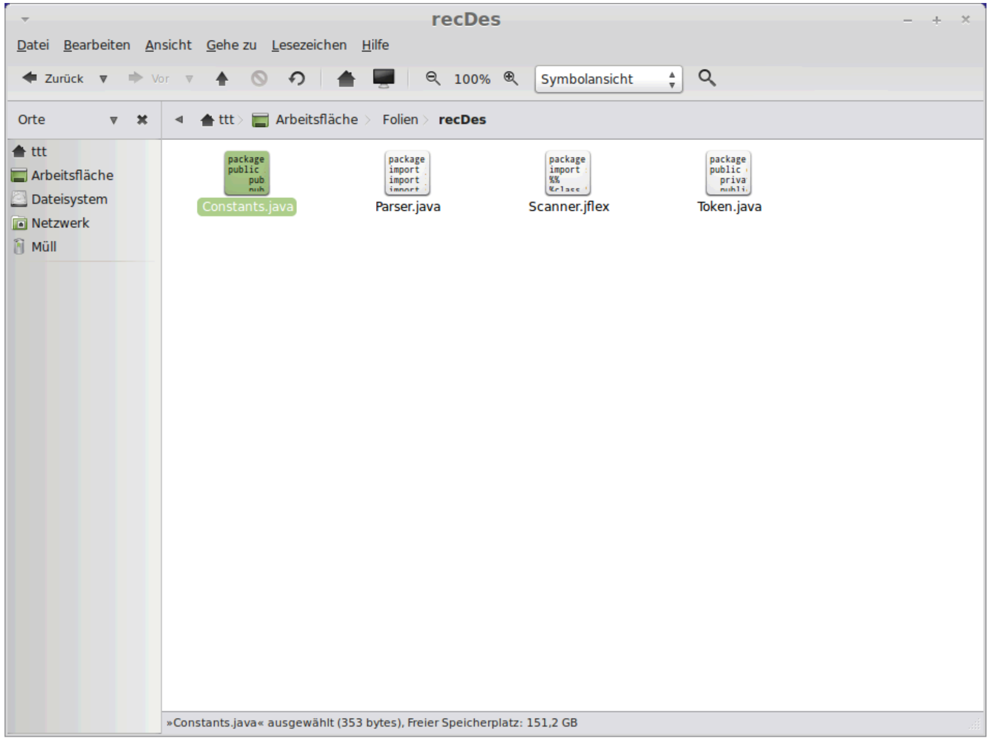
Script generated by TTT

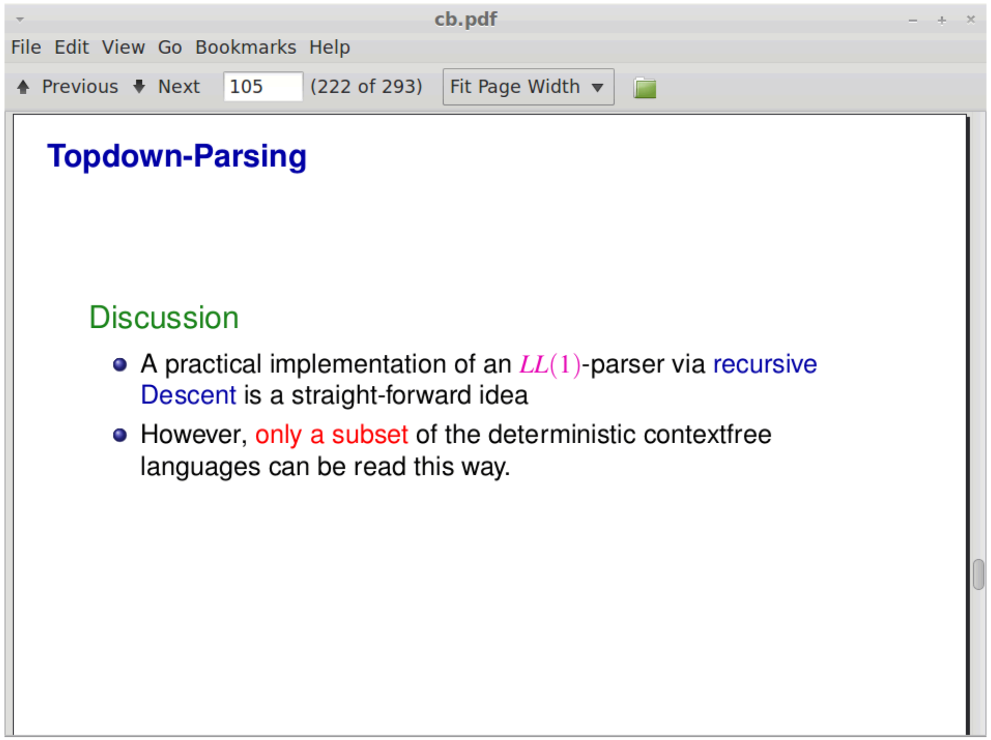
Title: Simon: Compilerbau (12.05.2014)

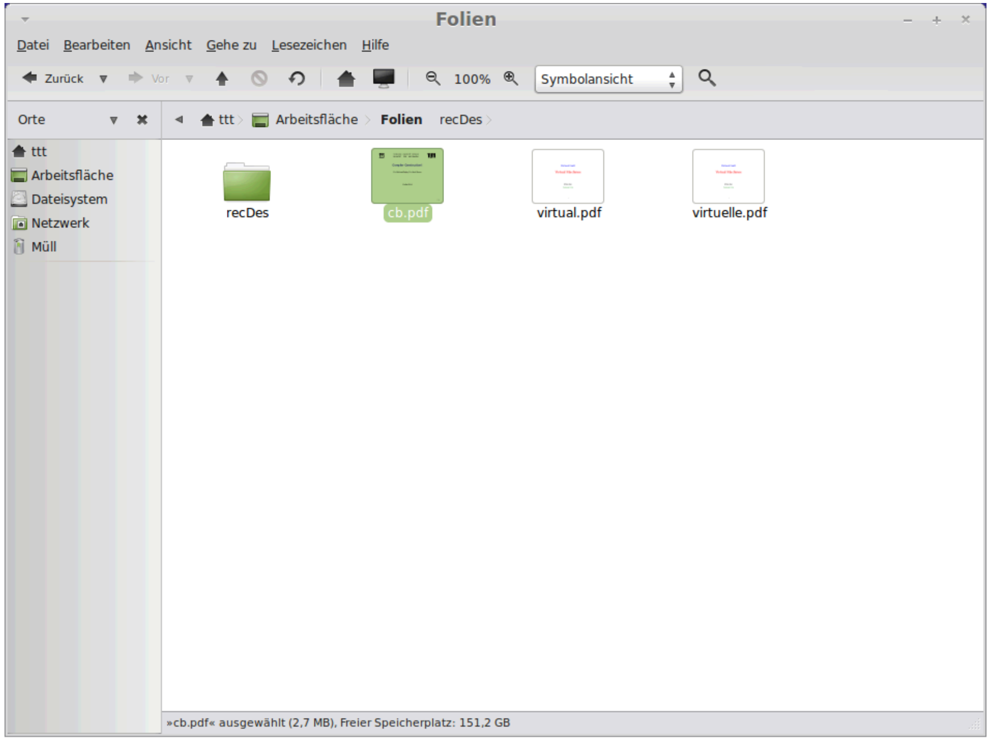
Date: Mon May 12 14:21:25 CEST 2014

Duration: 93:07 min

Pages: 67







Constants.java (~/Arbeitsfläche/Folien/recDes) - gedit

```java
package recDes;
public interface Constants {
    public static final int NULL = -1;
    public static final int PLUS = 0;
    public static final int MINUS = 1;
    public static final int MUL = 2;
    public static final int DIV = 3;
    public static final int LPAR = 4;
    public static final int RPAR = 5;
    public static final int NUMBER = 6;
}
```

Loading file '/home/ttt/Arbeitsfläche/Folien/recD... Java Tab Width: 8 Ln 1, Col 1 INS

Token.java (~/Arbeitsfläche/Folien/recDes) - gedit

```java
package recDes;
public class Token {
  private int   type;
  public  int   getType() { return type; }
  private int   value;
  public  int   getValue(){ return value; }

  public Token(int type,int value){
   this.type=type;
   this.value=value;
  }
  public Token(int type){ this(type,0); }
}
```

Loading file '/home/ttt/Arbeitsfläche/Folien/recD... Java Tab Width: 8 Ln 1, Col 1 INS

Terminal

```
ttt@tabletseidl6 ~/Arbeitsfläche/Folien/recDes $
```

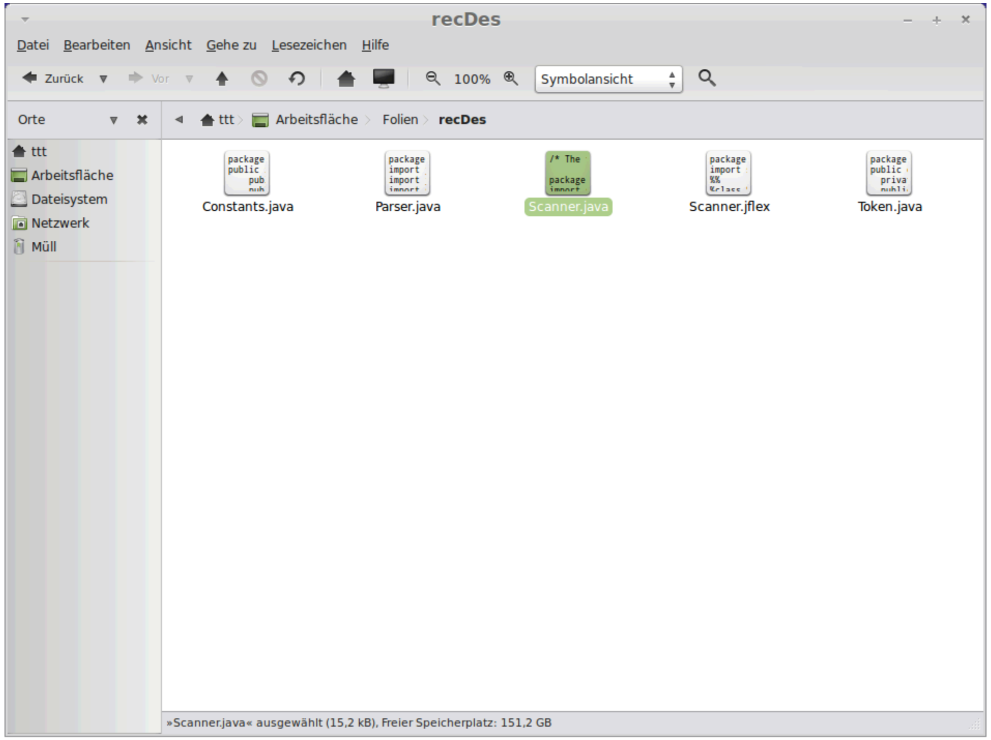
4 Objekte, freier Speicherplatz: 151,2 GB

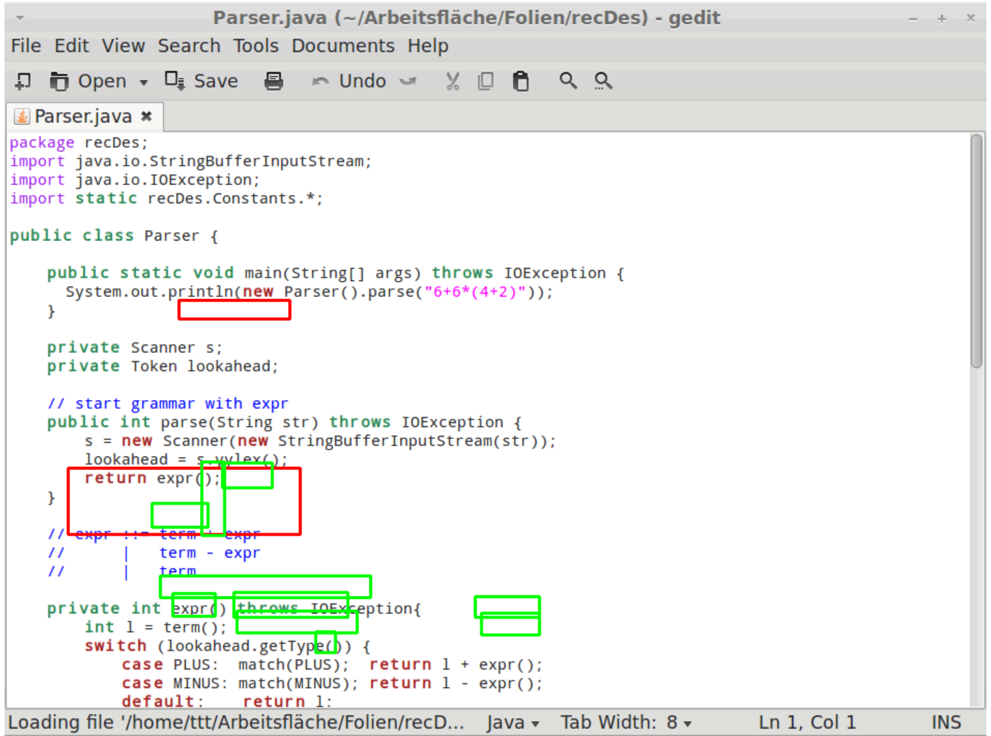
recDes

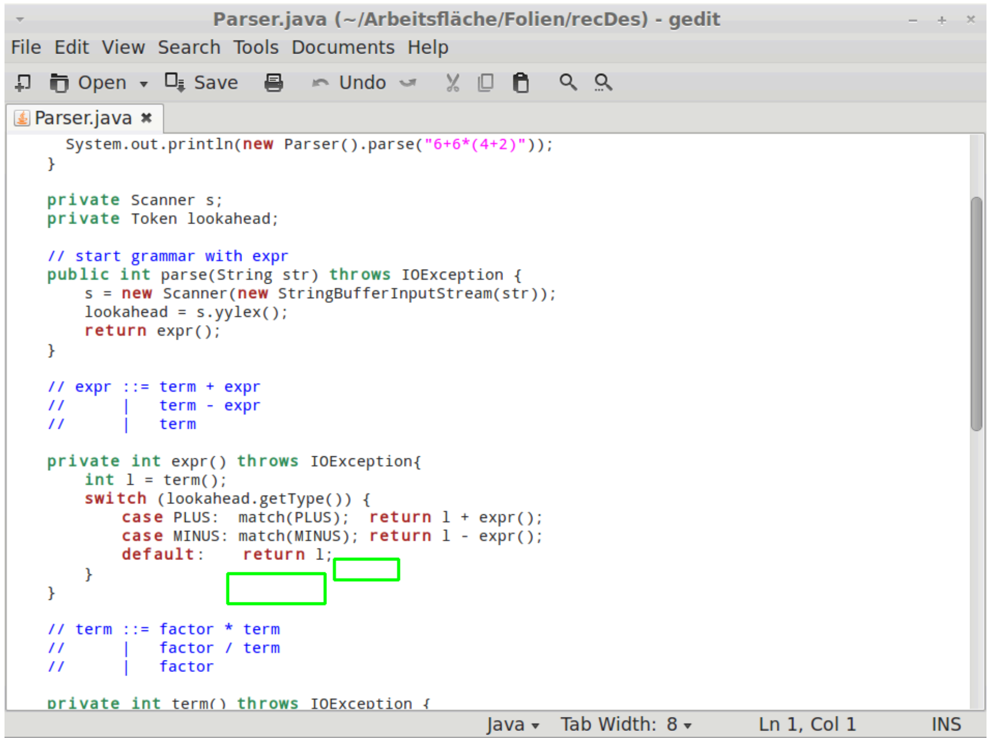
ttt › Arbeitsfläche › Folien › recDes

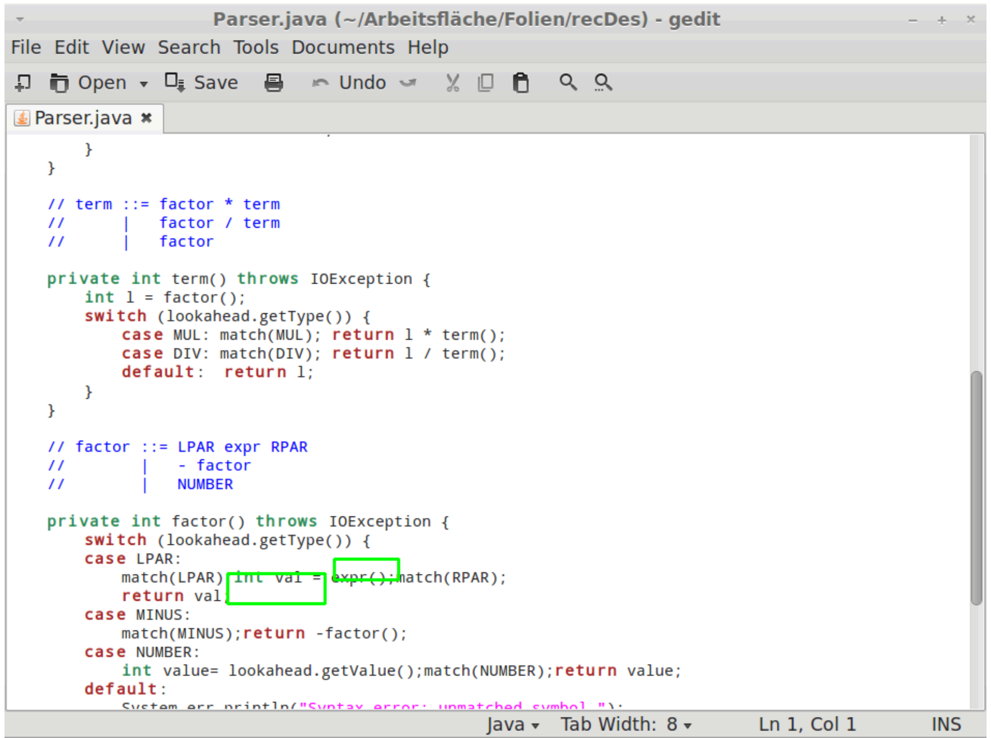
Constants.java Parser.java Scanner.java Scanner.jflex Token.java

5 Objekte, freier Speicherplatz: 151,2 GB


recDes
Datei Bearbeiten Ansicht Gehe zu Lesezeichen Hilfe
Zurück Vor 100% Symbolansicht
Orte
ttt
Arbeitsfläche
Dateisystem
Netzwerk
Müll
ttt Arbeitsfläche Folien recDes
Constants.java
Parser.java
Scanner.java
Scanner.jflex
Token.java
»Scanner.java« ausgewählt (15,2 kB), Freier Speicherplatz: 151,2 GB


Parser.java (~/Arbeitsfläche/Folien/recDes) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
Parser.java
package recDes;
import java.io.StringBufferInputStream;
import java.io.IOException;
import static recDes.Constants.*;
public class Parser {
public static void main(String[] args) throws IOException {
System.out.println(new Parser().parse("6+6*(4+2)"));
}
private Scanner s;
private Token lookahead;
// start grammar with expr
public int parse(String str) throws IOException {
s = new Scanner(new StringBufferInputStream(str));
lookahead = s.yylex();
return expr();
}
// expr ::= term + expr
// | term - expr
// | term
private int expr() throws IOException{
int l = term();
switch (lookahead.getType()) {
case PLUS: match(PLUS); return l + expr();
case MINUS: match(MINUS); return l - expr();
default: return l;
Loading file '/home/ttt/Arbeitsfläche/Folien/recD...
Java Tab Width: 8 Ln 1, Col 1 INS


Parser.java (~/Arbeitsfläche/Folien/recDes) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
Parser.java
System.out.println(new Parser().parse("6+6*(4+2)"));
}
private Scanner s;
private Token lookahead;
// start grammar with expr
public int parse(String str) throws IOException {
s = new Scanner(new StringBufferInputStream(str));
lookahead = s.yylex();
return expr();
}
// expr ::= term + expr
// | term - expr
// | term
private int expr() throws IOException{
int l = term();
switch (lookahead.getType()) {
case PLUS: match(PLUS); return l + expr();
case MINUS: match(MINUS); return l - expr();
default: return l;
}
}
// term ::= factor * term
// | factor / term
// | factor
private int term() throws IOException {
Java Tab Width: 8 Ln 1, Col 1 INS


Parser.java (~/Arbeitsfläche/Folien/recDes) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
Parser.java
}
}
// term ::= factor * term
// | factor / term
// | factor
private int term() throws IOException {
int l = factor();
switch (lookahead.getType()) {
case MUL: match(MUL); return l * term();
case DIV: match(DIV); return l / term();
default: return l;
}
}
// factor ::= LPAR expr RPAR
// | - factor
// | NUMBER
private int factor() throws IOException {
switch (lookahead.getType()) {
case LPAR:
match(LPAR); int val = expr(); match(RPAR);
return val;
case MINUS:
match(MINUS);return -factor();
case NUMBER:
int value= lookahead.getValue();match(NUMBER);return value;
default:
Java Tab Width: 8 Ln 1, Col 1 INS

Parser.java (~/Arbeitsfläche/Folien/recDes) - gedit

```java
        }
    }

    // factor ::= LPAR expr RPAR
    //          |   - factor
    //          |   NUMBER

    private int factor() throws IOException {
        switch (lookahead.getType()) {
        case LPAR:
            match(LPAR);int val = expr();match(RPAR);
            return val;
        case MINUS:
            match(MINUS);return -factor();
        case NUMBER:
            int value= lookahead.getValue();match(NUMBER);return value;
        default:
            System.err.println("Syntax error: unmatched symbol ");
            return 0;
        }
    }

    private void match(int type) throws IOException {
        if (lookahead.getType() == type)
            lookahead = s.yylex();
        else {
            System.err.println("Syntax error: unexpected Symbol ");
            System.exit(-1);
        }
    }
}
```

Java Tab Width: 8 Ln 9, Col 54 INS

Parser.java (~/Arbeitsfläche/Folien/recDes) - gedit

```java
            case DIV: match(DIV); return l / term();
            default:  return l;
        }
    }

    // factor ::= LPAR expr RPAR
    //          |   - factor
    //          |   NUMBER

    private int factor() throws IOException {
        switch (lookahead.getType()) {
        case LPAR:
            match(LPAR);int val = expr();match(RPAR);
            return val;
        case MINUS:
            match(MINUS);return -factor();
        case NUMBER:
            int value= lookahead.getValue();match(NUMBER);return value;
        default:
            System.err.println("Syntax error: unmatched symbol ");
            return 0;
        }
    }

    private void match(int type) throws IOException {
        if (lookahead.getType() == type)
            lookahead = s.yylex();
        else {
            System.err.println("Syntax error: unexpected Symbol ");
            System.exit(-1);
```

Java Tab Width: 8 Ln 59, Col 20 INS

Parser.java (~/Arbeitsfläche/Folien/recDes) - gedit

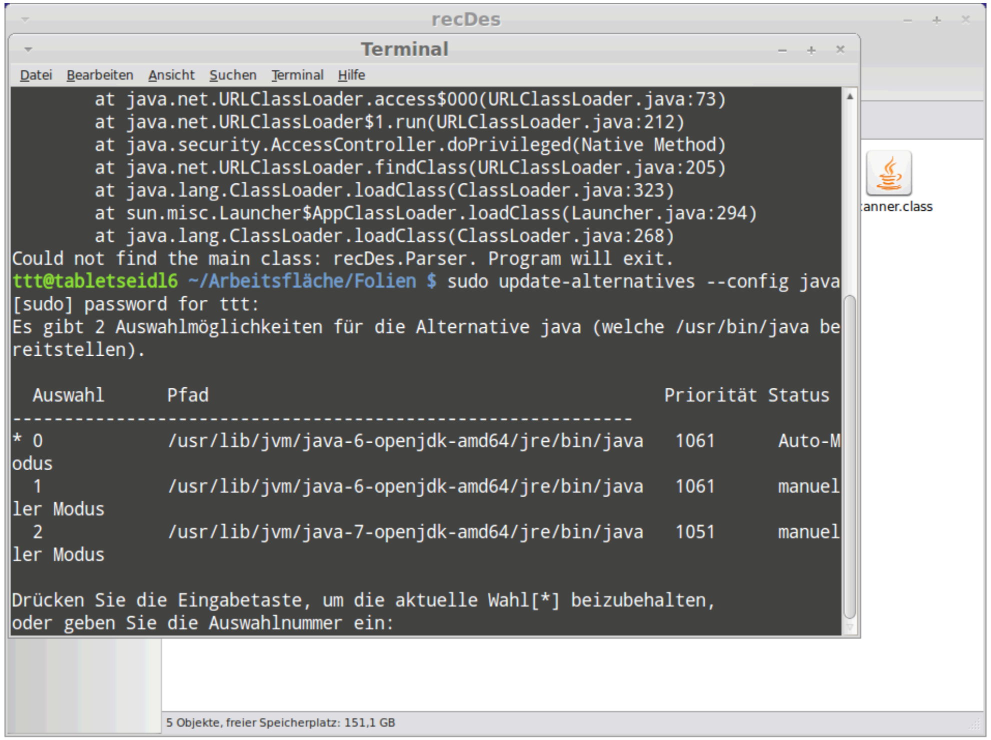
```java
import java.io.StringBufferInputStream;
import java.io.IOException;
import static recDes.Constants.*;

public class Parser {

    public static void main(String[] args) throws IOException {
      System.out.println(new Parser().parse("6+6*(4+2)"));
    }

    private Scanner s;
    private Token lookahead;

    // start grammar with expr
    public int parse(String str) throws IOException {
        s = new Scanner(new StringBufferInputStream(str));
        lookahead = s.yylex();
        return expr();
    }

    // expr ::= term + expr
    //       |   term - expr
    //       |   term

    private int expr() throws IOException{
        int l = term();
        switch (lookahead.getType()) {
            case PLUS:  match(PLUS);  return l + expr();
            case MINUS: match(MINUS); return l - expr();
            default:    return l;
```

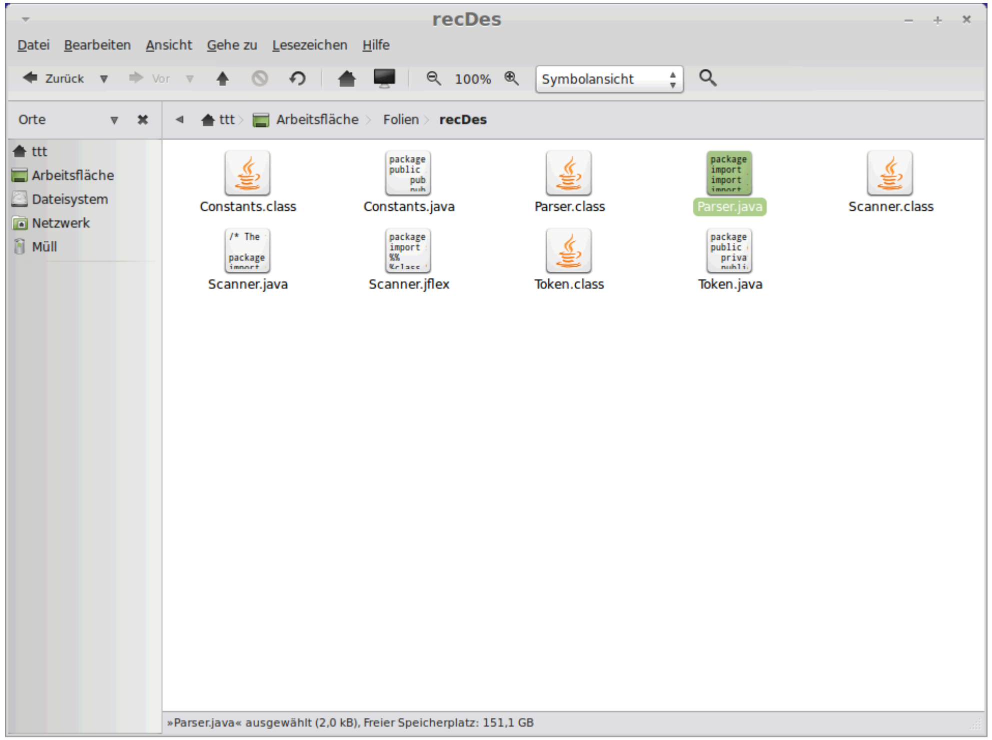
Java Tab Width: 8 Ln 42, Col 45 INS

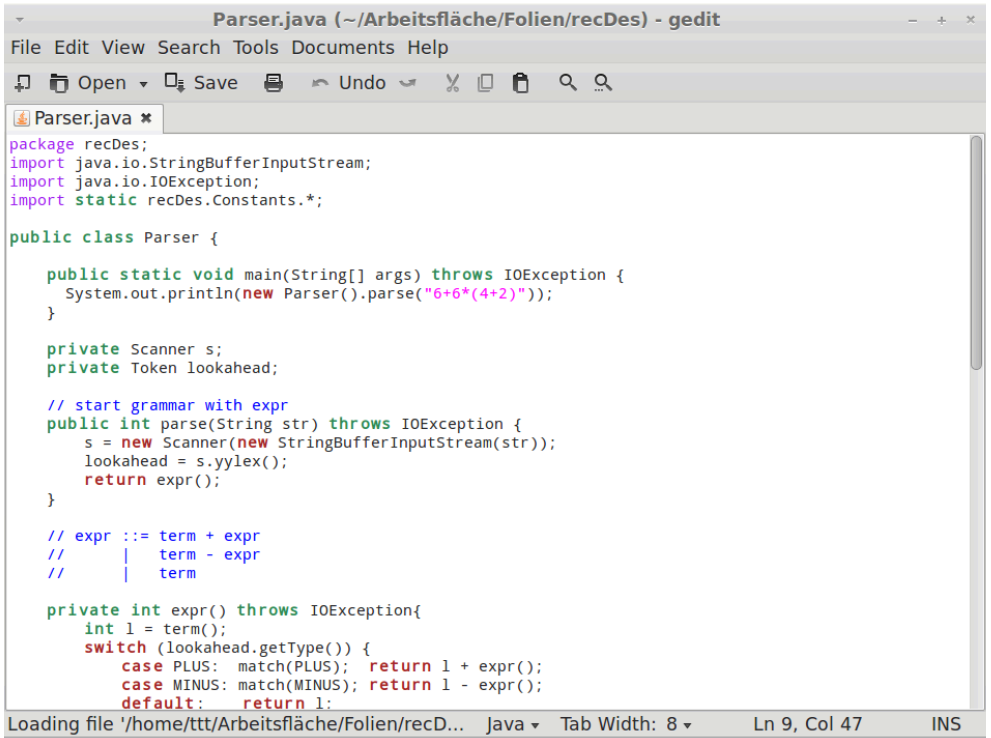
recDes — Terminal

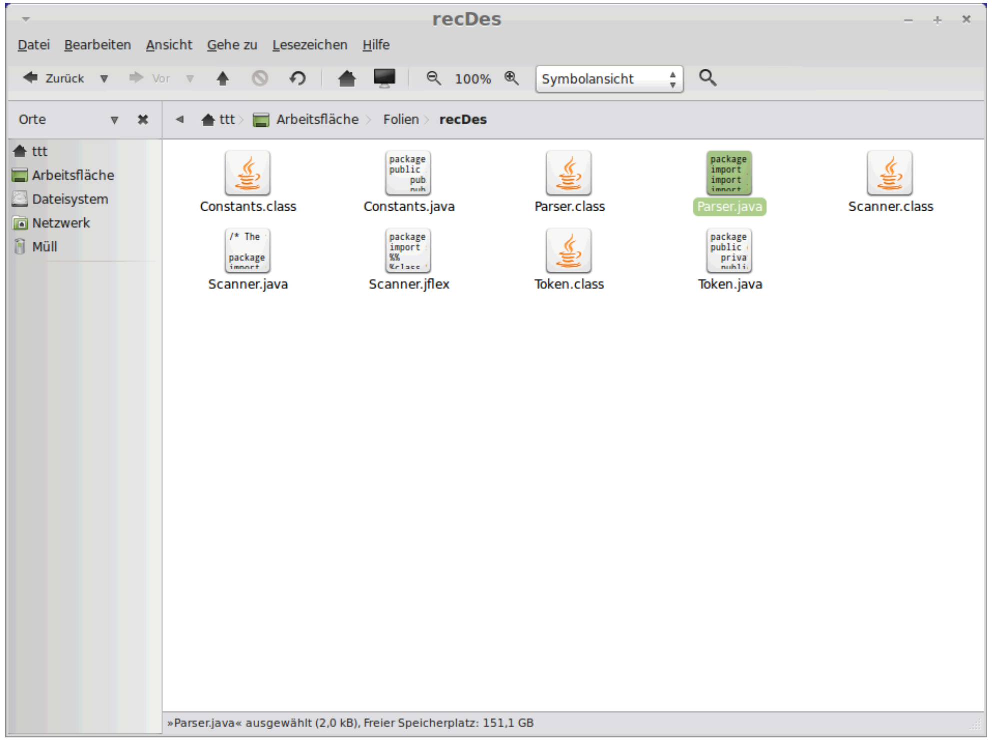
```
ttt@tabletseidl6 ~/Arbeitsfläche/Folien/recDes $ ..
..: command not found
ttt@tabletseidl6 ~/Arbeitsfläche/Folien/recDes $ cd ..
ttt@tabletseidl6 ~/Arbeitsfläche/Folien $ javac recDes/*.java
Note: recDes/Parser.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
ttt@tabletseidl6 ~/Arbeitsfläche/Folien $ java recDes.Parser
Exception in thread "main" java.lang.UnsupportedClassVersionError: recDes/Parser
 : Unsupported major.minor version 51.0
        at java.lang.ClassLoader.defineClass1(Native Method)
        at java.lang.ClassLoader.defineClass(ClassLoader.java:643)
        at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:14
2)
        at java.net.URLClassLoader.defineClass(URLClassLoader.java:277)
        at java.net.URLClassLoader.access$000(URLClassLoader.java:73)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:212)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:205)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:323)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:294)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:268)
Could not find the main class: recDes.Parser. Program will exit.
ttt@tabletseidl6 ~/Arbeitsfläche/Folien $
```

5 Objekte, freier Speicherplatz: 151,1 GB


recDes
Terminal
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
at java.net.URLClassLoader.access$000(URLClassLoader.java:73)
at java.net.URLClassLoader$1.run(URLClassLoader.java:212)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.URLClassLoader.findClass(URLClassLoader.java:205)
at java.lang.ClassLoader.loadClass(ClassLoader.java:323)
at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:294)
at java.lang.ClassLoader.loadClass(ClassLoader.java:268)
Could not find the main class: recDes.Parser. Program will exit.
ttt@tabletseidl6 ~/Arbeitsfläche/Folien $ sudo update-alternatives --config java
[sudo] password for ttt:
Es gibt 2 Auswahlmöglichkeiten für die Alternative java (welche /usr/bin/java be
reitstellen).
Auswahl Pfad Priorität Status
----
* 0 /usr/lib/jvm/java-6-openjdk-amd64/jre/bin/java 1061 Auto-M
odus
1 /usr/lib/jvm/java-6-openjdk-amd64/jre/bin/java 1061 manuel
ler Modus
2 /usr/lib/jvm/java-7-openjdk-amd64/jre/bin/java 1051 manuel
ler Modus
Drücken Sie die Eingabetaste, um die aktuelle Wahl[*] beizubehalten,
oder geben Sie die Auswahlnummer ein:
anner.class
5 Objekte, freier Speicherplatz: 151,1 GB


recDes
Datei Bearbeiten Ansicht Gehe zu Lesezeichen Hilfe
Zurück Vor 100% Symbolansicht
Orte
ttt Arbeitsfläche Folien recDes
ttt
Arbeitsfläche
Dateisystem
Netzwerk
Müll
Constants.class
Constants.java
Parser.class
Parser.java
Scanner.class
Scanner.java
Scanner.jflex
Token.class
Token.java
»Parser.java« ausgewählt (2,0 kB), Freier Speicherplatz: 151,1 GB


Parser.java (~/Arbeitsfläche/Folien/recDes) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
Parser.java
package recDes;
import java.io.StringBufferInputStream;
import java.io.IOException;
import static recDes.Constants.*;
public class Parser {
public static void main(String[] args) throws IOException {
System.out.println(new Parser().parse("6+6*(4+2)"));
}
private Scanner s;
private Token lookahead;
// start grammar with expr
public int parse(String str) throws IOException {
s = new Scanner(new StringBufferInputStream(str));
lookahead = s.yylex();
return expr();
}
// expr ::= term + expr
// | term - expr
// | term
private int expr() throws IOException{
int l = term();
switch (lookahead.getType()) {
case PLUS: match(PLUS); return l + expr();
case MINUS: match(MINUS); return l - expr();
default: return l;
Loading file '/home/ttt/Arbeitsfläche/Folien/recD... Java Tab Width: 8 Ln 9, Col 47 INS


recDes
Datei Bearbeiten Ansicht Gehe zu Lesezeichen Hilfe
Zurück Vor 100% Symbolansicht
Orte
ttt Arbeitsfläche Folien recDes
ttt
Arbeitsfläche
Dateisystem
Netzwerk
Müll
Constants.class
Constants.java
Parser.class
Parser.java
Scanner.class
Scanner.java
Scanner.jflex
Token.class
Token.java
»Parser.java« ausgewählt (2,0 kB), Freier Speicherplatz: 151,1 GB

Bottom-up Analysis

Attention:
Many grammars are not $LL(k)$!

A reason for that is:

Definition
Grammar G is called left-recursive, if

$$A \to^+ A\,\beta \qquad \text{for an} \quad A \in N,\ \beta \in (T \cup N)^*$$

Bottom-up Analysis

Attention:
Many grammars are not $LL(k)$!

A reason for that is:

Definition
Grammar G is called left-recursive, if

$$A \to^+ A\,\beta \qquad \text{for an} \quad A \in N,\ \beta \in (T \cup N)^*$$

Example:

$$\begin{array}{lcllll} E & \to & E + T & \mid & T & \\ T & \to & T * F & \mid & F & \\ F & \to & (\,E\,) & \mid & \text{name} & \mid \ \text{int} \end{array}$$

... is left-recursive

Bottom-up Analysis

Theorem:
Let a grammar G be reduced and left-recursive, then G is not $LL(k)$ for any k.

Proof:

Let $A \to A\,\beta \mid \alpha \quad \in P$
and A be reachable from S

Assumption: G is $LL(k)$

Bottom-up Analysis

Theorem:
Let a grammar G be reduced and left-recursive, then G is not $LL(k)$ for any k.

Proof:

Let $A \to A\,\beta \mid \alpha \quad \in P$
and A be reachable from S

Assumption: G is $LL(k)$

$\Rightarrow \text{First}_k(\alpha\,\beta^n\,\gamma)\ \cap$
$\quad \text{First}_k(\alpha\,\beta^{n+1}\,\gamma) = \emptyset$

Bottom-up Analysis

Theorem:

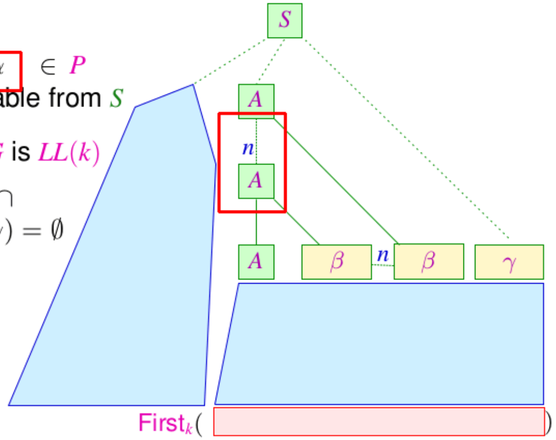
Let a grammar G be reduced and left-recursive, then G is not $LL(k)$ for any k.

Proof:

Let $A \to A\,\beta \mid \alpha \quad \in P$
and A be reachable from S

Assumption: G is $LL(k)$

$\Rightarrow \text{First}_k(\alpha\,\beta^n\,\gamma)\ \cap$
$\text{First}_k(\alpha\,\beta^{n+1}\,\gamma) = \emptyset$



Bottom-up Analysis

Theorem:

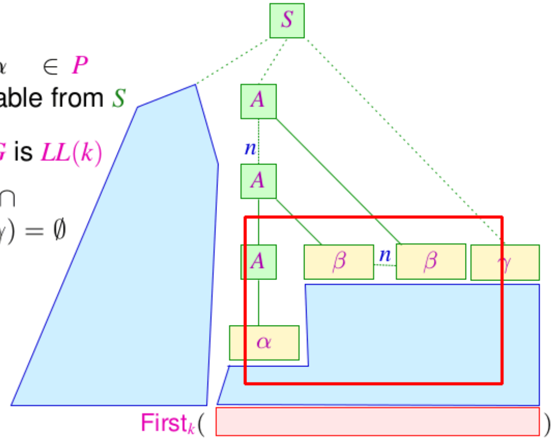
Let a grammar G be reduced and left-recursive, then G is not $LL(k)$ for any k.

Proof:

Let $A \to A\,\beta \mid \alpha \quad \in P$
and A be reachable from S

Assumption: G is $LL(k)$

$\Rightarrow \text{First}_k(\alpha\,\beta^n\,\gamma)\ \cap$
$\text{First}_k(\alpha\,\beta^{n+1}\,\gamma) = \emptyset$



Bottom-up Analysis

Theorem:

Let a grammar G be reduced and left-recursive, then G is not $LL(k)$ for any k.

Proof:

Let $A \to A\,\beta \mid \alpha \quad \in P$
and A be reachable from S

Assumption: G is $LL(k)$

$\Rightarrow \text{First}_k(\alpha\,\beta^n\,\gamma)\ \cap$
$\text{First}_k(\alpha\,\beta^{n+1}\,\gamma) = \emptyset$

Case 1: $\beta \to^* \epsilon$ — Contradiction !!!
Case 2: $\beta \to^* w \neq \epsilon \implies \text{First}_k(\alpha\,\beta^k\,\gamma)\ \cap\ \text{First}_k(\alpha\,\beta^{k+1}\,\gamma) \neq \emptyset$

Bottom-up Analysis

Theorem:

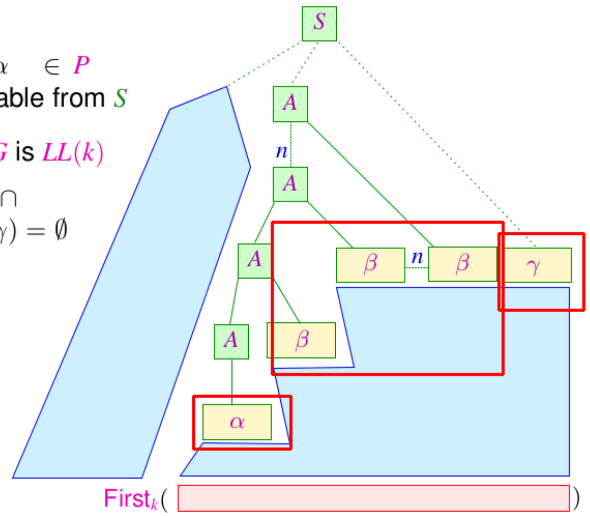
Let a grammar G be reduced and left-recursive, then G is not $LL(k)$ for any k.

Proof:

Let $A \to A\,\beta \mid \alpha \quad \in P$
and A be reachable from S

Assumption: G is $LL(k)$

$\Rightarrow$ First$_k$(α [redacted] γ) $\cap$
First$_k$(α [redacted] γ) $= \emptyset$



Bottom-up Analysis

Theorem:

Let a grammar G be reduced and left-recursive, then G is not $LL(k)$ for any k.

Proof:

Let $A \to A\,\beta \mid \alpha \quad \in P$
and A be reachable from S

Assumption: G is $LL(k)$

$\Rightarrow \text{First}_k(\alpha\,\beta^n\,\gamma) \cap$
$\text{First}_k(\alpha\,\beta^{n+1}\,\gamma) = \emptyset$

Case 1: $\beta \to^* \epsilon$ — Contradiction !!!
Case 2: $\beta \to^* w \neq \epsilon \implies \text{First}_k(\alpha\,\beta^k\,\gamma) \cap \text{First}_k(\alpha\,\beta^{k+1}\,\gamma) \neq \emptyset$

Shift-Reduce Parser

Donald Knuth

Idea:

We *delay* the decision whether to reduce until we know, whether the input matches the right-hand-side of a rule!

Construction: Shift-Reduce parser M_G^R

- The input is shifted successively to the pushdown.
- Is there a complete right-hand side (a handle) atop the pushdown, it is replaced (reduced) by the corresponding left-hand side

Bottom-up Analysis

Theorem:

Let a grammar G be reduced and left-recursive, then G is not $LL(k)$ for any k.

Proof:

Let $A \to A\,\beta \mid \alpha \quad \in P$
and A be reachable from S

Assumption: G is $LL(k)$

$\Rightarrow \text{First}_k(\alpha\,\beta^n\,\gamma) \cap$
$\text{First}_k(\alpha\,\beta^{n+1}\,\gamma) = \emptyset$

Case 1: $\beta \to^* \epsilon$ — Contradiction !!!
Case 2: $\beta \to^* w \neq \epsilon \implies \text{First}_k(\alpha\,\beta^k\,\gamma) \cap \text{First}_k(\alpha\,\beta^{k+1}\,\gamma) \neq \emptyset$

108 / 149

Shift-Reduce Parser

Example:

$$
\begin{array}{lll}
S & \to & A\,B \\
A & \to & a \\
B & \to & b
\end{array}
$$

The pushdown automaton:

States: $q_0, f, a, b, A, B, S;$
Start state: q_0
End state: f

q_0	a	$q_0\,a$
a	ϵ	A
A	b	$A\,b$
b	ϵ	B
$A\,B$	ϵ	S
$q_0\,S$	ϵ	f

Shift-Reduce Parser

Construction:

In general, we create an automaton $M_G^R = (Q, T, \delta, q_0, F)$ with:

- $Q = T \cup N \cup \{q_0, f\}$ (q_0, f fresh);
- $F = \{f\}$;
- Transitions:

$$\begin{array}{lll} \delta & = & \{(q, x, q\,x) \mid q \in Q, x \in T\} \;\cup \qquad // \text{ Shift-transitions} \\ & & \{(q\,\alpha, \epsilon, q\,A) \mid q \in Q, A \to \alpha \in P\} \;\cup \qquad // \text{ Reduce-transitions} \\ & & \{(q_0\,S, \epsilon, f)\} \qquad // \text{ finish} \end{array}$$

Shift-Reduce Parser

Construction:

In general, we create an automaton $M_G^R = (Q, T, \delta, q_0, F)$ with:

- $Q = T \cup N \cup \{q_0, f\}$ (q_0, f fresh);
- $F = \{f\}$;
- Transitions:

$$\begin{array}{lll} \delta & = & \{(q, x, q\,x) \mid q \in Q, x \in T\} \;\cup \qquad // \text{ Shift-transitions} \\ & & \{(q\,\alpha, \epsilon, q\,A) \mid q \in Q, A \to \alpha \in P\} \;\cup \qquad // \text{ Reduce-transitions} \\ & & \{(q_0\,S, \epsilon, f)\} \qquad // \text{ finish} \end{array}$$

Example-computation:

$$\begin{array}{lllll} (q_0, \; a\,b) & \vdash & (q_0\,a\,, \; b) & \vdash & (q_0\,A, \; b) \\ & \vdash & (q_0\,A\;b\,, \; \epsilon) & \vdash & (q_0\;A\,B\,, \; \epsilon) \\ & \vdash & (q_0\,S, \; \epsilon) & \vdash & (f, \; \epsilon) \end{array}$$

Shift-Reduce Parser

Observation:

- The sequence of reductions corresponds to a reverse rightmost-derivation for the input
- To prove correctnes, we have to prove:

$$(\epsilon, w) \vdash^* (A, \epsilon) \qquad \text{iff} \qquad A \to^* w$$

- The shift-reduce pushdown automaton M_G^R is in general also non-deterministic
- For a deterministic parsing-algorithm, we have to identify computation-states for reduction

$\Longrightarrow$ LR-Parsing

Shift-Reduce Parser

Construction:

In general, we create an automaton $M_G^R = (Q, T, \delta, q_0, F)$ with:

- $Q = T \cup N \cup \{q_0, f\}$ (q_0, f fresh);
- $F = \{f\}$;
- Transitions:

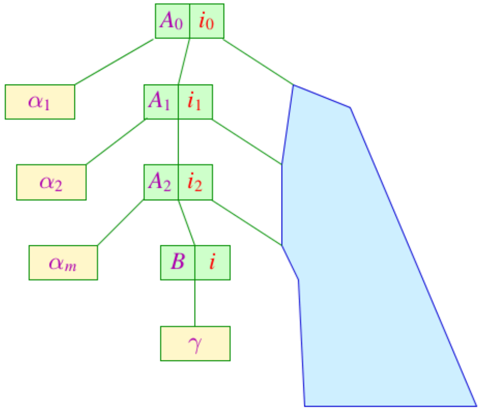
$$\begin{array}{lll} \delta & = & \{(q, x, q\,x) \mid q \in Q, x \in T\} \;\cup \qquad // \text{ Shift-transitions} \\ & & \{(q\,\alpha, \epsilon, q\,A) \mid q \in Q, A \to \alpha \in P\} \;\cup \qquad // \text{ Reduce-transitions} \\ & & \{(q_0\,S, \epsilon, f)\} \qquad // \text{ finish} \end{array}$$

Example-computation:

$$\begin{array}{lllll} (q_0, \; a\,b) & \vdash & (q_0\,a\,, \; b) & \vdash & (q_0\,A, \; b) \\ & \vdash & (q_0\,A\;b\,, \; \epsilon) & \vdash & (q_0\;A\,B\,, \; \epsilon) \\ & \vdash & (q_0\,S, \; \epsilon) & \vdash & (f, \; \epsilon) \end{array}$$

Bottom-up Analysis

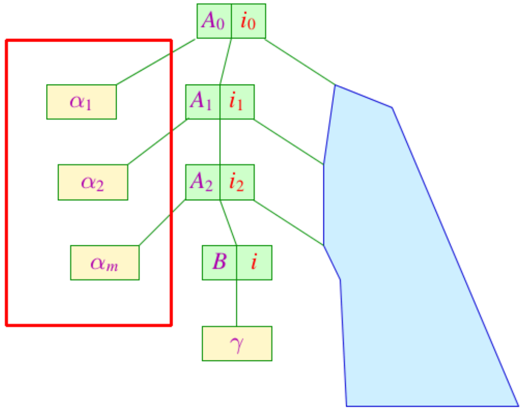
$\alpha\,\gamma$ is viable for $[B \to \gamma\bullet]$ iff $S \to_R^* \alpha\, B\, v$



... with $\alpha = \alpha_1 \ldots \alpha_m$

Bottom-up Analysis

$\alpha\,\gamma$ is viable for $[B \to \gamma\bullet]$ iff $S \to_R^* \alpha\, B\, v$

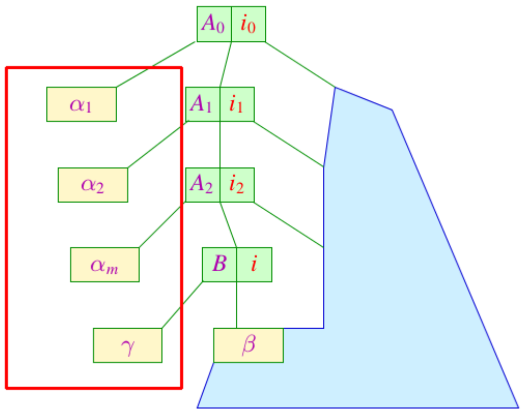


... with $\alpha = \alpha_1 \ldots \alpha_m$

Conversely, for an arbitrary valid word α' we can determine the set of all later on possibly matching rules ...

Bottom-up Analysis

The item $[B \to \gamma \bullet \beta]$ is called admissible for α' iff $S \to_R^* \alpha\, B\, v$ with $\alpha' = \alpha\,\gamma$:



... with $\alpha = \alpha_1 \ldots \alpha_m$

Characteristic Automaton

Observation:

The set of viable prefixes from $(N \cup T)^*$ for (admissible) items can be computed from the content of the shift-reduce parser's pushdown with the help of a finite automaton:

States: Items

Start state: $[S' \to \bullet\, S]$

Final states: $\{[B \to \gamma\bullet] \mid B \to \gamma \in P\}$

Transitions:

(1) $([A \to \alpha \bullet X\, \beta], X, [A \to \alpha\, X \bullet \beta])$, $X \in (N \cup T), A \to \alpha\, X\, \beta \in P$;

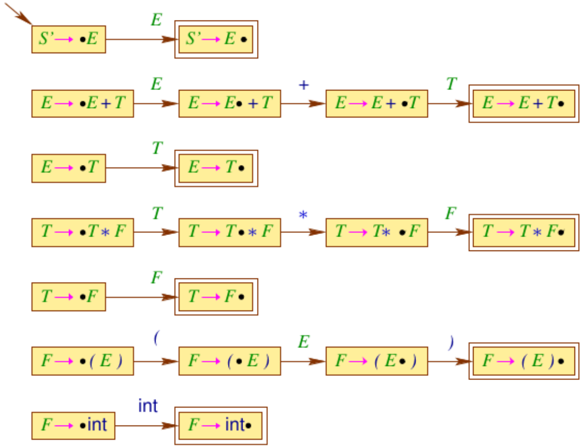
(2) $([A \to \alpha \bullet B\, \beta], \epsilon, [B \to \bullet\, \gamma])$, $A \to \alpha\, B\, \beta\,, \; B \to \gamma \in P$;

The automaton $c(G)$ is called characteristic automaton for G.

Characteristic Automaton

for example:

$$\begin{array}{lcl} E & \to & E+T \quad | \quad T \\ T & \to & T*F \quad | \quad F \\ F & \to & (\,E\,) \quad | \quad \text{int} \end{array}$$



Characteristic Automaton

Observation:
The set of viable prefixes from $(N \cup T)^*$ for (admissible) items can be computed from the content of the shift-reduce parser's pushdown with the help of a finite automaton:

States: Items

Start state: $[S' \to \bullet\, S]$

Final states: $\{[B \to \gamma \bullet] \mid B \to \gamma \in P\}$

Transitions:

(1) $([A \to \alpha \bullet X\,\beta], X, [A \to \alpha\, X \bullet \beta]), \quad X \in (N \cup T), A \to \alpha\, X\, \beta \in P;$

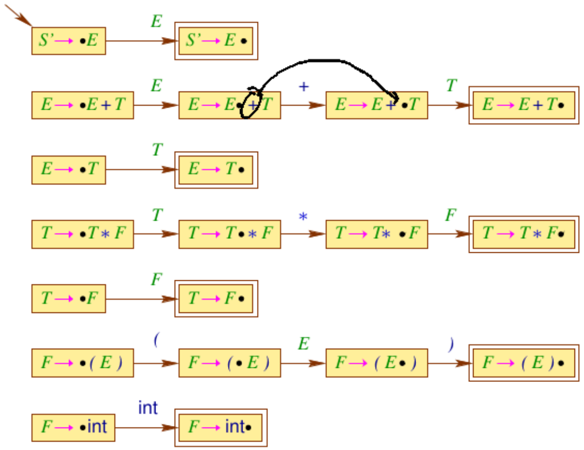
(2) $([A \to \alpha \bullet B\,\beta], \epsilon, [B \to \bullet\, \gamma]), \quad A \to \alpha\, B\, \beta\,,\; B \to \gamma \in P;$

The automaton $c(G)$ is called characteristic automaton for G.

Characteristic Automaton

for example:

$$\begin{array}{lcl} E & \to & E+T \quad | \quad T \\ T & \to & T*F \quad | \quad F \\ F & \to & (\,E\,) \quad | \quad \text{int} \end{array}$$



Characteristic Automaton

Observation:
The set of viable prefixes from $(N \cup T)^*$ for (admissible) items can be computed from the content of the shift-reduce parser's pushdown with the help of a finite automaton:

States: Items

Start state: $[S' \to \bullet\, S]$

Final states: $\{[B \to \gamma \bullet] \mid B \to \gamma \in P\}$

Transitions:

(1) $([A \to \alpha \bullet X\,\beta], X, [A \to \alpha\, X \bullet \beta]), \quad X \in (N \cup T), A \to \alpha\, X\, \beta \in P;$

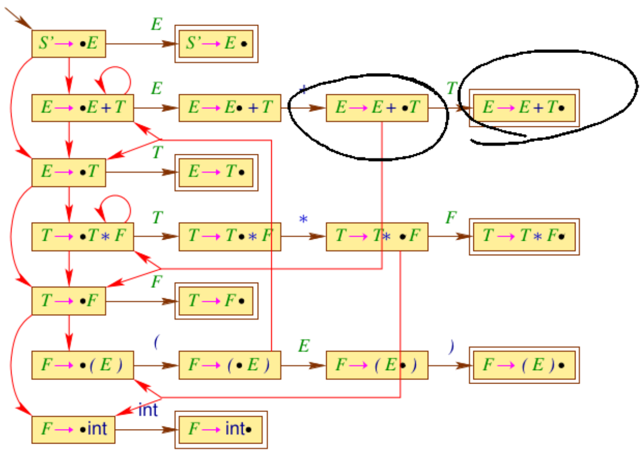
(2) $([A \to \alpha \bullet B\,\beta], \epsilon, [B \to \bullet\, \gamma]), \quad A \to \alpha\, B\, \beta\,,\; B \to \gamma \in P;$

The automaton $c(G)$ is called characteristic automaton for G.

Characteristic Automaton

for example:

$$\begin{array}{lcl} E & \to & E+T \mid T \\ T & \to & T*F \mid F \\ F & \to & (\,E\,) \mid \text{int} \end{array}$$

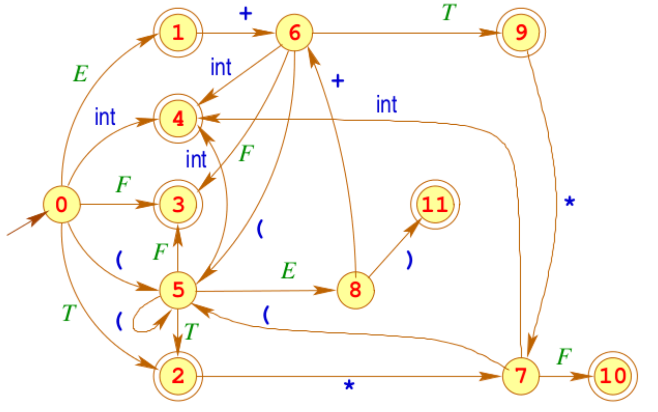


Canonical LR(0)-Automaton

The canonical $LR(0)$-automaton $LR(G)$ is created from $c(G)$ by:

1. performing arbitrarily many ϵ-transitions after every consuming transition
2. performing the powerset construction

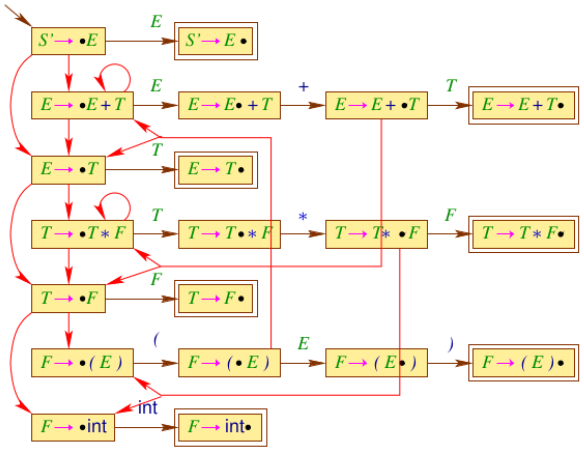
... for example:



Characteristic Automaton

for example:

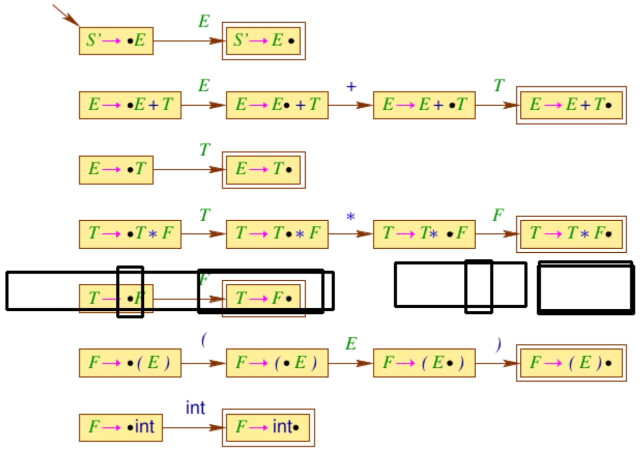
$$\begin{array}{lcl} E & \to & E+T \mid T \\ T & \to & T*F \mid F \\ F & \to & (\,E\,) \mid \text{int} \end{array}$$



Characteristic Automaton

for example:

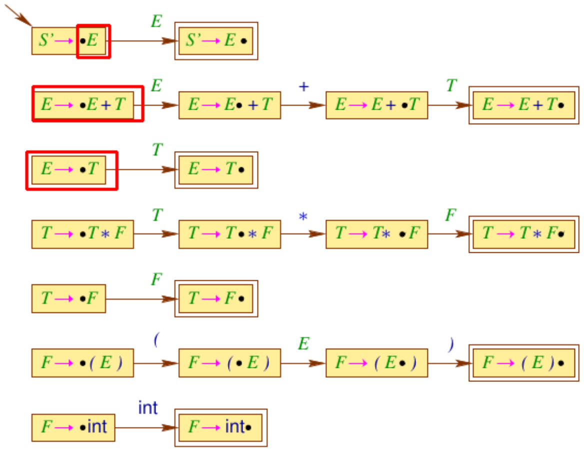
$$\begin{array}{lcl} E & \to & E+T \mid T \\ T & \to & T*F \mid F \\ F & \to & (\,E\,) \mid \text{int} \end{array}$$



Characteristic Automaton

for example:

$$\begin{array}{lcll} E & \to & E+T & \mid\ T \\ T & \to & T*F & \mid\ F \\ F & \to & (\ E\ ) & \mid\ \text{int} \end{array}$$

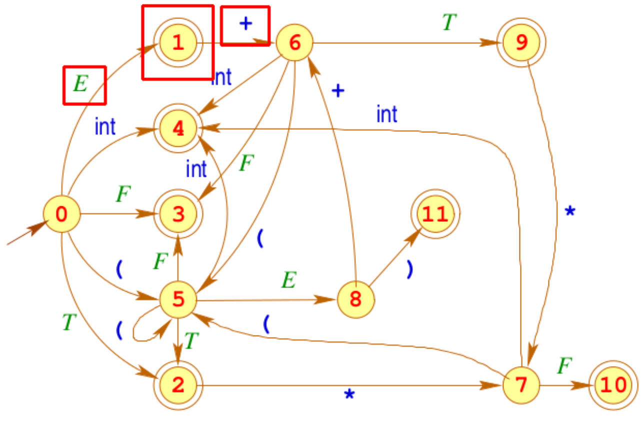


Canonical LR(0)-Automaton

The canonical $LR(0)$-automaton $LR(G)$ is created from $c(G)$ by:

1. performing arbitrarily many ϵ-transitions after every consuming transition
2. performing the powerset construction

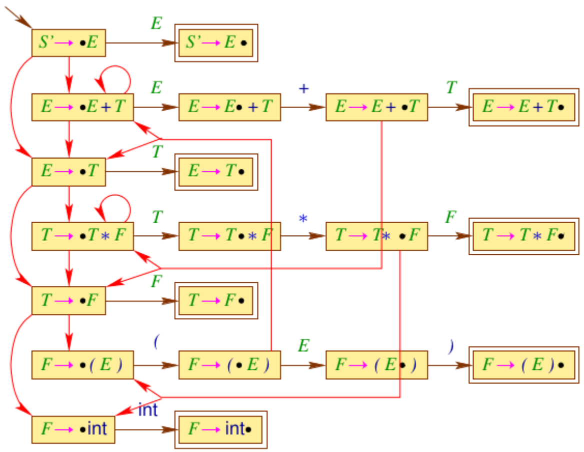
... for example:



Characteristic Automaton

for example:

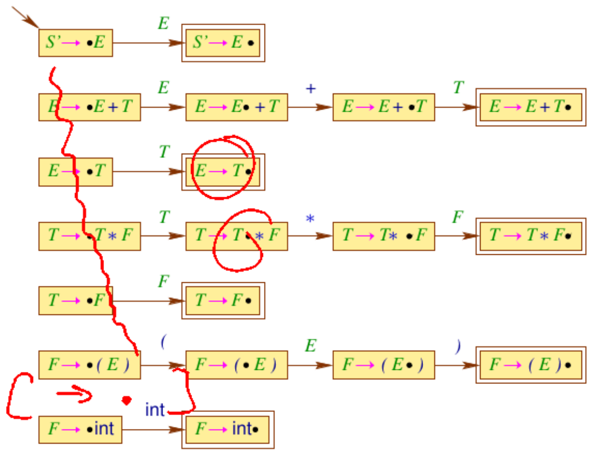
$$\begin{array}{lcll} E & \to & E+T & \mid\ T \\ T & \to & T*F & \mid\ F \\ F & \to & (\ E\ ) & \mid\ \text{int} \end{array}$$



Characteristic Automaton

for example:

$$\begin{array}{lcll} E & \to & E+T & \mid\ T \\ T & \to & T*F & \mid\ F \\ F & \to & (\ E\ ) & \mid\ \text{int} \end{array}$$



Characteristic Automaton

Observation:

The set of viable prefixes from $(N \cup T)^*$ for (admissible) items can be computed from the content of the shift-reduce parser's pushdown with the help of a finite automaton:

States: Items

Start state: $[S' \to \bullet\, S]$

Final states: $\{[B \to \gamma \bullet] \mid B \to \gamma \in P\}$

Transitions:

(1) $([A \to \alpha \bullet X \beta], X, [A \to \alpha X \bullet \beta]), \quad X \in (N \cup T), A \to \alpha X \beta \in P;$

(2) $([A \to \alpha \bullet B \beta], \epsilon, [B \to \bullet\, \gamma]), \quad A \to \alpha B \beta\,,\ B \to \gamma \in P;$

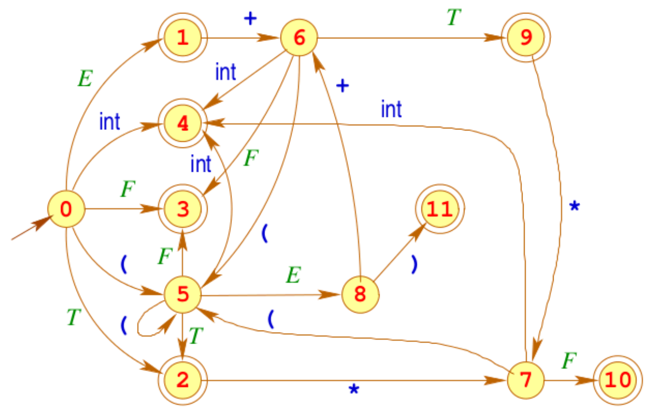
The automaton $c(G)$ is called characteristic automaton for G.

Canonical LR(0)-Automaton

The canonical $LR(0)$-automaton $LR(G)$ is created from $c(G)$ by:

1. performing arbitrarily many ϵ-transitions after every consuming transition
2. performing the powerset construction

... for example:

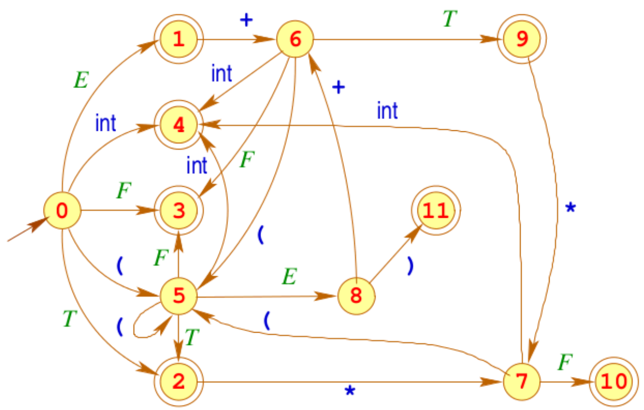


Canonical LR(0)-Automaton

The canonical $LR(0)$-automaton $LR(G)$ is created from $c(G)$ by:

1. performing arbitrarily many ϵ-transitions after every consuming transition
2. performing the powerset construction

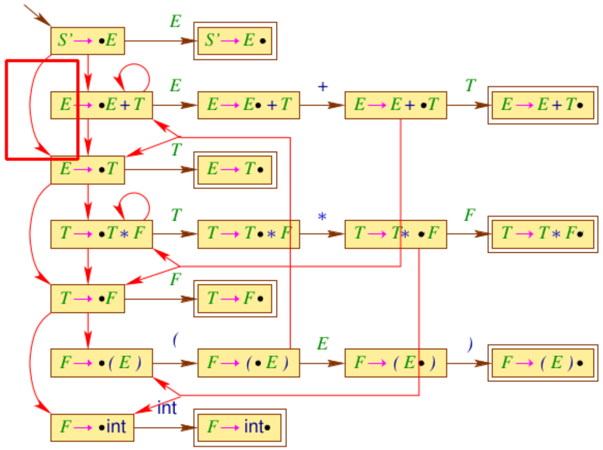
... for example:



Characteristic Automaton

for example:

$$E \to E + T \mid T$$
$$T \to T * F \mid F$$
$$F \to (\, E \,) \mid \text{int}$$

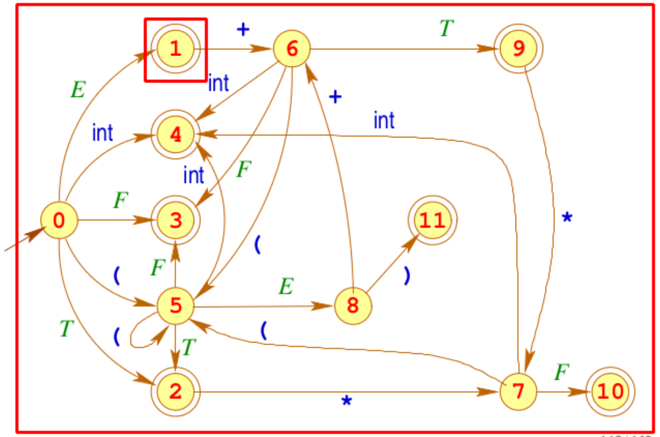


Canonical LR(0)-Automaton

The canonical $LR(0)$-automaton $LR(G)$ is created from $c(G)$ by:

1. performing arbitrarily many ϵ-transitions after every consuming transition
2. performing the powerset construction

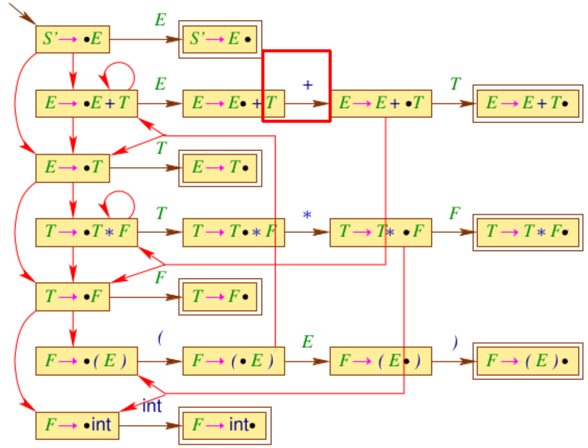
... for example:



Characteristic Automaton

for example:

$$
\begin{array}{lcl}
E & \to & E+T \quad | \quad T \\
T & \to & T*F \quad | \quad F \\
F & \to & (\,E\,) \quad | \quad \text{int}
\end{array}
$$



Canonical LR(0)-Automaton

Example:

$$
\begin{array}{lcl}
E & \to & E+T \quad | \quad T \\
T & \to & T*F \quad | \quad F \\
F & \to & (\,E\,) \quad | \quad \text{int}
\end{array}
$$

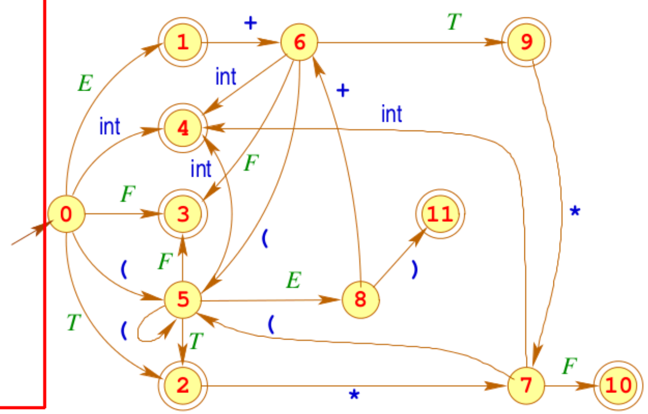
Therefore we determine:

Canonical LR(0)-Automaton

The canonical $LR(0)$-automaton $LR(G)$ is created from $c(G)$ by:

1. performing arbitrarily many ϵ-transitions after every consuming transition
2. performing the powerset construction

... for example:

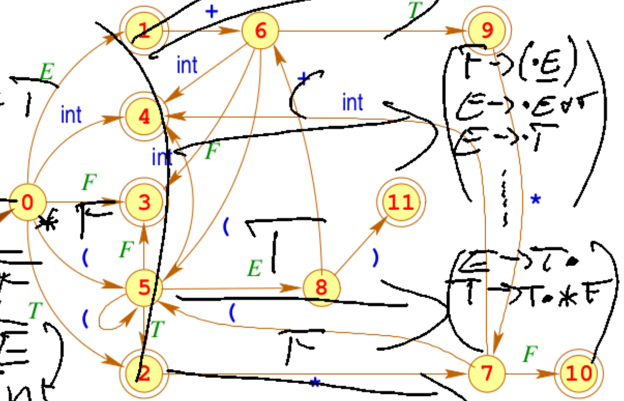


Canonical LR(0)-Automaton

The canonical $LR(0)$-automaton $LR(G)$ is created from $c(G)$ by:

1. performing arbitrarily many ϵ-transitions after every consuming transition
2. performing the powerset construction

... for example:



Canonical LR(0)-Automaton

Example:

$$\begin{array}{lcl} E & \to & E+T \mid T \\ T & \to & T*F \mid F \\ F & \to & (\,E\,) \mid \text{int} \end{array}$$

Therefore we determine:

$$q_0 = \{[S' \to \bullet E], [E \to \bullet E+T], [E \to \bullet T], [T \to \bullet T*F]\} [T \to \bullet F], [F \to \bullet (\,E\,)], [F \to \bullet \text{int}]\}$$

$$q_1 = \delta(q_0, E) = \{[S' \to E\bullet], [E \to E \bullet + T]\}$$

$$q_2 = \delta(q_0, T) = \{[E \to T\bullet], [T \to T \bullet * F]\}$$

$$q_3 = \delta(q_0, F) = \{[T \to F\bullet]\}$$

$$q_4 = \delta(q_0, \text{int}) = \{[F \to \text{int}\bullet]\}$$

Canonical LR(0)-Automaton

$$q_5 = \delta(q_0, (\,) = \{[F \to (\bullet E\,)], [E \to \bullet E+T], [E \to \bullet T], [T \to \bullet T*F], [T \to \bullet F], [F \to \bullet (\,E\,)], [F \to \bullet \text{int}]\}$$

$$q_6 = \delta(q_1, +) = \{[E \to E + \bullet T], [T \to \bullet T*F], [T \to \bullet F], [F \to \bullet (\,E\,)], [F \to \bullet \text{int}]\}$$

$$q_7 = \delta(q_2, *) = \{[T \to T * \bullet F], [F \to \bullet (\,E\,)], [F \to \bullet \text{int}]\}$$

$$q_8 = \delta(q_5, E) = \{[F \to (\,E\bullet)], [E \to E \bullet + T]\}$$

$$q_9 = \delta(q_6, T) = \{[E \to E+T\bullet], [T \to T \bullet * F]\}$$

$$q_{10} = \delta(q_7, F) = \{[T \to T*F\bullet]\}$$

$$q_{11} = \delta(q_8,)\,) = \{[F \to (\,E\,)\bullet]\}$$

Canonical LR(0)-Automaton

Observation:

The canonical $LR(0)$-automaton can be created directly from the grammar.
Therefore we need a helper function δ_ϵ^* (ϵ-closure)

$$\delta_\epsilon^*(q) = q \cup \{[B \to \bullet\gamma] \mid \exists\, [A \to \alpha \bullet B'\,\beta'] \in q, \; \beta \in (N \cup T)^* : \; B' \to^* B\,\beta\}$$

We define:

States: Sets of items;

Start state: $\delta_\epsilon^* \{[S' \to \bullet S]\}$

Final states: $\{q \mid \exists A \to \alpha \in P : \; [A \to \alpha\bullet] \in q\}$

Transitions: $\delta(q, X) = \delta_\epsilon^* \{[A \to \alpha X \bullet \beta] \mid [A \to \alpha \bullet X\,\beta] \in q\}$

Canonical LR(0)-Automaton

Observation:

The canonical $LR(0)$-automaton can be created directly from the grammar.
Therefore we need a helper function δ^*_ϵ (ϵ-closure)

$$\delta^*_\epsilon(q) = q \cup \{[B \to \bullet\, \gamma] \mid \exists\, [A \to \alpha \bullet B'\, \beta'] \in q, \; \beta \in (N \cup T)^* : \; B' \to^* B\, \beta\}$$

We define:

States: Sets of items;
Start state: $\delta^*_\epsilon\, \{[S' \to \bullet\, S]\}$
Final states: $\{q \mid \exists A \to \alpha \in P : \; [A \to \alpha\, \bullet] \in q\}$
Transitions: $\delta(q, X) = \delta^*_\epsilon\, \{[A \to \alpha\, X \bullet \beta] \mid [A \to \alpha \bullet X\, \beta] \in q\}$

LR(0)-Parser

Idea for a parser:

- The parser manages a viable prefix $\alpha = X_1 \ldots X_m$ on the pushdown and uses $LR(G)$, to identify reduction spots.
- It can reduce with $A \to \gamma$, if $[A \to \gamma\, \bullet]$ is admissible for α

Optimization:

We push the states instead of the X_i in order not to process the pushdown's content with the automaton anew all the time.
Reduction with $A \to \gamma$ leads to popping the uppermost $|\gamma|$ states and continue with the state on top of the stack and input A.

Attention:

This parser is only deterministic, if each final state of the canonical $LR(0)$-automaton is conflict free.

LR(0)-Parser

... for example:

$$q_1 = \{[S' \to E\, \bullet], [E \to E\, \bullet + T]\}$$
$$q_2 = \{[E \to T\, \bullet], [T \to T \bullet * F]\}$$
$$q_3 = \{[T \to F\, \bullet]\}$$
$$q_4 = \{[F \to \text{int}\, \bullet]\}$$
$$q_9 = \{[E \to E + T\, \bullet], [T \to T \bullet * F]\}$$
$$q_{10} = \{[T \to T * F\, \bullet]\}$$
$$q_{11} = \{[F \to (\, E\,)\, \bullet]\}$$

The final states q_1, q_2, q_9 contain more then one admissible item
$\Rightarrow$ non deterministic!

LR(0)-Parser

The construction of the $LR(0)$-parser:

States: $Q \cup \{f\}$ (f fresh)
Start state: q_0
Final state: f
Transitions:

Shift:	$(p, a, p\, q)$	if	$q = \delta(p, a) \neq \emptyset$
Reduce:	$(p\, q_1 \ldots q_m, \epsilon, p\, q)$	if	$[A \to X_1 \ldots X_m\, \bullet] \in q_m$, $q = \delta(p, A)$
Finish:	$(q_0\, p, \epsilon, f)$	if	$[S' \to S\bullet] \in p$

with $LR(G) = (Q, T, \delta, q_0, F)$.