Relating Chomsky Normal Form and Greibach Normal Form by Exponential Transposition

Jürgen Koslowski

Department of Theoretical Computer Science
Technical University Braunschweig

CT 2011, Vancouver, July 22
(last updated 2011-06-18)

http://www.iti.cs.tu-bs.de/~koslowj/RESEARCH

Overview

Overview

- ▷ We will look at familiar concepts

Overview

- We will look at familiar concepts
  - context-free grammars (CFG's) initially,

Overview

- We will look at familiar concepts
  - context-free grammars (CFG's) initially,
  - push-down automata (PDA's) later on

Overview

- We will look at familiar concepts
  - context-free grammars (CFG's) initially,
  - push-down automata (PDA's) later on

  from a slightly different angle.

Overview

- We will look at familiar concepts
  - context-free grammars (CFG's) initially,
  - push-down automata (PDA's) later on

  from a slightly different angle.
- This angle was initially suggested by work of Walters [1988], but can be exploited further.

Overview

- We will look at familiar concepts
  - context-free grammars (CFG's) initially,
  - push-down automata (PDA's) later on

  from a slightly different angle.
- This angle was initially suggested by work of Walters [1988], but can be exploited further.
- At issue is the use of node-labeled trees in the theory of formal languages.

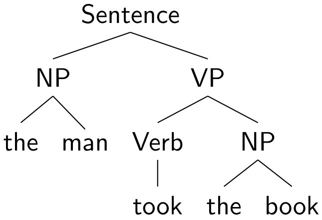
History: the man took the book

History: the man took the book

In his 1956 article "Three models for the description of language" (IRE Transactions on Information Theory (2): 113–124) Noam Chomsky published the first derivation tree as figure (22) with labeled nodes

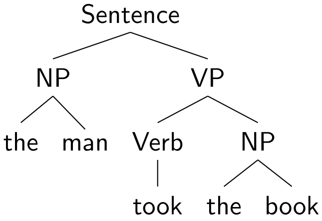
History: the man took the book

In his 1956 article “Three models for the description of language” (IRE Transactions on Information Theory (2): 113–124) Noam Chomsky published the first derivation tree as figure (22) with labeled nodes



History: the man took the book

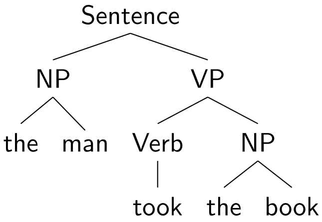
In his 1956 article "Three models for the description of language" (IRE Transactions on Information Theory (2): 113–124) Noam Chomsky published the first derivation tree as figure (22) with labeled nodes



- it shows the equivalence of two grammar-derivations displayed earlier (figure (21));

History: the man took the book

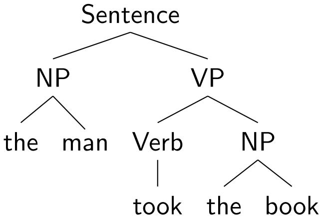
In his 1956 article "Three models for the description of language" (IRE Transactions on Information Theory (2): 113–124) Noam Chomsky published the first derivation tree as figure (22) with labeled nodes



- it shows the equivalence of two grammar-derivations displayed earlier (figure (21));
- it conveys the same phrase structure of "the man took the book" as the initially employed block diagram (figure (17)).

History: the man took the book

In his 1956 article "Three models for the description of language" (IRE Transactions on Information Theory (2): 113–124) Noam Chomsky published the first derivation tree as figure (22) with labeled nodes

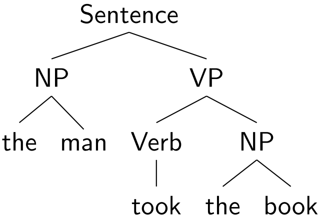


- it shows the equivalence of two grammar-derivations displayed earlier (figure (21));
- it conveys the same phrase structure of "the man took the book" as the initially employed block diagram (figure (17)).

Other such trees are used later to show the existence of non-equivalent derivations for certain sentences.

History: the man took the book

In his 1956 article “Three models for the description of language” (IRE Transactions on Information Theory (2): 113–124) Noam Chomsky published the first derivation tree as figure (22) with labeled nodes

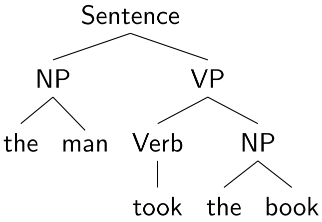


- it shows the equivalence of two grammar-derivations displayed earlier (figure (21));
- it conveys the same phrase structure of “the man took the book” as the initially employed block diagram (figure (17)).

Other such trees are used later to show the existence of non-equivalent derivations for certain sentences. However, they are employed just to aid visualization, not as an object of study in their own right.

History: the man took the book

In his 1956 article “Three models for the description of language” (IRE Transactions on Information Theory (2): 113–124) Noam Chomsky published the first derivation tree as figure (22) with labeled nodes



- it shows the equivalence of two grammar-derivations displayed earlier (figure (21));
- it conveys the same phrase structure of “the man took the book” as the initially employed block diagram (figure (17)).

Other such trees are used later to show the existence of non-equivalent derivations for certain sentences. However, they are employed just to aid visualization, not as an object of study in their own right.

Note the distinction between leaves and (capitalized) inner nodes.

On the finite vocabulary (= alphabet) V_P of his phrase-structure grammar (= semi-Thue rewriting system F + axioms), Chomsky remarks

On the finite vocabulary (= alphabet) V_P of his phrase-structure grammar (= semi-Thue rewriting system F + axioms), Chomsky remarks

> *In every interesting case there will be a terminal vocabulary V_T ($V_T \subseteq V_P$) that exactly characterizes the terminal strings, in the sense that every terminal string is a string in V_T and no symbol of V_T is rewritten in any of the rules of F .*

On the finite vocabulary (= alphabet) V_P of his phrase-structure grammar (= semi-Thue rewriting system F + axioms), Chomsky remarks

In every interesting case there will be a terminal vocabulary V_T ($V_T \subseteq V_P$) that exactly characterizes the terminal strings, in the sense that every terminal string is a string in V_T and no symbol of V_T is rewritten in any of the rules of F.

Chomsky's work inspired John Backus of the Algol 58 project in 1959 to develop most of the prevailing BNF-type presentation of CFG's:

On the finite vocabulary (= alphabet) V_P of his phrase-structure grammar (= semi-Thue rewriting system F + axioms), Chomsky remarks

In every interesting case there will be a terminal vocabulary V_T ($V_T \subseteq V_P$) that exactly characterizes the terminal strings, in the sense that every terminal string is a string in V_T and no symbol of V_T is rewritten in any of the rules of F .

Chomsky's work inspired John Backus of the Algol 58 project in 1959 to develop most of the prevailing BNF-type presentation of CFG's:

- the alphabet is partitioned into "terminals" and "nonterminals";

On the finite vocabulary (= alphabet) V_P of his phrase-structure grammar (= semi-Thue rewriting system F + axioms), Chomsky remarks

In every interesting case there will be a terminal vocabulary V_T ($V_T \subseteq V_P$) that exactly characterizes the terminal strings, in the sense that every terminal string is a string in V_T and no symbol of V_T is rewritten in any of the rules of F.

Chomsky's work inspired John Backus of the Algol 58 project in 1959 to develop most of the prevailing BNF-type presentation of CFG's:

- the alphabet is partitioned into "terminals" and "nonterminals";
- one nonterminal serves as axiom (rather than a finite set of words).

On the finite vocabulary (= alphabet) V_P of his phrase-structure grammar (= semi-Thue rewriting system F + axioms), Chomsky remarks

> *In every interesting case there will be a terminal vocabulary V_T ($V_T \subseteq V_P$) that exactly characterizes the terminal strings, in the sense that every terminal string is a string in V_T and no symbol of V_T is rewritten in any of the rules of F.*

Chomsky's work inspired John Backus of the Algol 58 project in 1959 to develop most of the prevailing BNF-type presentation of CFG's:

- the alphabet is partitioned into "terminals" and "nonterminals";
- one nonterminal serves as axiom (rather than a finite set of words).

This was fine-tuned by Peter Naur for the Revised Report on ALGOL 60, who also coined the name "Backus Normal Form".

On the finite vocabulary (= alphabet) V_P of his phrase-structure grammar (= semi-Thue rewriting system F + axioms), Chomsky remarks

In every interesting case there will be a terminal vocabulary V_T ($V_T \subseteq V_P$) that exactly characterizes the terminal strings, in the sense that every terminal string is a string in V_T and no symbol of V_T is rewritten in any of the rules of F.

Chomsky's work inspired John Backus of the Algol 58 project in 1959 to develop most of the prevailing BNF-type presentation of CFG's:

- the alphabet is partitioned into "terminals" and "nonterminals";
- one nonterminal serves as axiom (rather than a finite set of words).

This was fine-tuned by Peter Naur for the Revised Report on ALGOL 60, who also coined the name "Backus Normal Form". In 1964 Donald Knuth observed that this was not a normal form in any sense and suggested the term "Backus-Naur Form", saving the acronym.

On the finite vocabulary (= alphabet) V_P of his phrase-structure grammar (= semi-Thue rewriting system F + axioms), Chomsky remarks

In every interesting case there will be a terminal vocabulary V_T ($V_T \subseteq V_P$) that exactly characterizes the terminal strings, in the sense that every terminal string is a string in V_T and no symbol of V_T is rewritten in any of the rules of F .

Chomsky's work inspired John Backus of the Algol 58 project in 1959 to develop most of the prevailing BNF-type presentation of CFG's:

- the alphabet is partitioned into "terminals" and "nonterminals";
- one nonterminal serves as axiom (rather than a finite set of words).

This was fine-tuned by Peter Naur for the Revised Report on ALGOL 60, who also coined the name "Backus Normal Form". In 1964 Donald Knuth observed that this was not a normal form in any sense and suggested the term "Backus-Naur Form", saving the acronym.

Node-labeled trees in the 1960's also formed the basis for the new field of tree grammars/automata/languages, see Thatcher's survey of 1973.

Background on grammars and normal forms

Definition

A context-free grammar $\boldsymbol{G} = \langle \mathcal{N}, \mathcal{T}, S, \rightarrow \rangle$ consists of disjoint finite sets $\mathcal{N}$ of nonterminals and $\mathcal{T}$ of terminals, an axiom $S \in \mathcal{N}$, and a finite relation $\mathcal{N} \xrightarrow{\;\rightarrow\;} (\mathcal{T} + \mathcal{N})^*$ of so-called productions.

Background on grammars and normal forms

Definition

A context-free grammar $G = \langle \mathcal{N}, \mathcal{T}, S, \rightarrow \rangle$ consists of disjoint finite sets $\mathcal{N}$ of nonterminals and $\mathcal{T}$ of terminals, an axiom $S \in \mathcal{N}$, and a finite relation $\mathcal{N} \xrightarrow{\rightarrow} (\mathcal{T} + \mathcal{N})^*$ of so-called productions. G is said to be in

Background on grammars and normal forms

Definition

A context-free grammar $\boldsymbol{G} = \langle \mathcal{N}, \mathcal{T}, S, \rightarrow \rangle$ consists of disjoint finite sets $\mathcal{N}$ of nonterminals and $\mathcal{T}$ of terminals, an axiom $S \in \mathcal{N}$, and a finite relation $\mathcal{N} \overset{\rightarrow}{\longrightarrow} (\mathcal{T} + \mathcal{N})^*$ of so-called productions. $\boldsymbol{G}$ is said to be in

- weak Chomsky normal form (wCNF), if $\mathcal{N} \overset{\rightarrow}{\longrightarrow} (\mathcal{T} + \mathcal{N}^*)$;

Background on grammars and normal forms

Definition

A context-free grammar $\boldsymbol{G} = \langle \mathcal{N}, \mathcal{T}, S, \longrightarrow \rangle$ consists of disjoint finite sets $\mathcal{N}$ of nonterminals and $\mathcal{T}$ of terminals, an axiom $S \in \mathcal{N}$, and a finite relation $\mathcal{N} \xrightarrow{\rightarrow} (\mathcal{T} + \mathcal{N})^*$ of so-called productions. $\boldsymbol{G}$ is said to be in

- ▷ weak Chomsky normal form (wCNF), if $\mathcal{N} \xrightarrow{\rightarrow} (\mathcal{T} + \mathcal{N}^*)$;
- ▷ Chomsky normal form (CNF), if $\mathcal{N} \xrightarrow{\rightarrow} \left(\mathcal{T} + \mathcal{N}^2\right) + \{\varepsilon\}$;

Background on grammars and normal forms

Definition

A context-free grammar $\boldsymbol{G} = \langle \mathcal{N}, \mathcal{T}, S, \rightarrow \rangle$ consists of disjoint finite sets $\mathcal{N}$ of nonterminals and $\mathcal{T}$ of terminals, an axiom $S \in \mathcal{N}$, and a finite relation $\mathcal{N} \xrightarrow{\rightarrow} (\mathcal{T} + \mathcal{N})^*$ of so-called productions. $\boldsymbol{G}$ is said to be in

- ▷ weak Chomsky normal form (wCNF), if $\mathcal{N} \xrightarrow{\rightarrow} (\mathcal{T} + \mathcal{N}^*)$;
- ▷ Chomsky normal form (CNF), if $\mathcal{N} \xrightarrow{\rightarrow} \left(\mathcal{T} + \mathcal{N}^2\right) + \{\varepsilon\}$;
- ▷ Greibach normal form (GNF), if $\mathcal{N} \xrightarrow{\rightarrow} (\mathcal{T} \times \mathcal{N}^*) + \{\varepsilon\}$;

Background on grammars and normal forms

Definition

A context-free grammar $\boldsymbol{G} = \langle \mathcal{N}, \mathcal{T}, S, \rightarrow \rangle$ consists of disjoint finite sets $\mathcal{N}$ of nonterminals and $\mathcal{T}$ of terminals, an axiom $S \in \mathcal{N}$, and a finite relation $\mathcal{N} \xrightarrow{\rightarrow} (\mathcal{T} + \mathcal{N})^*$ of so-called productions. $\boldsymbol{G}$ is said to be in

- weak Chomsky normal form (wCNF), if $\mathcal{N} \xrightarrow{\rightarrow} (\mathcal{T} + \mathcal{N}^*)$;
- Chomsky normal form (CNF), if $\mathcal{N} \xrightarrow{\rightarrow} \left(\mathcal{T} + \mathcal{N}^2\right) + \{\varepsilon\}$;
- Greibach normal form (GNF), if $\mathcal{N} \xrightarrow{\rightarrow} (\mathcal{T} \times \mathcal{N}^*) + \{\varepsilon\}$;

with the technical provision for CNF and GNF that $Y \rightarrow \varepsilon$ implies $Y = S$, and in this case S must not appear on the right side of other productions.

Background on grammars and normal forms

Definition

A context-free grammar $\boldsymbol{G} = \langle \mathcal{N}, \mathcal{T}, S, \rightarrow \rangle$ consists of disjoint finite sets $\mathcal{N}$ of nonterminals and $\mathcal{T}$ of terminals, an axiom $S \in \mathcal{N}$, and a finite relation $\mathcal{N} \xrightarrow{\ \rightarrow\ } (\mathcal{T} + \mathcal{N})^*$ of so-called productions. $\boldsymbol{G}$ is said to be in

- weak Chomsky normal form (wCNF), if $\mathcal{N} \xrightarrow{\ \rightarrow\ } (\mathcal{T} + \mathcal{N}^*)$;
- Chomsky normal form (CNF), if $\mathcal{N} \xrightarrow{\ \rightarrow\ } \big(\mathcal{T} + \mathcal{N}^2\big) + \{\varepsilon\}$;
- Greibach normal form (GNF), if $\mathcal{N} \xrightarrow{\ \rightarrow\ } (\mathcal{T} \times \mathcal{N}^*) + \{\varepsilon\}$;

with the technical provision for CNF and GNF that $Y \rightarrow \varepsilon$ implies $Y = S$, and in this case S must not appear on the right side of other productions.

The derivation relation $(\mathcal{N} + \mathcal{T})^* \xrightarrow{\ \Rightarrow\ } (\mathcal{N} + \mathcal{T})^*$ consists of all pairs $\langle \alpha Y \beta, \alpha \omega \beta \rangle$ with $Y \rightarrow \omega$ and $\alpha, \beta \in (\mathcal{N} + \mathcal{T})^*$.

Background on grammars and normal forms

Definition

A context-free grammar $G = \langle \mathcal{N}, \mathcal{T}, S, \longrightarrow \rangle$ consists of disjoint finite sets $\mathcal{N}$ of nonterminals and $\mathcal{T}$ of terminals, an axiom $S \in \mathcal{N}$, and a finite relation $\mathcal{N} \xrightarrow{\;\to\;} (\mathcal{T} + \mathcal{N})^*$ of so-called productions. G is said to be in

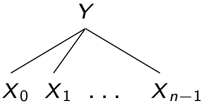
- weak Chomsky normal form (wCNF), if $\mathcal{N} \xrightarrow{\;\to\;} (\mathcal{T} + \mathcal{N}^*)$;
- Chomsky normal form (CNF), if $\mathcal{N} \xrightarrow{\;\to\;} \left(\mathcal{T} + \mathcal{N}^2\right) + \{\varepsilon\}$;
- Greibach normal form (GNF), if $\mathcal{N} \xrightarrow{\;\to\;} (\mathcal{T} \times \mathcal{N}^*) + \{\varepsilon\}$;

with the technical provision for CNF and GNF that $Y \to \varepsilon$ implies $Y = S$, and in this case S must not appear on the right side of other productions.

The derivation relation $(\mathcal{N} + \mathcal{T})^* \xrightarrow{\;\Rightarrow\;} (\mathcal{N} + \mathcal{T})^*$ consists of all pairs $\langle \alpha Y \beta, \alpha \omega \beta \rangle$ with $Y \to \omega$ and $\alpha, \beta \in (\mathcal{N} + \mathcal{T})^*$.

The words $w \in \mathcal{T}^*$ with $S \Rightarrow^* w$ constitute the language generated by G, where $\Rightarrow^*$ is the reflexive transitive hull of $\Rightarrow$.

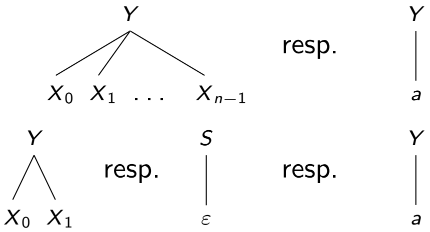
Traditional tree descriptions of normal-form productions:



resp.

(weak Chomsky)

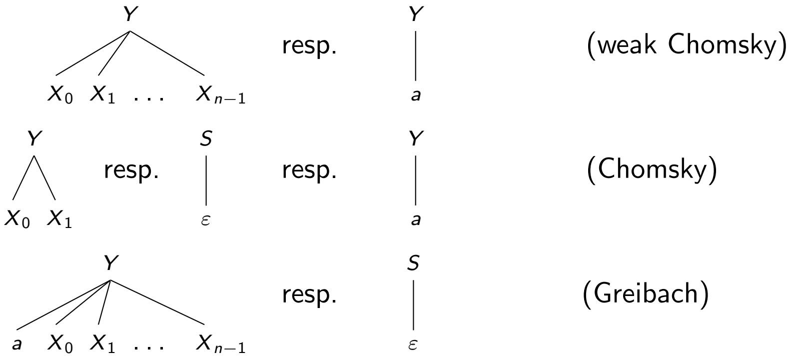
Traditional tree descriptions of normal-form productions:



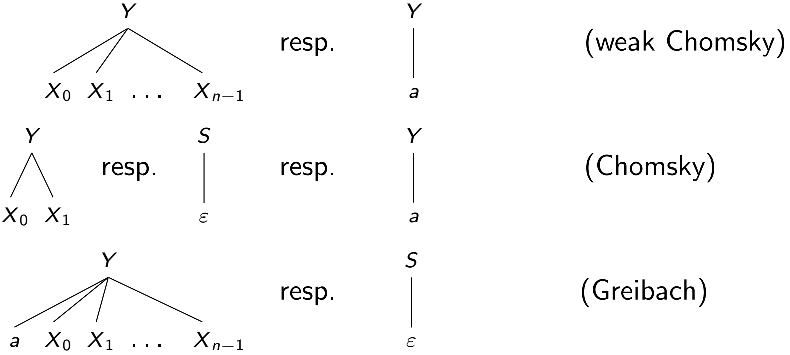
(weak Chomsky)

(Chomsky)

Traditional tree descriptions of normal-form productions:



Traditional tree descriptions of normal-form productions:



Strictly speaking, the tree for $S \longrightarrow \varepsilon$ is not correct; it should be just a leaf with nonterminal S. However, this is hard to distinguish from cases, where the derivation is not yet finished.

Alternatives?

Is it really necessary to lump terminals and nonterminals together into the set $\mathcal{V} := \mathcal{T} + \mathcal{N}$ on which to form the free monoid?

Alternatives?

Is it really necessary to lump terminals and nonterminals together into the set $\mathcal{V} := \mathcal{T} + \mathcal{N}$ on which to form the free monoid? No!

Alternatives?

Is it really necessary to lump terminals and nonterminals together into the set $\mathcal{V} := \mathcal{T} + \mathcal{N}$ on which to form the free monoid? No! First recall

Alternatives?

Is it really necessary to lump terminals and nonterminals together into the set $\mathcal{V} := \mathcal{T} + \mathcal{N}$ on which to form the free monoid? No! First recall

Definition (Multigraphs)

Let $\boldsymbol{D}$ be the free category on the graph with object set $\mathbb{N} + \{*\}$ with $n+1$ arrows d_i, $i < n$, and c from n to $*$, $n \in \mathbb{N}$. The category of multigraphs now is the functor-category $\boldsymbol{mgph} := [\boldsymbol{D}^{\mathsf{op}}, \boldsymbol{set}]$.

Alternatives?

Is it really necessary to lump terminals and nonterminals together into the set $\mathcal{V} := \mathcal{T} + \mathcal{N}$ on which to form the free monoid? No! First recall

Definition (Multigraphs)

Let $\boldsymbol{D}$ be the free category on the graph with object set $\mathbb{N} + \{*\}$ with $n+1$ arrows d_i, $i < n$, and c from n to $*$, $n \in \mathbb{N}$. The category of multigraphs now is the functor-category $\boldsymbol{mgph} := [\boldsymbol{D}^{\mathrm{op}}, \boldsymbol{set}]$.

For $\boldsymbol{D}^{\mathrm{op}} \xrightarrow{G} \boldsymbol{set}$ call the elements of G_* and of G_n, $n \in \mathbb{N}$, objects, resp., multiarrows.

Alternatives?

Is it really necessary to lump terminals and nonterminals together into the set $\mathcal{V} := \mathcal{T} + \mathcal{N}$ on which to form the free monoid? No! First recall

Definition (Multigraphs)

Let $\boldsymbol{D}$ be the free category on the graph with object set $\mathbb{N} + \{*\}$ with $n+1$ arrows d_i, $i < n$, and c from n to $*$, $n \in \mathbb{N}$. The category of multigraphs now is the functor-category $\boldsymbol{mgph} := [\boldsymbol{D}^{\mathsf{op}}, \boldsymbol{set}]$.

For $\boldsymbol{D}^{\mathsf{op}} \xrightarrow{G} \boldsymbol{set}$ call the elements of G_* and of G_n, $n \in \mathbb{N}$, objects, resp., multiarrows. If $f \in G_n$ satisfies $fd_i = X_i$, $i < n$, and $fc = Y$, besides the notation $Y \xrightarrow{\varphi} X_0 X_1 \ldots X_{n-1}$ we also use the circuit diagram

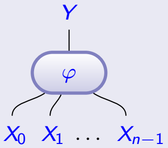
Alternatives?

Is it really necessary to lump terminals and nonterminals together into the set $\mathcal{V} := \mathcal{T} + \mathcal{N}$ on which to form the free monoid? No! First recall

Definition (Multigraphs)

Let $\boldsymbol{D}$ be the free category on the graph with object set $\mathbb{N} + \{*\}$ with $n+1$ arrows d_i, $i < n$, and c from n to $*$, $n \in \mathbb{N}$. The category of multigraphs now is the functor-category $\boldsymbol{mgph} := [\boldsymbol{D}^{\mathrm{op}}, \boldsymbol{set}]$.

For $\boldsymbol{D}^{\mathrm{op}} \xrightarrow{G} \boldsymbol{set}$ call the elements of G_* and of G_n, $n \in \mathbb{N}$, objects, resp., multiarrows. If $f \in G_n$ satisfies $fd_i = X_i$, $i < n$, and $fc = Y$, besides the notation $Y \xrightarrow{\varphi} X_0 X_1 \ldots X_{n-1}$ we also use the circuit diagram



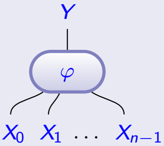
Alternatives?

Is it really necessary to lump terminals and nonterminals together into the set $\mathcal{V} := \mathcal{T} + \mathcal{N}$ on which to form the free monoid? No! First recall

Definition (Multigraphs)

Let $\boldsymbol{D}$ be the free category on the graph with object set $\mathbb{N} + \{*\}$ with $n+1$ arrows d_i, $i < n$, and c from n to $*$, $n \in \mathbb{N}$. The category of multigraphs now is the functor-category $\boldsymbol{mgph} := [\boldsymbol{D}^{\text{op}}, \boldsymbol{set}]$.

For $\boldsymbol{D}^{\text{op}} \xrightarrow{G} \boldsymbol{set}$ call the elements of G_* and of G_n, $n \in \mathbb{N}$, objects, resp., multiarrows. If $f \in G_n$ satisfies $fd_i = X_i$, $i < n$, and $fc = Y$, besides the notation $Y \xrightarrow{\varphi} X_0 X_1 \ldots X_{n-1}$ we also use the circuit diagram



- Nodes correspond multiarrows;

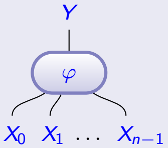
Alternatives?

Is it really necessary to lump terminals and nonterminals together into the set $\mathcal{V} := \mathcal{T} + \mathcal{N}$ on which to form the free monoid? No! First recall

Definition (Multigraphs)

Let $\boldsymbol{D}$ be the free category on the graph with object set $\mathbb{N} + \{*\}$ with $n+1$ arrows d_i, $i < n$, and c from n to $*$, $n \in \mathbb{N}$. The category of multigraphs now is the functor-category $\boldsymbol{mgph} := [\boldsymbol{D}^{\mathsf{op}}, \boldsymbol{set}]$.

For $\boldsymbol{D}^{\mathsf{op}} \xrightarrow{G} \boldsymbol{set}$ call the elements of G_* and of G_n, $n \in \mathbb{N}$, objects, resp., multiarrows. If $f \in G_n$ satisfies $fd_i = X_i$, $i < n$, and $fc = Y$, besides the notation $Y \xrightarrow{\varphi} X_0 X_1 \dots X_{n-1}$ we also use the circuit diagram



- Nodes correspond multiarrows;
- "wires" correspond to objects;

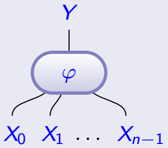
Alternatives?

Is it really necessary to lump terminals and nonterminals together into the set $\mathcal{V} := \mathcal{T} + \mathcal{N}$ on which to form the free monoid? No! First recall

Definition (Multigraphs)

Let $\boldsymbol{D}$ be the free category on the graph with object set $\mathbb{N} + \{*\}$ with $n+1$ arrows d_i, $i < n$, and c from n to $*$, $n \in \mathbb{N}$. The category of multigraphs now is the functor-category $\boldsymbol{mgph} := [\boldsymbol{D}^{\mathrm{op}}, \boldsymbol{set}]$.

For $\boldsymbol{D}^{\mathrm{op}} \xrightarrow{G} \boldsymbol{set}$ call the elements of G_* and of G_n, $n \in \mathbb{N}$, objects, resp., multiarrows. If $f \in G_n$ satisfies $fd_i = X_i$, $i < n$, and $fc = Y$, besides the notation $Y \xrightarrow{\varphi} X_0 X_1 \ldots X_{n-1}$ we also use the circuit diagram



- Nodes correspond multiarrows;
- "wires" correspond to objects;
- the direction is from top to bottom.

Definition (Bob Walters, 1988)

For a set $\mathcal{T}$ consider the multigraph $\mathcal{T}_{\langle 0 \rangle}$ with one object M, default multiarrows $M \xrightarrow{\mu_n} M^n$, $n \in \mathbb{N}$, and multiarrows $M \xrightarrow{a} M^0$, $a \in \mathcal{T}$.

Definition (Bob Walters, 1988)

For a set $\mathcal{T}$ consider the multigraph $\mathcal{T}_{\langle 0 \rangle}$ with one object M, default multiarrows $M \xrightarrow{\mu_n} M^n$, $n \in \mathbb{N}$, and multiarrows $M \xrightarrow{a} M^0$, $a \in \mathcal{T}$.

A CFG à la Walters over $\mathcal{T}$ is a faithful multigraph morphism $\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}_{\langle 0 \rangle}$ with $\boldsymbol{G}$ finite,

Definition (Bob Walters, 1988)

For a set $\mathcal{T}$ consider the multigraph $\mathcal{T}_{\langle 0 \rangle}$ with one object M, default multiarrows $M \xrightarrow{\mu_n} M^n$, $n \in \mathbb{N}$, and multiarrows $M \xrightarrow{a} M^0$, $a \in \mathcal{T}$.

A CFG à la Walters over $\mathcal{T}$ is a faithful multigraph morphism $\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}_{\langle 0 \rangle}$ with $\boldsymbol{G}$ finite, together with a distinguished object $\boldsymbol{G}$-object S.

Definition (Bob Walters, 1988)

For a set $\mathcal{T}$ consider the multigraph $\mathcal{T}_{\langle 0\rangle}$ with one object M, default multiarrows $M \xrightarrow{\mu_n} M^n$, $n \in \mathbb{N}$, and multiarrows $M \xrightarrow{a} M^0$, $a \in \mathcal{T}$.

A CFG à la Walters over $\mathcal{T}$ is a faithful multigraph morphism $\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}_{\langle 0\rangle}$ with $\boldsymbol{G}$ finite, together with a distinguished object $\boldsymbol{G}$-object S.

Comparison with a traditional CFG in wCNF shows that

- objects of $\boldsymbol{G}$ correspond to nonterminals;

Definition (Bob Walters, 1988)

For a set $\mathcal{T}$ consider the multigraph $\mathcal{T}_{\langle 0\rangle}$ with one object M, default multiarrows $M \xrightarrow{\mu_n} M^n$, $n \in \mathbb{N}$, and multiarrows $M \xrightarrow{a} M^0$, $a \in \mathcal{T}$.

A CFG à la Walters over $\mathcal{T}$ is a faithful multigraph morphism $\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}_{\langle 0\rangle}$ with $\boldsymbol{G}$ finite, together with a distinguished object $\boldsymbol{G}$-object S.

Comparison with a traditional CFG in wCNF shows that

- objects of $\boldsymbol{G}$ correspond to nonterminals;
- the 0-ary multiarrows $M \xrightarrow{a} M^0$ correspond to terminals, while $M \xrightarrow{\mu_0} M^0$ corresponds to the empty word ε;

Definition (Bob Walters, 1988)

For a set $\mathcal{T}$ consider the multigraph $\mathcal{T}_{\langle 0\rangle}$ with one object M, default multiarrows $M \xrightarrow{\mu_n} M^n$, $n \in \mathbb{N}$, and multiarrows $M \xrightarrow{a} M^0$, $a \in \mathcal{T}$.

A CFG à la Walters over $\mathcal{T}$ is a faithful multigraph morphism $\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}_{\langle 0\rangle}$ with $\boldsymbol{G}$ finite, together with a distinguished object $\boldsymbol{G}$-object S.

Comparison with a traditional CFG in wCNF shows that

- objects of $\boldsymbol{G}$ correspond to nonterminals;
- the 0-ary multiarrows $M \xrightarrow{a} M^0$ correspond to terminals, while $M \xrightarrow{\mu_0} M^0$ corresponds to the empty word ε;
- the γ-assignments of $\mathcal{T}_{\langle 0\rangle}$-multiarrows to $\boldsymbol{G}$-multiarrows correspond to productions.

Definition (Bob Walters, 1988)

For a set $\mathcal{T}$ consider the multigraph $\mathcal{T}_{\langle 0\rangle}$ with one object M, default multiarrows $M \xrightarrow{\mu_n} M^n$, $n \in \mathbb{N}$, and multiarrows $M \xrightarrow{a} M^0$, $a \in \mathcal{T}$.

A CFG à la Walters over $\mathcal{T}$ is a faithful multigraph morphism $\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}_{\langle 0\rangle}$ with $\boldsymbol{G}$ finite, together with a distinguished object $\boldsymbol{G}$-object S.

Comparison with a traditional CFG in wCNF shows that

- objects of $\boldsymbol{G}$ correspond to nonterminals;
- the 0-ary multiarrows $M \xrightarrow{a} M^0$ correspond to terminals, while $M \xrightarrow{\mu_0} M^0$ corresponds to the empty word ε;
- the γ-assignments of $\mathcal{T}_{\langle 0\rangle}$-multiarrows to $\boldsymbol{G}$-multiarrows correspond to productions.

Faithfulness prevents multiple copies of productions from occurring.

Definition (Bob Walters, 1988)

For a set $\mathcal{T}$ consider the multigraph $\mathcal{T}_{\langle 0\rangle}$ with one object M, default multiarrows $M \xrightarrow{\mu_n} M^n$, $n \in \mathbb{N}$, and multiarrows $M \xrightarrow{a} M^0$, $a \in \mathcal{T}$.

A CFG à la Walters over $\mathcal{T}$ is a faithful multigraph morphism $\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}_{\langle 0\rangle}$ with $\boldsymbol{G}$ finite, together with a distinguished object $\boldsymbol{G}$-object S.

Comparison with a traditional CFG in wCNF shows that

- objects of $\boldsymbol{G}$ correspond to nonterminals;
- the 0-ary multiarrows $M \xrightarrow{a} M^0$ correspond to terminals, while $M \xrightarrow{\mu_0} M^0$ corresponds to the empty word ε;
- the γ-assignments of $\mathcal{T}_{\langle 0\rangle}$-multiarrows to $\boldsymbol{G}$-multiarrows correspond to productions.

Faithfulness prevents multiple copies of productions from occurring.

To describe the language generated by γ as directly as possible, we take a different approach from that of Walters.

The generated language

[1] not to be confused with the graph-theoretic notion of this name

The generated language

- Freely extend γ to a multifunctor $\boldsymbol{G}^* \xrightarrow{\gamma^*} \mathcal{T}^*_{\langle 0 \rangle}$ (in analogy to forming the free category over a graph).

[1] not to be confused with the graph-theoretic notion of this name

The generated language

- Freely extend γ to a multifunctor $\boldsymbol{G}^* \xrightarrow{\gamma^*} \mathcal{T}^*_{\langle 0 \rangle}$ (in analogy to forming the free category over a graph).
- The γ^*-image of the hom-set $\langle S, \varepsilon \rangle \boldsymbol{G}^*$ in $\langle M, M^0 \rangle \mathcal{T}^*_{\langle 0 \rangle}$ consists of certain tree-like composite diagrams with some leaves in $\mathcal{T}$.

[1] not to be confused with the graph-theoretic notion of this name

The generated language

- Freely extend γ to a multifunctor $\boldsymbol{G}^* \xrightarrow{\gamma^*} \mathcal{T}^*_{\langle 0\rangle}$ (in analogy to forming the free category over a graph).
- The γ^*-image of the hom-set $\langle S, \varepsilon\rangle \boldsymbol{G}^*$ in $\langle M, M^0\rangle \mathcal{T}^*_{\langle 0\rangle}$ consists of certain tree-like composite diagrams with some leaves in $\mathcal{T}$.
- The yields of these diagrams, *i.e.*, the words formed by their leaves, constitute the language generated by γ.

[1] not to be confused with the graph-theoretic notion of this name

The generated language

- Freely extend γ to a multifunctor $\boldsymbol{G}^* \xrightarrow{\gamma^*} \mathcal{T}^*_{\langle 0 \rangle}$ (in analogy to forming the free category over a graph).
- The γ^*-image of the hom-set $\langle S, \varepsilon \rangle \boldsymbol{G}^*$ in $\langle M, M^0 \rangle \mathcal{T}^*_{\langle 0 \rangle}$ consists of certain tree-like composite diagrams with some leaves in $\mathcal{T}$.
- The yields of these diagrams, *i.e.*, the words formed by their leaves, constitute the language generated by γ.

We would like to identify words generated in this fashion with their diagrams.

[1]not to be confused with the graph-theoretic notion of this name

The generated language

- Freely extend γ to a multifunctor $\boldsymbol{G}^* \xrightarrow{\gamma^*} \mathcal{T}^*_{\langle 0 \rangle}$ (in analogy to forming the free category over a graph).
- The γ^*-image of the hom-set $\langle S, \varepsilon \rangle \boldsymbol{G}^*$ in $\langle M, M^0 \rangle \mathcal{T}^*_{\langle 0 \rangle}$ consists of certain tree-like composite diagrams with some leaves in $\mathcal{T}$.
- The yields of these diagrams, *i.e.*, the words formed by their leaves, constitute the language generated by γ.

We would like to identify words generated in this fashion with their diagrams. Hence we consider $\mathcal{T}_{\langle 0 \rangle}$ as a "reflexive multigraph[1]" with the default multiarrows being distinguished.

[1] not to be confused with the graph-theoretic notion of this name

The generated language

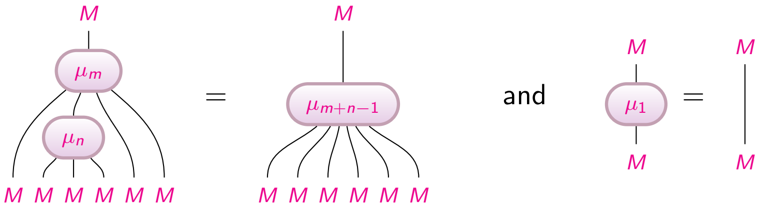
- Freely extend γ to a multifunctor $\boldsymbol{G}^* \xrightarrow{\gamma^*} \mathcal{T}^*_{\langle 0 \rangle}$ (in analogy to forming the free category over a graph).
- The γ^*-image of the hom-set $\langle S, \varepsilon \rangle \boldsymbol{G}^*$ in $\langle M, M^0 \rangle \mathcal{T}^*_{\langle 0 \rangle}$ consists of certain tree-like composite diagrams with some leaves in $\mathcal{T}$.
- The yields of these diagrams, *i.e.*, the words formed by their leaves, constitute the language generated by γ.

We would like to identify words generated in this fashion with their diagrams. Hence we consider $\mathcal{T}_{\langle 0 \rangle}$ as a "reflexive multigraph[1]" with the default multiarrows being distinguished.

The intention is to have the default multiarrows obey certain identifications in the free multicategory $\mathcal{T}^*_{\langle 0 \rangle}$; hence its construction needs to be revised:

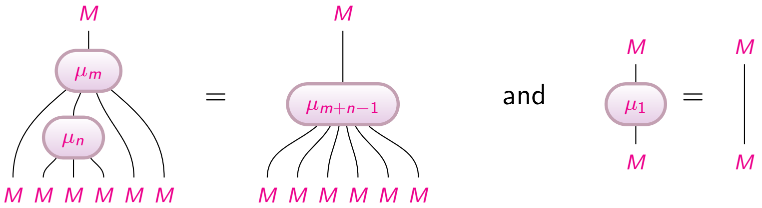
[1] not to be confused with the graph-theoretic notion of this name

Identifications in $\mathcal{T}^*_{\langle 0 \rangle}$

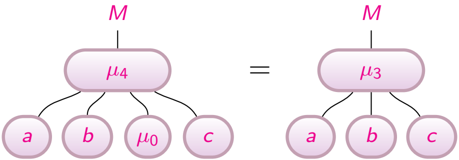


for $m, n \in \mathbb{N}$, $m > 0$.

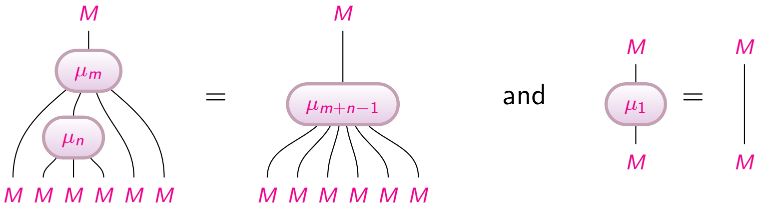
Identifications in $\mathcal{T}^*_{\langle 0 \rangle}$



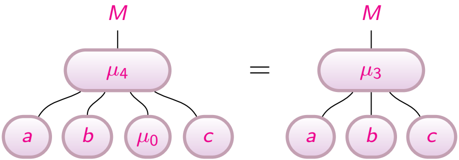
for $m, n \in \mathbb{N}$, $m > 0$. This allows the elimination of μ_0-leaves, *e.g.*,



Identifications in $\mathcal{T}^*_{\langle 0 \rangle}$

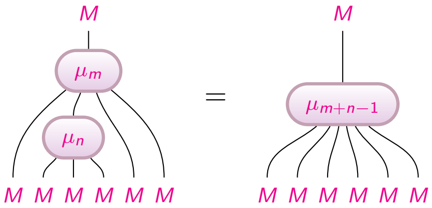


for $m, n \in \mathbb{N}$, $m > 0$. This allows the elimination of μ_0-leaves, *e.g.*,

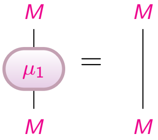


Any generated $w \in \mathcal{T}^*$ appears as yield directly underneath $\mu_{|w|}$.

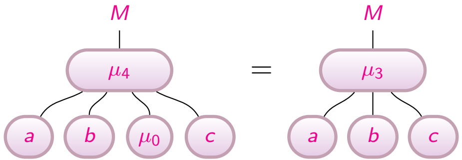
Identifications in $\mathcal{T}^*_{\langle 0 \rangle}$



and



for $m, n \in \mathbb{N}$, $m > 0$. This allows the elimination of μ_0-leaves, *e.g.*,



Any generated $w \in \mathcal{T}^*$ appears as yield directly underneath $\mu_{|w|}$.

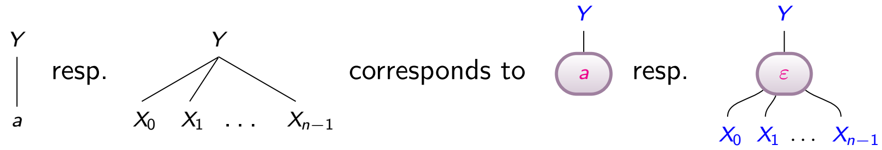
This motivates us to write ε not only for μ_0, but also for μ_n, $n \in \mathbb{N}$.

Where's the beef?

There is an obvious translation between the traditional tree-view and the multiarrow view of productions, and hence of derivation diagrams:

Where's the beef?

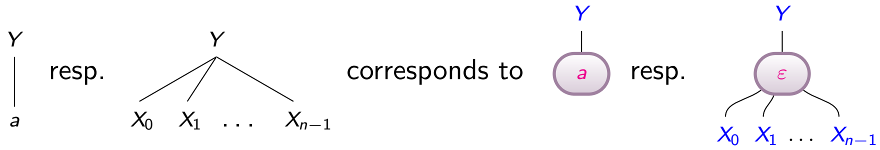
There is an obvious translation between the traditional tree-view and the multiarrow view of productions, and hence of derivation diagrams:



where $a \in \mathcal{T}^{\varepsilon} := \mathcal{T} + \{\varepsilon\}$.

Where's the beef?

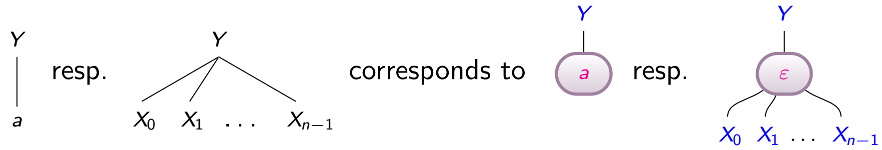
There is an obvious translation between the traditional tree-view and the multiarrow view of productions, and hence of derivation diagrams:



where $a \in \mathcal{T}^{\varepsilon} := \mathcal{T} + \{\varepsilon\}$. The nonterminals are labeling nodes on the left, but wires on the right!

Where's the beef?

There is an obvious translation between the traditional tree-view and the multiarrow view of productions, and hence of derivation diagrams:

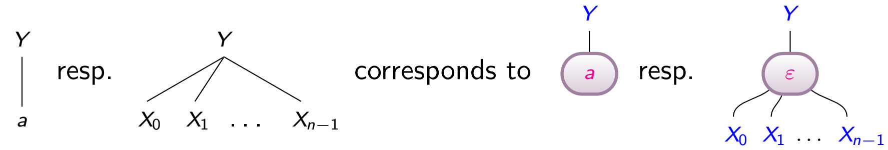


where $a \in \mathcal{T}^{\varepsilon} := \mathcal{T} + \{\varepsilon\}$. The nonterminals are labeling nodes on the left, but wires on the right!

Besides a certain elegance of the new approach and the better handling of ε-productions ("peanuts"),

Where's the beef?

There is an obvious translation between the traditional tree-view and the multiarrow view of productions, and hence of derivation diagrams:



where $a \in \mathcal{T}^{\varepsilon} := \mathcal{T} + \{\varepsilon\}$. The nonterminals are labeling nodes on the left, but wires on the right!

Besides a certain elegance of the new approach and the better handling of ε-productions ("peanuts"), how do we "sell" this to computer scientists or the tree-people (Ents?), who seem to be perfectly happy with the traditional approach?

Normal forms in the multi-setting

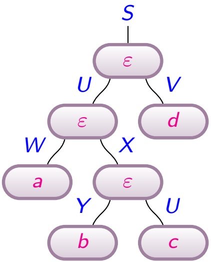
It makes sense to adapt the algorithm for transforming a wCNF grammar into CNF to the multigraph setting.

Normal forms in the multi-setting

It makes sense to adapt the algorithm for transforming a wCNF grammar into CNF to the multigraph setting. (We will not present this algorithm.)

Normal forms in the multi-setting

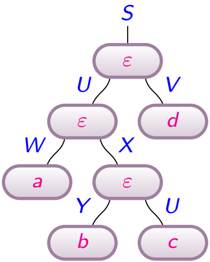
It makes sense to adapt the algorithm for transforming a wCNF grammar into CNF to the multigraph setting. (We will not present this algorithm.)



Consider derivation diagrams for a CFG à la Walters in CNF, i.e., elements of $\langle S, \varepsilon \rangle \boldsymbol{G}^*$ with multiarrows labeled by elements of $\mathcal{T}^\varepsilon$ and wires labeled by $\boldsymbol{G}$-objects.

Normal forms in the multi-setting

It makes sense to adapt the algorithm for transforming a wCNF grammar into CNF to the multigraph setting. (We will not present this algorithm.)

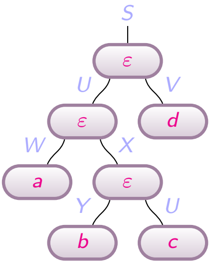


Consider derivation diagrams for a CFG à la Walters in CNF, i.e., elements of $\langle S, \varepsilon \rangle \boldsymbol{G}^*$ with multiarrows labeled by elements of $\mathcal{T}^\varepsilon$ and wires labeled by $\boldsymbol{G}$-objects.

Disregarding for the moment the wire-labels,

Normal forms in the multi-setting

It makes sense to adapt the algorithm for transforming a wCNF grammar into CNF to the multigraph setting. (We will not present this algorithm.)

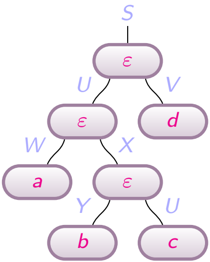


Consider derivation diagrams for a CFG à la Walters in CNF, i.e., elements of $\langle S, \varepsilon \rangle \boldsymbol{G}^*$ with multiarrows labeled by elements of $\mathcal{T}^{\varepsilon}$ and wires labeled by $\boldsymbol{G}$-objects.

Disregarding for the moment the wire-labels,

Normal forms in the multi-setting

It makes sense to adapt the algorithm for transforming a wCNF grammar into CNF to the multigraph setting. (We will not present this algorithm.)

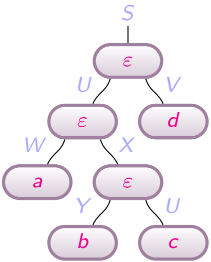


Consider derivation diagrams for a CFG à la Walters in CNF, i.e., elements of $\langle S, \varepsilon \rangle \boldsymbol{G}^*$ with multiarrows labeled by elements of $\mathcal{T}^{\varepsilon}$ and wires labeled by $\boldsymbol{G}$-objects.

Disregarding for the moment the wire-labels, these are instances of the inductive datatype of binary trees with leaves in $\mathcal{T}$:

Normal forms in the multi-setting

It makes sense to adapt the algorithm for transforming a wCNF grammar into CNF to the multigraph setting. (We will not present this algorithm.)



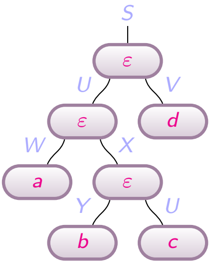
Consider derivation diagrams for a CFG à la Walters in CNF, i.e., elements of $\langle S, \varepsilon \rangle \boldsymbol{G}^*$ with multiarrows labeled by elements of $\mathcal{T}^\varepsilon$ and wires labeled by $\boldsymbol{G}$-objects.

Disregarding for the moment the wire-labels, these are instances of the inductive datatype of binary trees with leaves in $\mathcal{T}$:

$$\boldsymbol{btree}\,\mathcal{T} = \boldsymbol{tip}\,\mathcal{T} \mid \boldsymbol{bin}(\boldsymbol{btree}\,\mathcal{T}, \boldsymbol{btree}\,\mathcal{T}) = \mathcal{T} + \boldsymbol{btree}\,\mathcal{T} \times \boldsymbol{btree}\,\mathcal{T}$$

Normal forms in the multi-setting

It makes sense to adapt the algorithm for transforming a wCNF grammar into CNF to the multigraph setting. (We will not present this algorithm.)



Consider derivation diagrams for a CFG à la Walters in CNF, i.e., elements of $\langle S, \varepsilon \rangle \boldsymbol{G}^*$ with multiarrows labeled by elements of $\mathcal{T}^{\varepsilon}$ and wires labeled by $\boldsymbol{G}$-objects.

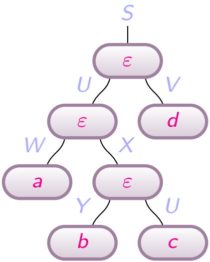
Disregarding for the moment the wire-labels, these are instances of the inductive datatype of binary trees with leaves in $\mathcal{T}$:

$$\boldsymbol{btree}\,\mathcal{T} = \boldsymbol{tip}\,\mathcal{T} \mid \boldsymbol{bin}(\boldsymbol{btree}\,\mathcal{T}, \boldsymbol{btree}\,\mathcal{T}) = \mathcal{T} + \boldsymbol{btree}\,\mathcal{T} \times \boldsymbol{btree}\,\mathcal{T}$$

Some programmers [*cf.*, Bird, deMoor, Ex. 1.13, 1.14] know this to be isomorphic to the inductive datatype of general trees with all nodes in $\mathcal{T}$:

Normal forms in the multi-setting

It makes sense to adapt the algorithm for transforming a wCNF grammar into CNF to the multigraph setting. (We will not present this algorithm.)



Consider derivation diagrams for a CFG à la Walters in CNF, i.e., elements of $\langle S, \varepsilon\rangle \boldsymbol{G}^*$ with multiarrows labeled by elements of $\mathcal{T}^\varepsilon$ and wires labeled by $\boldsymbol{G}$-objects.

Disregarding for the moment the wire-labels, these are instances of the inductive datatype of binary trees with leaves in $\mathcal{T}$:

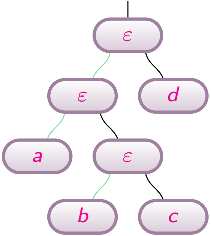
$$\boldsymbol{btree}\,\mathcal{T} = \boldsymbol{tip}\,\mathcal{T} \mid \boldsymbol{bin}(\boldsymbol{btree}\,\mathcal{T}, \boldsymbol{btree}\,\mathcal{T}) = \mathcal{T} + \boldsymbol{btree}\,\mathcal{T} \times \boldsymbol{btree}\,\mathcal{T}$$

Some programmers [*cf.*, Bird, deMoor, Ex. 1.13, 1.14] know this to be isomorphic to the inductive datatype of general trees with all nodes in $\mathcal{T}$:

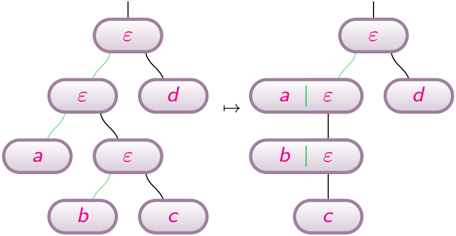
$$\boldsymbol{gtree}\,\mathcal{T} = \boldsymbol{node}(\mathcal{T}, \boldsymbol{listl}(\boldsymbol{gtree}\,\mathcal{T})) = \mathcal{T} \times \boldsymbol{listl}(\boldsymbol{gtree}\,\mathcal{T})$$

The isomorphism from $btree$ to $gtree$ is just uncurrying:

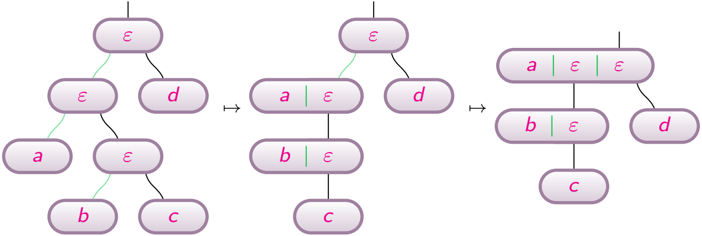
The isomorphism from $btree$ to $gtree$ is just uncurrying:



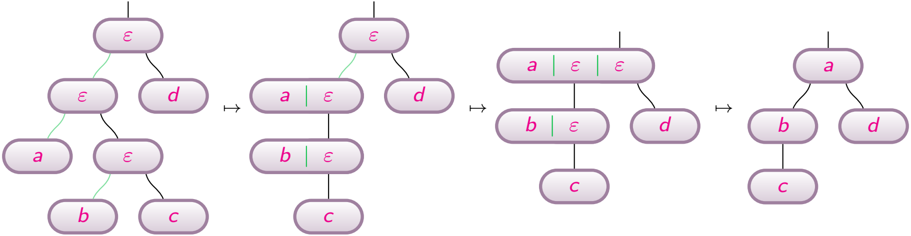
The isomorphism from $\boldsymbol{btree}$ to $\boldsymbol{gtree}$ is just uncurrying:



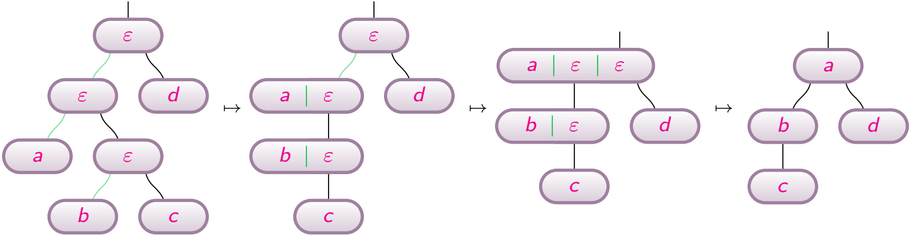
The isomorphism from $\boldsymbol{btree}$ to $\boldsymbol{gtree}$ is just uncurrying:



The isomorphism from ***btree*** to ***gtree*** is just uncurrying:

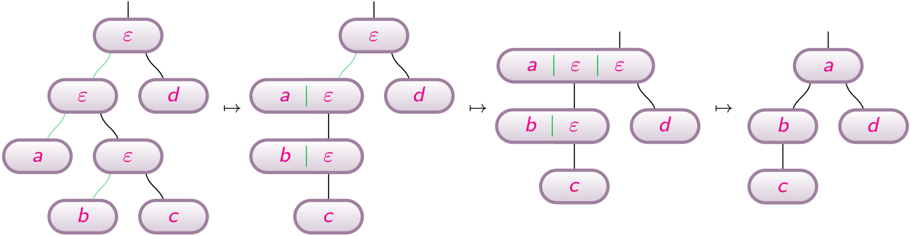


The isomorphism from $\boldsymbol{btree}$ to $\boldsymbol{gtree}$ is just uncurrying:



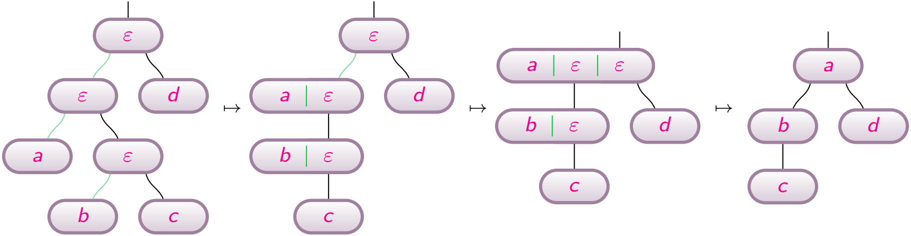
- ▷ obtain "composite nodes" by collapsing the green wires;

The isomorphism from $btree$ to $gtree$ is just uncurrying:



- ▷ obtain "composite nodes" by collapsing the green wires;
- ▷ revert to ordinary nodes by concatenation of labels.

The isomorphism from $\boldsymbol{btree}$ to $\boldsymbol{gtree}$ is just uncurrying:

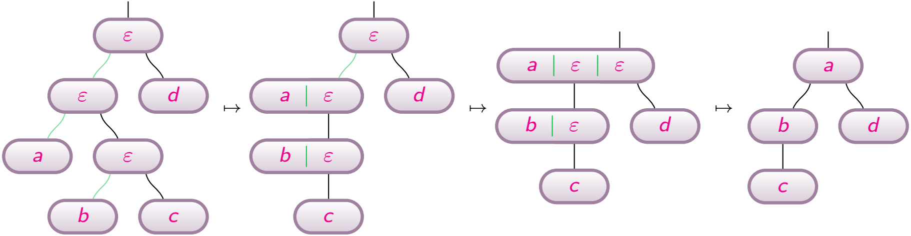


- obtain "composite nodes" by collapsing the green wires;
- revert to ordinary nodes by concatenation of labels.

Flattening these diagrams to strings requires parentheses; on the left the ε-nodes then serve as implicit left application operators.

$$(a(bc))d \qquad \mapsto \qquad a(b(c), d)$$

The isomorphism from ***btree*** to ***gtree*** is just uncurrying:

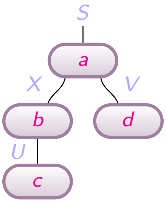


- ▷ obtain "composite nodes" by collapsing the green wires;
- ▷ revert to ordinary nodes by concatenation of labels.

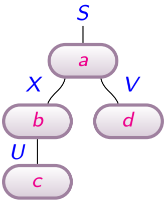
Flattening these diagrams to strings requires parentheses; on the left the ε-nodes then serve as implicit left application operators.

$$(a(bc))d \qquad \mapsto \qquad a(b(c), d)$$

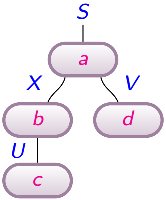
(There is a second (better?) isomorphism utilizing reverse Polish notation.)



Recovering the wire-labels,

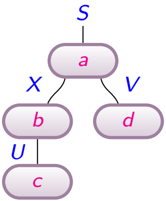


Recovering the wire-labels,



Recovering the wire-labels, the resulting diagram ought to be interpreted as derivation in a GNF grammar with conventional productions

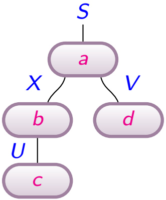
$$S \longrightarrow aXV \quad , \quad X \longrightarrow bU \quad , \quad U \longrightarrow c \quad , \quad V \longrightarrow d$$



Recovering the wire-labels, the resulting diagram ought to be interpreted as derivation in a GNF grammar with conventional productions

$$S \longrightarrow aXV \quad , \quad X \longrightarrow bU \quad , \quad U \longrightarrow c \quad , \quad V \longrightarrow d$$

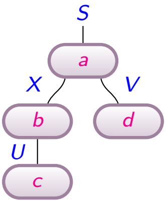
Of course, this diagram no longer lives in the reflexive multigraph $\mathcal{T}_{\langle 0 \rangle}$, but rather in $\mathcal{T}_{\langle \mathbb{N} \rangle}$,



Recovering the wire-labels, the resulting diagram ought to be interpreted as derivation in a GNF grammar with conventional productions

$$S \longrightarrow aXV \quad , \quad X \longrightarrow bU \quad , \quad U \longrightarrow c \quad , \quad V \longrightarrow d$$

Of course, this diagram no longer lives in the reflexive multigraph $\mathcal{T}_{\langle 0 \rangle}$, but rather in $\mathcal{T}_{\langle \mathbb{N} \rangle}$, where all hom-sets coincide with $\mathcal{T} + \{\varepsilon\}$.

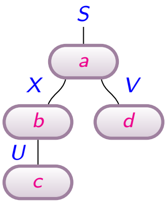


Recovering the wire-labels, the resulting diagram ought to be interpreted as derivation in a GNF grammar with conventional productions

$$S \to aXV \quad , \quad X \to bU \quad , \quad U \to c \quad , \quad V \to d$$

Of course, this diagram no longer lives in the reflexive multigraph $\mathcal{T}_{\langle 0 \rangle}$, but rather in $\mathcal{T}_{\langle \mathbb{N} \rangle}$, where all hom-sets coincide with $\mathcal{T} + \{\varepsilon\}$.

Currying the non-nullary productions of a GNF and splitting the results clearly yields an equivalent CNF:

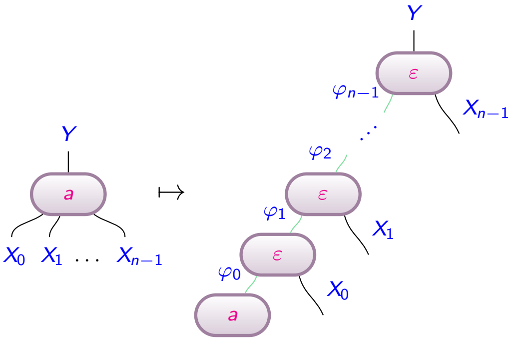


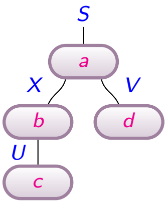
Recovering the wire-labels, the resulting diagram ought to be interpreted as derivation in a GNF grammar with conventional productions

$$S \to aXV \quad , \quad X \to bU \quad , \quad U \to c \quad , \quad V \to d$$

Of course, this diagram no longer lives in the reflexive multigraph $\mathcal{T}_{\langle 0 \rangle}$, but rather in $\mathcal{T}_{\langle \mathbb{N} \rangle}$, where all hom-sets coincide with $\mathcal{T} + \{\varepsilon\}$.

Currying the non-nullary productions of a GNF and splitting the results clearly yields an equivalent CNF:





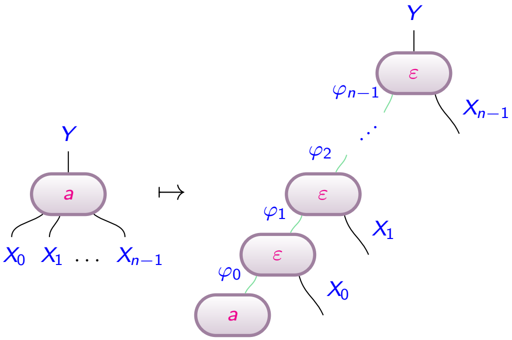
Recovering the wire-labels, the resulting diagram ought to be interpreted as derivation in a GNF grammar with conventional productions

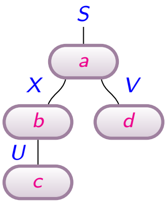
$$S \to aXV \quad , \quad X \to bU \quad , \quad U \to c \quad , \quad V \to d$$

Of course, this diagram no longer lives in the reflexive multigraph $\mathcal{T}_{\langle 0 \rangle}$, but rather in $\mathcal{T}_{\langle \mathbb{N} \rangle}$, where all hom-sets coincide with $\mathcal{T} + \{\varepsilon\}$.

Currying the non-nullary productions of a GNF and splitting the results clearly yields an equivalent CNF:

Not being able to recover previously collapsed green wires creates no problems.





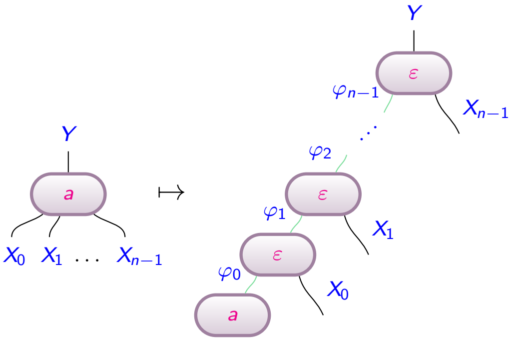
Recovering the wire-labels, the resulting diagram ought to be interpreted as derivation in a GNF grammar with conventional productions

$$S \to aXV \quad , \quad X \to bU \quad , \quad U \to c \quad , \quad V \to d$$

Of course, this diagram no longer lives in the reflexive multigraph $\mathcal{T}_{\langle 0 \rangle}$, but rather in $\mathcal{T}_{\langle \mathbb{N} \rangle}$, where all hom-sets coincide with $\mathcal{T} + \{\varepsilon\}$.

Currying the non-nullary productions of a GNF and splitting the results clearly yields an equivalent CNF:

Not being able to recover previously collapsed green wires creates no problems. The new objects can be mapped on the old ones.



The issue of recursiveness

Conversely, constructing a GNF from a CNF requires uncurrying certain left-combinations of CNF productions to obtain single GNF productions.

The issue of recursiveness

Conversely, constructing a GNF from a CNF requires uncurrying certain left-combinations of CNF productions to obtain single GNF productions. For only finitely many such combinations to exist, the CNF-multigraph G must not admit left feedback (or not be left-recursive).

The issue of recursiveness

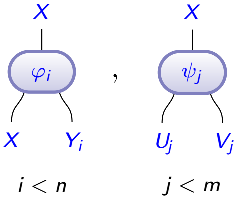
Conversely, constructing a GNF from a CNF requires uncurrying certain left-combinations of CNF productions to obtain single GNF productions. For only finitely many such combinations to exist, the CNF-multigraph $\boldsymbol{G}$ must not admit left feedback (or not be left-recursive).

Adapting classical grammar techniques, eliminating direct feedback at X can be achieved by new nonterminals $\bar{X}$ and X' and new productions:

The issue of recursiveness

Conversely, constructing a GNF from a CNF requires uncurrying certain left-combinations of CNF productions to obtain single GNF productions. For only finitely many such combinations to exist, the CNF-multigraph $\boldsymbol{G}$ must not admit left feedback (or not be left-recursive).

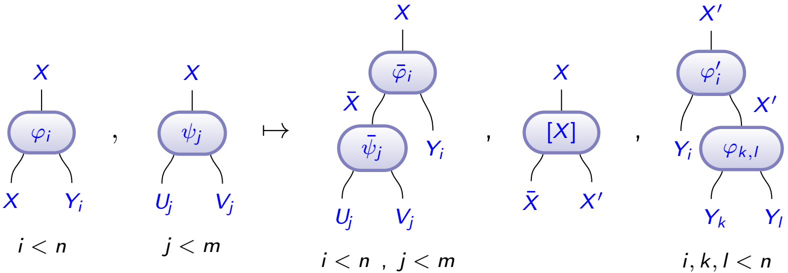
Adapting classical grammar techniques, eliminating direct feedback at X can be achieved by new nonterminals $\bar{X}$ and X' and new productions:



The issue of recursiveness

Conversely, constructing a GNF from a CNF requires uncurrying certain left-combinations of CNF productions to obtain single GNF productions. For only finitely many such combinations to exist, the CNF-multigraph $\boldsymbol{G}$ must not admit left feedback (or not be left-recursive).

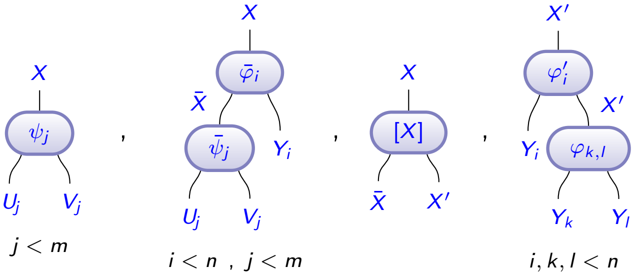
Adapting classical grammar techniques, eliminating direct feedback at X can be achieved by new nonterminals $\bar{X}$ and X' and new productions:



The issue of recursiveness

Conversely, constructing a GNF from a CNF requires uncurrying certain left-combinations of CNF productions to obtain single GNF productions. For only finitely many such combinations to exist, the CNF-multigraph $\boldsymbol{G}$ must not admit left feedback (or not be left-recursive).

Adapting classical grammar techniques, eliminating direct feedback at X can be achieved by new nonterminals $\bar{X}$ and X' and new productions:



Of course, delayed feedback has to be eliminated as well.

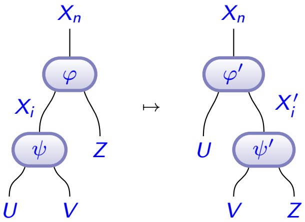
Of course, delayed feedback has to be eliminated as well. This requires a recursive procedure with some arbitrary ordering on the G-objects:

Of course, delayed feedback has to be eliminated as well. This requires a recursive procedure with some arbitrary ordering on the $\boldsymbol{G}$-objects:

Once the objects X_i, $i < n$, have been treated, perform "re-associations"

Of course, delayed feedback has to be eliminated as well. This requires a recursive procedure with some arbitrary ordering on the G-objects:

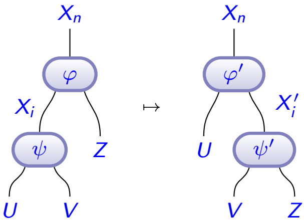
Once the objects X_i, $i < n$, have been treated, perform “re-associations”



until no multiarrows of the form $X_n \xrightarrow{\varphi} X_i\, Z$ with $i < n$ are left.

Of course, delayed feedback has to be eliminated as well. This requires a recursive procedure with some arbitrary ordering on the G-objects:

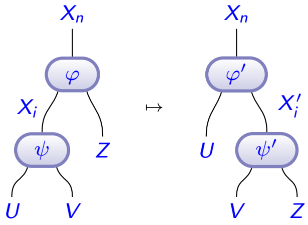
Once the objects X_i , $i < n$, have been treated, perform "re-associations"



until no multiarrows of the form $X_n \xrightarrow{\varphi} X_i\, Z$ with $i < n$ are left. Then eliminate direct feedback at X_n as described above.

Of course, delayed feedback has to be eliminated as well. This requires a recursive procedure with some arbitrary ordering on the G-objects:

Once the objects X_i, $i < n$, have been treated, perform "re-associations"

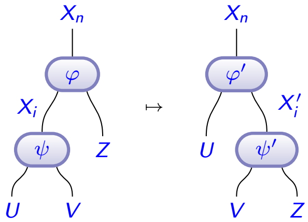


until no multiarrows of the form $X_n \xrightarrow{\varphi} X_i\, Z$ with $i < n$ are left. Then eliminate direct feedback at X_n as described above.

This is an expensive operation as it can square the size of the grammar, *i.e.*, the sum over the symbols in all productions.

Of course, delayed feedback has to be eliminated as well. This requires a recursive procedure with some arbitrary ordering on the $\boldsymbol{G}$-objects:

Once the objects X_i , $i < n$, have been treated, perform "re-associations"



until no multiarrows of the form $X_n \xrightarrow{\varphi} X_i\, Z$ with $i < n$ are left. Then eliminate direct feedback at X_n as described above.

This is an expensive operation as it can square the size of the grammar, *i.e.*, the sum over the symbols in all productions. Uncurrying, like all classical algorithms, can lead to another squaring in size.

The regular case and automata

Interpreting (faithful) morphisms of reflexive(!) graphs, $G \xrightarrow{\gamma} \mathcal{T}_{\langle 1 \rangle}$ with G finite, as regular $\mathcal{T}$-grammars provided Walters' motivation.

The regular case and automata

Interpreting (faithful) morphisms of reflexive(!) graphs, $G \xrightarrow{\gamma} \mathcal{T}_{\langle 1 \rangle}$ with G finite, as regular $\mathcal{T}$-grammars provided Walters' motivation.

Thinking of such graph morphisms instead as labeled transition systems, possibly with ε-transitions, nonterminals turn into states.

The regular case and automata

Interpreting (faithful) morphisms of reflexive(!) graphs, $G \xrightarrow{\gamma} \mathcal{T}_{\langle 1 \rangle}$ with G finite, as regular $\mathcal{T}$-grammars provided Walters' motivation.

Thinking of such graph morphisms instead as labeled transition systems, possibly with ε-transitions, nonterminals turn into states. Specifying initial and final states results in the notion of finite state automaton (FSA).

The regular case and automata

Interpreting (faithful) morphisms of reflexive(!) graphs, $G \xrightarrow{\gamma} \mathcal{T}_{\langle 1 \rangle}$ with G finite, as regular $\mathcal{T}$-grammars provided Walters' motivation.

Thinking of such graph morphisms instead as labeled transition systems, possibly with ε-transitions, nonterminals turn into states. Specifying initial and final states results in the notion of finite state automaton (FSA).

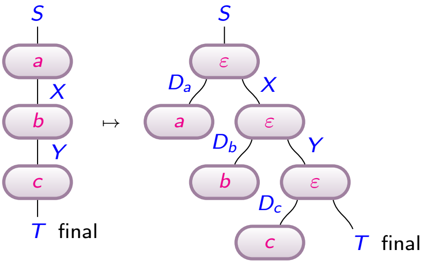
Now we can apply currying to the transitions of such an automaton:

The regular case and automata

Interpreting (faithful) morphisms of reflexive(!) graphs, $G \xrightarrow{\gamma} \mathcal{T}_{\langle 1 \rangle}$ with G finite, as regular $\mathcal{T}$-grammars provided Walters' motivation.

Thinking of such graph morphisms instead as labeled transition systems, possibly with ε-transitions, nonterminals turn into states. Specifying initial and final states results in the notion of finite state automaton (FSA).

Now we can apply currying to the transitions of such an automaton:

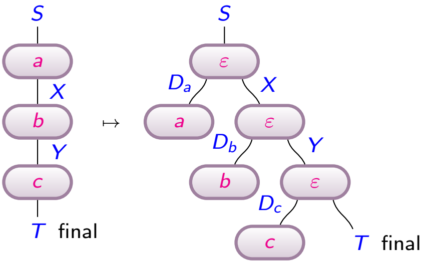


The regular case and automata

Interpreting (faithful) morphisms of reflexive(!) graphs, $G \xrightarrow{\gamma} \mathcal{T}_{\langle 1 \rangle}$ with G finite, as regular $\mathcal{T}$-grammars provided Walters' motivation.

Thinking of such graph morphisms instead as labeled transition systems, possibly with ε-transitions, nonterminals turn into states. Specifying initial and final states results in the notion of finite state automaton (FSA).

Now we can apply currying to the transitions of such an automaton:



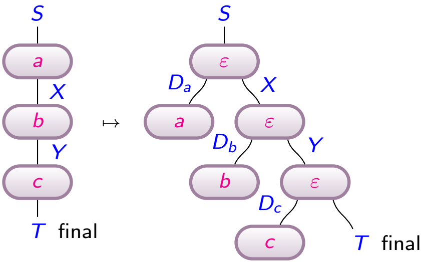
This essentially results in a right-linear and hence regular grammar that is in CNF iff the automaton does not have ε-transitions.

The regular case and automata

Interpreting (faithful) morphisms of reflexive(!) graphs, $\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}_{\langle 1 \rangle}$ with $\boldsymbol{G}$ finite, as regular $\mathcal{T}$-grammars provided Walters' motivation.

Thinking of such graph morphisms instead as labeled transition systems, possibly with ε-transitions, nonterminals turn into states. Specifying initial and final states results in the notion of finite state automaton (FSA).

Now we can apply currying to the transitions of such an automaton:



This essentially results in a right-linear and hence regular grammar that is in CNF iff the automaton does not have ε-transitions.

Final states provide an externally imposed mechanism for termination, as $\mathcal{T}_{\langle 1 \rangle}$ has no default for this.

Extend the notion of CFG to faithful multigraph morphisms $\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}_{\langle \mathbb{N} \rangle}$.

Extend the notion of CFG to faithful multigraph morphisms $G \xrightarrow{\gamma} \mathcal{T}_{\langle\mathbb{N}\rangle}$.

These can be turned into push-down automata (PDA), provided that

Extend the notion of CFG to faithful multigraph morphisms $\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}_{\langle\mathbb{N}\rangle}$.

These can be turned into push-down automata (PDA), provided that

- strings(!) of nonterminals are interpreted as (internal) states;

Extend the notion of CFG to faithful multigraph morphisms $G \xrightarrow{\gamma} \mathcal{T}_{\langle\mathbb{N}\rangle}$.

These can be turned into push-down automata (PDA), provided that

- strings(!) of nonterminals are interpreted as (internal) states;
- only left-derivations are considered, which corresponds to the constrained access to the stack via its top.

Extend the notion of CFG to faithful multigraph morphisms $G \xrightarrow{\gamma} \mathcal{T}_{\langle \mathbb{N} \rangle}$.

These can be turned into push-down automata (PDA), provided that

- strings(!) of nonterminals are interpreted as (internal) states;
- only left-derivations are considered, which corresponds to the constrained access to the stack via its top.

The initial state is S , while the canonical final state is ε .

Extend the notion of CFG to faithful multigraph morphisms $\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}_{\langle\mathbb{N}\rangle}$.

These can be turned into push-down automata (PDA), provided that

- strings(!) of nonterminals are interpreted as (internal) states;
- only left-derivations are considered, which corresponds to the constrained access to the stack via its top.

The initial state is S, while the canonical final state is ε.

In particular, CFG's in GNF correspond to push-down automata without ε-transitions, Extend the notion of CFG to faithful multigraph morphisms $\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}_{\langle \mathbb{N} \rangle}$.

These can be turned into push-down automata (PDA), provided that

- strings(!) of nonterminals are interpreted as (internal) states;
- only left-derivations are considered, which corresponds to the constrained access to the stack via its top.

The initial state is S, while the canonical final state is ε.

In particular, CFG's in GNF correspond to push-down automata without ε-transitions, and the transformation into GNF can be viewed as the elimination of ε-transitions from a general PDA.

Extend the notion of CFG to faithful multigraph morphisms $\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}_{\langle\mathbb{N}\rangle}$.

These can be turned into push-down automata (PDA), provided that

- strings(!) of nonterminals are interpreted as (internal) states;
- only left-derivations are considered, which corresponds to the constrained access to the stack via its top.

The initial state is S, while the canonical final state is ε.

In particular, CFG's in GNF correspond to push-down automata without ε-transitions, and the transformation into GNF can be viewed as the elimination of ε-transitions from a general PDA.

Also note that CFG-induced PDA's are pure in the sense of having only one external state.

Extend the notion of CFG to faithful multigraph morphisms $\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}_{\langle\mathbb{N}\rangle}$.

These can be turned into push-down automata (PDA), provided that

- strings(!) of nonterminals are interpreted as (internal) states;
- only left-derivations are considered, which corresponds to the constrained access to the stack via its top.

The initial state is S, while the canonical final state is ε.

In particular, CFG's in GNF correspond to push-down automata without ε-transitions, and the transformation into GNF can be viewed as the elimination of ε-transitions from a general PDA.

Also note that CFG-induced PDA's are pure in the sense of having only one external state. Still they can accept any context-free language.

Extend the notion of CFG to faithful multigraph morphisms $G \xrightarrow{\gamma} \mathcal{T}_{\langle \mathbb{N} \rangle}$.

These can be turned into push-down automata (PDA), provided that

- strings(!) of nonterminals are interpreted as (internal) states;
- only left-derivations are considered, which corresponds to the constrained access to the stack via its top.

The initial state is S, while the canonical final state is ε.

In particular, CFG's in GNF correspond to push-down automata without ε-transitions, and the transformation into GNF can be viewed as the elimination of ε-transitions from a general PDA.

Also note that CFG-induced PDA's are pure in the sense of having only one external state. Still they can accept any context-free language.

Conventionally, PDA's are defined with external states. Eliminating the stack then leads to FSA's, also called 0-PDA's.

Extend the notion of CFG to faithful multigraph morphisms $G \xrightarrow{\gamma} \mathcal{T}_{\langle\mathbb{N}\rangle}$.

These can be turned into push-down automata (PDA), provided that

- strings(!) of nonterminals are interpreted as (internal) states;
- only left-derivations are considered, which corresponds to the constrained access to the stack via its top.

The initial state is S, while the canonical final state is ε.

In particular, CFG's in GNF correspond to push-down automata without ε-transitions, and the transformation into GNF can be viewed as the elimination of ε-transitions from a general PDA.

Also note that CFG-induced PDA's are pure in the sense of having only one external state. Still they can accept any context-free language.

Conventionally, PDA's are defined with external states. Eliminating the stack then leads to FSA's, also called 0-PDA's. As seen above, FSA's can also be realized by pure PDA's, where the stack is limited to depth 1.

(Multi-)Graph Comprehension

The familiar bijection for labeled transition systems

$$\frac{G \xrightarrow{\gamma} \mathcal{T}_{\langle 1 \rangle}}{\mathcal{T}_{\langle 1 \rangle} \xrightarrow{\Gamma} \boldsymbol{rel}}$$

(faithful graph morphism)

(graph morphism)

readily carries over to the multi-setting:

$$\frac{G \xrightarrow{\gamma} \mathcal{T}_{\langle \mathbb{N} \rangle}}{\mathcal{T}_{\langle \mathbb{N} \rangle} \xrightarrow{\Gamma} \boldsymbol{rel}}$$

(faithful multigraph morphism)

(multigraph morphism)

(Multi-)Graph Comprehension

The familiar bijection for labeled transition systems

$$\frac{\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}_{\langle 1 \rangle}}{\mathcal{T}_{\langle 1 \rangle} \xrightarrow{\Gamma} \boldsymbol{rel}} \qquad \begin{array}{l}\text{(faithful graph morphism)}\\ \text{(graph morphism)}\end{array}$$

readily carries over to the multi-setting:

$$\frac{\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}_{\langle \mathbb{N} \rangle}}{\mathcal{T}_{\langle \mathbb{N} \rangle} \xrightarrow{\Gamma} \boldsymbol{rel}} \qquad \begin{array}{l}\text{(faithful multigraph morphism)}\\ \text{(multigraph morphism)}\end{array}$$

where the multigraph structure of $\boldsymbol{rel}$ is given by $\times$.

(Multi-)Graph Comprehension

The familiar bijection for labeled transition systems

$$\frac{\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}_{\langle 1 \rangle}}{\mathcal{T}_{\langle 1 \rangle} \xrightarrow{\Gamma} \boldsymbol{rel}}$$

(faithful graph morphism)

(graph morphism)

readily carries over to the multi-setting:

$$\frac{\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}_{\langle \mathbb{N} \rangle}}{\mathcal{T}_{\langle \mathbb{N} \rangle} \xrightarrow{\Gamma} \boldsymbol{rel}}$$

(faithful multigraph morphism)

(multigraph morphism)

where the multigraph structure of $\boldsymbol{rel}$ is given by $\times$.

Both bijections remain valid, if $\mathcal{T}_{\langle 1 \rangle}$ and $\mathcal{T}_{\langle \mathbb{N} \rangle}$ are replaced by an arbitrary graph, resp., multigraph as control.

(Multi-)Graph Comprehension

The familiar bijection for labeled transition systems

$$\frac{\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}_{\langle 1 \rangle}}{\mathcal{T}_{\langle 1 \rangle} \xrightarrow{\Gamma} \boldsymbol{rel}} \qquad \begin{array}{l}\text{(faithful graph morphism)}\\ \text{(graph morphism)}\end{array}$$

readily carries over to the multi-setting:

$$\frac{\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}_{\langle \mathbb{N} \rangle}}{\mathcal{T}_{\langle \mathbb{N} \rangle} \xrightarrow{\Gamma} \boldsymbol{rel}} \qquad \begin{array}{l}\text{(faithful multigraph morphism)}\\ \text{(multigraph morphism)}\end{array}$$

where the multigraph structure of $\boldsymbol{rel}$ is given by $\times$.

Both bijections remain valid, if $\mathcal{T}_{\langle 1 \rangle}$ and $\mathcal{T}_{\langle \mathbb{N} \rangle}$ are replaced by an arbitrary graph, resp., multigraph as control. Restricting to finite $\boldsymbol{G}$ imposes appropriate finiteness conditions on the "denominators".

Closing up under "vertical" composition, we obtain bijections

$$\frac{\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}^*_{\langle 1\rangle}}{\mathcal{T}^*_{\langle 1\rangle} \xrightarrow{\Gamma} \boldsymbol{rel}} \qquad \begin{array}{l}\text{(faithful functor)}\\ \text{(lax functor)}\end{array}$$

Moving to the free monoidal category in the multi-setting yields

$$\frac{\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}^\star_{\langle \mathbb{N}\rangle}}{\mathcal{T}^\star_{\langle \mathbb{N}\rangle} \xrightarrow{\Gamma} \boldsymbol{rel}} \qquad \begin{array}{l}\text{(faithful strict monoidal functor)}\\ \text{(strict monoidal lax functor)}\end{array}$$

Closing up under "vertical" composition, we obtain bijections

$$\frac{\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}^{*}_{\langle 1 \rangle}}{\mathcal{T}^{*}_{\langle 1 \rangle} \xrightarrow{\Gamma} \boldsymbol{rel}}$$

(faithful functor)

(lax functor)

Moving to the free monoidal category in the multi-setting yields

$$\frac{\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}^{\star}_{\langle \mathbb{N} \rangle}}{\mathcal{T}^{\star}_{\langle \mathbb{N} \rangle} \xrightarrow{\Gamma} \boldsymbol{rel}}$$

(faithful strict monoidal functor)

(strict monoidal lax functor)

where $\boldsymbol{G}$ now is a (strict monoidal) category.

Closing up under "vertical" composition, we obtain bijections

$$\frac{\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}^{*}_{\langle 1 \rangle}}{\mathcal{T}^{*}_{\langle 1 \rangle} \xrightarrow{\Gamma} \boldsymbol{rel}}$$

(faithful functor)

(lax functor)

Moving to the free monoidal category in the multi-setting yields

$$\frac{\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}^{\star}_{\langle \mathbb{N} \rangle}}{\mathcal{T}^{\star}_{\langle \mathbb{N} \rangle} \xrightarrow{\Gamma} \boldsymbol{rel}}$$

(faithful strict monoidal functor)

(strict monoidal lax functor)

where $\boldsymbol{G}$ now is a (strict monoidal) category. Again, general (strict monoidal) categories can serve as controls instead of $\mathcal{T}^{*}_{\langle 1 \rangle}$, resp., $\mathcal{T}^{\star}_{\langle \mathbb{N} \rangle}$.

Closing up under "vertical" composition, we obtain bijections

$$\frac{\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}^{*}_{\langle 1 \rangle}}{\mathcal{T}^{*}_{\langle 1 \rangle} \xrightarrow{\Gamma} \boldsymbol{rel}} \qquad \begin{array}{l}\text{(faithful functor)} \\ \text{(lax functor)}\end{array}$$

Moving to the free monoidal category in the multi-setting yields

$$\frac{\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}^{\star}_{\langle \mathbb{N} \rangle}}{\mathcal{T}^{\star}_{\langle \mathbb{N} \rangle} \xrightarrow{\Gamma} \boldsymbol{rel}} \qquad \begin{array}{l}\text{(faithful strict monoidal functor)} \\ \text{(strict monoidal lax functor)}\end{array}$$

where $\boldsymbol{G}$ now is a (strict monoidal) category. Again, general (strict monoidal) categories can serve as controls instead of $\mathcal{T}^{*}_{\langle 1 \rangle}$, resp., $\mathcal{T}^{\star}_{\langle \mathbb{N} \rangle}$.

Oplax (monoidal) natural transformations turn out to be the appropriate type of morphism in the "denominators" to encode simulations.

Closing up under "vertical" composition, we obtain bijections

$$\frac{\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}^*_{\langle 1 \rangle}}{\mathcal{T}^*_{\langle 1 \rangle} \xrightarrow{\Gamma} \boldsymbol{rel}} \qquad \begin{array}{l}\text{(faithful functor)} \\ \text{(lax functor)}\end{array}$$

Moving to the free monoidal category in the multi-setting yields

$$\frac{\boldsymbol{G} \xrightarrow{\gamma} \mathcal{T}^\star_{\langle \mathbb{N} \rangle}}{\mathcal{T}^\star_{\langle \mathbb{N} \rangle} \xrightarrow{\Gamma} \boldsymbol{rel}} \qquad \begin{array}{l}\text{(faithful strict monoidal functor)} \\ \text{(strict monoidal lax functor)}\end{array}$$

where $\boldsymbol{G}$ now is a (strict monoidal) category. Again, general (strict monoidal) categories can serve as controls instead of $\mathcal{T}^*_{\langle 1 \rangle}$, resp., $\mathcal{T}^\star_{\langle \mathbb{N} \rangle}$.

Oplax (monoidal) natural transformations turn out to be the appropriate type of morphism in the "denominators" to encode simulations.

Instead of $\boldsymbol{rel}$, matrix categories over other rigs yield further instances of this phenomenon, like probabilistic or weighted transition systems.

Obvious questions

Obvious questions

- What about general grammars with unrestricted productions

$$\mathcal{V}^* \times \mathcal{N} \times \mathcal{V}^* \xrightarrow{\rightarrow} \mathcal{V}^* \quad \text{where} \quad \mathcal{V} := \mathcal{T} + \mathcal{N}$$

(the left side has to contain at least one nonterminal)?

Obvious questions

- What about general grammars with unrestricted productions

$$\mathcal{V}^* \times \mathcal{N} \times \mathcal{V}^* \xrightarrow{\ \to\ } \mathcal{V}^* \quad \text{where} \quad \mathcal{V} := \mathcal{T} + \mathcal{N}$$

(the left side has to contain at least one nonterminal)?

These generate all semidecidable languages, which are precisely those recognizable by Turing machines.

Obvious questions

- What about general grammars with unrestricted productions

$$\mathcal{V}^* \times \mathcal{N} \times \mathcal{V}^* \xrightarrow{\;\to\;} \mathcal{V}^* \quad \text{where} \quad \mathcal{V} := \mathcal{T} + \mathcal{N}$$

  (the left side has to contain at least one nonterminal)?

  These generate all semidecidable languages, which are precisely those recognizable by Turing machines.

- What about “polygraphs” and consequently “polycategories”?

Obvious questions

- What about general grammars with unrestricted productions

$$\mathcal{V}^* \times \mathcal{N} \times \mathcal{V}^* \xrightarrow{\ \rightarrow\ } \mathcal{V}^* \quad \text{where} \quad \mathcal{V} := \mathcal{T} + \mathcal{N}$$

  (the left side has to contain at least one nonterminal)?

  These generate all semidecidable languages, which are precisely those recognizable by Turing machines.

- What about "polygraphs" and consequently "polycategories"?

  The well-developed theory of "planar" polycategories and poly-bicategories, where the poly-2-cells can have finitely many inputs and outputs, *cf.*, [Cockett, Koslowski, Seely: TAC 11(2)] and [Koslowski: TAC 14(7)],

Obvious questions

- What about general grammars with unrestricted productions

$$\mathcal{V}^* \times \mathcal{N} \times \mathcal{V}^* \xrightarrow{\ \to\ } \mathcal{V}^* \quad \text{where} \quad \mathcal{V} := \mathcal{T} + \mathcal{N}$$

  (the left side has to contain at least one nonterminal)?

  These generate all semidecidable languages, which are precisely those recognizable by Turing machines.

- What about "polygraphs" and consequently "polycategories"?

  The well-developed theory of "planar" polycategories and poly-bicategories, where the poly-2-cells can have finitely many inputs and outputs, *cf.*, [Cockett, Koslowski, Seely: TAC 11(2)] and [Koslowski: TAC 14(7)], is based on logical considerations (calculus of 2-sided sequents) and uses cut along single wires as "vertical" composition.

Obvious questions

- What about general grammars with unrestricted productions

$$\mathcal{V}^* \times \mathcal{N} \times \mathcal{V}^* \xrightarrow{\ \rightarrow\ } \mathcal{V}^* \quad \text{where} \quad \mathcal{V} := \mathcal{T} + \mathcal{N}$$

  (the left side has to contain at least one nonterminal)?

  These generate all semidecidable languages, which are precisely those recognizable by Turing machines.

- What about "polygraphs" and consequently "polycategories"?

  The well-developed theory of "planar" polycategories and poly-bicategories, where the poly-2-cells can have finitely many inputs and outputs, *cf.*, [Cockett, Koslowski, Seely: TAC 11(2)] and [Koslowski: TAC 14(7)], is based on logical considerations (calculus of 2-sided sequents) and uses cut along single wires as "vertical" composition. It would seem to be incomatible with the replacement process of general grammars.

How can we extend pure PDA's as above in a finite fashion in order to handle general grammars?

How can we extend pure PDA's as above in a finite fashion in order to handle general grammars?

Even for left derivations, the first subword to which a production can be applied need not be a prefix of the current stack; in fact the depth of its occurrence is unbounded.

How can we extend pure PDA's as above in a finite fashion in order to handle general grammars?

Even for left derivations, the first subword to which a production can be applied need not be a prefix of the current stack; in fact the depth of its occurrence is unbounded. *E.g.*, consider productions $AB \longrightarrow EF$ and $CD \longrightarrow B$, and a current stack of the form $A^n CD \ldots$.

How can we extend pure PDA's as above in a finite fashion in order to handle general grammars?

Even for left derivations, the first subword to which a production can be applied need not be a prefix of the current stack; in fact the depth of its occurrence is unbounded. *E.g.*, consider productions $AB \longrightarrow EF$ and $CD \longrightarrow B$, and a current stack of the form $A^nCD\ldots$.

- We can allow to look deeper into the stack, beyond the top element.

How can we extend pure PDA's as above in a finite fashion in order to handle general grammars?

Even for left derivations, the first subword to which a production can be applied need not be a prefix of the current stack; in fact the depth of its occurrence is unbounded. *E.g.*, consider productions $AB \longrightarrow EF$ and $CD \longrightarrow B$, and a current stack of the form $A^n CD \ldots$.

- We can allow to look deeper into the stack, beyond the top element. As long as the depth is bounded, this does not help.

How can we extend pure PDA's as above in a finite fashion in order to handle general grammars?

Even for left derivations, the first subword to which a production can be applied need not be a prefix of the current stack; in fact the depth of its occurrence is unbounded. *E.g.*, consider productions $AB \longrightarrow EF$ and $CD \longrightarrow B$, and a current stack of the form $A^n CD \ldots$.

- We can allow to look deeper into the stack, beyond the top element. As long as the depth is bounded, this does not help.
- We can add finitely many external states to examine subwords on the stack for being left sides of productions.

How can we extend pure PDA's as above in a finite fashion in order to handle general grammars?

Even for left derivations, the first subword to which a production can be applied need not be a prefix of the current stack; in fact the depth of its occurrence is unbounded. *E.g.*, consider productions $AB \longrightarrow EF$ and $CD \longrightarrow B$, and a current stack of the form $A^n CD \ldots$.

- We can allow to look deeper into the stack, beyond the top element. As long as the depth is bounded, this does not help.
- We can add finitely many external states to examine subwords on the stack for being left sides of productions. However, his does not help in remembering the prefix before the first such subword.

How can we extend pure PDA's as above in a finite fashion in order to handle general grammars?

Even for left derivations, the first subword to which a production can be applied need not be a prefix of the current stack; in fact the depth of its occurrence is unbounded. *E.g.*, consider productions $AB \longrightarrow EF$ and $CD \longrightarrow B$, and a current stack of the form $A^n CD \ldots$.

- We can allow to look deeper into the stack, beyond the top element. As long as the depth is bounded, this does not help.
- We can add finitely many external states to examine subwords on the stack for being left sides of productions. However, his does not help in remembering the prefix before the first such subword.
- We can add the ability to "bend wires out of the way" until we find a first left hand side of a production, and "bend the wires back" later.

How can we extend pure PDA's as above in a finite fashion in order to handle general grammars?

Even for left derivations, the first subword to which a production can be applied need not be a prefix of the current stack; in fact the depth of its occurrence is unbounded. *E.g.*, consider productions $AB \longrightarrow EF$ and $CD \longrightarrow B$, and a current stack of the form $A^n CD \ldots$.

- We can allow to look deeper into the stack, beyond the top element. As long as the depth is bounded, this does not help.
- We can add finitely many external states to examine subwords on the stack for being left sides of productions. However, his does not help in remembering the prefix before the first such subword.
- We can add the ability to "bend wires out of the way" until we find a first left hand side of a production, and "bend the wires back" later. Recall that we only have to deal with finitely many nonterminals.

How can we extend pure PDA's as above in a finite fashion in order to handle general grammars?

Even for left derivations, the first subword to which a production can be applied need not be a prefix of the current stack; in fact the depth of its occurrence is unbounded. *E.g.*, consider productions $AB \longrightarrow EF$ and $CD \longrightarrow B$, and a current stack of the form $A^n CD \ldots$.

- We can allow to look deeper into the stack, beyond the top element. As long as the depth is bounded, this does not help.
- We can add finitely many external states to examine subwords on the stack for being left sides of productions. However, his does not help in remembering the prefix before the first such subword.
- We can add the ability to "bend wires out of the way" until we find a first left hand side of a production, and "bend the wires back" later. Recall that we only have to deal with finitely many nonterminals.

A combination of the last two ideas indeed will do the trick.

Polygraphs with linear adjoints

Recall that planar poly-bicategories as well as polycategories admit a very nice notion of so-called linear adjoints, together with a mate calculus.

Polygraphs with linear adjoints

Recall that planar poly-bicategories as well as polycategories admit a very nice notion of so-called linear adjoints, together with a mate calculus.

In analogy with the notion of reflexive (multi-)graph, where implicitly a set of equations is specified by which to factor the absolutely free (multi-)category, we consider adjoint polygraphs, which are supposed to already contain the units and counits of the "free polycategory with linear adjunctions" over it as distinguished poly-2-cells.

Polygraphs with linear adjoints

Recall that planar poly-bicategories as well as polycategories admit a very nice notion of so-called linear adjoints, together with a mate calculus.

In analogy with the notion of reflexive (multi-)graph, where implicitly a set of equations is specified by which to factor the absolutely free (multi-)category, we consider adjoint polygraphs, which are supposed to already contain the units and counits of the "free polycategory with linear adjunctions" over it as distinguished poly-2-cells.

Of course, $\mathcal{T}_{\langle\mathbb{N}\rangle}$ needs to be replaced by an obvious polygraph $\mathcal{T}^{\langle\mathbb{N}\rangle}_{\langle\mathbb{N}\rangle}$, where again all hom-sets coincide with $\mathcal{T} + \{\varepsilon\}$.

Polygraphs with linear adjoints

Recall that planar poly-bicategories as well as polycategories admit a very nice notion of so-called linear adjoints, together with a mate calculus.

In analogy with the notion of reflexive (multi-)graph, where implicitly a set of equations is specified by which to factor the absolutely free (multi-)category, we consider adjoint polygraphs, which are supposed to already contain the units and counits of the "free polycategory with linear adjunctions" over it as distinguished poly-2-cells.

Of course, $\mathcal{T}_{\langle\mathbb{N}\rangle}$ needs to be replaced by an obvious polygraph $\mathcal{T}^{\langle\mathbb{N}\rangle}_{\langle\mathbb{N}\rangle}$, where again all hom-sets coincide with $\mathcal{T} + \{\varepsilon\}$.

In machine terms we obtain pure 2-PDA's, which are more powerful than pure PDA's, as they can recognize shuffles of context-free languages (which need not be context-free anymore).

Polygraphs with linear adjoints

Recall that planar poly-bicategories as well as polycategories admit a very nice notion of so-called linear adjoints, together with a mate calculus.

In analogy with the notion of reflexive (multi-)graph, where implicitly a set of equations is specified by which to factor the absolutely free (multi-)category, we consider adjoint polygraphs, which are supposed to already contain the units and counits of the "free polycategory with linear adjunctions" over it as distinguished poly-2-cells.

Of course, $\mathcal{T}_{\langle\mathbb{N}\rangle}$ needs to be replaced by an obvious polygraph $\mathcal{T}_{\langle\mathbb{N}\rangle}^{\langle\mathbb{N}\rangle}$, where again all hom-sets coincide with $\mathcal{T} + \{\varepsilon\}$.

In machine terms we obtain pure 2-PDA's, which are more powerful than pure PDA's, as they can recognize shuffles of context-free languages (which need not be context-free anymore).

2-PDA's with external states are well-known to be equivalent to Turing machines.

To do

- ▷ Do pure 2-PDA's suffice to simulate Turing machines for decision problems?

To do

- Do pure 2-PDA's suffice to simulate Turing machines for decision problems?
- Even if they do, external states may still be useful for computational problems (even for 1-PDA's).

To do

- Do pure 2-PDA's suffice to simulate Turing machines for decision problems?
- Even if they do, external states may still be useful for computational problems (even for 1-PDA's). What is the "right" way of adding external states, short of grafting them on?

To do

- Do pure 2-PDA's suffice to simulate Turing machines for decision problems?
- Even if they do, external states may still be useful for computational problems (even for 1-PDA's). What is the "right" way of adding external states, short of grafting them on?
- What about transducers, *i.e.*, how should output be handled?

To do

- Do pure 2-PDA's suffice to simulate Turing machines for decision problems?
- Even if they do, external states may still be useful for computational problems (even for 1-PDA's). What is the "right" way of adding external states, short of grafting them on?
- What about transducers, *i.e.*, how should output be handled?
- Work out the details for polygraph comprehension.

Thank you!