A century of parentheses languages

with some amazing returns

ICTCS 2010, Camerino

Stefano Crespi Reghizzi

Politecnico di Milano
Parentheses have appeared in algebraic writing in the XV-XVI century. Erasmus of Rotterdam calls them lunulae. Earlier and until the XVIII century, overline vinculum had been used for grouping literals into a term

\[
\overline{aa + bb}^m \text{ instead of } \left(\overline{aa + bb}\right)^m
\]

Abstracting from the contents of parenthesized expressions, Walter von Dyck’s (1856-1934) name has been given to the formal language every computer science student knows.
The alphabet has just two letters: the open/close parentheses \( (, ) \) or \textit{begin, end}, etc.

The language \( Cl(\varepsilon) \) is the equivalence class of all strings such that repeated deletions of well-parenthesized digram \( () \) reduce the string to the empty one \( \varepsilon \).

\[
(((()))() \Rightarrow ((()))() = (())() \Rightarrow (())() \Rightarrow ()() \Rightarrow () = \varepsilon
\]

The equivalence class such that, after all deletions, the string, say, \( ) \) is obtained is another formal language: \( Cl(\varepsilon) \) instead of \( Cl(\varepsilon) \).
Properties of Dyck’s languages

Modest generalization: several matching pairs in the alphabet:

\( (,) \), \([,]\), \{"\}, \ldots

Obvious revision of cancellation rule.

**Concatenating** two or more times two such strings produces a Dyck string.

**Reversing** a Dyck string produces a Dyck language over the alphabet

\[
([]) \xrightarrow{reversal} \left(\right) \text{open} \quad \right \} \text{close}
\]

Substituting a Dyck phrase for a character, say \( (, \), changes the equivalence class from \( Cl(\varepsilon) \) to \( Cl(\left(\right)) \) i.e., one closed paren in excess.
Noam Chomsky’s Context-Free grammar [1956] (or Bar-Hillel’s Categorial g.) generates the Dyck language:

\[
S \rightarrow SS \quad \text{a phrase is the concatenation of 2 phrases}
\]

\[
S \rightarrow (S) \quad \text{a phrase is a parenthesized phrase}
\]

\[
S \rightarrow \varepsilon \quad \text{a phrase is the empty string}
\]

Word membership/parsing problem: given a string, is it a Dyck string? Deterministic push-down (LIFO) machine:

- Push on reading (
- Pop on reading ) and recognize if empty

Time complexity is linear (real-time).
and ... queues

\begin{parenthesis}
equipped with a FIFO memory, a queue (or Post) machine recognizes the Anti-Dyck language [Vauquelin, Franchi-Zannettacci 1979], where “no parentheses match”. Cancellation rule:

contains no closing parens

\[
\begin{array}{c}
( \overbrace{([ \_ \_ )]} \Rightarrow ([ \_ ]) \\
([ \_ ]) \Rightarrow [] \Rightarrow \varepsilon
\end{array}
\]

Such languages are generated by breadth-first context-free grammars [Allevi, Cherubini, CR 1988].
\end{parenthesis}
Parentheses unwelcome!

When parentheses proliferate they are hard to read. The number of parentheses can be reduced assigning precedences to operators:

\[ 5 \times 3 + 8 \div 3 \times 9 + 7 \text{ instead of } (5 \times 3) + ((8 \div (3 \times 9)) + 7) \]

\[ \times \text{ (and } \div \text{) takes precedence over } + \text{, } + \text{ yields precedence to } \div \text{ (and } \times \text{), } + \text{ yields to } + \text{ (association from right to left)} \]

Some people hate parentheses: Jan Lukasiewicz [1924] would write (without vincula) in reverse polish notation:

\[ 5 \ 3 \times \ 8 \ 3 \ 9 \times \ 7 \div \ 7 \+ \ 7 \+ \]
Floyd [1963]: operator precedence grammars

Idea: between all terminal characters there is a precedence relation:

- yields $\Leftarrow$, takes $\Rightarrow$
- equal-in-precedence, $\equiv$, between opening-closing pairs.

Compilers have extensively used Floyd grammars until the invention of deterministic methods (LL(k) Lewis et al. 1966, and LR(k) Knuth 1966);
Still popular for fast parsing of expressions. Precedence relations are easily computed by grammar inspection.
Example: arithmetic expression with plus and times and with parens.

**Grammar**:

\[
E \rightarrow E + T \mid T, \quad T \rightarrow T \times F \mid F, \quad F \rightarrow (E) \mid a
\]

**Precedence matrix**:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>+</th>
<th>×</th>
<th>(</th>
<th>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td>&gt;</td>
<td>&gt;</td>
<td></td>
<td>&gt;</td>
</tr>
<tr>
<td>+</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>×</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&gt;</td>
<td>&lt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>(</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>≡</td>
</tr>
<tr>
<td>)</td>
<td>&gt;</td>
<td>&gt;</td>
<td></td>
<td>&gt;</td>
<td></td>
</tr>
</tbody>
</table>
Easy: syntax subtrees are delimited by $\langle \ldots \rangle$, and may include subtrees separated by $\div$. No need to perform reductions from left to right.

\[
\begin{align*}
\vdash &\quad a \times (a + a) \vdash \\
\vdash &\quad \langle a \rangle \times \langle (\langle a \rangle + \langle a \rangle) \rangle \vdash \\
\vdash &\quad \langle a \rangle \times \langle (\langle E + T \rangle) \rangle \vdash \\
\vdash &\quad \langle a \rangle \times \langle (\div E) \rangle \vdash \\
\vdash &\quad \langle a \rangle \times F \vdash \\
\vdash &\quad \ldots \quad \ldots \\
\end{align*}
\]

Clearly, $\langle \ldots \rangle$ act as parentheses. I’ll return to Floyd after various parenthesis models.
## Similarities with Regular languages REG

<table>
<thead>
<tr>
<th>Few Properties of REG</th>
<th>preserved by</th>
<th>CF</th>
<th>Deterministic CF (DCF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>all Boolean</td>
<td>UNION</td>
<td>COMPL</td>
<td></td>
</tr>
<tr>
<td>Concatenation</td>
<td>YES</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Kleene Star</td>
<td>YES</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>unique min. det. machine / grammar</td>
<td>NO</td>
<td>NO</td>
<td></td>
</tr>
</tbody>
</table>

Several scientists looked for “better” subfamilies of DCF:

- Parenthesis grammars [McNaughton 1967, Knuth 1967], Tree automata [Thatcher 1967]
- Balanced Grammars [Berstel & Boasson 2002]
- Visibly Push-Down Automata VPD [Alur & Madhusudan 2004]
Parentheses induce well-nested structures on strings. CF grammar rules are parenthesized [McNaughton]:

\[ E \rightarrow E + E \mid v \quad \text{becomes} \quad E \rightarrow (E + E) \mid (v) \]

The ambiguous phrase corresponds to different paren phrases

\[ ((v + v) + v) \quad (v + (v + v)) \]

**Parentheses Languages** PL are DCF and math. similar to REG.

Very similar to **tree languages** [Thatcher 1967]
REG-like properties of parentheses grammars

- Uniqueness of minimal grammar (in backwards-deterministic form).
- Grammars having the same set of rule patterns (stencils) define a Boolean algebra of languages.
- Non-Counting (aperiodic) REG languages [McNaughton, Papert, Schützenberger] have counterparts within parentheses languages [CR, Guida, Mandrioli 1978]

Similar definitions and properties have been stated [Thomas] in the framework of tree automata.
Surprisingly Dyck is not a parenthesis language:

\[
( ) ( ) \\
\text{missing external parens}
\]

\[
( ( ) ( ) ) \ldots ( ) \\
\text{unbounded}
\]

PL are not closed under concatenation and Kleene star. [Knuth 1967] asked: is a given CF language a parenslanguage? The answer involves an equivalent definition of well-parenthesizing for an alphabet including parens and possibly other “internal” letters.

A string is balanced if

- # open parens = # of closed parens
- in every prefix, # open parens ≥ # closed parens

Dyck phrases are exactly the balanced strings.
Letters associated to open paren

Every letter in a string is an associate of an open paren:

\[
\left( 0 \ c_1 \ (2 \ c_3) \ 4 \ (5) \ 6 \right)_7
\]

letter 1 is associate of \(0\), letter 3 of \(2\), letter 4 of \(0\)

[Knuth] A CF language is a parens language iff

- every phrase is balanced and
- every open parens has bounded number of associates.

\[
S \rightarrow XY \quad X \rightarrow (c)X \mid d \quad Y \rightarrow (Y(c)) \mid e
\]

is a parens language, though grammar is not parenthesized.
Paren nesting in human and artificial languages

- Natural languages rarely exhibit deeply nested structures
- Although in principle they are possible
  
  \[
  \text{der Mann der die Frau die das Kind das die Katze füttert sieht liebt schlägt.}
  \]
- Inner clauses are rarely marked by paresns or by words acting as opening / closing tags
- Good writers moderately use *parentheticals*, because they depart from the main subject
Parens in computer languages

All programming languages have parenthetical constructs, perhaps exaggeratedly so in Algol 68

begin...end, do...od, if...fi, case... esac

Mark-up or semi-structured (web) documents (e.g. XML) are deeply and widely nested; visible open/close tags delimit structures:

```xml
<div id="accessorapido">
  <ul>
    <li><a href="#barranavigazione">Navigazione</a></li>
    <li><a href="#avvisi">Avvisi</a></li>
    <li><a href="#contenutoprincipale">cont</a></li>
    <li><a href="#barrainformazzioniaggiuntive">informazi</a></li>
  </ul>
</div>
```
Balanced grammars [Berstel & Boasson 2002]

Allow RegExpr in right-hand sides of rules:

\[ S \rightarrow (Y^*) \quad Y \rightarrow [ ] \]

phrases: \( () \), \( ([ ]) \), \( ([ ][ ]) \), \( ([ ][ ][ ]) \), …

Several properties of regular languages hold for balanced languages:
Boolean closures, uniqueness of minimal grammar.

An elegant formalism for XML-like languages.
Visibly PushDown languages [Alur & Madhusudan 2004]

Restricted type of deterministic pushdown machine. Motivated by:

- model-checking of programs (i.e. \(\infty\) state systems)
- XML

VPD Push-Down Machine:

- pushes a call (=open) letter, changing state
- pops on a return (=close) letter, if a matching call letter is on top, changing state
- changes state on a return letter, if stack empty
- on an internal letter, changes state without using stack
- accepts by final state.
VPD versus balanced lang. and REG

- 3-partite alphabet: \( \Sigma_{\text{call}} \cup \Sigma_{\text{return}} \cup \Sigma_{\text{internal}} \)

- no bijection open-close: e.g., ( ] and ( )

- unbalanced returns may occur as prefix of a word, and unbalanced calls as suffix

- REGULAR \( \subset \) BALANCED \( \subset \) VPD \( \subset \) Deterministic CF

- determinization, minimization, uniqueness

- closure and decidability properties \( \approx \) to REGULAR

- real-time deterministic parsing
Examples

- Dyck equivalence class ‘)’: 
  \[( ( )) ] \text{ parsed as } \left( ( ) \right) \text{ not as } ( ( ) )

- Program execution modelled by VPD, e.g., abnormal termination of procedure call:
  \[ c_{main} c_A c_B \hat{\text{instruction}} \cdots r_B c_B \hat{\text{instr.}} \hat{\text{except.}} \]
Plenty of research on VPD

- Monadic Second Order Logic of VPD languages
- Grammatical formulations of VPD have been provided
- \( \omega \) (infinitary) languages
- Bisimulation equivalence
- Decidable problems
- VPD games
- Complexity of membership problem w.r.t. grammar size [La Torre, Napoli, Parente 2006]
- Comparison with synchronized [Caucal] and height-deterministic [Nowotka & Srba] languages
But are VPD languages that new?

- The precedence relations of VPD languages have a particular form
- A FL grammar with such form of precedences generates a VPD lang.
- The same formal properties hold for FL and VPD
  - Boolean closure [CR, Mandrioli, Martin 1978]
  - Reversal
  - Closures under concatenation, star, suffix, prefix [CR & Mandrioli 2010]
Precedence relations of VPD

Structure of FL grammar of a VPD lang.:

Precedence relations:

<table>
<thead>
<tr>
<th></th>
<th>$c \in \Sigma_{call}$</th>
<th>$r \in \Sigma_{return}$</th>
<th>$s \in \Sigma_{internal}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Sigma_{call}$</td>
<td>$&lt;$</td>
<td>$\equiv$</td>
<td>$&lt;$</td>
</tr>
<tr>
<td>$\Sigma_{return}$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
</tr>
<tr>
<td>$\Sigma_{internal}$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
</tr>
</tbody>
</table>

A language is VPD iff it is generated by a FL grammar with such precedences.
Limitations of VPD

- Open / close tags must be letter-disjoint
  
  \[
  \underbrace{a \ a \ \ldots \ b \ b}, \quad \underbrace{c \ c \ \ldots \ d \ d}
  \]

  contradict

  \[
  \underbrace{e \ e \ \ldots \ ac \ ac}
  \]

- Fixed syntax structure in many cases not structurally adequate
  
  bad \[ \underbrace{3 + 5 \times 7} \]  \quad good \[ \underbrace{3 + 5 \times 7} \]

Both cases are correctly handled by Floyd grammars.
Floyd lang. as generalized parens lang.

- Functional notation uses nested parens:
  \[ ADD(a, MULT(c, ADD(d, e))) \]

- more readable in *infix notation* with precedences:
  \[ a \ ADD b \ MULT ( c \ ADD d) \]

- For ternary operators
  \[ IF\_THEN\_ELSE(c_1, s_2, s_3) \]

  *mixfix notation* imitates natural language

\[ IF \ c_1 \ THEN \ s_2 \ ELSE \ s_3 \]

with precedences:

- \[ IF \div \ THEN \div \ ELSE \]
- \[ IF < \ 1\text{st symbol of} \ c_1 \]
- \[ \text{last symbol of} \ c_1 > \ THEN \]
Conclusione semiseria

[dal Breve glossario di retorica e metrica]
Parentesi o frase incidentale è l’aggiunta di elementi non necessari o di precisazioni all’interno di una frase. È segnalata dalle parentesi o dalle virgole.

Sono contrito di avervi intrattenuto per un’ora parlandovodi elementi non necessari!
O forse l’informatica teorica è la scienza del non-necessario . . . ma illuminante?