LR Parsing

Kilian Evang

Seminar für Sprachwissenschaft
Universität Tübingen

2007-01-24

Outline

Properties of LR Parsing

- bottom-up
- directional (consumes input from left to right)
- related to Earley parsing
- handles left recursion
- handles ε-rules in its more sophisticated form
- fast - parses in linear time
- deterministic

Properties of LR Parsing

- bottom-up
- directional (consumes input from left to right)
- related to Earley parsing
- handles left recursion
- handles ε-rules in its more sophisticated form
- fast - parses in linear time
- deterministic

Properties of LR Parsing

- bottom-up
- directional (consumes input from left to right)
- related to Earley parsing
- handles left recursion
- handles ε-rules in its more sophisticated form
- fast - parses in linear time
- deterministic

Properties of LR Parsing

- bottom-up
- directional (consumes input from left to right)
- related to Earley parsing
- handles left recursion
- handles ε-rules in its more sophisticated form
- fast - parses in linear time
- deterministic

Properties of LR Parsing

- bottom-up
- directional (consumes input from left to right)
- related to Earley parsing
- handles left recursion
- handles ε-rules in its more sophisticated form
- fast - parses in linear time
- deterministic

Properties of LR Parsing

- bottom-up
- directional (consumes input from left to right)
- related to Earley parsing
- handles left recursion
- handles ε-rules in its more sophisticated form
- fast - parses in linear time
- deterministic

Properties of LR Parsing

- bottom-up
- directional (consumes input from left to right)
- related to Earley parsing
- handles left recursion
- handles ε-rules in its more sophisticated form
- fast - parses in linear time
- deterministic

The idea

- The central task of the parser is to find *handles* in the input.
- *Peter goes on a trip with his wife*
- *Peter goes PP with his wife*
- $4 + 3 \times 5$
- $4 + 15$
- In parsing, a lot depends on finding the right handles as early as possible.

The idea

- The central task of the parser is to find *handles* in the input.
- *Peter goes on a trip with his wife*
- *Peter goes PP with his wife*
- $4 + 3 \times 5$
- $4 + 15$
- In parsing, a lot depends on finding the right handles as early as possible.

The idea

- The central task of the parser is to find *handles* in the input.
- *Peter goes on a trip with his wife*
- *Peter goes PP with his wife*
- $4 + 3 \times 5$
- $4 + 15$
- In parsing, a lot depends on finding the right handles as early as possible.

The idea

- The central task of the parser is to find *handles* in the input.
- *Peter goes on a trip with his wife*
- *Peter goes PP with his wife*
- $4 + 3 \times 5$
- $4 + 15$
- In parsing, a lot depends on finding the right handles as early as possible.

The idea

- The central task of the parser is to find *handles* in the input.
- *Peter goes on a trip with his wife*
- *Peter goes PP with his wife*
- $4 + 3 \times 5$
- $4 + 15$
- In parsing, a lot depends on finding the right handles as early as possible.

The idea

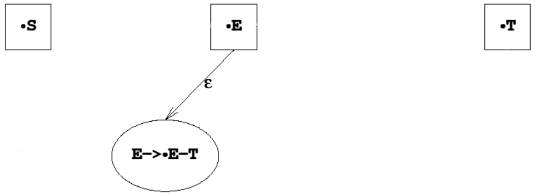
- The central task of the parser is to find *handles* in the input.
- *Peter goes on a trip with his wife*
- *Peter goes PP with his wife*
- $4 + 3 \times 5$
- $4 + 15$
- In parsing, a lot depends on finding the right handles as early as possible.

The idea (cont.)

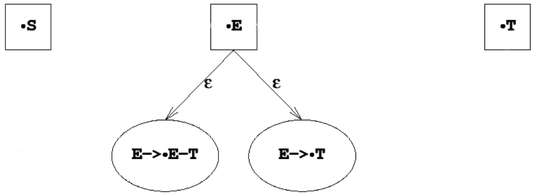
- A sequence of tokens can be a handle only if it could have been derived from some non-terminal in that position, which in turn could have been derived. . . and so on, going all the way back up to the start symbol.
- To take that into consideration, LR parsers use a top down component like the Earley parser.

Building an LR Parser

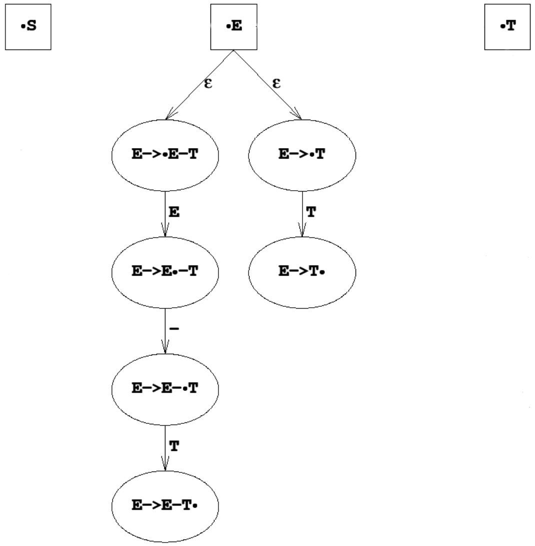
Building an LR Parser



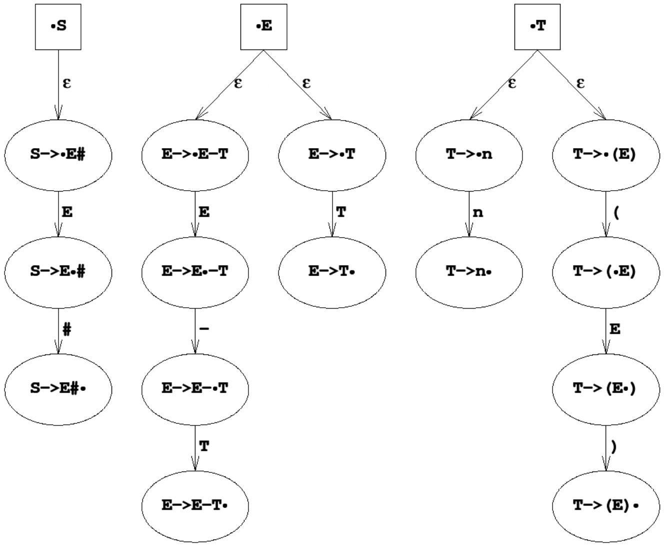
Building an LR Parser



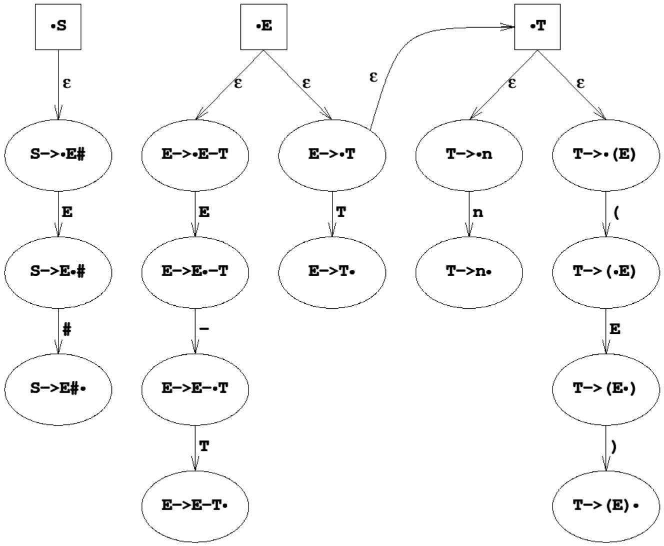
Building an LR Parser



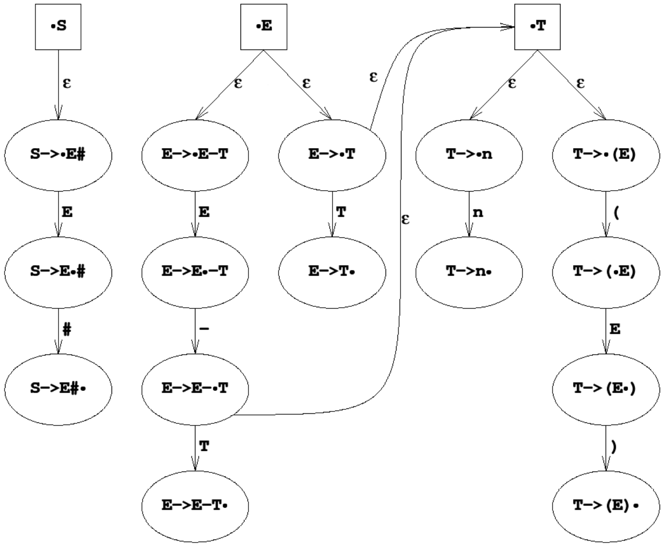
Building an LR Parser



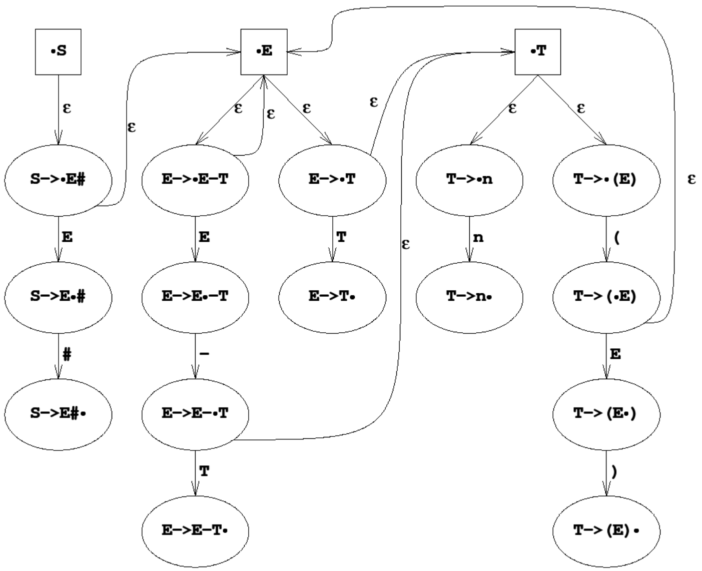
Building an LR Parser



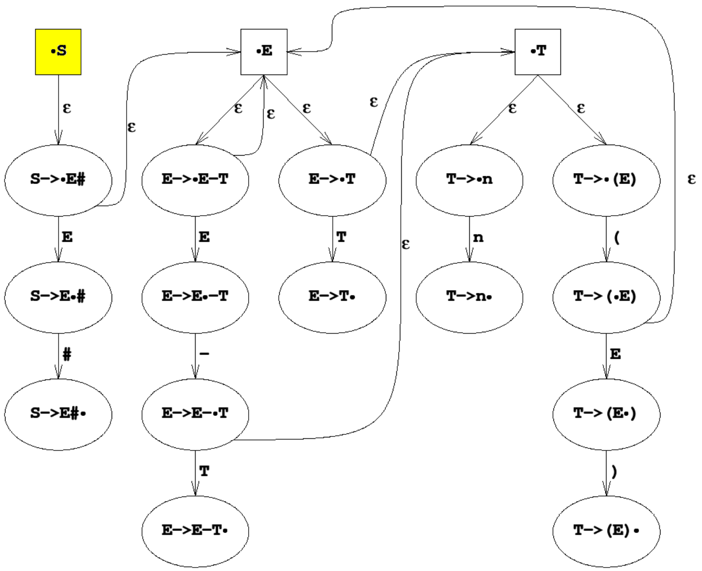
Building an LR Parser



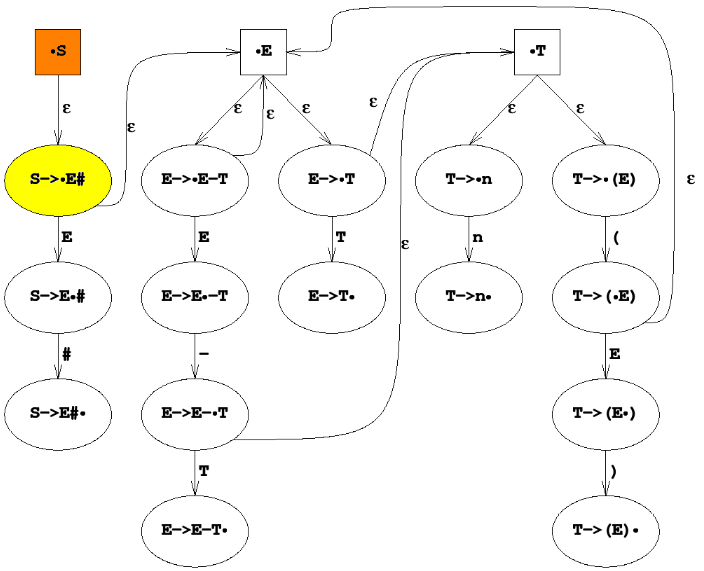
Building an LR Parser



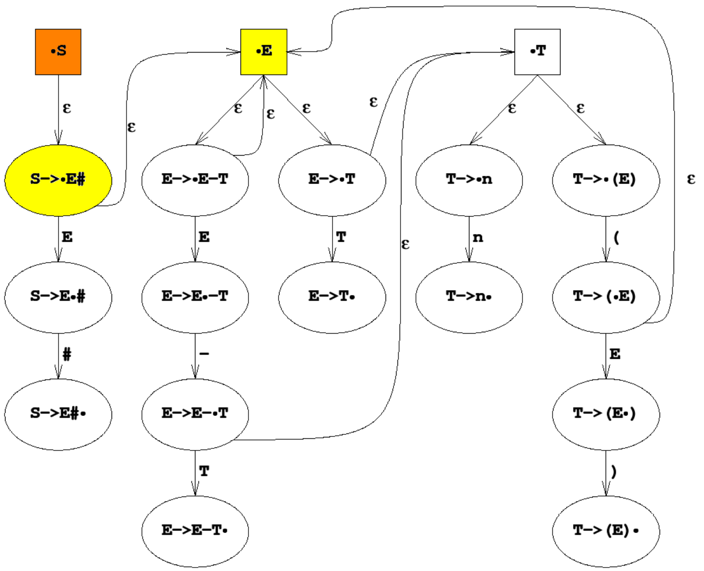
Using an LR Parser



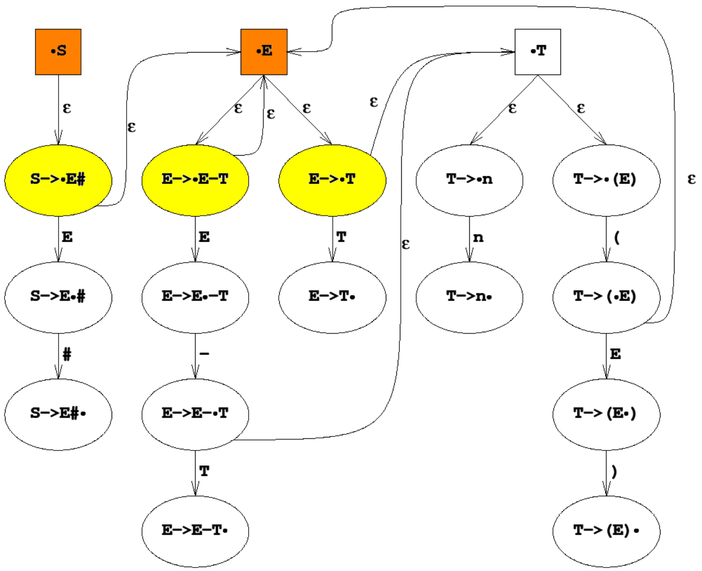
Using an LR Parser



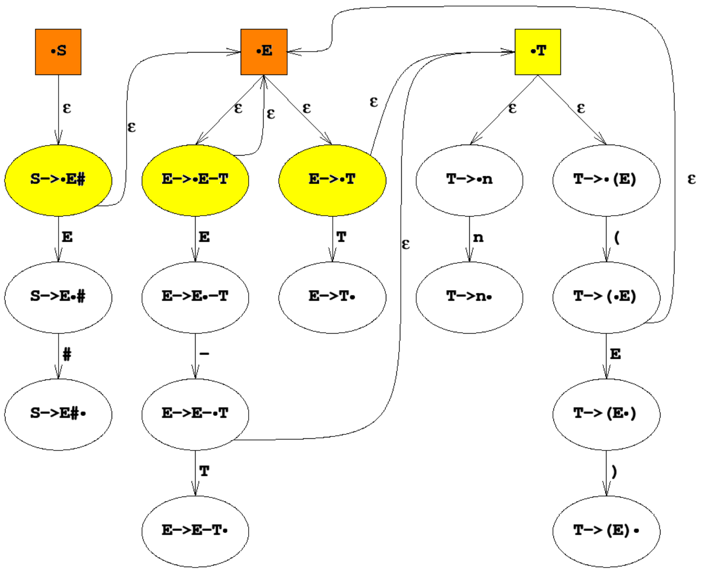
Using an LR Parser



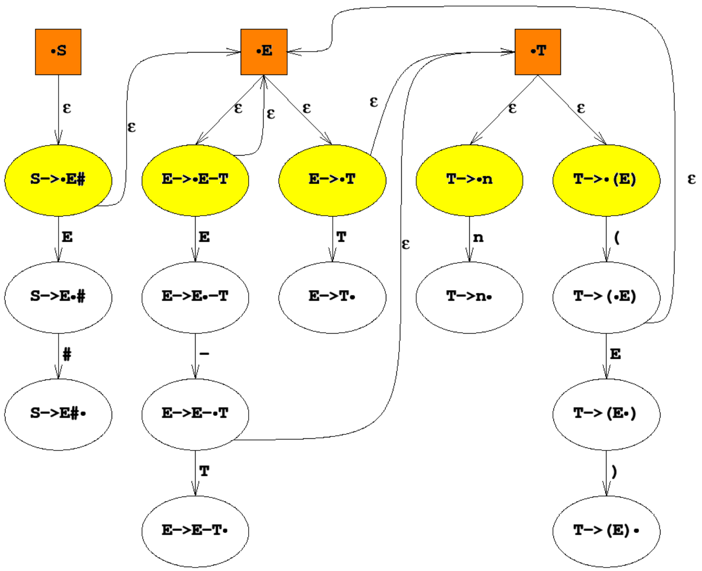
Using an LR Parser



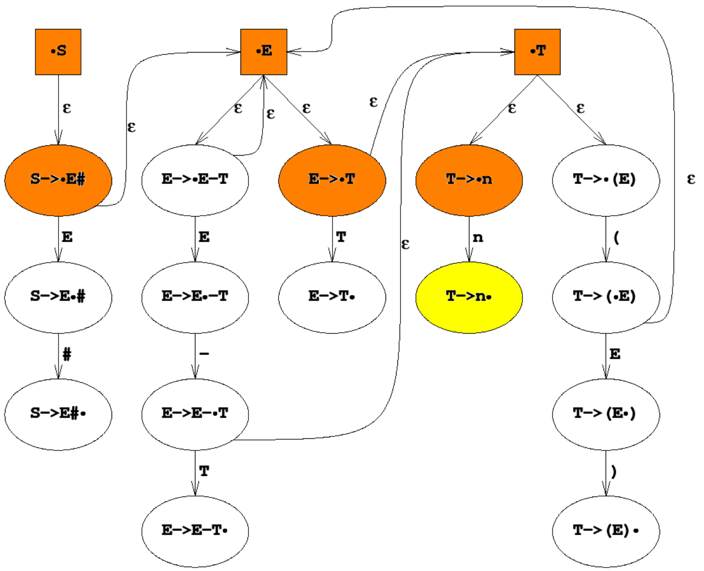
Using an LR Parser



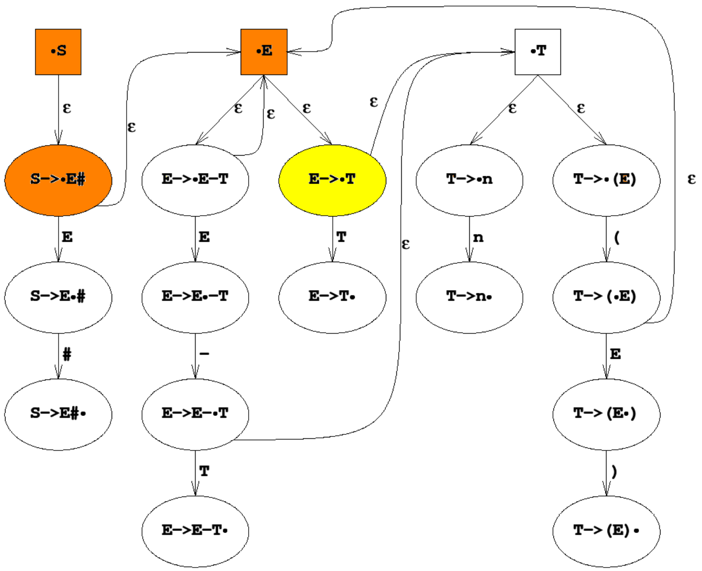
Using an LR Parser



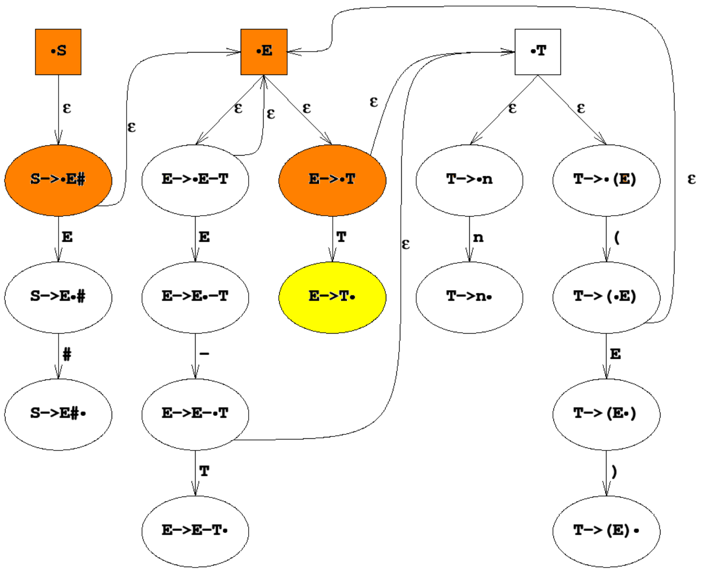
Using an LR Parser



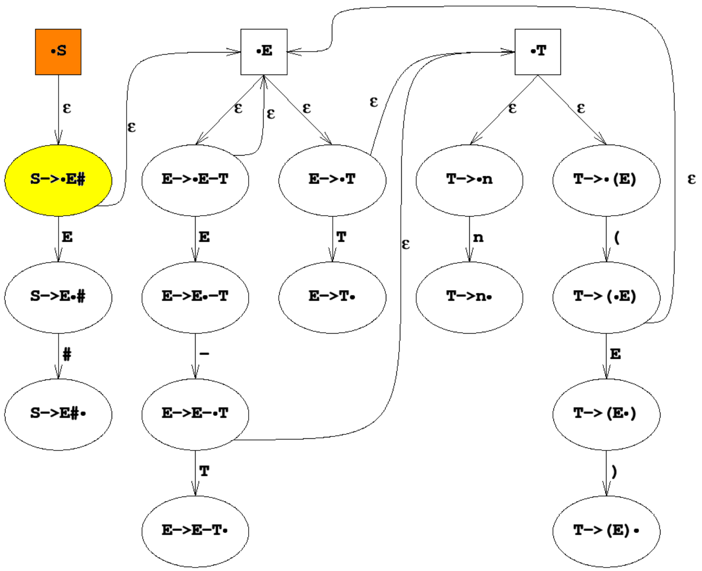
Using an LR Parser



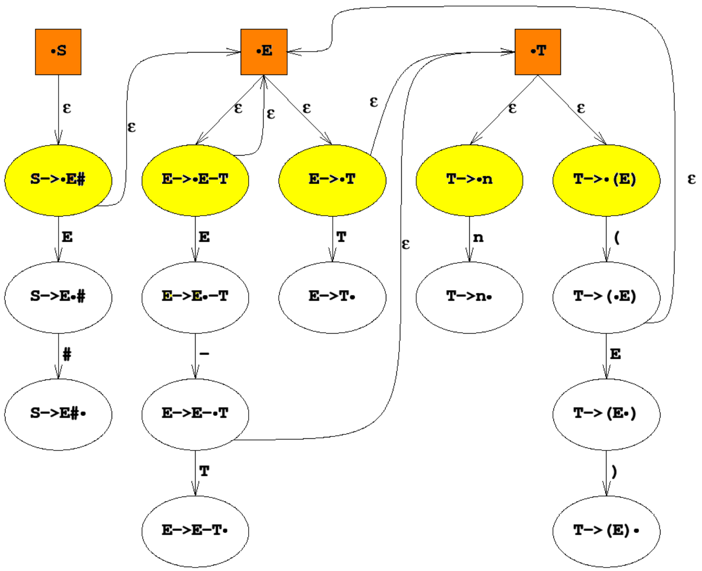
Using an LR Parser



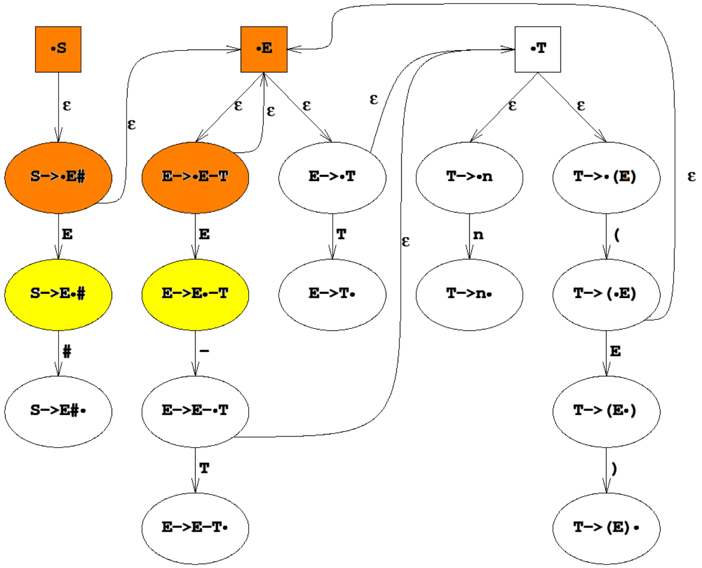
Using an LR Parser



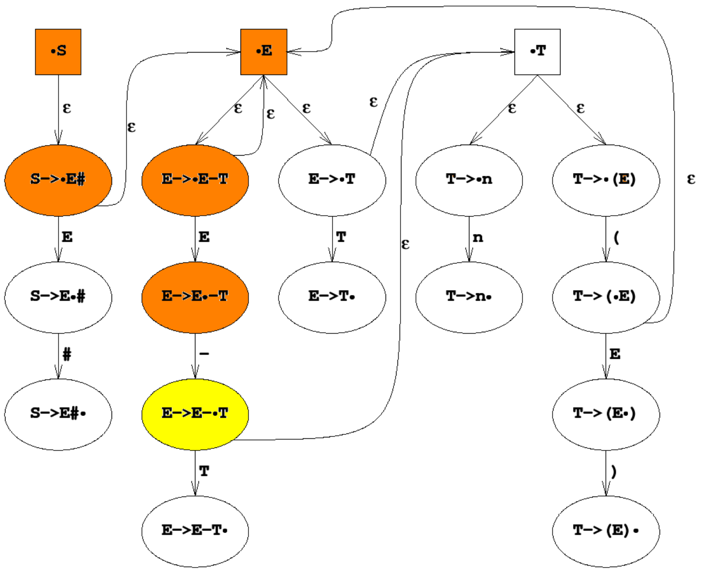
Using an LR Parser



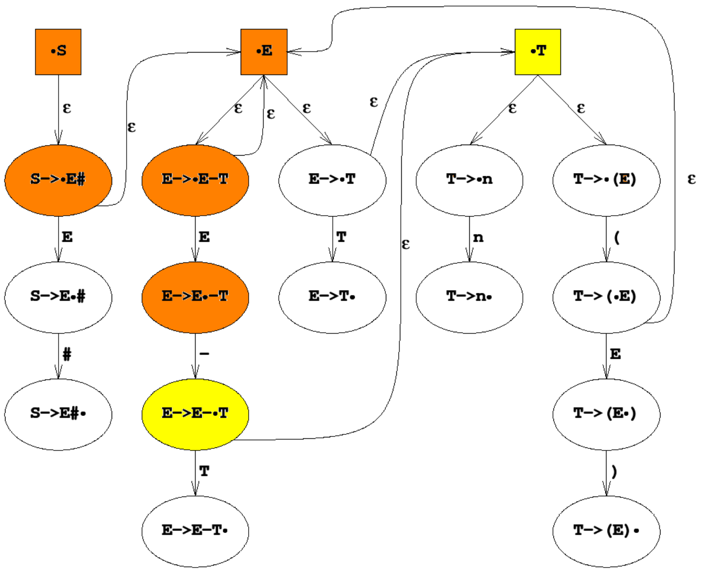
Using an LR Parser



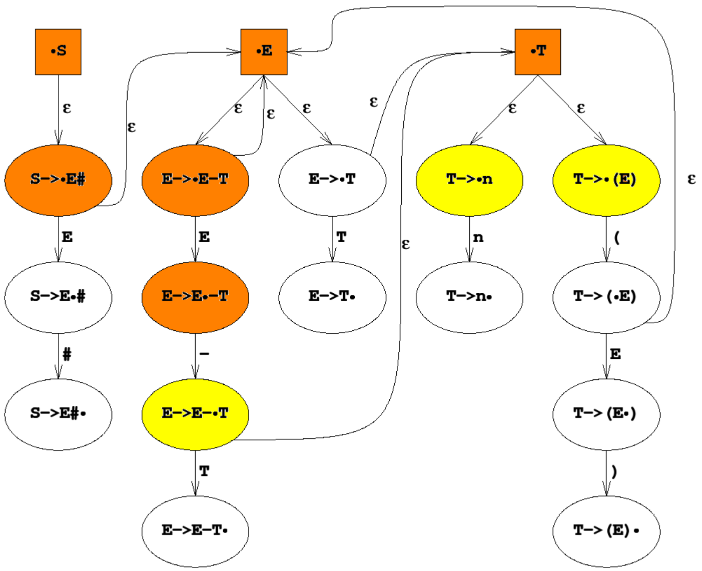
Using an LR Parser



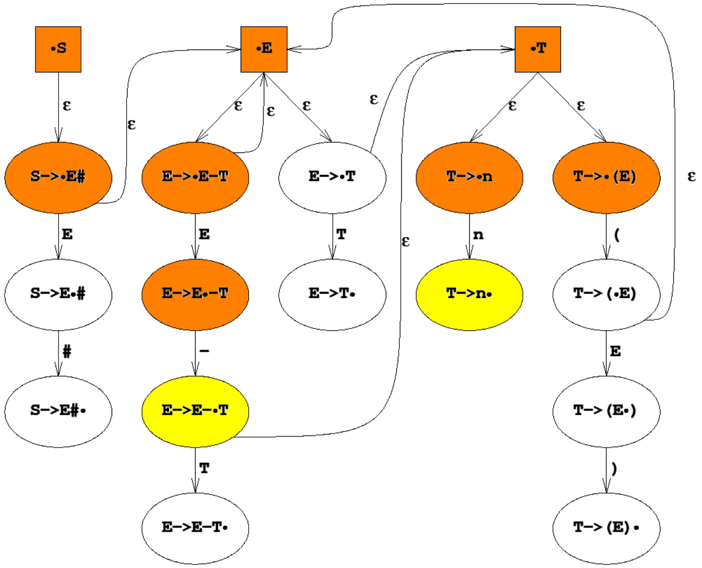
Using an LR Parser



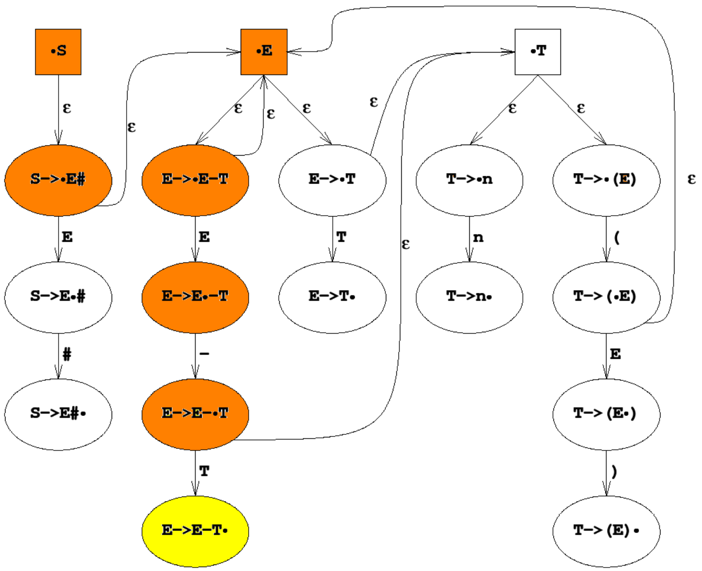
Using an LR Parser



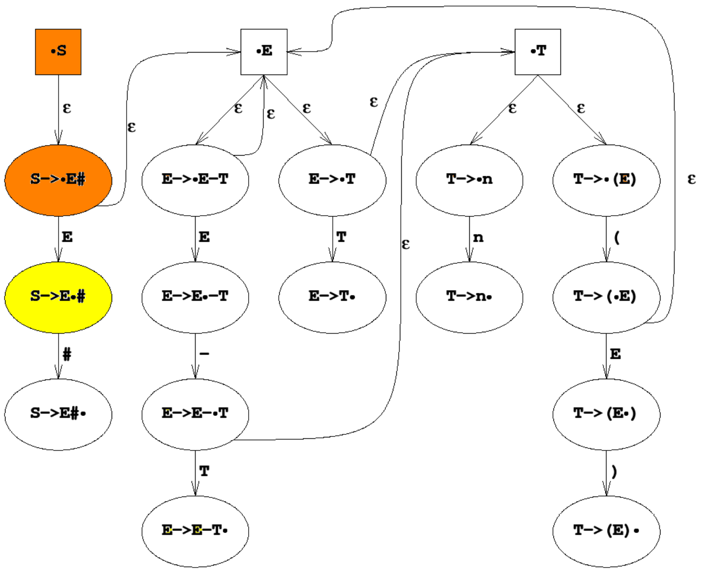
Using an LR Parser



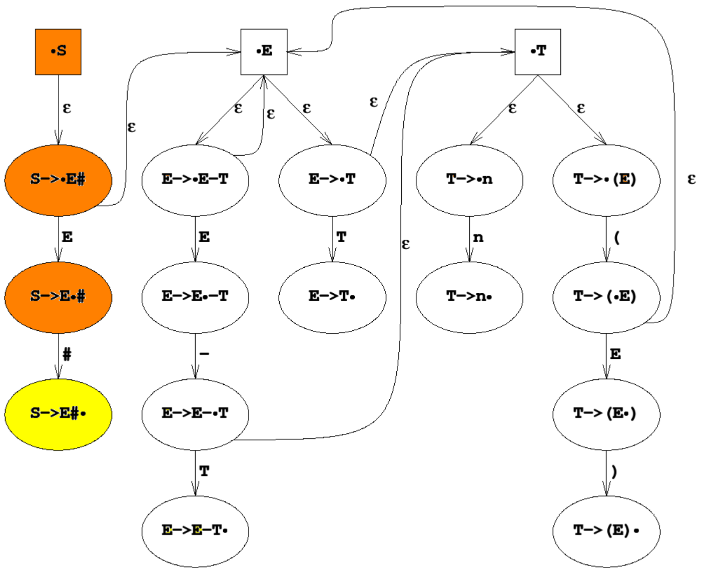
Using an LR Parser



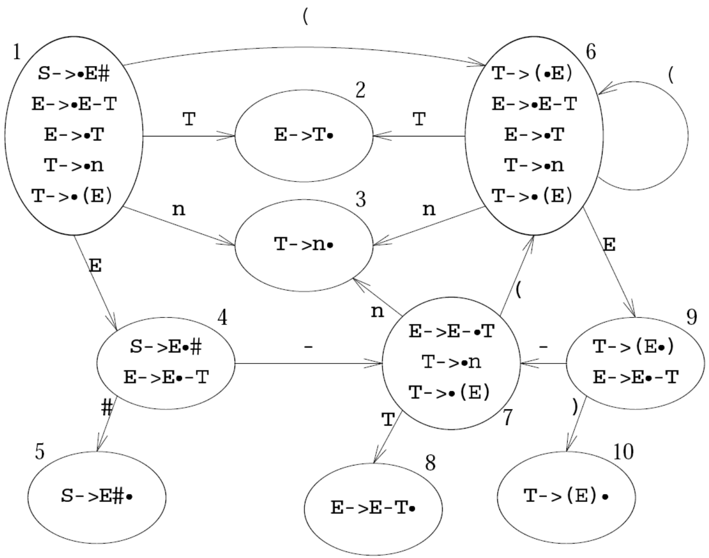
Using an LR Parser



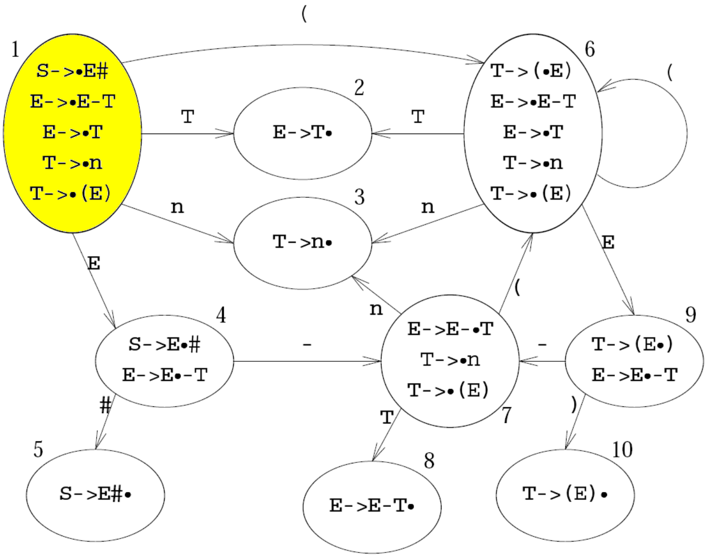
Using an LR Parser



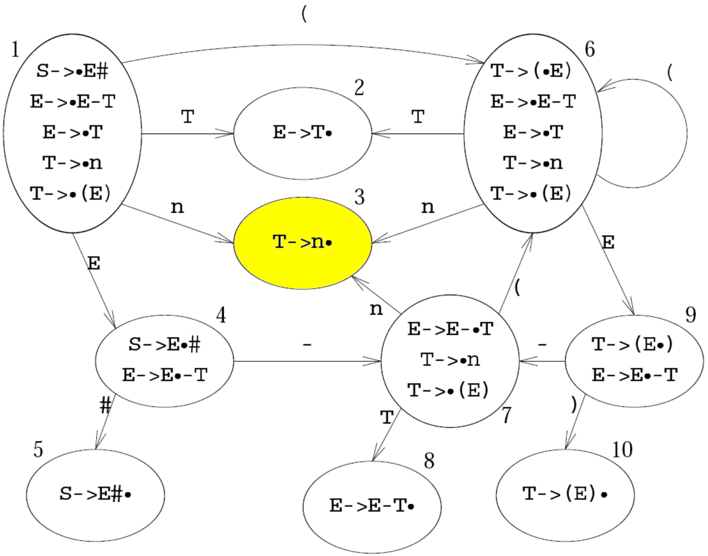
A deterministic handle recognizer



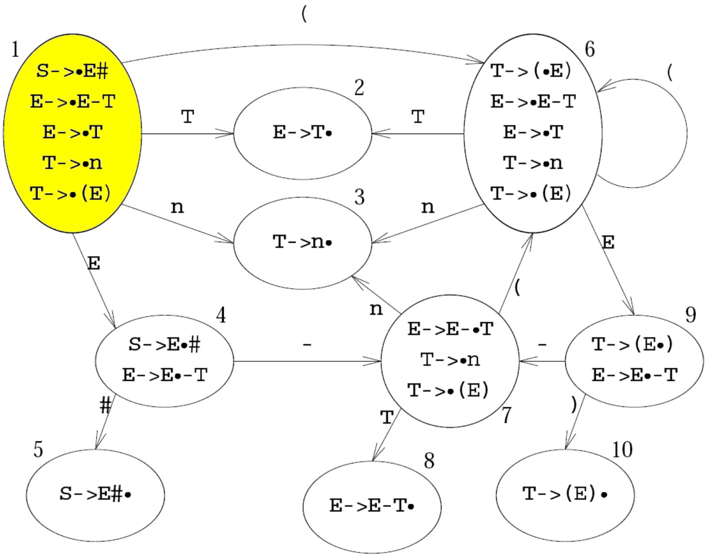
A deterministic handle recognizer



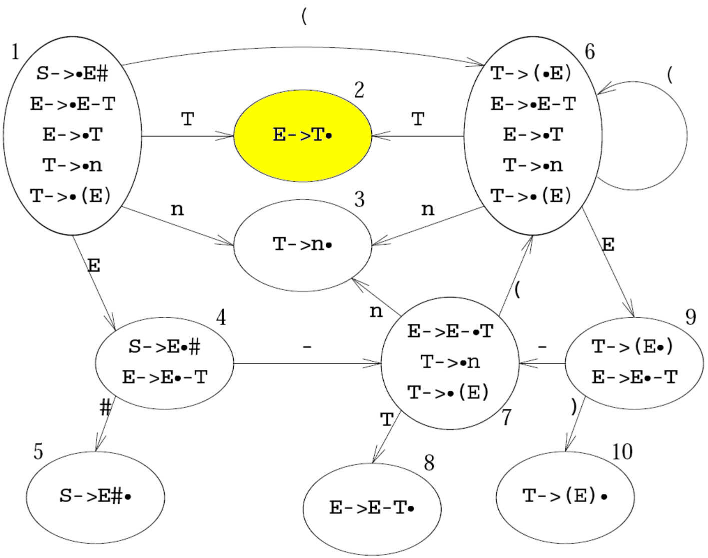
A deterministic handle recognizer



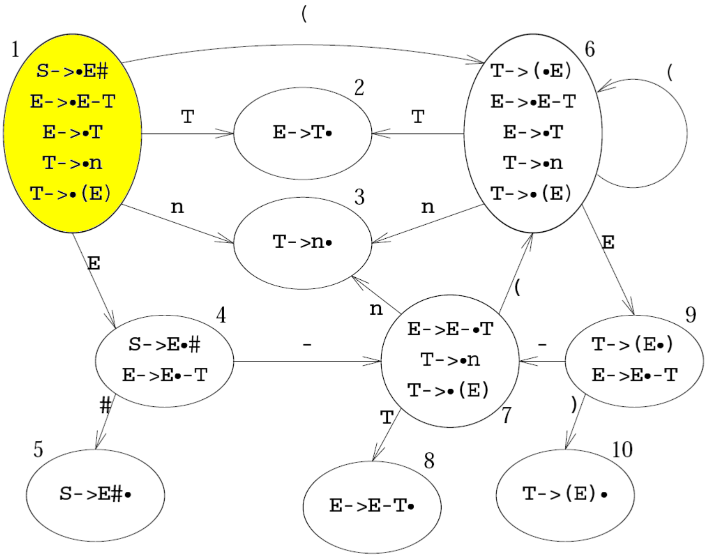
A deterministic handle recognizer



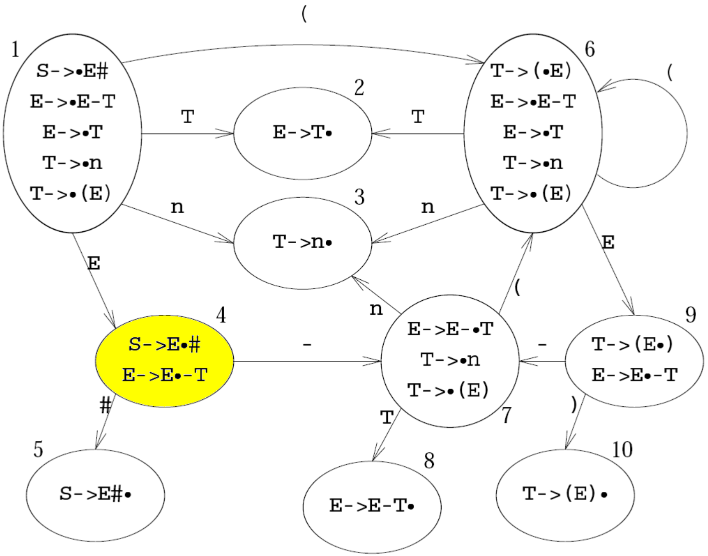
A deterministic handle recognizer



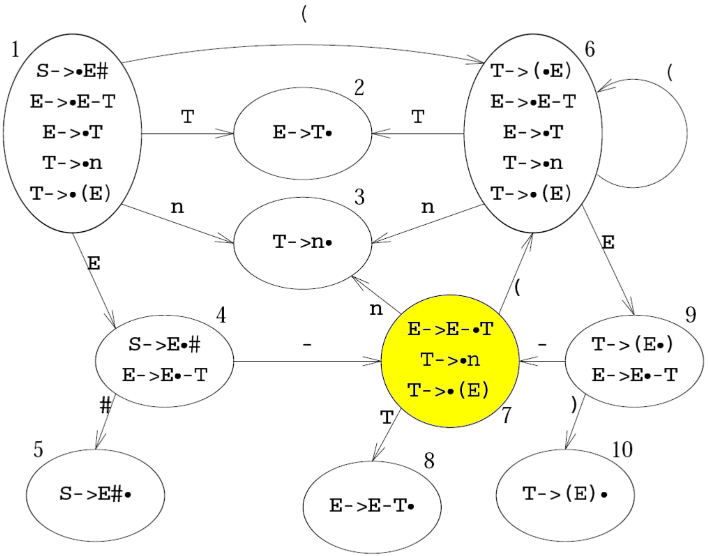
A deterministic handle recognizer



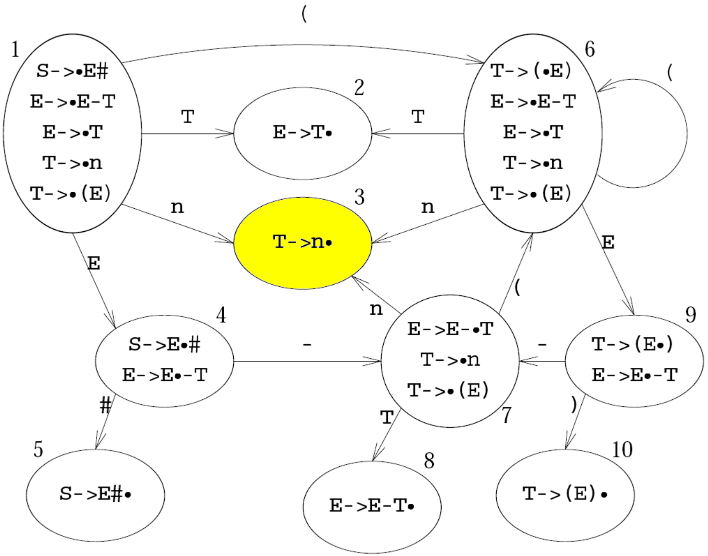
A deterministic handle recognizer



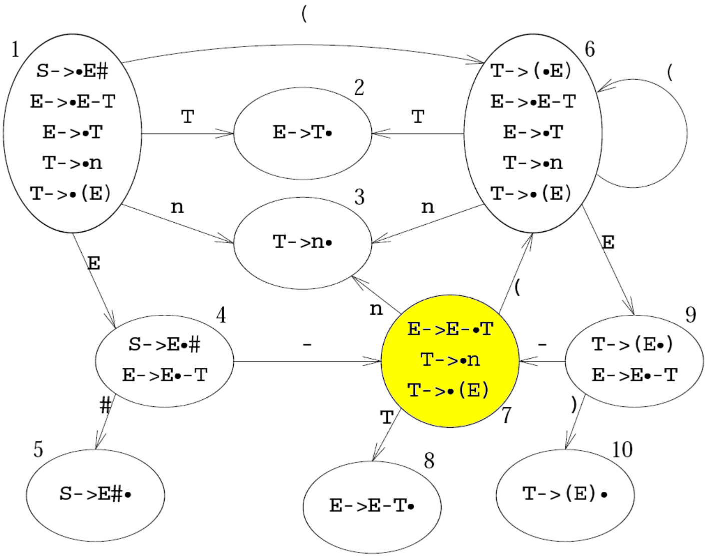
A deterministic handle recognizer



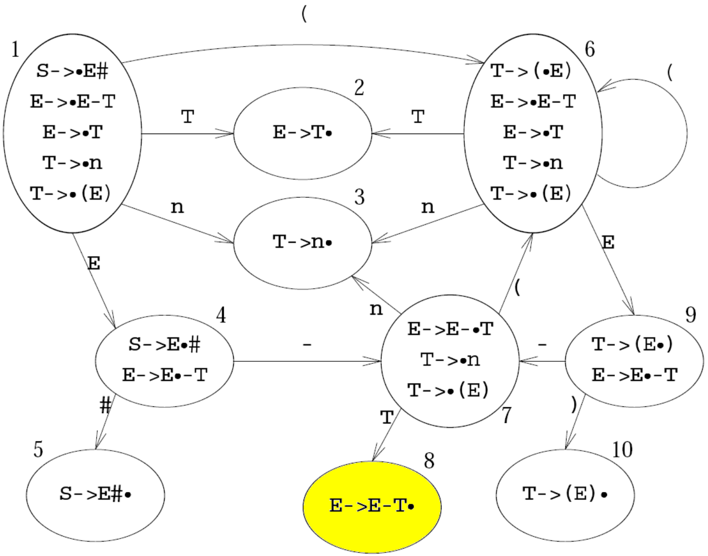
A deterministic handle recognizer



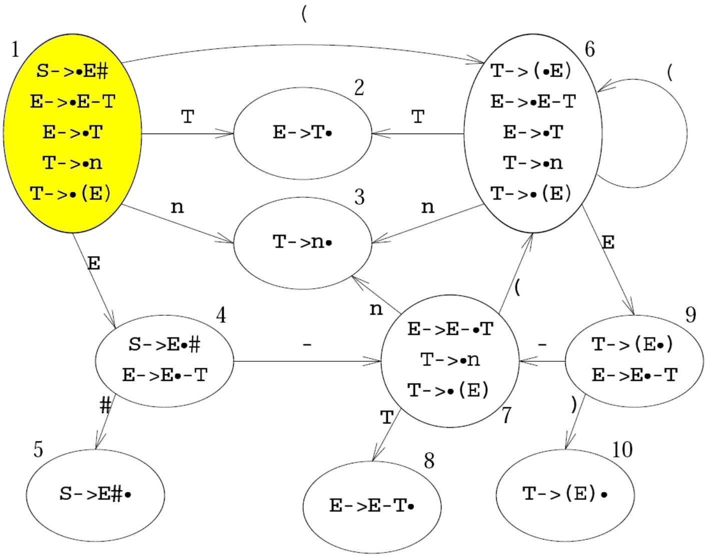
A deterministic handle recognizer



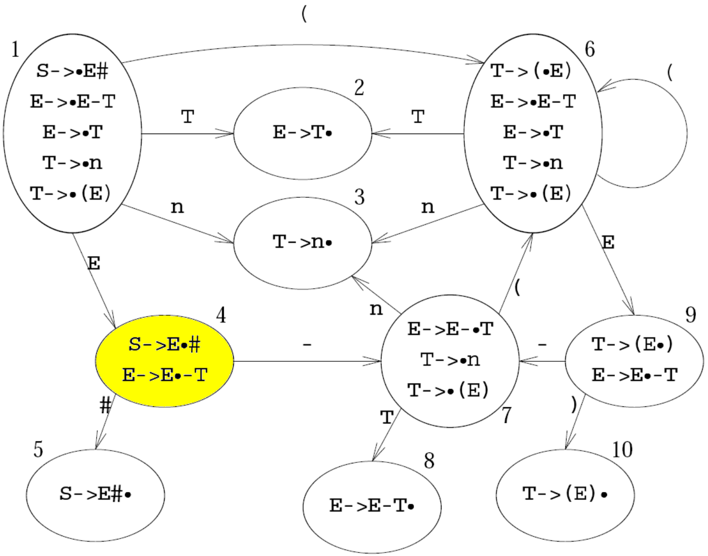
A deterministic handle recognizer



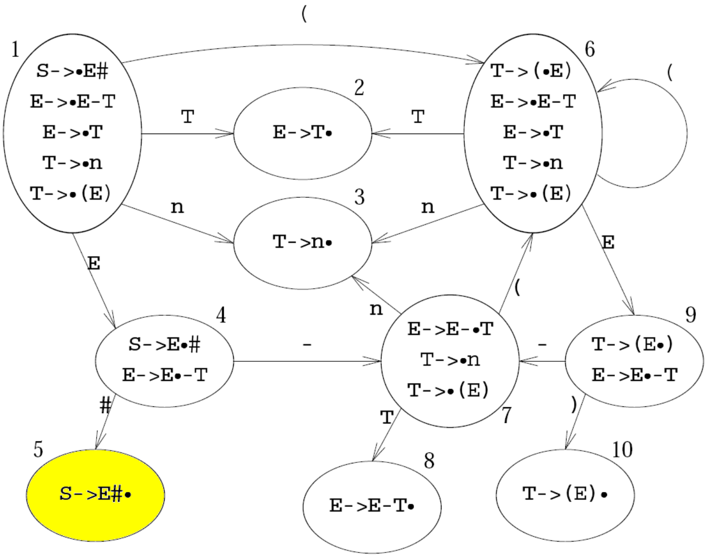
A deterministic handle recognizer



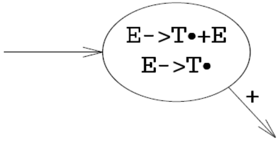
A deterministic handle recognizer



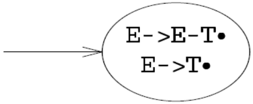
A deterministic handle recognizer



Conflicts caused by non-LR(0) grammars

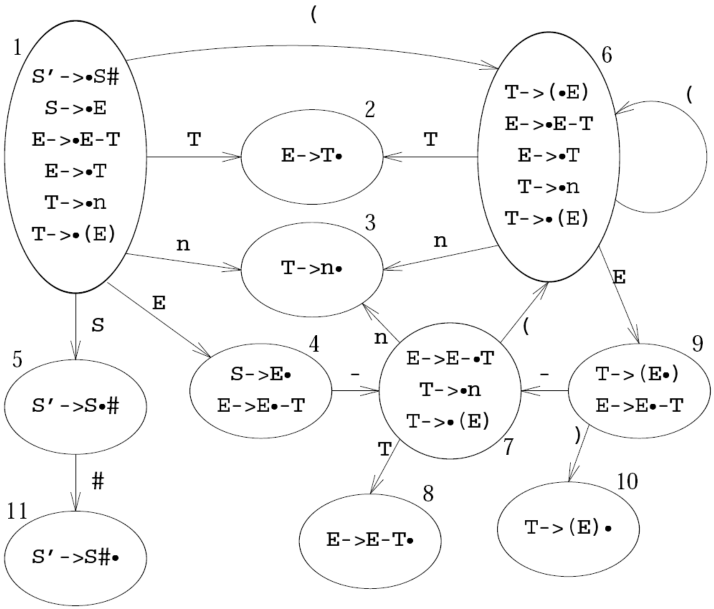


shift/reduce conflict
(on +)



reduce/reduce conflict
(always)

An inadequate handle recognizer



Lookahead

- The 0 in $LR(0)$ stands for 0 symbols lookahead.
- Grammars which are not $LR(0)$ may still be $LR(1)$, or $LR(k)$ with some higher k.
- LR parsers can be built for such grammars by introducing lookahead.
- Lookahead makes reductions depend on what follows the handle.

Lookahead

- The 0 in $LR(0)$ stands for 0 symbols lookahead.
- Grammars which are not $LR(0)$ may still be $LR(1)$, or $LR(k)$ with some higher k.
- LR parsers can be built for such grammars by introducing lookahead.
- Lookahead makes reductions depend on what follows the handle.

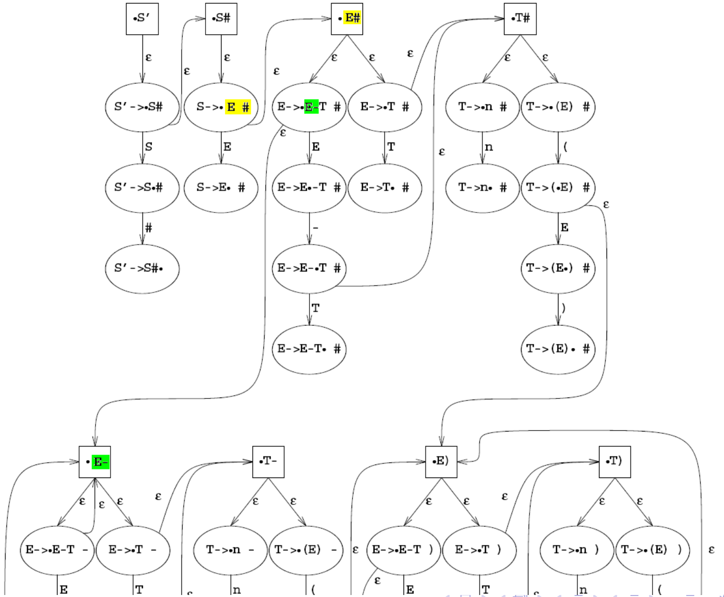
Lookahead

- The 0 in $LR(0)$ stands for 0 symbols lookahead.
- Grammars which are not $LR(0)$ may still be $LR(1)$, or $LR(k)$ with some higher k.
- LR parsers can be built for such grammars by introducing lookahead.
- Lookahead makes reductions depend on what follows the handle.

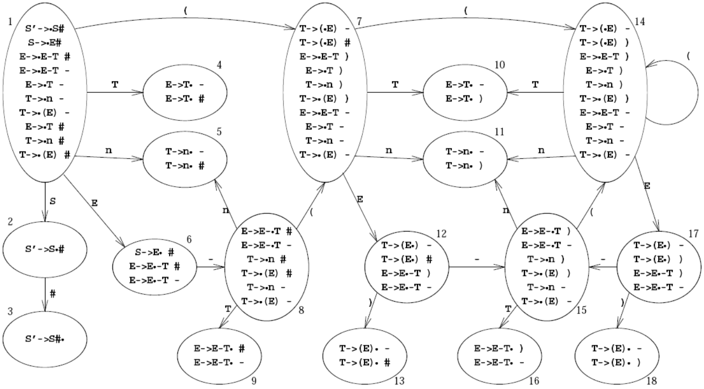
Lookahead

- The 0 in $LR(0)$ stands for 0 symbols lookahead.
- Grammars which are not $LR(0)$ may still be $LR(1)$, or $LR(k)$ with some higher k.
- LR parsers can be built for such grammars by introducing lookahead.
- Lookahead makes reductions depend on what follows the handle.

A non-deterministic handle recognizer with lookahead



A deterministic handle recognizer with lookahead



A table representation of our LR(1) parser

	n	-	(	)	#	S	E	T
1	s5	e	s7	e	e	s2	s6	s4
2	e	e	e	e	s3			
3/acc								
4	e	r4	e	e	r4			
5	e	r5	e	e	r5			
6	e	s8	e	e	r2			
7	s11	e	s14	e	e		s12	s10
8	s5	e	s7	e	e			s9
9	e	r3	e	e	r3			
10	e	r4	e	r4	e			
11	e	r5	e	r5	e			
12	e	s15	e	s13	e			
13	e	r6	e	e	r6			
14	s11	e	s14	e	e		s17	s10
15	s11	e	s14	e	e			s16
16	e	r3	e	r3	e			
17	e	s15	e	s18	e			
18	e	r6	e	r6	e			

Limitations of $LR(k)$ parsing

- Although an $LR(2)$ parser is more powerful than an $LR(1)$ in that it can handle some grammars that the other cannot, the emphasis is on *some*. (Grune + Jacobs)

Some more properties of $LR(k)$ parsing

- Any $LR(k)$ grammar with $k > 1$ can be transformed into an $LR(k-1)$ grammar, thus to $LR(1)$, but not always to $LR(0)$.
- A language that has an $LR(1)$ grammar is called *deterministic*.
- Many programming languages are $LR(1)$. Well-known exceptions are C++, Perl and Python.

Some more properties of $LR(k)$ parsing

- Any $LR(k)$ grammar with $k > 1$ can be transformed into an $LR(k-1)$ grammar, thus to $LR(1)$, but not always to $LR(0)$.
- A language that has an $LR(1)$ grammar is called *deterministic*.
- Many programming languages are $LR(1)$. Well-known exceptions are C++, Perl and Python.

Some more properties of $LR(k)$ parsing

- Any $LR(k)$ grammar with $k > 1$ can be transformed into an $LR(k-1)$ grammar, thus to $LR(1)$, but not always to $LR(0)$.
- A language that has an $LR(1)$ grammar is called *deterministic*.
- Many programming languages are $LR(1)$. Well-known exceptions are C++, Perl and Python.

Some more properties of $LR(k)$ parsing

- Any $LR(k)$ grammar with $k > 1$ can be transformed into an $LR(k-1)$ grammar, thus to $LR(1)$, but not always to $LR(0)$.
- A language that has an $LR(1)$ grammar is called *deterministic*.
- Many programming languages are $LR(1)$. Well-known exceptions are C++, Perl and Python.

Some more properties of $LR(k)$ parsing (cont.)

- An $LR(1)$ parser has the *immediate error detection property*: It will stop at the first incorrect token.
- The parsing tables of LR parsers with lookahead quickly assume gargantuan size. A way around that is LALR parsing, a very widely used variation of LR.

Some more properties of $LR(k)$ parsing (cont.)

- An $LR(1)$ parser has the *immediate error detection property*: It will stop at the first incorrect token.
- The parsing tables of LR parsers with lookahead quickly assume gargantuan size. A way around that is LALR parsing, a very widely used variation of LR.

Summary

- You have seen how to construct and use $LR(k)$ parsers, the strongest deterministic parsers possible and possibly the strongest linear-time parsers known.
- Key concepts to bear in mind: Handle recognition, items that call for specific reductions, use of nondeterministic and deterministic automata, lookahead

Summary

- You have seen how to construct and use $LR(k)$ parsers, the strongest deterministic parsers possible and possibly the strongest linear-time parsers known.
- Key concepts to bear in mind: Handle recognition, items that call for specific reductions, use of nondeterministic and deterministic automata, lookahead

Summary

- You have seen how to construct and use $LR(k)$ parsers, the strongest deterministic parsers possible and possibly the strongest linear-time parsers known.
- Key concepts to bear in mind: Handle recognition, items that call for specific reductions, use of nondeterministic and deterministic automata, lookahead

References

Dick Grune & Ceriel Jacobs (1990). *Parsing Techniques.* Amsterdam. pp. 200-213

Content, examples and all diagrams taken from there.