An enduring trail of language characterizations via homomorphism

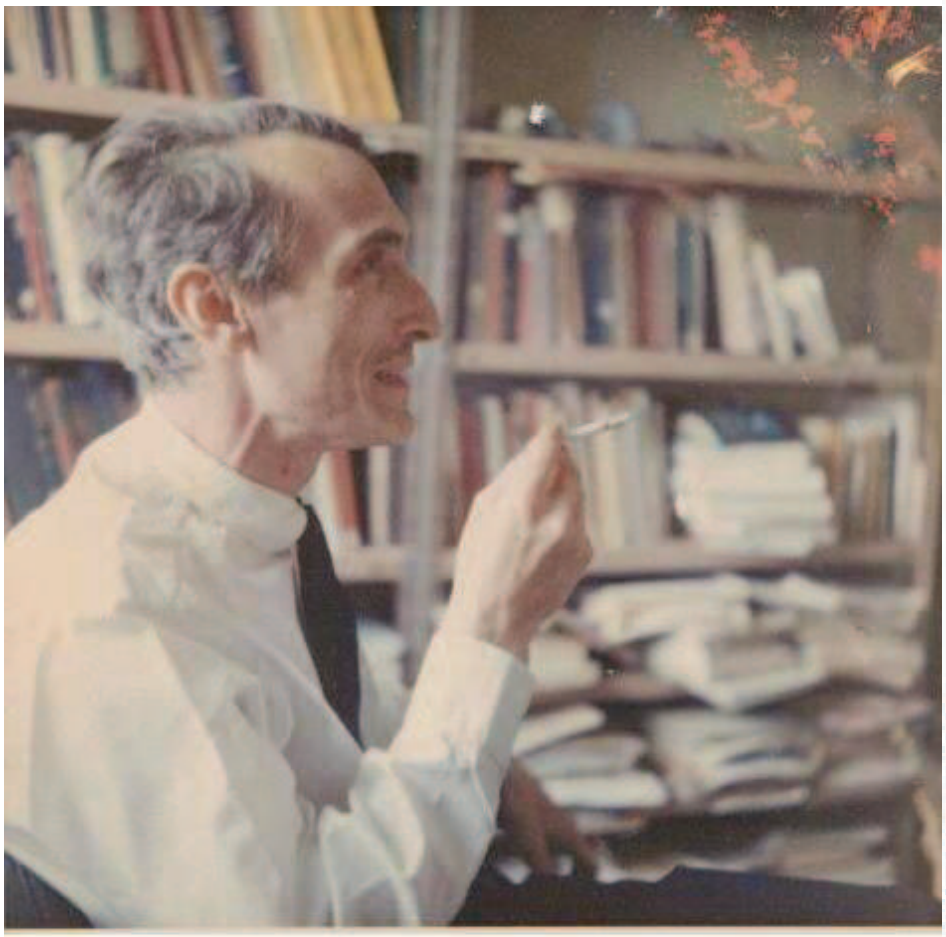
Stefano Crespi Reghizzi
and Pierluigi San Pietro

DEIB - Politecnico di Milano

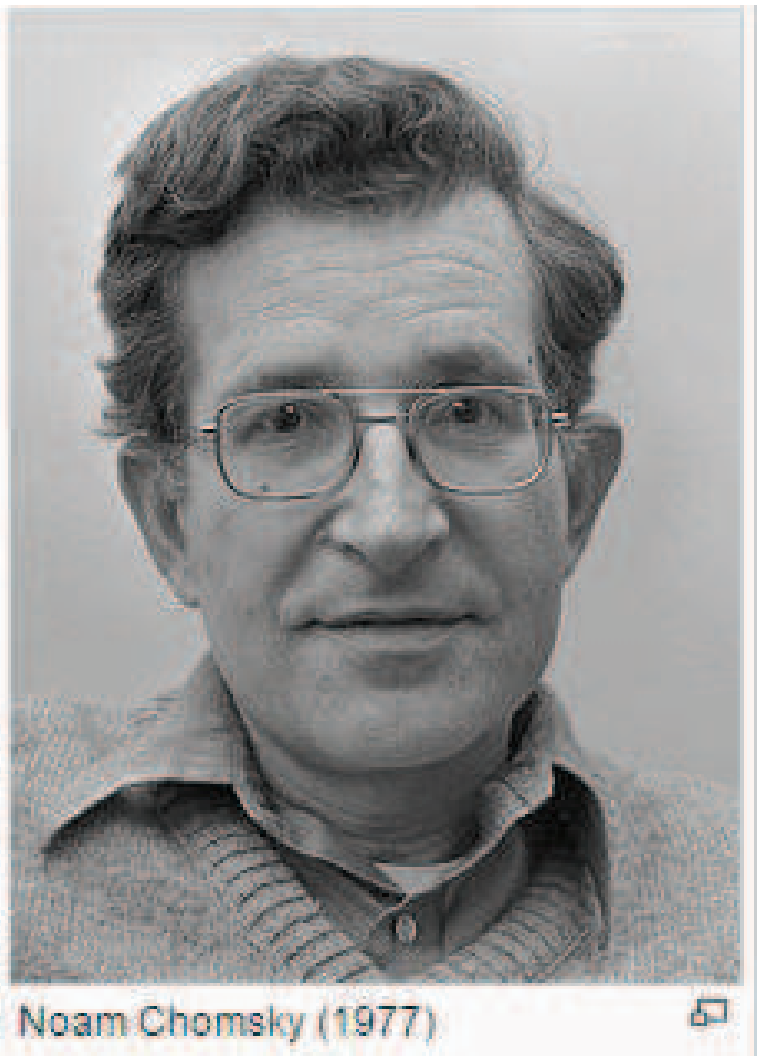
stefano.crespireghizzi@polimi.it

Conf. dedicated to the scientific legacy of M. P. Schützenberger, Bordeaux, March 2016

Schützenberger and Chomsky and their Representation Theorem



1960, environ. Antony. Marco dans son bureau

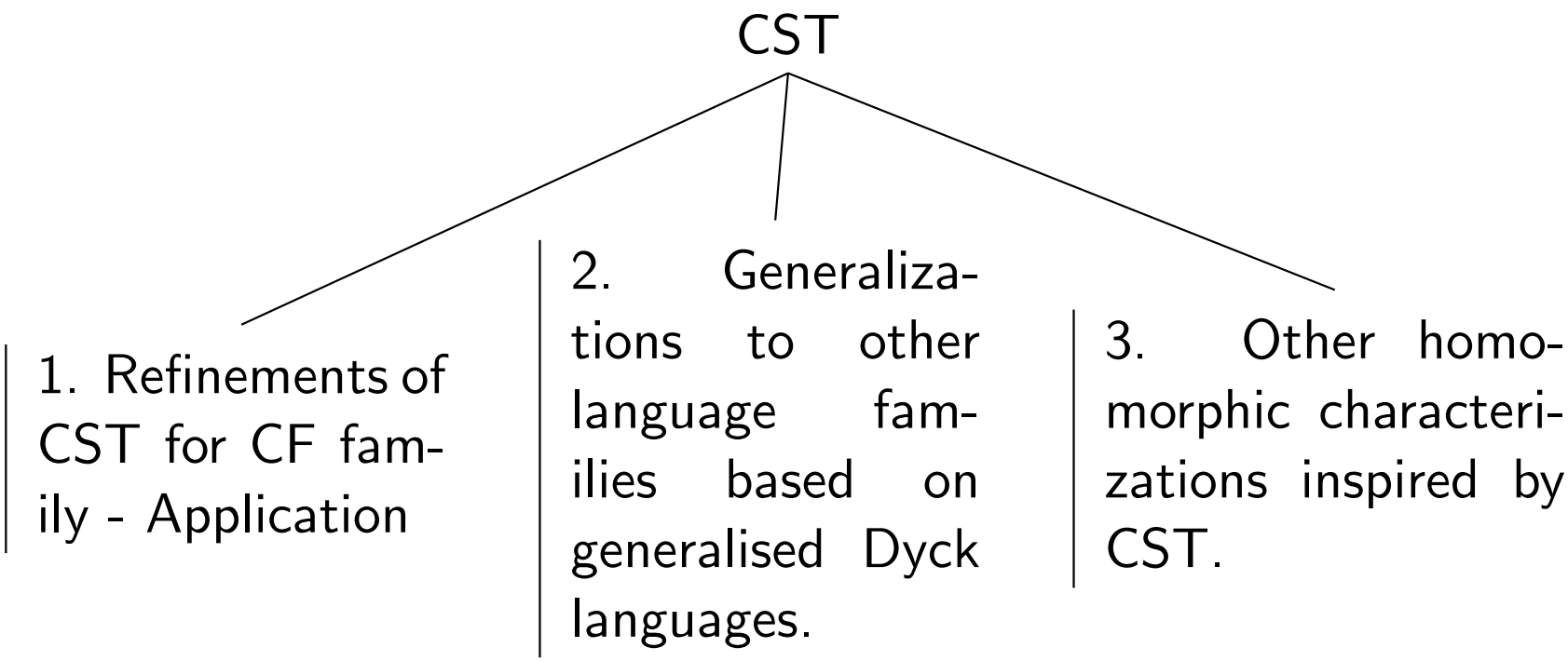


Noam Chomsky (1977)

Outline: Chomsky-Schützenberger theorem CST

- Parentheses, brackets and parenthetical constructs occur as a syntactic constituent in natural language, math, animal behavior; foremost, in logical formulas and computer languages.
- Walter von Dyck defined pure parenthetical languages as congruence classes of words
- Noam Chomsky's phrase-structure grammars, of which type 2 (Context-Free) is the most successful, generate all sorts of tree-like syntactic constructs, including Dyck languages
- His "representation theorem" with Schützenberger [1962], CST : CF languages coincide with the languages obtained from a Dyck language by first intersecting with a regular (finite-state) language that acts as filter, then transliterating or deleting each parenthesis.

Outline: developments and descendants of CST



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:

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

- 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.

Parenthetical constructs in artificial languages

- All *programming languages* have parenthetical constructs, perhaps exaggeratedly so in Algol 68, where open/closed brackets have palindromic codes:

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

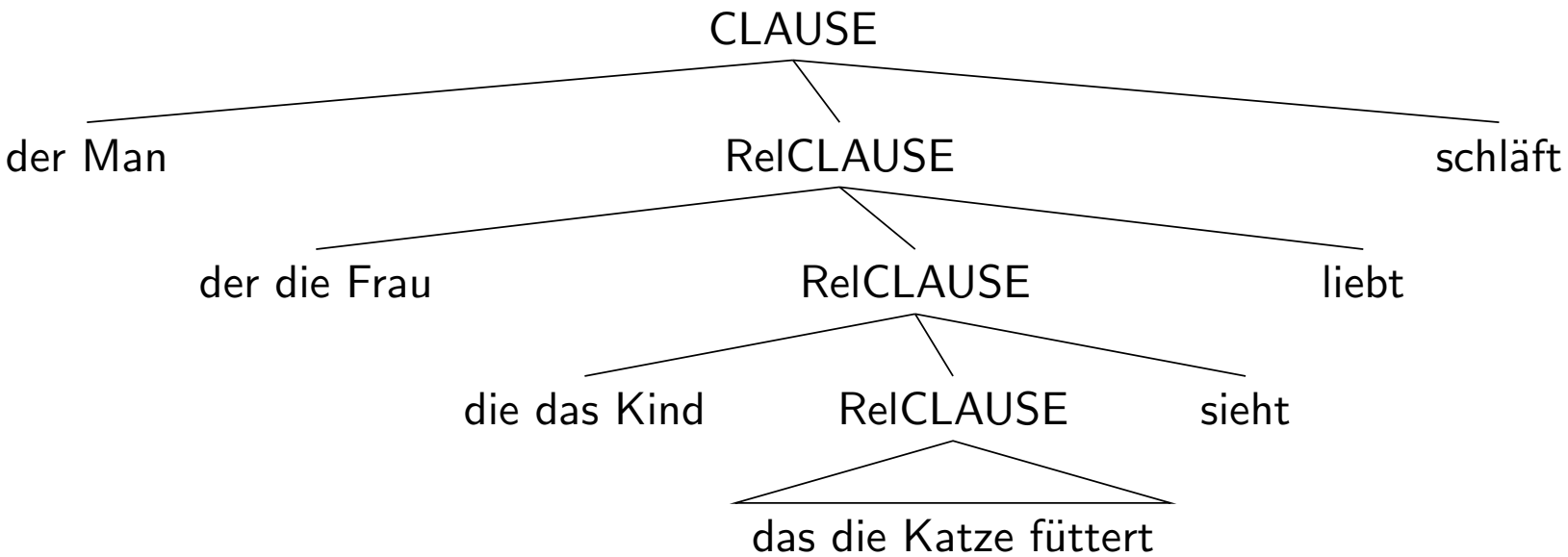
- J. McCarthy [1958] LISP language uses only one type of parentheses, but plenty of them to the disappontment of programmers who frequently forget some or add too many:

```
(if nil
    (list 1 2 "foo")
    (list 3 4 "bar")) )
```

- Many web documents are in the XML notation, which encloses each field between specific tags:
  $< /TITLE > \ldots < /TITLE >$

Parenthetical structures in human languages

Natural language sentences rarely exhibit deeply nested structures although they are grammatically correct, e.g. :



Inner clauses are variously delimited by words acting as opening / closing tags. Here relative pronouns open and verbs close.
A long-standing debate among linguists: what are the essential features of a parenthetical?

Dyck's language as congruence class

- The alphabet Σ is bipartite into open-closed brackets: $(,)$, $[,]$, $\{,\}, \ldots$
  (Algebra denotes brackets as $x_i, \bar{x}_i$)
- In computer science, the Dyck language with $k \geq 1$ bracket pairs, D_k, is such that the brackets are well-balanced, in the following sense.

Pair deletion rule

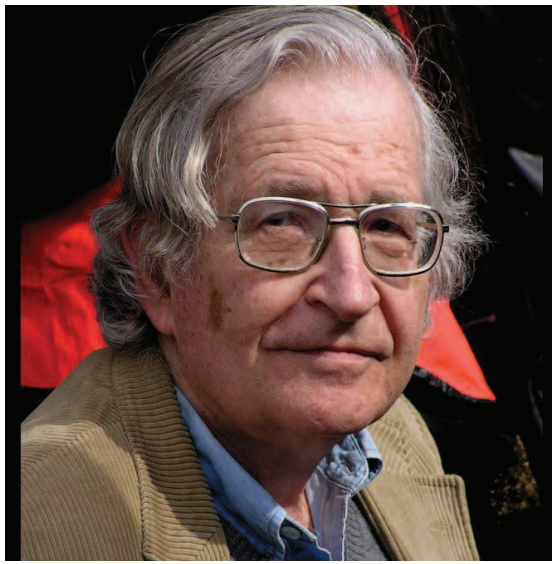
D_k is the congruence class of all words such that repeated deletions of an adjacent symbol pair, such as $[\,]$, reduce the word to the empty one, ε:

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

The equivalence $x_i\,\bar{x}_i \sim \varepsilon$ generates the congruence.

Context-Free grammars, push-down machine

1956 Noam Chomsky invented the *type 2* or CF grammars made by rewriting rules, later renamed by computer scientists BNF grammars. He studied the first algorithm to recognize if a word is "grammatical".



- The algorithm is a special Turing machine called push-down automaton PDA, with a finite-state RAM memory and an unbounded "push-down stack" memory (Last-In-First-Out)
- Language families generated by C.F. grammars and recognized by non-deterministic PDAs coincide.

Grammars, push-downs and Dyck languages

Chomsky's Context-Free grammar of Dyck language:

$$S \to \left(S\right) S \qquad \text{a phrase is the concatenation of a bracketed}$$
$$S \to \left[S\right] S \qquad \text{phrase and a phrase;}$$
$$S \to \varepsilon \qquad \text{a phrase is the empty word;}$$

Word membership/parsing problem: input word, is a Dyck phrase?

Deterministic push-down machine recognizes Dyck words:

- On reading (or [, push stack symbol $A_)$ or $A_]$;
- On reading), if $A_)$ is on top of stack, pop, i.e., remove top symbol;
- At the end, recognize the word if stack empty.

The idea of Chomsky and Schützenberger [1962]

- A CF language is typified by various constructs: lists of items, ternary structures **if** ... **then** ... **else** ..., and also well-nested structures **begin** ... **end**
- But Dyck purely consists of nested and concatenated brackets.
- Can the ∞ variety of constructs be obtained, starting from Dyck and
  - constraining the brackets that are adjacent
  - transliterating each bracket to some character of the language alphabet? Yes!

Every CF grammar is a combination of

- local constraints on well-formedness expressible as a finite-state language, and
- global constraints expressing well-nestedness of some elements.

Dyck language: the seed of every CF language

The alphabets are: Σ for lang. L and Δ for lang. Dyck.

Chomsky Schützenberger Theorem, CST

L is context-free if, and only if

$\exists$ Dyck language D, $\exists$ regular language R

$\exists$ alphabetic transliteration τ from Δ to Σ such that

$$L = \tau(D \cap R)$$

First, lang. R acts as filter to remove useless strings from D. Then transliteration (homomorphism) maps a bracket on a different symbol of Σ or nothing.

CST Example - Secondary school

- In secondary schools 3 layers of parentheses admitted. Consider such language L, for brevity without curly braces:
- From outermost to innermost: ~~curly~~, square , round

bad: $(( )\,[])$ good: $[(\,)\,(\,)]\,[\,]$

Filter by forbidding adjacencies: $([,\ )[,$, and also ((and [[if nesting of round-round and square-square are excluded.

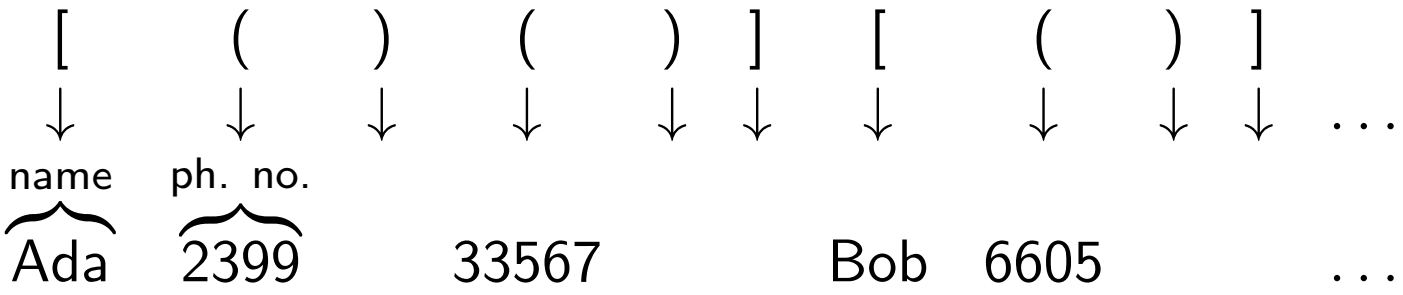
This veto is easily expressed by a *regular* language

$$R = \overline{\underbrace{\Sigma^* \cdot ([\cdot \Sigma^*}_{\text{strings without ([}} \quad \cup \quad \Sigma^* \cdot](\cdot \Sigma^*}$$

$L = D_2 \cap R$ Transliteration is not needed in this case since the alphabets of D_2 and L coincide and all brackets are preserved.

CST Example - My phone contacts

The same Dyck language D_2 (alphabet $\Delta = \{(,), [,]\}$, with a different filter and transliteration, generates the language L of my phone contacts:



Filter R vetoes the adjacent pairs: ((, ([, [[,](

Transliteration τ is :

symbol	τ
[	name
(	ph. no.
] and)	null

Chomsky & Schützenberger presentation i

STUDIES IN LOGIC

AND THE

FOUNDATIONS OF MATHEMATICS

L. E. J. BROUWER / E. W. BETH / A. HEYTING

EDITORS

Computer Programming and Formal Systems

Editors

P. BRAFFORT

and

D. HIRSCHBERG

NORTH-HOLLAND PUBLISHING COMPANY

AMSTERDAM

THE ALGEBRAIC THEORY OF CONTEXT-FREE LANGUAGES*

N. CHOMSKY

Massachusetts Institute of Technology

AND

M. P. SCHÜTZENBERGER

Harvard University

1. LINGUISTIC MOTIVATION

We will be concerned here with several classes of sentence-generating devices that are closely related, in various ways, to the grammars of both natural languages and artificial languages of various kinds. By a *language* we will mean simply a set of strings in some finite set V of symbols called the *vocabulary* of the language. By a *grammar* we mean a set of rules that give a recursive enumeration of the strings belonging to the language. We will say that the grammar *generates* these strings. (Thinking of natural languages, we would call the generated strings *sentences*; in algebraic parlance they would ordinarily be called *words* and the vocabulary would be called an *alphabet*; regarding a grammar as specifying a programming language, the strings would be called *programs*; we will generally use the neutral term *strings*).

For a class of grammars to have linguistic interest, there must be a procedure that assigns to any pair (σ, G), where σ is a string and G a grammar of this class, a satisfactory *structural description* of the string σ with respect to the grammar G. In particular, the structural description should indicate that the string σ is a well-formed sentence of the language $L(G)$ generated by G, where this is the case. If it is, the structural description should contain grammatical information that provides the basis for explaining how σ is understood by speakers who have internalized the grammar G; if it is not, the structural description might indicate in what respects σ deviates from well-formedness.

* This work was supported in part by the U. S. Army Signal Corps, the Air Force Office of Scientific Research, and the Office of Naval Research; and in part by the National Science Foundation; and in part by a grant from the Commonwealth Fund.

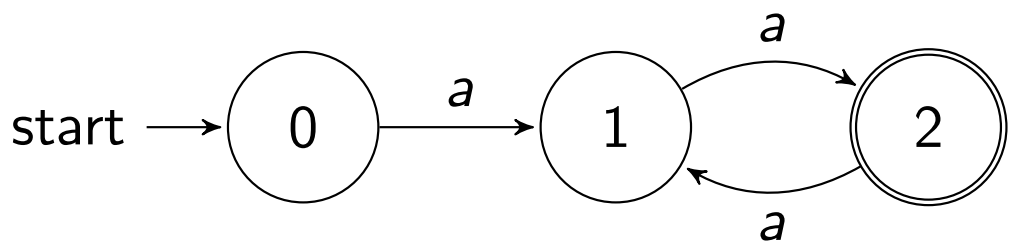
"An alternative characterization of families of CF languages" I

Proposition 1 [now known as Y. Medvedev Theor. but Chomsky gives credit for it to Schütz.]

For any regular event [now language] B on alphabet Σ we can find a standard regular event [now Strictly Locally Testable lang. of degree 2] A on alphabet Δ and a homomorphism [letter-to-letter transliteration] $h : \Delta \to \Sigma$ such that $B = h(A)$.

The immediate proof is suggested by the example:

"An alternative characterization of families of CF languages" II



$B = (aa)^+$

- $A =$ set of words over $\Delta = \{\langle 0,1,a\rangle, \langle 1,2,a\rangle, \langle 2,1,a\rangle\}$ s.t.: start with $\langle 0,1,a\rangle$, end with $\langle 1,2,a\rangle$, and only contain the pairs $\langle 0,1,a\rangle\,\langle 1,2,a\rangle$, $\langle 1,2,a\rangle\,\langle 2,1,a\rangle$, $\langle 2,1,a\rangle\,\langle 1,2,a\rangle$ as substrings.
- Such languages specified by a *sliding window of width 2* are 2-Strictly Locally Testable
- Homomorphism: $h(x) = a$ for all $x \in \Delta$.

"An alternative characterization of families of CF languages"- C.S.T. I

They say: "We can generalize *Proposition* 1 to CF languages"

Proposition 2

Any CF language L on alphabet Σ is given by an integer k, a standard regular event R on alphabet $\Delta_k = \{x_i, \bar{x}_i \mid 1 \leq i \leq k\}$, a [letter-to-letter, erasing] homomorphism $h : \Delta_k \to \Sigma \cup \{\varepsilon\}$, and the rule

$$L = h(D_k \cap R)$$

- The wording did not care to explicit the obvious fact that any language defined by this equation is CF. This has occasionally induced others to claim a CST property also in cases where the statement holds only in one direction.

"An alternative characterization of families of CF languages"- C.S.T. II

- If L is a *regular* language, Dyck is replaced by the free monoid Δ^* (Medvedev Theor.)
- The original proof has been essentially reproduced many times in books and papers.
  - Label all grammar rules and pick as many brackets for the Dyck language
  - Write a CF grammar that has bracketed rules: $A \to [_5 BC]_5$ and $A \to [_3\, c\,]_3$
  - Two consecutive steps in a leftmost derivation create a pair of permitted adjacent left brackets.

    A pair of adjacent] [brackets is created in the position $B \overset{\downarrow}{\cdot} C$. All other bracket pairs are forbidden by the 2-SLT language R.
  - Transliterate each $[_3$ to c and erase all other brackets.

Parameters of proposition $L = h(D_k \cap R)$ I

alphabet size, type of homomorphism, subclass of regular language

The parameters more investigated are: Dyck alphabet size, whether the homomorphism is erasing, and their relationship.

	Dyck alphabet versus erasing	homomorphism h	
		erases some brackets	doesn't erase
alphabet $\|D_k\|$	depends on $\|G\|$	Ch. & Schüt. [1962] J. Berstel [1979]	A. Okhotin [2012]
	depends on $\|\Sigma\|$ but not on $\|G\|$	R. Stanley [1965]	C-R & P. San Pietro [2016]

Doing without erasures and having a grammar-independent alphabet can be viewed as a tighter specification of the language.

Parameters of proposition $L = h(D_k \cap R)$ II

alphabet size, type of homomorphism, subclass of regular language

Less attention has been given to the regular language.

$R \in$ subclass of regular languages	
strictly locally testable with sliding window width 2; or width > 2	regular languages generated by *simple linear CF grammars*, "flower automata", S. Hirose & M. Yoneda [1985]

Refinements of proposition $L = h(D_k \cap R)$ I

- Ch. & Schüt. [1962] assume the numbered grammar rules are identified by a number and are in Chomsky normal form:
  $\lambda_1 : A \to B\,C,\ \lambda_2 : A \to a$.
  Dyck grammar rules are : $A \to [_\lambda B\,C\,]_\lambda$
- R. Stanley [1965] is similar to Ch. & Schüt. but each bracket with label λ is encoded by $\lambda + 2$ brackets:

$$\begin{array}{c|c} [_\lambda &]_\lambda \\ \hline [_1 \underbrace{[\ldots[}_{\lambda}\ [_2 &]_2\ \underbrace{]\ldots]}_{\lambda}\ ]_1 \end{array}$$

Terminal rules $A \to a$ are bracketed as $A \to [_a\, a\,]_a$. Hence the number of bracket pairs is $|\Sigma| + 3$. Many more brackets are erased by homomorphism h than in CST construction.

Refinements of proposition $L = h(D_k \cap R)$ II

- Berstel [1979] mentions that using rules in Greibach normal form, $A \to aB_1B_2 \dots B_m$, the number of brackets erased to obtain a word $w \in L$ is linearly bounded by the length of w.
- Okhotin [2012] assumes rules are in Double Greibach normal form $A \to aB_1B_2 \dots B_m b$.
  - For *even length* words it is simple to use the bracketing $A \to [_\lambda B_1B_2 \dots B_m]_\lambda$ and the letter-to-letter homomorphism $[_\lambda \xrightarrow{h} a, \ ]_\lambda \xrightarrow{h} b$.
  - For odd length words, a *non-erasing letter-to-word* homomorphism is used.
- C-R & P. San Pietro [2016] also assume *Double Greibach normal form* and
  - encode each of Okhotin brackets with an $m-$ *tuple* of brackets, taken over an alphabet of $j \geq 2$ digits, i.e., using a base j representation

Refinements of proposition $L = h(D_k \cap R)$ III

- by choosing m large enough, the number j, i.e. the Dyck alphabet size, is polynomial in the size of alphabet Σ.
- approach is analogous to the restatement of Medvedev Theorem in C-R & P. San Pietro [2012] using an alphabet independent from the NFA size.
- regular language R remains SLT, but with sliding window of width > 2.

- Hirose & Yoneda [1985] use for language R another subclass of regular languages, those generated by a minimal linear CF grammar, where "minimal" means *one* nonterminal. The corresponding NFA has loops in q_0 and from q_0 straight-line paths to final states.

Subclasses of context-free languages I

Ch. & Schütz. observe that by forbidding additional bracket pairs in the strictly locally testable language R "we have an independent definition of the notion *linear language*"
In much the same way they give a general definition of *meta-linear language*. They conclude:

> *"Propositions 1 and 2 thus provide for the possibility of very natural definitions of the full class of CF languages, and various subfamilies of the class, independently of the [taxonomic] approach taken in preceding sections".*

A piece of work in this direction by P. Dömosi & S. Okawa [2001] characterizes the family of slender CF languages, a subclass of Linear CF, for which the number of words of the same length is bounded by a constant.

Subclasses of context-free languages II

Using the properties of slender languages studied by Paun & Salomaa [1995] they obtain the characterization $L = h(D \cap R)$ where

- D is a linear Dyck language over a grammar-independent alphabet, and
- the regular language R is a UDL Union of Double Loops:

$$\bigcup_{i=1\ldots k} \{u_i v_i^* w_i x_i^* y_i\} \qquad \text{for some words } u_i, v_i \ldots.$$

Generalizations for other language models

A CST confers a Patent of Nobility to novel language models!

- M. Kanazawa [2013] has a CST for Simple Context-Free Tree Grammars, related to the Tree-Adjoining Grammars, for which D. Weir [1988] had already stated a CST.
- R. Yoshinaka et al. [2010] have a CST for the family of Multiple Context-Free languages. a family between CF and Context-Sensitive invented by H. Seki et al. [1991]
- Cherubini et al. [1995] introduce a Generalized Dyck language for Multi-Push-Down languages, MPD.
  An *MPD language of order* $n \geq 1$ is recognized by a non-deterministic state-less machine equipped with n ordered push-down stacks.
  This family includes the *Tree Adjoining Grammar* TAG family (Cherubini & San Pietro [2000])

Multi-Push-Down languages

Generalized Dyck Language of order n, $D^{(n)}$: the alphabet has one or more $(n+1)$-tuples: $a, a^{(1)}, \ldots, a^{(n)}$, $b, b^{(1)}, \ldots, b^{(n)}, \ldots$
(The Dyck alphabet and the CF languages correspond to case $n = 1$, i.e., one push-down stack.)
Each of the n projections of $D^{(n)}$ on $a, a^{(j)}, b, b^{(j)}, \ldots$ is a Dyck language:

			projection on first Dyck alphabet								
a	b	$b^{(1)}$	a	$a^{(1)}$	$a^{(1)}$	$-$	a	$a^{(1)}$	$-$	$-$	$-$
a	b	$b^{(1)}$	a	$a^{(1)}$	$a^{(1)}$	$a^{(2)}$	a	$a^{(1)}$	$a^{(2)}$	$b^{(2)}$	$a^{(2)}$
a	b	$-$	a	$-$	$-$	$a^{(2)}$	a	$-$	$a^{(2)}$	$b^{(2)}$	$a^{(2)}$
			projection on 2nd Dyck alphabet								

A *cancellation rule* defines Generalized Dyck languages.

Theorem.

A language L is MPD of order n iff
$L = h(R \cap$ *Generalized Dyck of order* $n)$, with h and R as in CST.

A rare application of CST I

Parsing with C-S representation, M. Holden [2011]

- CST elegance has attracted many theoreticians, but very few, if any, real applications have been derived from it.
- A likely explanation: to exploit CST for tasks such as *parsing* or *grammar inference*, one has to solve the inverse problem caused by the homomorphism.
- Hulden [2009] proposes a cubic-time parser (as Earley and CKY) and discusses how to enhance it for probabilistic parsing.

A rare application of CST II

Parsing with C-S representation, M. Holden [2011]

Hulden polynomial-time CF parsing algorithm

Language representation

$$L(G) = h\left(g^{-1}\left(D \cap R\right)\right) \begin{cases} D \text{ encodes grammar derivations} \\ h \text{ deletes brackets} \\ g \text{ deletes terminal symbols} \end{cases}$$

To parse a sentence $w = w_1 w_2 \ldots w_n$:

1. Calculate $h^{-1}(w)$. This is a DFA that accepts only w with arbitrarily many brackets interspersed.
2. Calculate $R \cap h^{-1}(w)$. This DFA accepts all the locally correct parses of w but also unbalanced words.
3. Extract from (2) the set of words where brackets are balanced. The result precisely contains the correct parses of w.

. . . experimental results are not available.

Other homomorphic characterizations I

AntiDyck instead of Dyck, Queue automaton

- Replace LIFO stack with a First-In-First-Out queue memory
- Queue machine by Emil Post contemporary (1936) and computationally equivalent to Turing machine
- Simple state-less queue machine model recognizes the Anti-Dyck language (Vauquelin & Franchi-Zannettacci [1980]), such that (almost) no parentheses are well-nested. Anti-Dick is defined, as Dyck, by a deletion rule :

$$\underline{(}\ \overbrace{(\,[}^{\text{contains no closed paren.}}\ \underline{)}\,)\,] \Longrightarrow (\,[\,)\,] = \underline{(}\,[\,\underline{)}\,] \Longrightarrow [\,] \Longrightarrow \varepsilon$$

View $(_i$ and $)_i$ as arrival/departure of customer at a service:

Anti-Dyck $\Rightarrow$ service in arrival order	Dyck $\Rightarrow$ service in reversed order
$(_1 \quad (_2\,(_3\,)_1\,(_4\,)_2\,)_3\,)_4$	$(_1\,(_2\,(_3\,)_3\,(_4\,)_4\,)_2\,)_1$

Other homomorphic characterizations II

AntiDyck instead of Dyck, Queue automaton

- Languages such as Anti-Dyck are generated by Breadth-first CF grammars of Allevi et al. [1988].
- BCF grammars formally identical to CF grammars, but derivations in breadth-first order.

A $\frac{1}{2}$ CST by Cherubini et al. [1990]

Let L be a BCF language, Then $\exists$ an Anti-Dyck language, a 2-SLT language R and a homomorphism h s.t. $L = h(\text{AntiDyck} \cap R)$

The converse is false, because BCF languages are not closed by $\cap$ with regular languages.

Homomorphic characterizations of Recursive Enumerable languages

An idea is to characterize the Recursively Enumerable family by means of the homomorphism of some simpler language E, which plays the role of Dyck for CF languages. Intersection with a regular language is no longer needed. Typical characterizations take the form

$$L \in RE \iff L = h(E) \quad \text{where } E = L_1 \cap L_2 \text{ and } L_1, L_2 \text{ are:}$$

- Ginsburg et al. [1967]: two deterministic CF languages
- Baker & Book [1974]: two linear CF languages
- Hirose et al. [1985]: two minimal linear languages
- further restricted to smaller classes in Okawa & Hirose [2001], which also contains a survey.

A negative result for Context-Sensitive Languages I

S. Okawa et al. [1986]

terminal alphabet: $\Sigma = \{a, b, \ldots\}$
Dyck: D_{Δ_k} over $\Delta_k = \{a, a', b, b', \ldots\} \cup \{c_1, c_1', c_2, c_2', \ldots, c_k, c_k'\}$
Projection π from Δ_k to Σ (same role as homomorphism in CST)

Definition

$$\text{family } \mathcal{F} \text{ is } k-\textit{expressed by family } \mathcal{F}' \iff$$

$$\forall L \subseteq \Sigma^* \text{ in } \mathcal{F}, \exists \text{ language } L' \subseteq \Delta_k^* \text{ in } \mathcal{F}' : L = \pi\left(L' \cap D_{\Delta_k}\right) \quad (1)$$

$$\text{family } \mathcal{F} \text{ is } k-\textit{characterized by family } \mathcal{F}' \iff (1) \wedge$$

$$\forall L' \subseteq \Delta_k^* \text{ in } \mathcal{F}' : \pi\left(L' \cap D_{\Delta_k}\right) \text{ is in } \mathcal{F} \quad (2)$$

A negative result for Context-Sensitive Languages II

S. Okawa et al. [1986]

Theorem

For any family $\mathcal{F}'$ closed under ϵ-free homomorphism, for any integer $k \geq 2$ family Context Sensitive is not k-characterized by $\mathcal{F}'$.

The proof considers a lang. in $(Rec.Enum. - Context\ Sensitive)$

Corollary

For any family $\mathcal{F}$ closed under ϵ-free homomorphism, the Context Sensitive family cannot be of the form

$$\{\pi\left(L \cap D_{\Delta_k}\right) \mid L \in \mathcal{F}, k \geq 0\}$$

The characterizability of the *Context Sensitive* family using a Dyck language remains unsettled for $k < 2$ or if the homomorphism is restricted in some way.

CST for Weighted Context-Free languages I

Weighted automata not only recognize a word, but compute a *weight function*. CST characterizations of weighted languages are actively pursued:

- A weighted version of CST by A. Salomaa & M. Soittola [1978] uses weights from a *commutative semiring*.
- Recent models of weighted push-down automata allow more general domains for the weights, e.g., average computations on the reals for probabilistic machines.
- Weighted CF grammars with weights from a so-called *valuation monoid*, M. Droste & H. Vogler [2013]:

CST for Weighted Context-Free languages II

> *. . . we show that any quantitative CF lang. arises as the image of the intersection of a Dyck lang. and a recognizable lang. under a suitable weighted morphism, and also as the image of a Dyck lang. and a recognizable series under a free monoid morphism [and conversely]. . . . The classical CST is contained in the weighted result by considering the Boolean semiring.*

Further CST characterizations for different weighted automata or for different weight domains are: L. Herrman & H. Vogler [2015] and T. Denkinger [2015].

Half-serious Conclusion

[from **The short glossary of rhetorics and metrics**:]

> *Parenthesis or parenthetical clause is the addition of unnecessary elements or delucidations into a sentence. It is marked off by round or square brackets parentheses, dashes or commas.*

Should I apologize for imposing one hour of unnecessary elements on my distinguished audience? My lawyer, M. D. Murray of the University of Michigan Law School, says definitely NO in his:

> *For the Love of Parentheticals: The Story of Parenthetical Usage in Synthesis, Rhetoric, Economics, and Narrative Reasoning*

I just recounted Schüzenberger's lively heritage on the essential role of parentheses!

References I

B. S. Baker and R. V. Book.
Reversal-bounded multipushdown machines.
J. Comput. Syst. Sci., 8(3):315–332, 1974.

J. Berstel.
Transductions and Context-Free Languages.
Teubner, Stuttgart, 1979.

L. Breveglieri, A. Cherubini, C. Citrini, and S. Crespi-Reghizzi.
Multi-push-down languages and grammars.
Int. J. Found. Comput. Sci., 7(3):253–292, 1996.

A. Cherubini and P. San Pietro.
Tree adjoining languages and multipushdown languages.
Theory Comput. Syst., 33(4):257–293, 2000.

N. Chomsky and M. Schützenberger.
The algebraic theory of context-free languages.
In Brafford and Hirschenber, editors, *Computer programming and formal systems*, pages 118–161.
North-Holland, Amsterdam, 1963.

S. Crespi Reghizzi and P. San Pietro.
From regular to strictly locally testable languages.
Int. J. Found. Comput. Sci., 23(8):1711–1728, 2012.

References II

S. Crespi-Reghizzi and P. San Pietro.

The missing case in Chomsky-Schützenberger theorem.
In *Language and Automata Theory and Applications - 10th International Conference, LATA 2016*, pages 345–358, 2016.

T. Denkinger.

A Chomsky-Schützenberger representation theorem for weighted multiple context-free languages.
In *Proceedings of the 12th International Conference on Finite-State Methods and Natural Language Processing (FSMNLP 2015)*, 2015.

P. Dömösi and S. Okawa.

A Chomsky-Schützenberger-Stanley type characterization of the class of slender context-free languages.
Acta Cybern., 15(1):25–32, 2001.

M. Droste and H. Vogler.

The Chomsky-Schützenberger theorem for quantitative context-free languages.
In *Developments in Language Theory - 17th International Conference, DLT 2013, Marne-la-Vallée, France, June 18-21, 2013. Proceedings*, volume 7907 of *Lecture Notes in Computer Science*, pages 203–214. Springer, 2013.

S. Ginsburg, S. A. Greibach, and M. A. Harrison.

One-way stack automata.
J. ACM, 14(2):389–418, 1967.

References III

L. Herrmann and H. Vogler.

A Chomsky-Schützenberger theorem for weighted automata with storage.
In *Algebraic Informatics - 6th International Conference, CAI 2015, Stuttgart, Germany, September 1-4, 2015. Proceedings*, volume 9270 of *Lecture Notes in Computer Science*, pages 115–127. Springer, 2015.

S. Hirose, S. Okawa, and M. Yoneda.

A homomorphic characterization of recursively enumerable languages.
Theor. Comput. Sci., 35:261 – 269, 1985.

S. Hirose and M. Yoneda.

On the Chomsky and Stanley's homomorphic characterization of context-free languages.
Theor. Comput. Sci., 36:109–112, 1985.

M. Hulden.

Parsing cfgs and pcfgs with a chomsky-schützenberger representation.
In *Proceedings of the 4th Conference on Human Language Technology: Challenges for Computer Science and Linguistics*, LTC'09, pages 151–160, Berlin, Heidelberg, 2011. Springer-Verlag.

M. Kanazawa.

Multidimensional trees and a Chomsky-Schützenberger-Weir representation theorem for simple context-free tree grammars.
Journal of Logic and Computation, 2014.

References IV

Y. T. Medvedev.

On the class of events representable in a finite automaton.
In E. F. Moore, editor, *Sequential machines – Selected papers (translated from Russian)*, pages 215–227. Addison-Wesley, New York, NY, USA, 1964.

S. Okawa and S. Hirose.

Homomorphic characterizations of recursively enumerable languages with very small language classes.
Theor. Comput. Sci., 250(1-2):55–69, 2001.

S. Okawa, S. Hirose, and M. Yoneda.

On the impossibility of the homomorphic characterization of context-sensitive languages.
Theor. Comput. Sci., 44:225–228, 1986.

A. Okhotin.

Non-erasing variants of the Chomsky—Schützenberger theorem.
In *Proceedings of the 16th International Conference on Developments in Language Theory*, DLT'12, pages 121–129, Berlin, Heidelberg, 2012. Springer-Verlag.

A. Salomaa and M. Soittola.

Automata-Theoretic Aspects of Formal Power Series.
Texts and Monographs in Computer Science. Springer, 1978.

R. J. Stanley.

Finite state representations of context-free languages.
M.I.T. Res. Lab. Electron. Quart. Progr. Rept., 76(1):276–279, 1965.

References V

D. J. Weir.

Characterizing Mildly Context-Sensitive Grammar Formalisms.
PhD thesis, Sciences, 1988.

R. Yoshinaka, Y. Kaji, and H. Seki.

Chomsky-Schützenberger-type characterization of multiple context-free languages.
In *Language and Automata Theory and Applications, 4th International Conference, LATA 2010*, volume 6031 of *Lecture Notes in Computer Science*, pages 596–607. Springer, 2010.