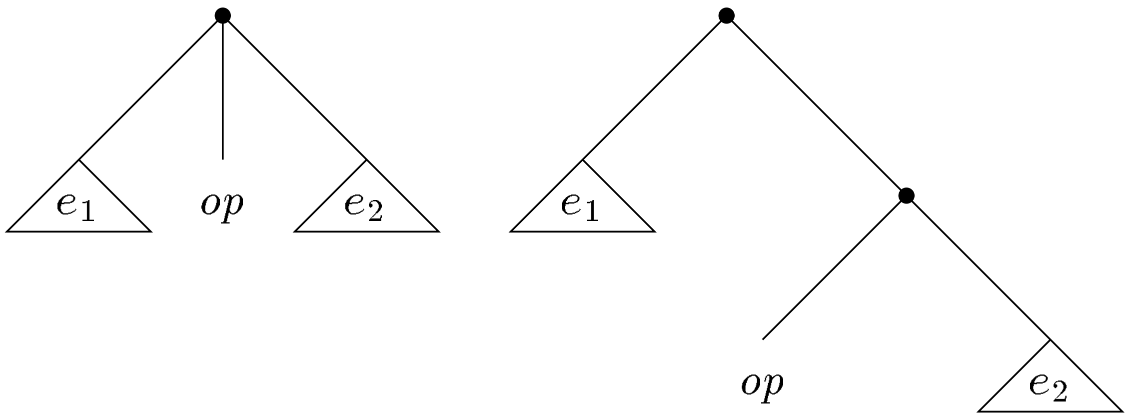
Regular Right Part Grammars

Reinhard Wilhelm
Universität des Saarlandes
wilhelm@cs.uni-sb.de

May 21, 1997

Rrp–grammars – Motivation

- Left recursion destroys the LL(k)–property.
- Factorization destroys the structure of the grammar. Disadvantageous for further processing of syntax trees.



- Non-embedded recursion as in lists, sums, and products can be expressed by regular expressions.

Arithmetic Expressions

$G_e = (\{S, E, T, F\}, \{\textbf{id}, (,), +, -, *, /\}, p, S)$,
where p is the following mapping

$$
\begin{array}{lcl}
S & \rightarrow & E \\
E & \rightarrow & T\{\{+\,|\,-\}\,T\}^* \\
T & \rightarrow & F\{\{*\,|\,/\}\,F\}^* \\
F & \rightarrow & (E)\,|\,\textbf{id}
\end{array}
$$

- embedded recursion, $E \stackrel{*}{\Longrightarrow} (E)$, remains,

- non-embedded recursion, $E \stackrel{*}{\Longrightarrow} E + T + \cdots + T$, replaced by iteration.

rrp–grammars, Definition

Regular right part Grammar (rrp–grammar)
$G = (V_N, V_T, p, S)$ where
$p : V_N \;\rightarrow\; RA$ is a mapping of non-terminals to RA, the set of regular expressions over $V_N \cup V_T$.

Relation $\underset{R,lm}{\Longrightarrow}$ on RA, "directly leftmost rrp–derives", is defined by:

$$
\begin{array}{llll}
w\,X\,\beta & \underset{R,lm}{\Longrightarrow} & w\,\alpha\,\beta & \text{with } \alpha = p(X) \\
w\,(r_1\,|\ldots|\,r_n)\,\beta & \underset{R,lm}{\Longrightarrow} & w\,r_i\,\beta & \text{for } 1 \leq i \leq n \\
w\,(r)^*\,\beta & \underset{R,lm}{\Longrightarrow} & w\,\beta & \\
w\,(r)^*\,\beta & \underset{R,lm}{\Longrightarrow} & w\,r\,(r)^*\,\beta &
\end{array}
$$

An Example Derivation

An rrp–derivation for **id** $+$ **id** $*$ **id** (rrp–grammar G_e):

$$
\begin{aligned}
S &\underset{R,lm}{\Longrightarrow} E \underset{R,lm}{\Longrightarrow} T\{\{+|-\}T\}^* \\
&\underset{R,lm}{\Longrightarrow} F\{\{*|/\}F\}^*\{\{+|-\}T\}^* \\
&\underset{R,lm}{\Longrightarrow} \{(E)|\textbf{id}\}\{\{*|/\}F\}^*\{\{+|-\}T\}^* \\
&\underset{R,lm}{\Longrightarrow} \textbf{id}\{\{*|/\}F\}^*\{\{+|-\}T\}^* \\
&\underset{R,lm}{\Longrightarrow} \textbf{id}\{\{+|-\}T\}^* \\
&\underset{R,lm}{\Longrightarrow} \textbf{id}\{+|-\}T\{\{+|-\}T\}^* \\
&\underset{R,lm}{\Longrightarrow} \textbf{id} + T\{\{+|-\}T\}^* \\
&\underset{R,lm}{\Longrightarrow} \textbf{id} + F\{\{*|/\}F\}^*\{\{+|-\}T\}^* \\
&\underset{R,lm}{\Longrightarrow} \textbf{id} + \{(E)|\textbf{id}\}\{\{*|/\}F\}^*\{\{+|-\}T\}^* \\
&\underset{R,lm}{\Longrightarrow} \textbf{id} + \textbf{id}\{\{*|/\}F\}^*\{\{+|-\}T\}^* \\
&\underset{R,lm}{\Longrightarrow} \textbf{id} + \textbf{id}\{*|/\}F\{\{*|/\}F\}^*\{\{+|-\}T\}^* \\
&\underset{R,lm}{\Longrightarrow} \textbf{id} + \textbf{id} * F\{\{*|/\}F\}^*\{\{+|-\}T\}^* \\
&\underset{R,lm}{\Longrightarrow} \textbf{id} + \textbf{id} * \{(E)|\textbf{id}\}\{\{*|/\}F\}^*\{\{+|-\}T\}^* \\
&\underset{R,lm}{\Longrightarrow} \textbf{id} + \textbf{id} * \textbf{id}\{\{*|/\}F\}^*\{\{+|-\}T\}^* \\
&\underset{R,lm}{\Longrightarrow} \textbf{id} + \textbf{id} * \textbf{id}\{\{+|-\}T\}^* \\
&\underset{R,lm}{\Longrightarrow} \textbf{id} + \textbf{id} * \textbf{id}
\end{aligned}
$$

Parser for rrp–Grammars

Non-determinism in rrp–derivations lies in

- alternatives $(r_1 \,|\, \ldots \,|\, r_n)$: which to select?
- iterations $(r)^*$: unroll or eliminate?

Lookahead is used to remove the non-determinism.

$FIRST_1$

Split computation of $FIRST_1$ for rrp–grammars into

- computation of ε–productivity (eps),
- computation of ε-free $FIRST_1$ (ε-$\mathit{ffi}(r_i)$).

ε–Productivity:

$$
\begin{array}{l}
eps(a) = false, \quad \text{for } a \in V_T \\
eps(\varepsilon) = true \\
eps(r^*) = true \\
eps(X) = eps(r), \text{ if } p(X) = r \text{ for } X \in V_N \\
eps((r_1|\ldots|r_n)) = \bigvee_{i=1}^{n} eps(r_i) \\
eps((r_1\ldots r_n)) = \bigwedge_{i=1}^{n} eps(r_i)
\end{array}
$$

For all non-terminal of G_e holds: $eps(X) = false$

ε-free $FIRST_1$:

$$\varepsilon\text{-}\mathit{ffi}(\varepsilon) = \emptyset$$
$$\varepsilon\text{-}\mathit{ffi}(a) = \{a\}$$
$$\varepsilon\text{-}\mathit{ffi}(r^*) = \varepsilon\text{-}\mathit{ffi}(r)$$
$$\varepsilon\text{-}\mathit{ffi}(X) = \varepsilon\text{-}\mathit{ffi}(r), \text{ if } p(X) = r$$
$$\varepsilon\text{-}\mathit{ffi}((r_1|\ldots|r_n)) = \bigcup_{1 \leq i \leq n} \varepsilon\text{-}\mathit{ffi}(r_i)$$
$$\varepsilon\text{-}\mathit{ffi}((r_1 \ldots r_n)) = \bigcup_{1 \leq j \leq n} \{\varepsilon\text{-}\mathit{ffi}(r_j) \mid \bigwedge_{1 \leq i < j} eps(r_i)\}$$

The ε-$\mathit{ffi}(r_i)$– and therefore the $FIRST_1$–sets for G_e are:
$FIRST_1(S) = FIRST_1(E) = FIRST_1(T) = FIRST_1(F) = \{(, \textbf{id}\}$

rrp–Items

$(X, \alpha, \beta, \gamma)$ is an **rrp–item**, written as $[X \rightarrow \beta.\alpha\gamma]$, if $X \in V_N$ and α, β, γ are words over V_N, V_T and the metacharacters $(,),\ ^*, |, \varepsilon$,
$p(X) = \beta\alpha\gamma$ and α is regular subexpression of $\beta\alpha\gamma$.

Note: The regular expressions should be fully parenthesized to avoid ambiguities.

Examples:

rrp–items

$$[E \rightarrow .\{T\{\{+|-\}T\}^*\}]$$
$$[E \rightarrow \{.T\{\{+|-\}T\}^*\}]$$
$$[E \rightarrow \{T.\{\{+|-\}T\}^*\}]$$
$$[E \rightarrow \{T\{\{.+|-\}T\}^*\}]$$
$$[E \rightarrow \{T\{\{+|-\}.T\}^*\}]$$

no rrp–items:

$$[E \rightarrow \{T\{\{+.|-\}T\}^*\}]$$
$$[E \rightarrow \{T\{\{+|-\}T.\}^*\}]$$
$$[E \rightarrow \{T\{\{+|-\}T\}^*.\}]$$

$FOLLOW_1$

$FOLLOW_1$ has to be generalized from occurrences of non-terminals to rrp–items.

$$FOLLOW_1([S' \rightarrow .S]) = \{\#\}$$

$$FOLLOW_1([X \rightarrow \cdots(r_1|\cdots|.r_i|\cdots|r_n)\cdots]) = \\ FOLLOW_1([X \rightarrow \cdots.(r_1|\cdots|r_i|\cdots|r_n)\cdots])$$

$$FOLLOW_1([X \rightarrow \cdots(\cdots .r_i r_{i+1}\cdots)\cdots]) = \\ \varepsilon\text{-}\mathit{ffi}(r_{i+1}) \cup \begin{cases} FOLLOW_1([X \rightarrow \cdots(\cdots r_i . r_{i+1}\cdots)\cdots]), \\ \quad \text{if } eps(r_{i+1}) = \mathit{true} \\ \emptyset \quad \text{otherwise} \end{cases}$$

$$FOLLOW_1([X \rightarrow \cdots(r_1\cdots r_{n-1}.r_n)\cdots]) = \\ FOLLOW_1([X \rightarrow \cdots.(r_1\cdots r_{n-1}r_n)\cdots])$$

$$FOLLOW_1([X \rightarrow \cdots(.r)^*\cdots]) = \\ \varepsilon\text{-}\mathit{ffi}(r) \cup FOLLOW_1([X \rightarrow \cdots.(r)^*\cdots])$$

$$FOLLOW_1([X \rightarrow .r]) = \bigcup FOLLOW_1([Y \rightarrow \cdots.X\cdots])$$

$FOLLOW_1$ – Example

The $FOLLOW_1$–sets for some items of grammar G_e are:

$$FOLLOW_1([S \rightarrow \ .E]) = \{\#\}$$

$$\begin{aligned}
&FOLLOW_1([E \rightarrow \ T.\{\{\{+|-\}T\}^*\}]) = \\
&\quad FOLLOW_1([E \rightarrow \ .\{T\{\{+|-\}T\}^*\}]) = \\
&\quad FOLLOW_1([S \rightarrow \ .E]) \cup FOLLOW_1([F \rightarrow \ (.E)]) = \\
&\quad (\{\#\} \cup FOLLOW_1([F \rightarrow \ (.E)])) = \{), \#\}
\end{aligned}$$

$$FOLLOW_1([T \rightarrow \ F\{.\{\{*|/\}F\}^*\}]) = \{+, -,), \#\}$$

RLL(1)–Grammar

$G = (V_N, V_T, p, S)$ is an **RLL(1)–grammar**, if for all items $[X \rightarrow \cdots .(r_1|\cdots|r_n)\cdots]$:

$FIRST_1(r_i) \oplus_1 FOLLOW_1([X \rightarrow \cdots .(r_1|\cdots|r_n)\cdots]) \cap FIRST_1(r_j) \oplus_1 FOLLOW_1([X \rightarrow \cdots .(r_1|\cdots|r_n)\cdots]) = \emptyset$ for all $i \neq j$,

and for all Items $[X \rightarrow \cdots .(r)^* \cdots]$:
$FIRST_1(r) \cap FOLLOW_1([X \rightarrow \cdots .(r)^* \cdots]) = \emptyset$ and $eps(r) = false$.

G_e is an RLL(1)–grammar.

RLL(1)–Parser

- A deterministic push down automaton.
- Control is a **parser table** $M : It_G \times V_T \rightarrow It_G \cup \{error\}$.
- M is consulted for alternatives and iterations, i.e. items of the form $[X \rightarrow \cdots .(r_1|\cdots|r_n)\cdots]$ and $[X \rightarrow \cdots .(r)^* \cdots]$.
- $M[\rho, a]$ for the actual topmost item ρ and next input symbol a indicates the next item or $error$.
- $(\#[S' \rightarrow .S], w\#)$ — start configuration.
- $(\#[S' \rightarrow S.], \#)$ — accepting configuration.

The other transitions are:

$\delta([X \rightarrow \cdots .a \cdots], a) = [X \rightarrow \cdots a. \cdots]$

$\delta([X \rightarrow \cdots .Y \cdots], \varepsilon) =$
$\quad [X \rightarrow \cdots .Y \cdots][Y \rightarrow .p(Y)]$

$\delta([X \rightarrow \cdots .Y \cdots][Y \rightarrow p(Y).], \varepsilon) =$
$\quad [X \rightarrow \cdots Y. \cdots]$

RLL(1)–Parser Generator

Input: RLL(1)–grammar G, $FIRST_1$ and $FOLLOW_1$ for G.

Output: Parsertable M of RLL(1)-Parser for G.

Method: For $[X \rightarrow \cdots .(r_1|\cdots|r_n)\cdots]$ set
$M([X \rightarrow \cdots .(r_1|\cdots|r_n)\cdots], a) =$
$[X \rightarrow \cdots(\cdots|.r_i|\cdots)\cdots]$,
for $a \in FIRST_1(r_i)$
and if additionally $\varepsilon \in FIRST_1(r_i)$ also
for $a \in FOLLOW_1([X \rightarrow \cdots .(r_1|\cdots|r_n)\cdots])$.

For $[X \rightarrow \cdots .(r)^* \cdots]$ set

$$M([X \rightarrow \cdots .(r)^* \cdots], a) =$$

$$\begin{cases} [X \rightarrow \cdots(.r)^* \cdots] & \text{if } a \in FIRST_1(r) \\ [X \rightarrow \cdots(r)^* . \cdots] & \text{if } a \in FOLLOW_1([X \rightarrow \cdots .(r)^* \cdots]) \end{cases}$$

Set all yet unfilled entries to $error$.

Example RRL(1)–Parser

Parsertable for grammar G_e (reflected along the diagonal)

	$[E \rightarrow T.\{\{+\mid-\}T\}^*]$	$[T \rightarrow F.\{\{*\mid/\}F\}^*]$
$+$	$[E \rightarrow T\{\{.+\mid-\}T\}^*]$	$[T \rightarrow F\{\{*\mid/\}F\}^*.]$
$-$	$[E \rightarrow T\{\{+\mid.-\}T\}^*]$	$[T \rightarrow F\{\{*\mid/\}F\}^*.]$
$\#$	$[E \rightarrow T\{\{+\mid-\}T\}^*.]$	$[T \rightarrow F\{\{*\mid/\}F\}^*.]$
$)$	$[E \rightarrow T\{\{+\mid-\}T\}^*.]$	$[T \rightarrow F\{\{*\mid/\}F\}^*.]$
$*$	$error$	$[T \rightarrow F\{\{.*\mid/\}F\}^*]$
$/$	$error$	$[T \rightarrow F\{\{*\mid./\}F\}^*]$

Recursive Descent RLL(1)–Parser

```
program parser;
    var nextsym: symbol;
    proc scan;
        (* reads next input symbol into nextsym *)
    proc error(message: string);
        (* issues error message and stops parser *)
    proc accept;
        (* successful termination *)
    p_progr(X_0 → α_0);
    p_progr(X_1 → α_1);
        ⋮
    p_progr(X_n → α_n);
begin
    scan;
    X_0;
    if nextsym = "#"
        then accept
        else error("# expected ")
    fi
end .
```

$p\_progr(X \;\to\; \alpha) =$

proc X;
begin
$\quad progr([X \;\to\; .\alpha])$
end ;

$progr([X \;\to\; \cdots .(\alpha_1|\alpha_2|\cdots|\alpha_{k-1}|\alpha_k)\cdots]) =$

case $nextsym$ **in**

$FiFo([X \;\to\; \cdots(.\alpha_1|\alpha_2|\cdots|\alpha_{k-1}|\alpha_k)\cdots])$:
$\quad progr([X \;\to\; \cdots(.\alpha_1|\alpha_2|\cdots|\alpha_{k-1}|\alpha_k)\cdots])$;

$FiFo([X \;\to\; \cdots(\alpha_1|.\alpha_2|\cdots|\alpha_{k-1}|\alpha_k)\cdots])$:
$\quad progr([X \;\to\; \cdots(\alpha_1|.\alpha_2|\cdots|\alpha_{k-1}|\alpha_k)\cdots])$;

$\vdots$

$FiFo([X \;\to\; \cdots(\alpha_1|\alpha_2|\cdots|.\alpha_{k-1}|\alpha_k)\cdots])$:
$\quad progr([X \;\to\; \cdots(\alpha_1|\alpha_2|\cdots|.\alpha_{k-1}|\alpha_k)\cdots])$;

otherwise $progr([X \;\to\; \cdots(\alpha_1|\alpha_2|\cdots|\alpha_{k-1}|.\alpha_k)\cdots])$;

endcase

$progr([X \;\to\; \cdots .(\alpha_1\alpha_2\cdots\alpha_k)\cdots]) =$

$progr([X \;\to\; \cdots(.\alpha_1\alpha_2\cdots\alpha_k)\cdots])$;
$progr([X \;\to\; \cdots(\alpha_1 .\alpha_2\cdots\alpha_k)\cdots])$;
$\vdots$
$progr([X \;\to\; \cdots(\alpha_1\alpha_2\cdots .\alpha_k)\cdots])$;

$progr([X \;\rightarrow\; \cdots .(\alpha)^* \cdots]) =$

while $nextsym$ **in** $FIRST_1(\alpha)$ **do**

$\quad progr([X \;\rightarrow\; \cdots .\alpha \cdots])$

od

$progr([X \;\rightarrow\; \cdots .(\alpha)^+ \cdots]) =$

repeat $progr([X \;\rightarrow\; \cdots .\alpha \cdots])$

until $nextsym$ **not in** $FIRST_1(\alpha)$

$progr([X \;\rightarrow\; \cdots .\varepsilon \cdots]) = \;;$

For $a \in V_T$ is

$progr([X \;\rightarrow\; \cdots .a \cdots]) =$

if $nextsym = a$ **then** $scan$

else $error(\text{" } a \text{ expected "})$

fi

For $Y \in V_N$ is

$progr([X \;\rightarrow\; \cdots .Y \cdots]) = Y$

Parser for Arithmetic Expressions

```
program parser;
var  nextsym: string;
proc  scan;
   (*reads next input symbol into  nextsym*)

proc  error (message: string);
   (*issues error message and stops parser*)

proc  accept;
   (*successful termination*)

proc  S;
   begin  E end ;
```

```
proc E;
  begin T;
    while nextsym ∈ {"+"|"−"} do
      case nextsym in
        {"+"} :     if nextsym = "+" then scan
        else error("+ expected") fi;
        otherwise if nextsym = "−" then scan
                    else error("− expected") fi;
      endcase;
      T
    od;
  end ;

proc T;
  begin F;
    while nextsym ∈ {"∗"|"/"} do
      case nextsym in
        {"∗"} :     if nextsym = "∗" then scan
                    else error("∗ expected") fi;
        otherwise if nextsym = "/" then scan
        else error("/ expected") fi;
      endcase;
      F
    od;
  end ;
```

```
proc  F;
  begin
        case  nextsym in
           {"("} :        E;
                          if nextsym = ")" then scan
                          else error(" ) expected") fi;
           otherwise if nextsym ="id" then scan
                          else error("id expected") fi;
        endcase;
  end ;

begin
scan;
S;
if  nextsym = "#"
then  accept
else  error("# expected")
fi
end .
```