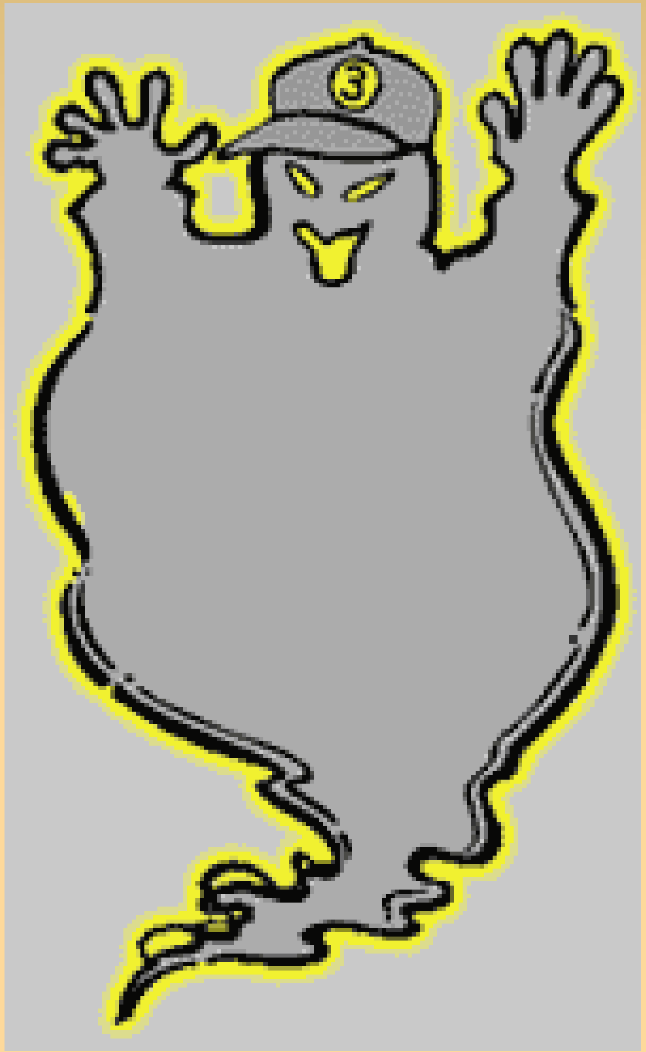
Earley Parser: completed@p

completed@p

Earley Parser: active@p

active@p

Earley Parser: predicted@p



predicted@p

Earley Parser: The Four Sets, cont.

- **itemset@p-1 -** items available just before sigma@p;
- **completed@p -** items that have become completed after sigma@p;
- **active@p -** non-completed items after sigma@p:
- **predicted@p** - the set of newly predicted items.

Earley Parser:The Scanner

The Scanner :

looks at **sigma@p** -> goes through **itemset@p-1**

-> makes copies of all items that contain **•sigma**

-> changes them to **sigma •** -> adds them...

a) to the set **completed@p** if the item@p was completed ...or...

b) to the set **active@p** if the **item@p** is not yet completed

Earley Parser:The Scanner, cont.

Rules not containing •sigma are discarded!

Earley Parser: The Completer

The Completer inspects **completed@p**, which contains the completely recognized items and can now be ***reduced***.

Earley Parser: The Completer, cont.

For each item of the form **R --> sigma@m** the Completer goes to **itemset@(m-1)**, and calls the Scanner; which goes to work on **R**.

Earley Parser: The Completer

The Scanner will make copies of all items in **itemset@(m-1)** featuring a •R, replace the •R by R• and store them in either **completed@p** or **active@p**. At this stage items could be added to the set **completed@p**.

Earley Parser: The Completer

Eventually the Completer stops completing.

(When it has completely completed

the set **completed@p** :))

Earley Parser: The Predictor

The Predictor goes through the sets **active@p** (which was filled by the Scanner) and **predicted@p** (which is empty initially), and considers all non-terminals which have a • before them.

Earley Parser: The Predictor, cont.

For each expected non-terminal **N** and each rule for that non-terminal **N --> P...,** the Predictor adds an item to the set **predicted@p**.

Earley Parser: The Predictor, cont.

This may introduce new predicted non-terminals (for instance, P) to **predicted@p** which causes more work for the Predictor.

Earley Parser: The Predictor, cont.

Eventually the Predictor stops predicting.

Earley Parser: Recognition

The sets **active@p** and **predicted@p** together form the new **itemset@p**. If the completed set for the last symbol in the input contains an item **S-->...•@1**. Then the input is recognized.

Earley Parser: Example

Consider an example with the following grammar and the input: **a - a + a.**

S --> E

E --> EQF

E --> F

Q --> +

Q --> -

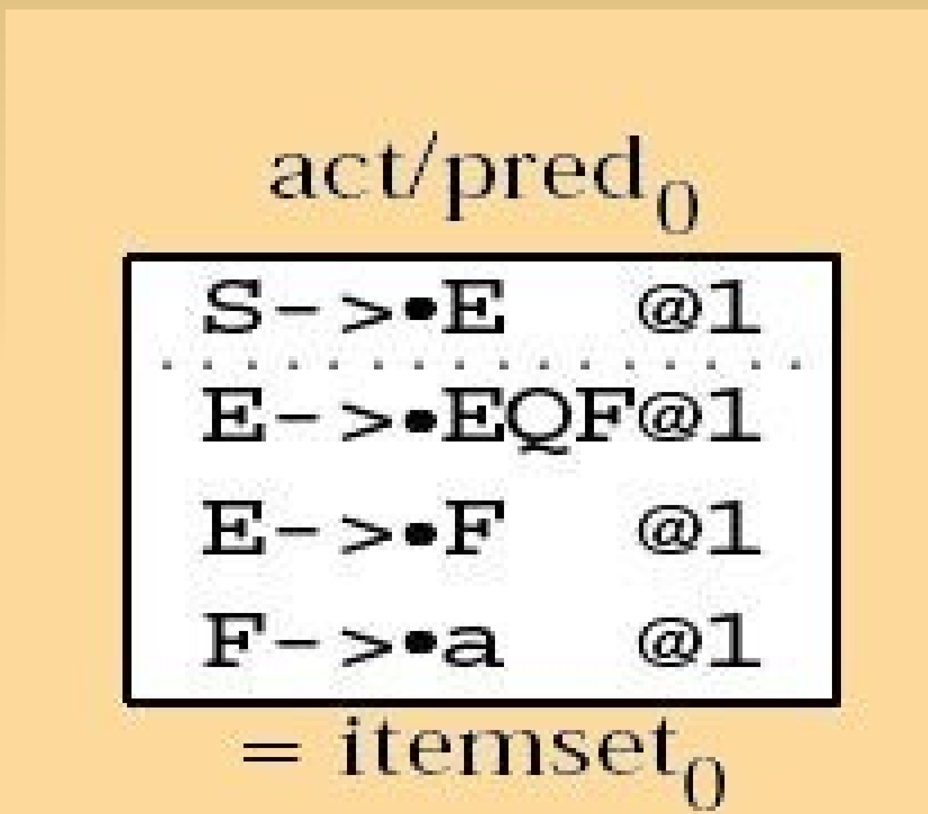
F --> a

Earley Parser: Example, cont.

There is one **Predictor, Scanner** and **Completer** stage for each symbol.

Parsing begins by **calling the Predictor** on the initial active set containing **S --> E@1** which generates **itemset@0.**

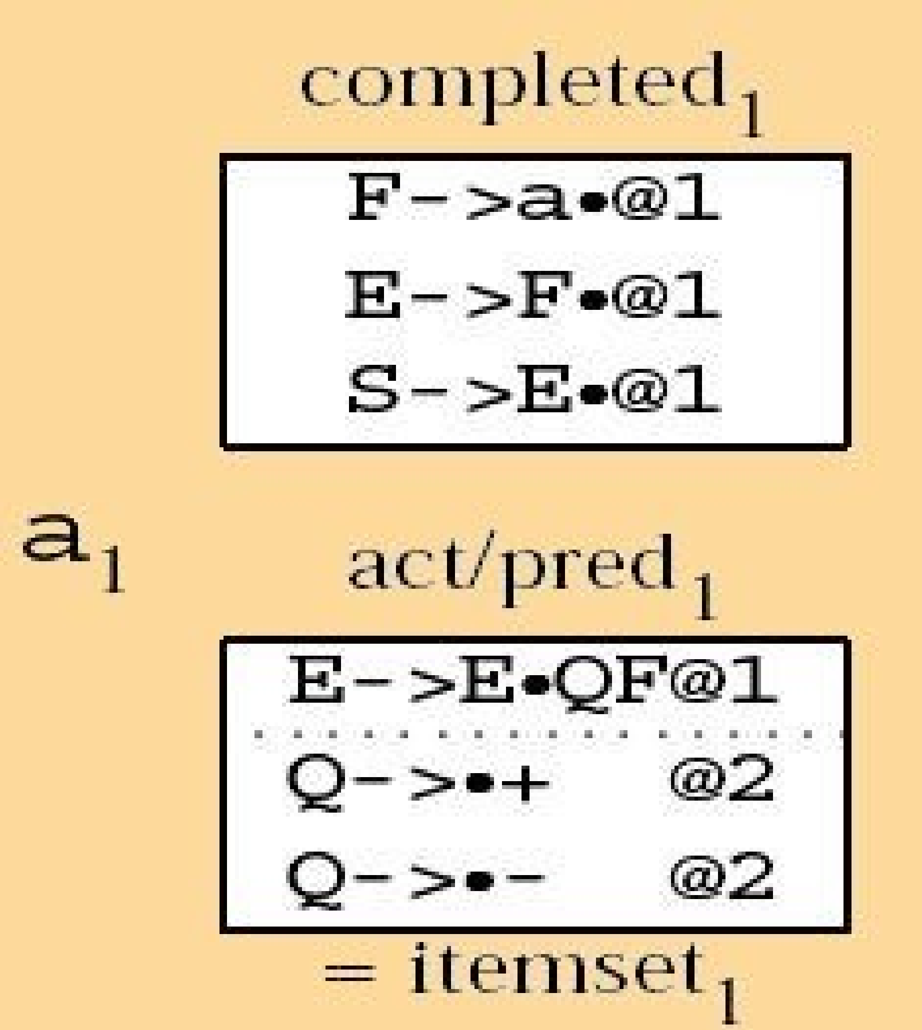
Earley Parser: δ@0



The Predictor, reads **active@0, {S-> •E@1 }** and **predicted@0**, which is initially empty, and fills the set **predicted@0**.

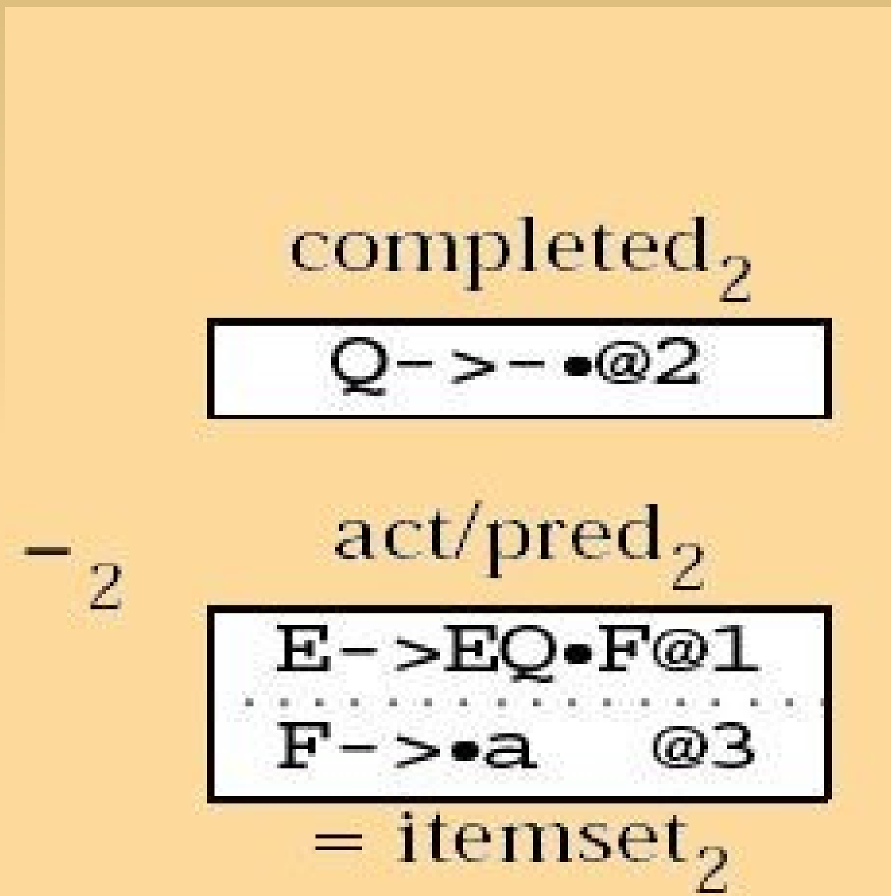
{act.@0} U {pred.@0} = {itemset@0}

Earley Parser: δ@1



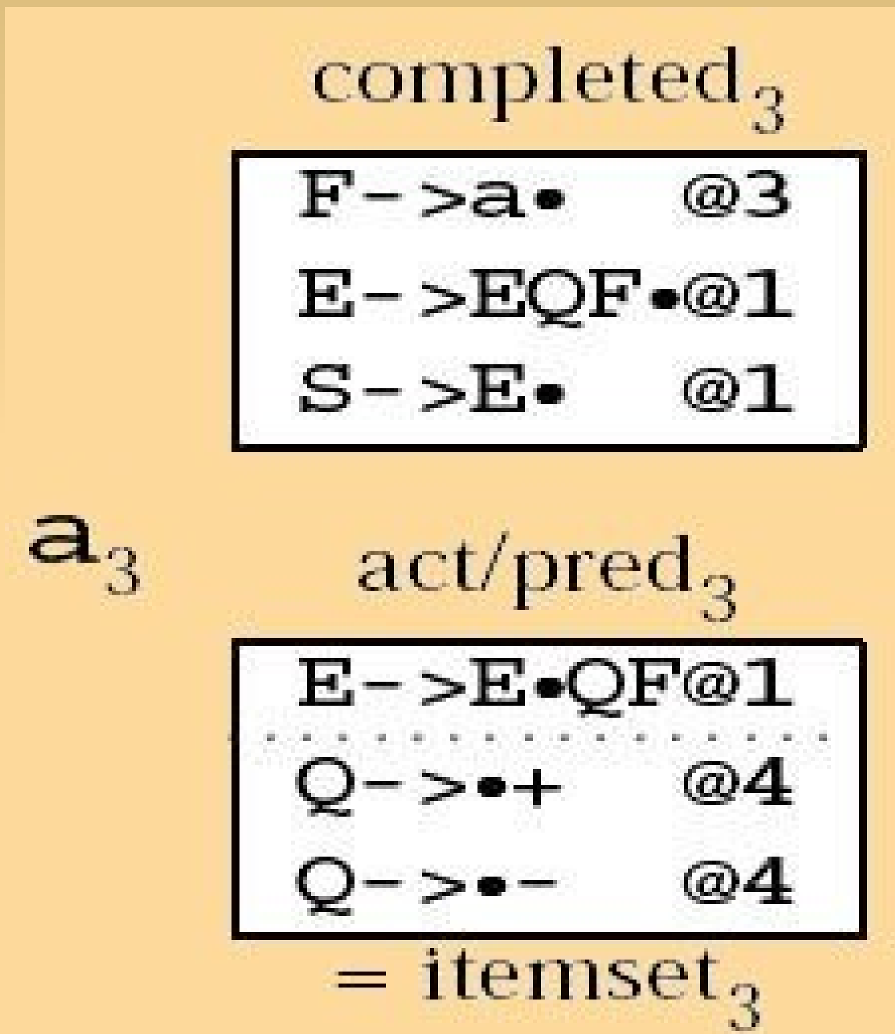
After scanning **δ@1** the Completer completes some rules, and puts the other possible rules in **active@1**. Predictor makes predictions from those that are in the active set.

Earley Parser: δ@2



Continue as before until the input is consumed.

Earley Parser: δ@3



As you can see we already have few possibilities...

Earley Parser: δ@4

completed_4

Q->+•@4

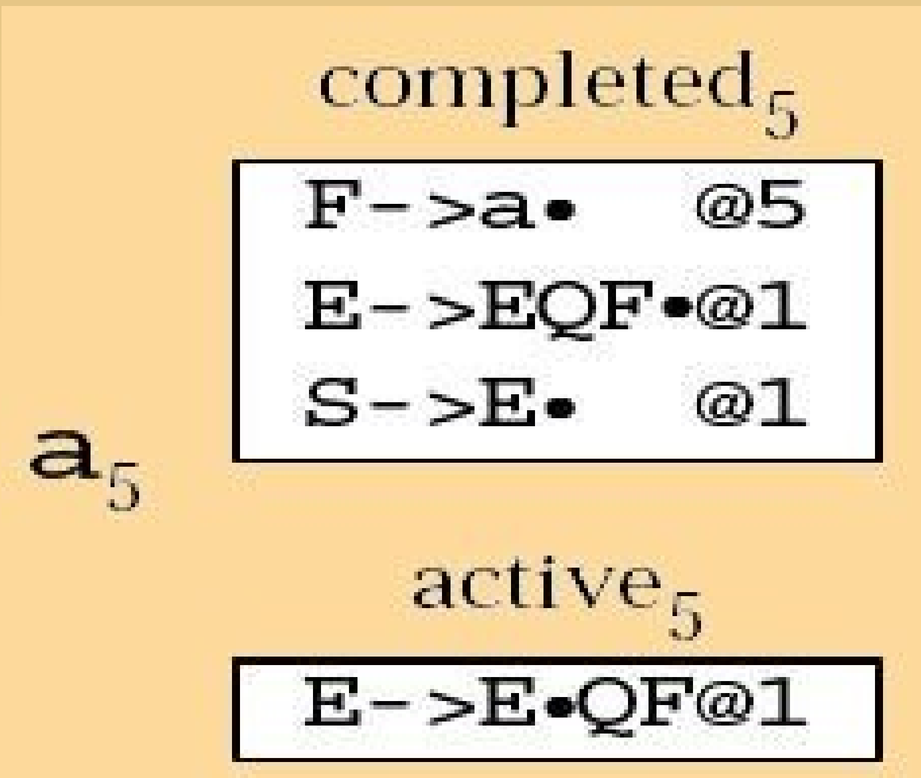
$+_4$

act/pred_4

E->EQ•F@1
F->•a @5

$= \text{itemset}_4$

Earley Parser: δ@5



S --> E• @1 is in the set completed and the last input symbol has been read.

Therefore the sentence is recognized!!!

Earley Parser: Comparison to CYK

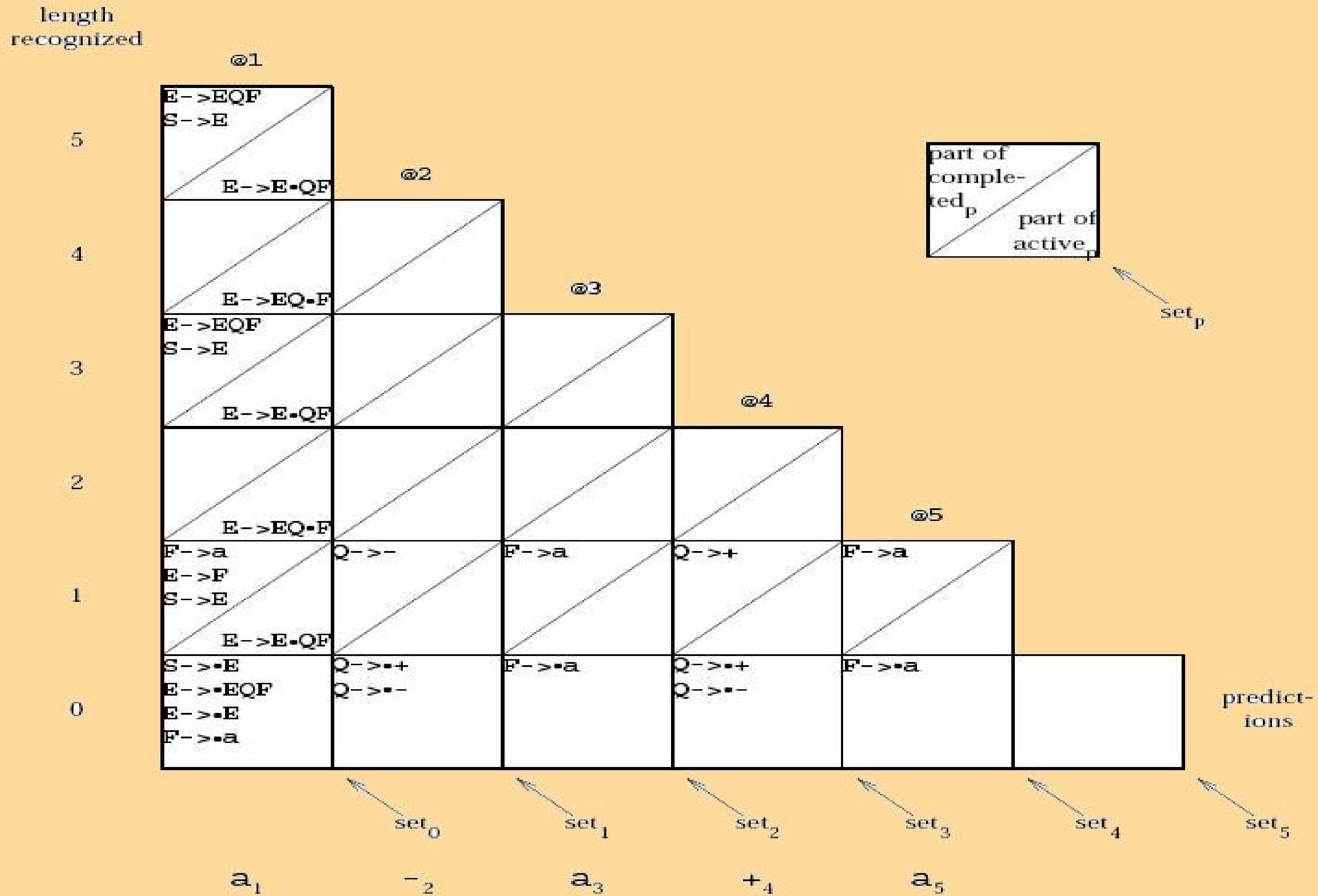
Similarities:

- are Chart Parsers
- worst case memory requirements $O(n^2)$
- worst case time complexity $O(n^3)$
- use bottom-up recognition
- use a top-down parser to build trees

Earley Parser: Comparison to CYK

The Early Parser however eliminates rules which will not be useful as we go along, with non ambiguous grammars such as the example shown we get a worst time complexity of $O(n^2)$.

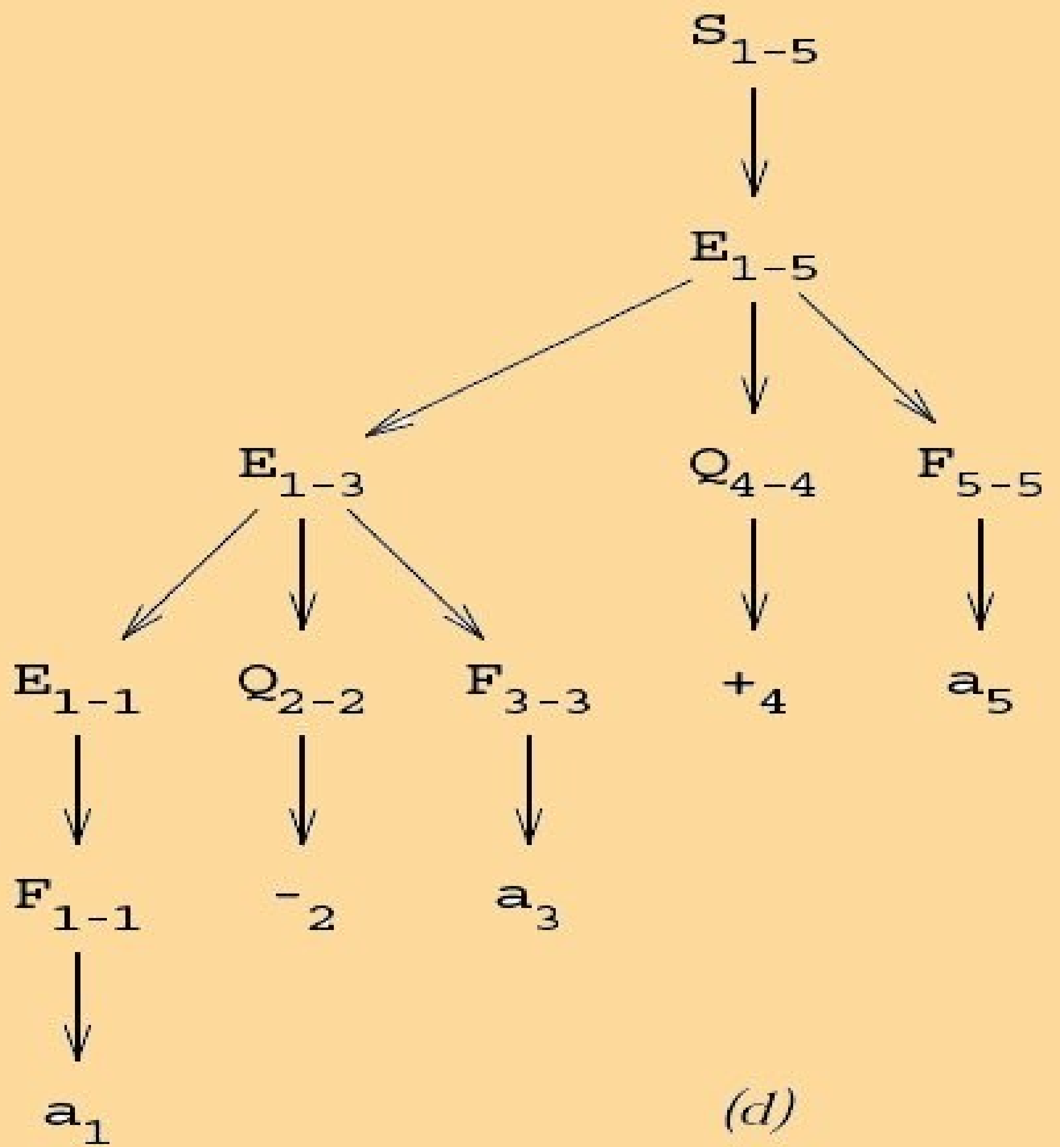
Earley Parser: Recognition Chart



Earley Parser: CYK Recognition Chart

length recognized	@1	@2	@3	@4	@5
5	E S				
4					
3	E S		E S		
2					
1	F E S	Q	F E S	Q	F E S
	a_1	$-_2$	a_3	$+_4$	a_5

Earley Parser: Parsing Tree



As with the CYK parser, a simple top-down Unger-type parser can be used to reconstruct all possible parse trees from a chart.

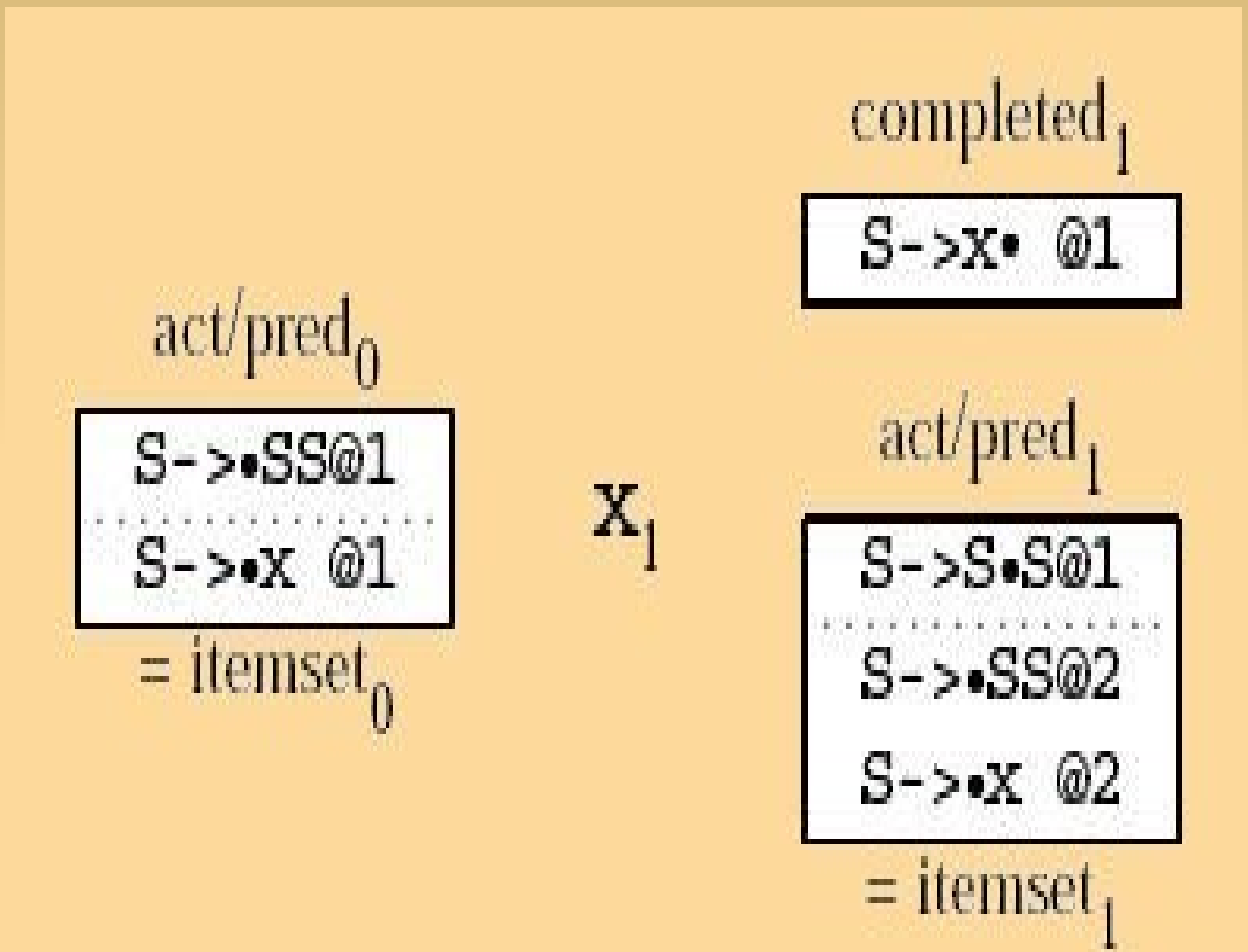
Earley Parser: A Worse Example

We get worst case behaviour when we have to deal with ambiguous grammars like:

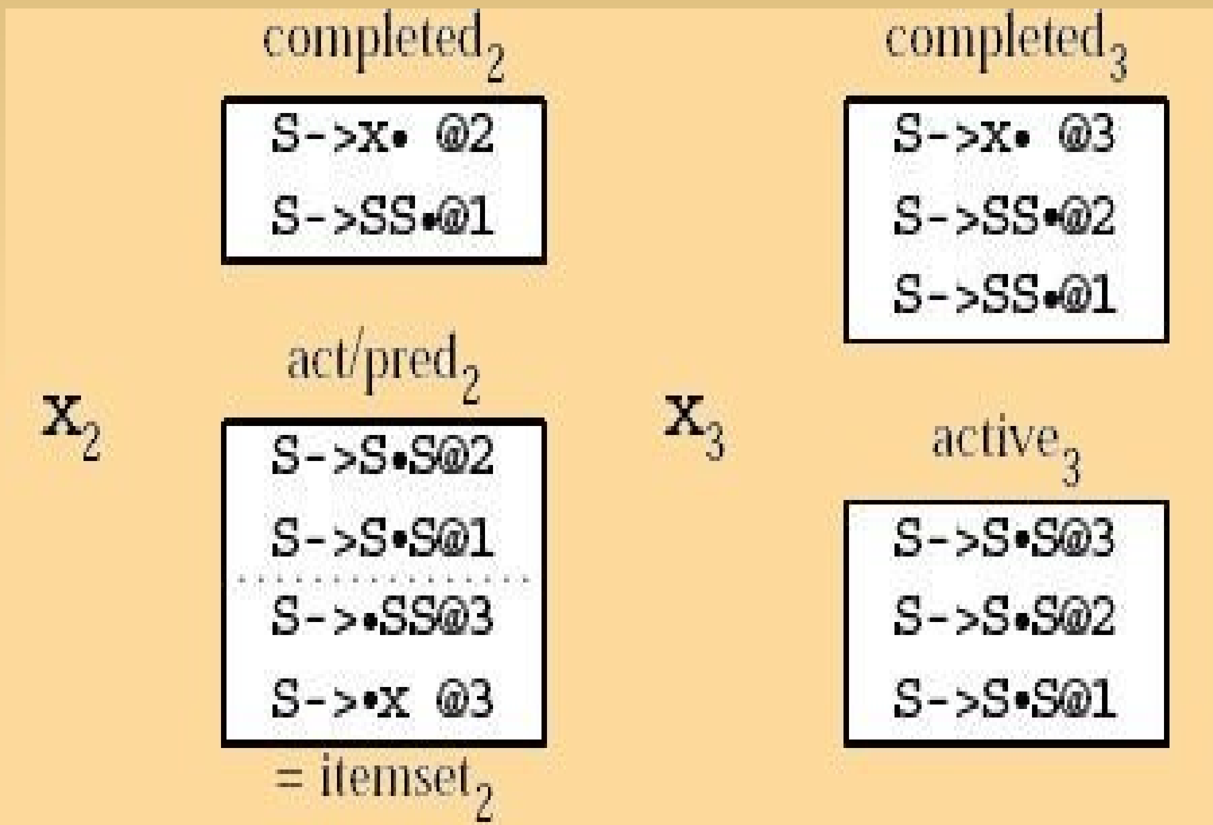
S --> SS

S --> x

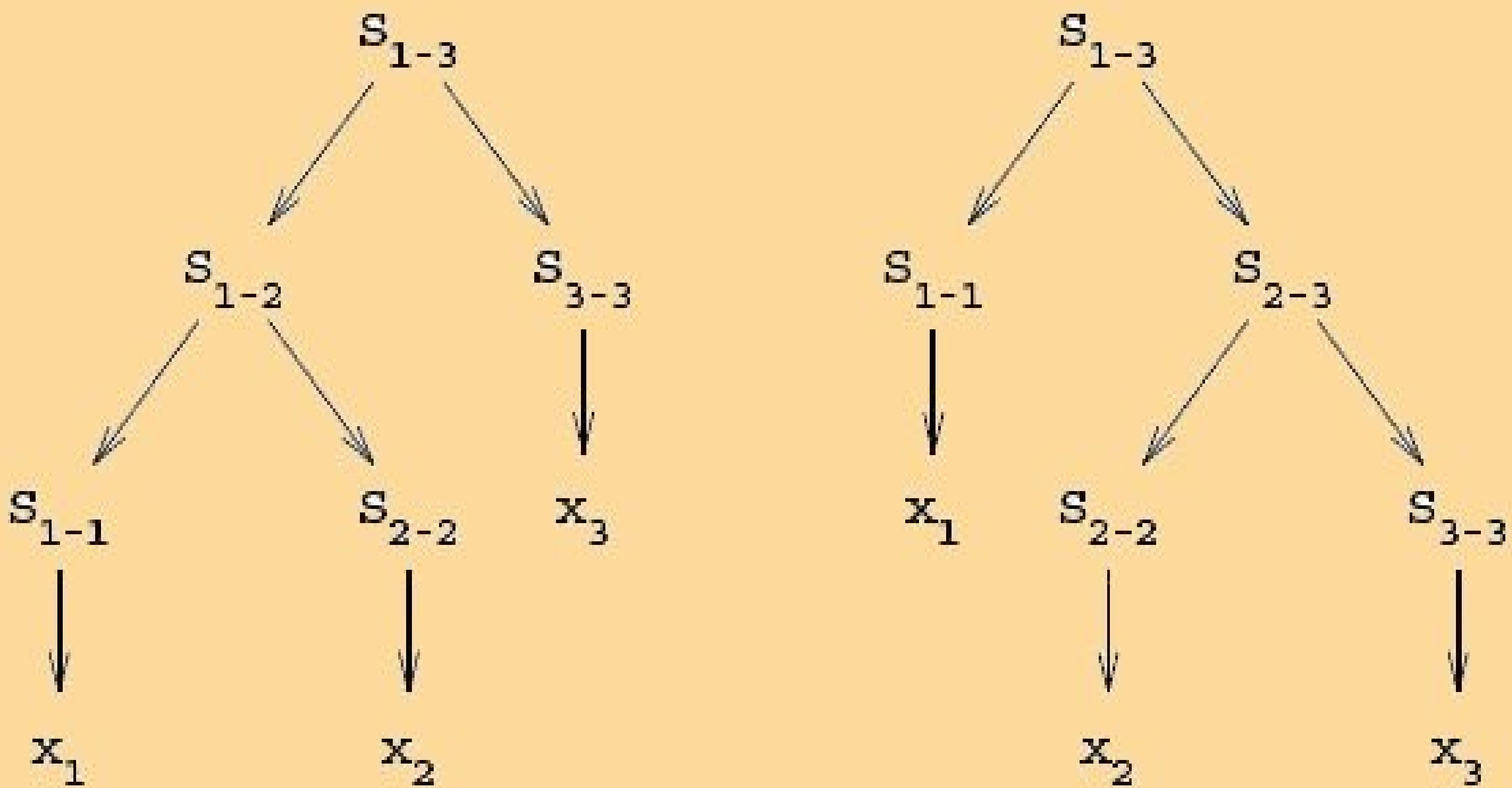
Earley Parser: A Worse Example, cont.



Earley Parser: A Worse Example, cont.



Earley Parser: A Worse Example, cont.



Earley Parser: A Worse Example, cont.

The **active@p** and **predicted@p** sets keep growing untill the final symbol is read. When building a parse tree from the resulting chart we find two possible derivations, but if the input would be longer the the situation would be worse!

Earley Parser: **ε-rules**

The Earley parser doesn't like ε-rules!

(Does **anybody** like them?)

Earley Parser: **ε-rules,** cont.

Consider the following non-e-free grammar with the input a a / a.

S --> E E --> EQF E --> F

Q --> *

Q --> /

Q --> e

F --> a

Earley Parser: **ε-rules**, cont.

After reading a1 we have a situation where every time the predictor predicts a •Q it must also predict a Q•

Earley Parser: **ε-rules**, cont.

act/pred$_1$

E->E•QF@1
E->EQ•F@1
Q->•x @2
Q->•/ @2
F->•a @2

= itemset$_1$

This can effect the behaviour of the Completer which is working on itemset@1.

Earley Parser: **ε-rules,** cont.

In the end we can find a parse with this grammar.

Earley Parser: **ε-rules,** cont.

What would happen to the itemset if we had a rule Q --> QQ ?

Earley Parser: **ε-rules**, cont.

An Early parser would resolve it but not without inefficiency.

E --> E•QF   E --> EQ•F

Q --> •QQ   Q --> Q•Q   Q --> QQ•

Q --> *

Q --> /   ε-rules add significantly to the

F --> a   complexity time

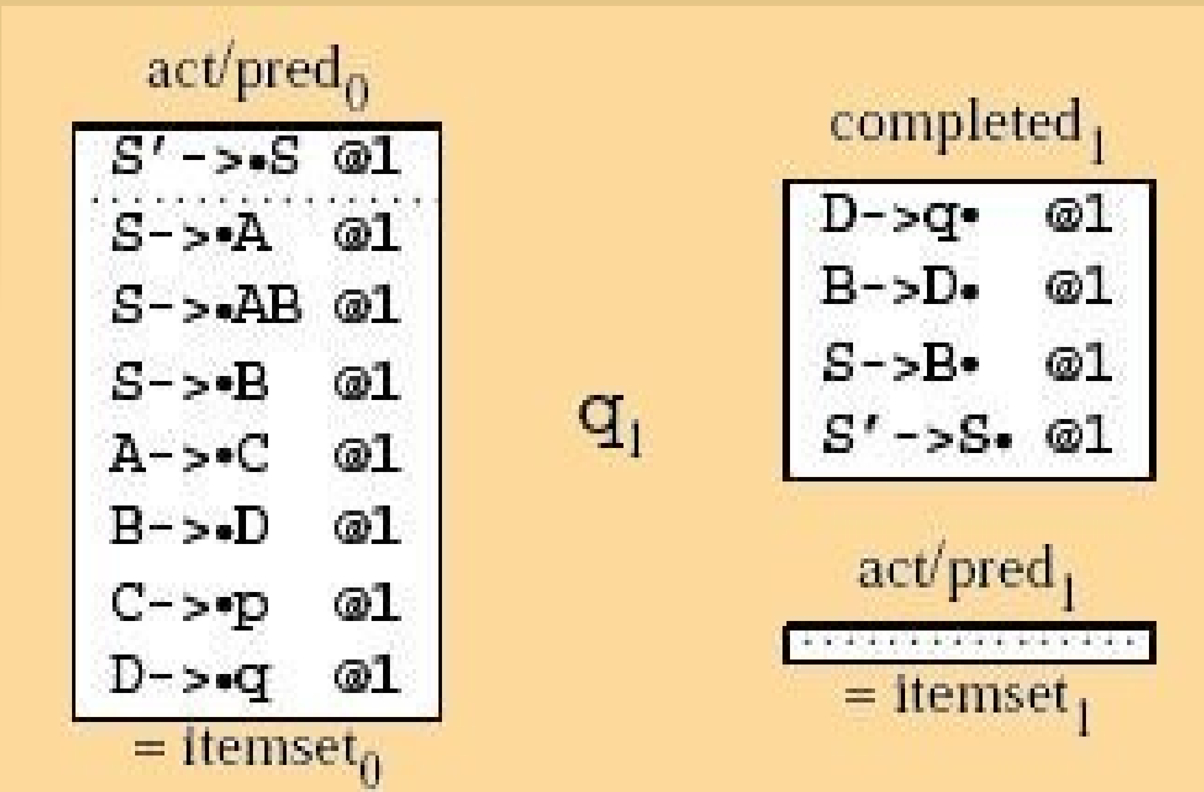
Earley Parser: Prediction Lookahead

Prediction Lookahead reduces the number of incorrect predictions made by the Predictor by considering next input symbol before adding items to **predicted@p**. It uses a set of FIRST terminal symbols, for each non terminal.

Earley Parser: Prediction Lookahead

S -> A | AB | B	FIRST(S) = {p, q}
A -> C	FIRST(A) = {p}
B -> D	FIRST(B) = {q}
C -> p	FIRST(C) = {p}
D -> q	FIRST(D) = {q}

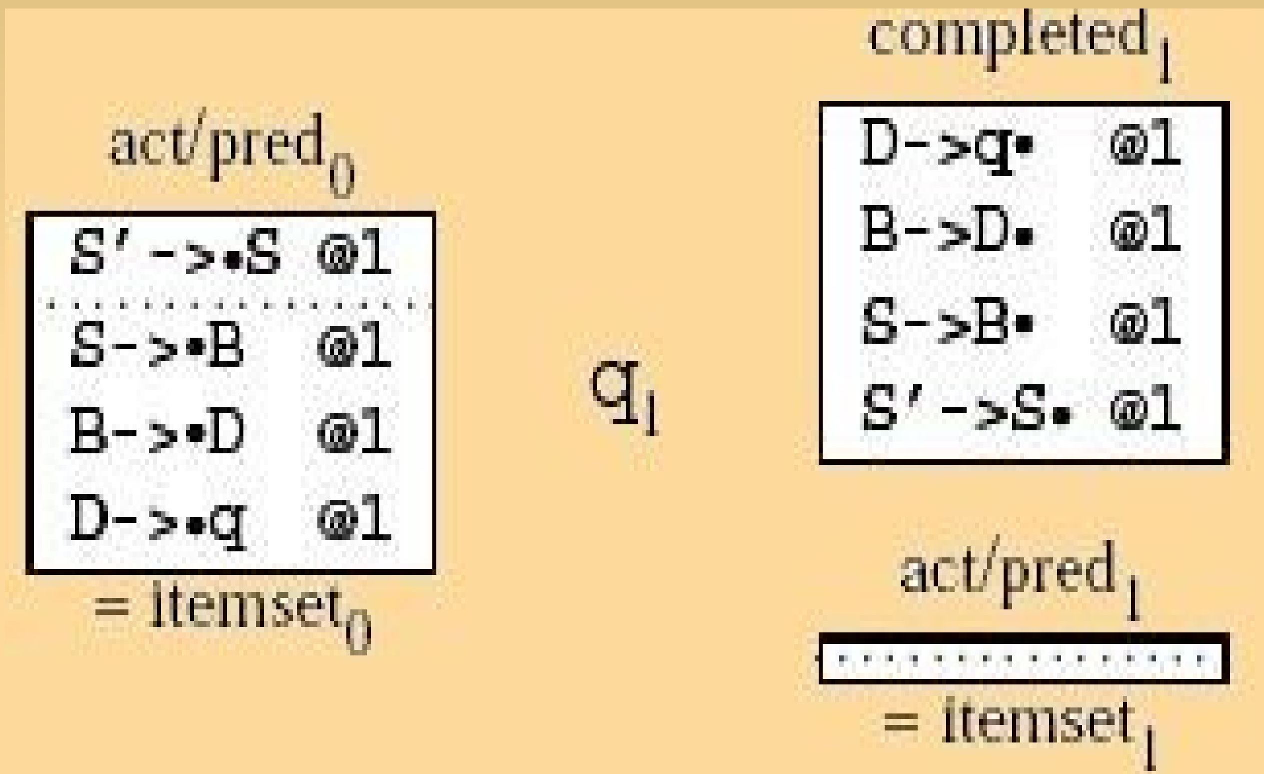
Earley Parser: Prediction Lookahead



Without

lookahead

Earley Parser: Prediction Lookahead



With lookahead

Earley Parser: Conclusion

Earley Parser shows a very successful combination of strong sides of top-down and bottom-up methods, handles well left recursion and ε-rules, and, being armoured by lookahead, takes the optimal possible amount of memory.

Earley Parser: Conclusion

Earley rules!