A century of parentheses languages

with some amazing returns

ICTCS 2010, Camerino

Stefano Crespi Reghizzi

Politecnico di Milano

Parentheses in mathematical notation

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

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

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.

Dyck's language

The alphabet has just two letters: the open/close parentheses $(,)$ or $begin$, end, etc.
The language $Cl\left(\varepsilon\right)$ is the equivalence class of all strings such that repeated deletions of well-parenthesized digram $()$ reduce the string to the empty one ε.

$$(()())() \Rightarrow (()\cancel{()})() = (())() \Rightarrow (\cancel{()})() \Rightarrow \cancel{()}() \Rightarrow \cancel{()} = \varepsilon$$

The equivalence class such that, after all deletions, the string, say, $))$ is obtained is another formal language: $Cl\left(\;))\;\right)$ instead of $Cl\left(\varepsilon\right)$.

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

$$([\,])\ \overset{reversal}{\Longrightarrow}\quad \overbrace{\ )\ }^{open}\ ]\,[\ \overbrace{\ (\ }^{close}$$

Substituting a Dyck phrase for a character, say $($, changes the equivalence class from $Cl(\varepsilon)$ to $Cl(\,\big)\,)$ i.e., one closed paren in excess.

Dyck's languages, grammars, push-downs and ...

Noam Chomsky's Context-Free grammar [1956] (or Bar-Hillel's Categorial g.) generates the Dyck language:

$$S \rightarrow S\ S \qquad \text{a phrase is the concatenation of 2 phrases}$$

$$S \rightarrow \Big(S \Big) \qquad \text{a phrase is a parenthesized phrase}$$

$$S \rightarrow \varepsilon \qquad \text{a phrase is the empty string}$$

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:

$$\underline{(}\ \overbrace{(\ [}^{\text{contains no closing parens}}\ \underline{)}\ )\ ] \Longrightarrow (\ [\ )\ ]$$

$$\underline{(}\ [\ \underline{)}\ ] \Longrightarrow [\ ] \Longrightarrow \varepsilon$$

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 <u>precedences</u> to operators:

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

$\times$ (and $\div$) takes precedence over $+$	$\times \gtrdot +$
$+$ yields precedence to $\div$ (and $\times$)	$+ \lessdot \div$
$+$ yields to $+$ (association from right to left)	$+ \lessdot +$

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

$$\overline{5\;3\times}\;\;\overline{\overline{8\;\overline{3\;9\times}\;\div}\;7+}\;+$$

Floyd [1963]: operator precedence grammars

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

- yields $\lessdot$,   takes $\gtrdot$
- equal-in-precedence, $\doteq$, 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.

Parsing with precedence 1

Example: arithmetic expression with plus and times and with parens.

Grammar : $E \to E + T \mid T, \qquad T \to T \times F \mid F, \qquad F \to (E) \mid$

Precedence matrix:

	a	$+$	$\times$	$($	$)$
a		$\gtrdot$	$\gtrdot$		$\gtrdot$
$+$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\gtrdot$
$\times$	$\lessdot$	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
$($	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$
$)$		$\gtrdot$	$\gtrdot$		$\gtrdot$

Parsing with precedence 2

Easy: syntax subtrees are delimited by $\lessdot \ldots \gtrdot$, and may include subtrees separated by $\doteq$. No need to perform reductions from left to right.

$$
\begin{array}{ccccccccc}
\vdash & a & \times & (& a & + & a &) & \dashv \\
\vdash \lessdot & a \gtrdot & \times \lessdot & (& \lessdot a \gtrdot & + \lessdot & a \gtrdot &) \gtrdot & \dashv \\
\vdash \lessdot & a \gtrdot & \times \lessdot & (\lessdot & E & + & T & \gtrdot) \gtrdot & \dashv \\
\\
\vdash \lessdot & a \gtrdot & \times \lessdot & (\doteq & & E & &) \gtrdot & \dashv \\
\vdash \lessdot & a \gtrdot & \times & & & F & & \gtrdot & \dashv \\
\ldots & \ldots & & & & & & &
\end{array}
$$

Clearly, $\lessdot \ldots \gtrdot$ act as parentheses.
I'll return to Floyd after various parenthesis models.

Similarities with Regular languages REG

Few Properties of REG	preserved by	CF	Deterministic CF (DCF)
all Boolean		UNION	COMPL
Concatenation		YES	NO
Kleene Star		YES	NO
unique min. det. machine / grammar		NO	NO

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 grammars and tree languages

Parentheses induce well-nested structures on strings.
CF grammar rules are parenthesized [McNaughton]:

$$E \to E + E \mid v \quad \text{becomes} \quad E \to \big(E + E\big) \mid \big(v\big)$$

The ambiguous phrase $\overbrace{\overbrace{v+v}^{E}+v}^{E}$ $\quad$ $\overbrace{v+\overbrace{v+v}^{E}}^{E}$
corresponds to different paren phrases

$$((v+v)+v) \qquad (v+(v+v))$$

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, Schutzenberger] have counterparts within parentheses languages [CR, Guida, Mandrioli 1978]

Similar definitions and properties have been stated [Thomas] in the framework of tree automata.

Further on: balanced strings and grammars 1

Surprisingly Dyck is not a parenthesis language:

$$\underbrace{(\,)(\,)}_{\text{missing external parens}} \qquad (\,\underbrace{(\,)(\,)(\,)\ldots(\,)}_{\text{unbounded}}\,)$$

PL are not closed under concatenation and Kleene star. [Knuth 1967] asked: is a given CF language a parens language? 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 $\geq \#$ closed parens

Dyck phrases are exactly the balanced strings.

Letters associated to open paren

Every letter in a string is an <u>associate</u> of an open paren:

$$(_0\, c_1\, (_2\, c_3\,)_4 (_5\,)_6\,)_7$$

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.

$$S \to XY \qquad X \to (\,)c)X) \mid (d \qquad Y \to (Y(c)) \mid e)$$

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

  $der\ Mann\ der\ die\ Frau\ die\ das\ Kind\ das\ die\ Katze\ füttert\ sieht\ liebt\ schlä$
- inner clauses are rarely marked by parens 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

$$begin \ldots end, \qquad do \ldots od, \qquad if \ldots fi, \qquad case \ldots esac$$

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

```
<div id="accessorapido">
    <ul>
    <li><a href="#barranavigazione">Navigaz
    <li><a href="#avvisi">Avvisi</a></li>
    <li><a href="#contenutoprincipale">cont
    <li><a href="#barrainformazioniaggiunti
    </ul>
</div>
```

alanced grammars [Berstel & Boasson 200

Allow RegExpr in righhand sides of rules:

$$S \rightarrow \big(Y^*\big) \qquad Y \rightarrow \big[\,\big]$$

phrases: $(),\ ([\,]),\ ([\,]\,[\,]),\ \ldots$

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. ∞ state systems)
- XML

VPD Push-Down Machine:

- pushes a <u>call</u> (=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: $\overbrace{\Sigma_{call}}^{\text{open parens}} \cup \overbrace{\Sigma_{return}}^{\text{closed parens}} \cup \overbrace{\Sigma_{internal}}^{\text{others}}$
- no bijection open-close: e.g., $(\,]$ and $(\,)$
- unbalanced returns may occur as prefix of a word, and unbalanced calls as suffix

$$\overbrace{r_1 r_2}^{\text{unbal. returns}} \quad \overbrace{c_1 c_1 \ldots r_1 r_1}^{\text{balanced}} \; \underbrace{i_1}_{\text{internal}} \; \overbrace{c_1 c_1 \ldots \underbrace{i_2}_{\text{internal}} \; r_2 \ldots \underbrace{i_2}_{\text{internal}} \; r_2}^{\text{balanced}} \overbrace{c_2 \ldots c_2}^{\text{unbal. calls}}$$

- 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 '$)$':

$$(\,(\,)))\text{ parsed as }\overbrace{(\,(\,))}\,)\text{ not as }(\,\overbrace{(\,))}\,)$$

- Program execution modelled by VPD, e.g., abnormal termination of procedure call:

$$c_{main}c_A\,\overbrace{c_B\,i_{instruction}\cdots r_B}\,c_B i_{instr.} i_{except.}$$

Plenty of research on VPD

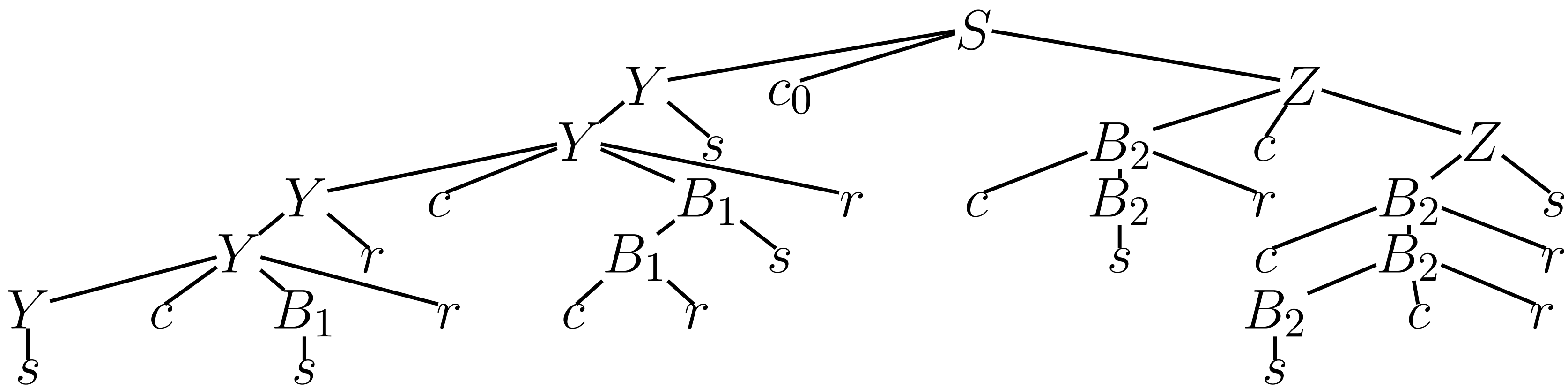
- Monadic Second Order Logic of VPD languages
- Grammatical formulations of VPD have been provided
- ω (infinitary) languages
- Bisimulation equivalence
- Decidable problems
- VPD games
- Complexity of membership problem w.r.t. grammar size [La Torre, Napoli, Parente 2006]
- Comparison with synchronized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

But are VPD languages that new?

- Floyd [1963] Operator Precedence [1963] Languages FL strictly include VPD [CR & Mandrioli 2009]
- 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:

	$c \in \Sigma_{call}$	$r \in \Sigma_{return}$	$s \in \Sigma_{internal}$
Σ_{call}	$\lessdot$	$\doteq$	$\lessdot$
Σ_{return}	$\gtrdot$	$\gtrdot$	$\gtrdot$
$\Sigma_{internal}$	$\gtrdot$	$\gtrdot$	$\gtrdot$

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

$$\overbrace{a\ \overbrace{a\ \ldots\ b}\ b} \qquad \overbrace{c\ \overbrace{c\ \ldots\ d}\ d}$$

contradict

$$\overbrace{e\ \overbrace{e\ \ldots\ \color{red}{ac}}\ \color{red}{ac}}$$

- fixed syntax structure in many cases not structurally adequate

bad $\overbrace{\overbrace{3 + 5} \times 7}$ good $\overbrace{3 + \overbrace{5 \times 7}}$

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 $\underline{\text{ADD}}$ b $\underline{\text{MULT}}$ (c $\underline{\text{ADD}}$ d)
- For ternary operators
  $IF\_THEN\_ELSE(c_1, s_2, s_3)$
  mixfix notation imitates natural language

$$\underline{IF}\ c_1\ \underline{THEN}\ s_2\ \underline{ELSE}\ s_3$$

  with precedences:
  - $\underline{IF} \doteq \underline{THEN} \doteq \underline{ELSE}$
  - $\underline{IF} \lessdot$ 1st symbol of c_1
  - last symbol of $c_1 \gtrdot \underline{THEN}$

Conclusione semiseria

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

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