Formal grammars after the Cuban missile crisis

Alexander Okhotin

St. Petersburg State University, Russia

WDCM 2022 session in honour of Victor L. Selivanov
the 28th of October, A. D. 2022

Part I

Introduction

Formal grammars: a model of syntax

- Substrings with known properties, one after another.

Formal grammars: a model of syntax

- Substrings with known properties, one after another.

"A noun phrase, followed by a verb phrase,
is a sentence" ($S \to NP\ VP$).

Formal grammars: a model of syntax

- Substrings with known properties, one after another.

"A noun phrase, followed by a verb phrase,
is a sentence" ($S \to NP\ VP$).

- *Immediate constituent analysis*, later Chomsky.

Formal grammars: a model of syntax

- Substrings with known properties, one after another.

"A noun phrase, followed by a verb phrase,
is a sentence" ($S \to NP\ VP$).

- *Immediate constituent analysis*, later Chomsky.
- Formal grammars: now a classical subject.

Formal grammars: a model of syntax

- Substrings with known properties, one after another.

"A noun phrase, followed by a verb phrase,
is a sentence" ($S \to NP\ VP$).

- *Immediate constituent analysis*, later Chomsky.
- Formal grammars: now a classical subject.
- Ongoing research: new models, new ideas.

Formal grammars: a model of syntax

- Substrings with known properties, one after another.

"A noun phrase, followed by a verb phrase,
is a sentence" ($S \to NP\ VP$).

- *Immediate constituent analysis*, later Chomsky.
- Formal grammars: now a classical subject.
- Ongoing research: new models, new ideas.
- Classroom presentation rooted in the early 1960s.

Formal grammars before October 1962

- Chomsky, 1956: phrase-structure grammars, definition by rewriting.

$$S \Longrightarrow \text{NP VP} \Longrightarrow \ldots \Longrightarrow \text{Time flies like an arrow}$$

Formal grammars before October 1962

- Chomsky, 1956: phrase-structure grammars, definition by rewriting.

$$S \Longrightarrow \text{NP VP} \Longrightarrow \ldots \Longrightarrow \text{Time flies like an arrow}$$

- Chomsky, 1959: more general rewriting systems.

Formal grammars before October 1962

- Chomsky, 1956: phrase-structure grammars, definition by rewriting.

$$S \Longrightarrow \text{NP VP} \Longrightarrow \ldots \Longrightarrow \text{Time flies like an arrow}$$

- Chomsky, 1959: more general rewriting systems.
- ALGOL 60 committee: Backus–Naur form.

Formal grammars before October 1962

- Chomsky, 1956: phrase-structure grammars, definition by rewriting.

$$S \Longrightarrow \text{NP VP} \Longrightarrow \ldots \Longrightarrow \text{Time flies like an arrow}$$

- Chomsky, 1959: more general rewriting systems.
- ALGOL 60 committee: Backus–Naur form.
- Parikh, 1961: commutative image is semilinear.

Formal grammars before October 1962

- Chomsky, 1956: phrase-structure grammars, definition by rewriting.

$$S \Longrightarrow \text{NP VP} \Longrightarrow \ldots \Longrightarrow \text{Time flies like an arrow}$$

- Chomsky, 1959: more general rewriting systems.
- ALGOL 60 committee: Backus–Naur form.
- Parikh, 1961: commutative image is semilinear.
- Bar-Hillel et al., 1961: pumping lemma, undecidable properties.

Formal grammars before October 1962

- Chomsky, 1956: phrase-structure grammars, definition by rewriting.

$$S \Longrightarrow \text{NP VP} \Longrightarrow \ldots \Longrightarrow \text{Time flies like an arrow}$$

- Chomsky, 1959: more general rewriting systems.
- ALGOL 60 committee: Backus–Naur form.
- Parikh, 1961: commutative image is semilinear.
- Bar-Hillel et al., 1961: pumping lemma, undecidable properties.
- Various authors: early recursive descent parsing.

October 1962: the Cuban missile crisis

Old systematics

- Chomsky (1956): grammars as rewriting $A \to BC$.

Old systematics

- Chomsky (1956): grammars as rewriting $A \to BC$.
- Chomsky (1957): generalized rewriting $AB \to CD$.

Old systematics

- Chomsky (1956): grammars as rewriting $A \to BC$.
- Chomsky (1957): generalized rewriting $AB \to CD$.

The Chomsky hierarchy

Old systematics

- Chomsky (1956): grammars as rewriting $A \to BC$.
- Chomsky (1957): generalized rewriting $AB \to CD$.

The Chomsky hierarchy

0. Recursively enumerable sets.

Old systematics

- Chomsky (1956): grammars as rewriting $A \to BC$.
- Chomsky (1957): generalized rewriting $AB \to CD$.

The Chomsky hierarchy

0. Recursively enumerable sets.
1. NSPACE(n)

Old systematics

- Chomsky (1956): grammars as rewriting $A \to BC$.
- Chomsky (1957): generalized rewriting $AB \to CD$.

The Chomsky hierarchy

0. Recursively enumerable sets.
1. NSPACE(n) (presumed "context-sensitive grammars")

Old systematics

- Chomsky (1956): grammars as rewriting $A \to BC$.
- Chomsky (1957): generalized rewriting $AB \to CD$.

The Chomsky hierarchy

0. Recursively enumerable sets.
1. NSPACE(n) (presumed "context-sensitive grammars")
2. Grammars (henceforth named "context-free")

Old systematics

- Chomsky (1956): grammars as rewriting $A \to BC$.
- Chomsky (1957): generalized rewriting $AB \to CD$.

The Chomsky hierarchy

0. Recursively enumerable sets.
1. NSPACE(n) (presumed "context-sensitive grammars")
2. Grammars (henceforth named "context-free")
3. Regular languages: DSPACE(const).

Old systematics

- Chomsky (1956): grammars as rewriting $A \to BC$.
- Chomsky (1957): generalized rewriting $AB \to CD$.

The Chomsky hierarchy

0. Recursively enumerable sets.
1. NSPACE(n) (presumed "context-sensitive grammars")
2. Grammars (henceforth named "context-free")
3. Regular languages: DSPACE(const).

✓ Ancient complexity theory, important in its time.

Old systematics

- Chomsky (1956): grammars as rewriting $A \to BC$.
- Chomsky (1957): generalized rewriting $AB \to CD$.

The Chomsky hierarchy

0. Recursively enumerable sets.
1. NSPACE(n) (presumed "context-sensitive grammars")
2. Grammars (henceforth named "context-free")
3. Regular languages: DSPACE(const).

✓ Ancient complexity theory, important in its time.

- $AB \to CD$: strange nondeterministic computation.

A terminological issue

- Chomsky's *"context-free grammars"*:
  a chance name for a fundamental concept.

A terminological issue

- Chomsky's *"context-free grammars"*:
  a chance name for a fundamental concept.
- Chomsky's *"context-sensitive"*: abandoned direction.

A terminological issue

- Chomsky's *"context-free grammars"*:
  a chance name for a fundamental concept.
- Chomsky's *"context-sensitive"*: abandoned direction.
- None of today's major models use contexts.

A terminological issue

- Chomsky's *"context-free grammars"*:
  a chance name for a fundamental concept.
- Chomsky's *"context-sensitive"*: abandoned direction.
- None of today's major models use contexts.
- Why the strange name "context-free", then?

A terminological issue

- Chomsky's *"context-free grammars"*:
  a chance name for a fundamental concept.
- Chomsky's *"context-sensitive"*: abandoned direction.
- None of today's major models use contexts.
- Why the strange name "context-free", then?
- Main effect: new students led astray.

A terminological issue

- Chomsky's *"context-free grammars"*:
  a chance name for a fundamental concept.
- Chomsky's *"context-sensitive"*: abandoned direction.
- None of today's major models use contexts.
- Why the strange name "context-free", then?
- Main effect: new students led astray.
- Proposed name:

A terminological issue

- Chomsky's *"context-free grammars"*:
  a chance name for a fundamental concept.
- Chomsky's *"context-sensitive"*: abandoned direction.
- None of today's major models use contexts.
- Why the strange name "context-free", then?
- Main effect: new students led astray.
- Proposed name:

ordinary grammars

A terminological issue

- Chomsky's *"context-free grammars"*:
  a chance name for a fundamental concept.
- Chomsky's *"context-sensitive"*: abandoned direction.
- None of today's major models use contexts.
- Why the strange name "context-free", then?
- Main effect: new students led astray.
- Proposed name:

ordinary grammars

- Other grammar families: in relation to the ordinary.

Part II

Grammar families

Intuitive syntactic desriptions

- Substrings with known properties, one after another.

Intuitive syntactic desriptions

- Substrings with known properties, one after another.

Example (the Dyck language)

- ε is well-nested.

Intuitive syntactic desriptions

- Substrings with known properties, one after another.

Example (the Dyck language)

- ε is well-nested.
- if w is well-nested, then so is awb.

Intuitive syntactic desriptions

- Substrings with known properties, one after another.

Example (the Dyck language)

- ε is well-nested.
- if w is well-nested, then so is awb.
- if u and v are well-nested, then so is uv.

Intuitive syntactic desriptions

- Substrings with known properties, one after another.

Example (the Dyck language)

- ε is well-nested.
- if w is well-nested, then so is awb.
- if u and v are well-nested, then so is uv.

- As a grammar: $S \to \varepsilon \mid aSb \mid SS$.

Intuitive syntactic desriptions

- Substrings with known properties, one after another.

Example (the Dyck language)

- ε is well-nested.
- if w is well-nested, then so is awb.
- if u and v are well-nested, then so is uv.

- As a grammar: $S \to \varepsilon \mid aSb \mid SS$.

Definition

Ordinary grammar: $G = (\Sigma, N, R, S)$, with rules

$$A \to X_1 \dots X_\ell \qquad (\ell \geqslant 0,\ X_1, \dots, X_\ell \in \Sigma \cup N)$$

Intuitive syntactic desriptions

- Substrings with known properties, one after another.

Example (the Dyck language)

- ε is well-nested.
- if w is well-nested, then so is awb.
- if u and v are well-nested, then so is uv.

- As a grammar: $S \to \varepsilon \mid aSb \mid SS$.

Definition

Ordinary grammar: $G = (\Sigma, N, R, S)$, with rules

$$A \to X_1 \dots X_\ell \qquad (\ell \geqslant 0,\ X_1, \dots, X_\ell \in \Sigma \cup N)$$

- "$\to$" denotes implication in the other direction.

Form of syntactic desriptions

"if u and v are well-nested, then so is uv"

Form of syntactic desriptions

"if u and v are well-nested, then so is uv"

- "well-nested": syntactic category.

Form of syntactic desriptions

"if u and v are well-nested, then so is uv"

- "well-nested": syntactic category.
- u and v: constituents.

Form of syntactic desriptions

"if u and v are well-nested, then so is uv"

- "well-nested": syntactic category.
- u and v: constituents.
- uv: operation on constituents.

Form of syntactic desriptions

"if u and v are well-nested, then so is uv"

- "well-nested": syntactic category.
- u and v: constituents.
- uv: operation on constituents.
- Quantifier over u and v: logical operation.

Form of syntactic desriptions

"if u and v are well-nested, then so is uv"

- "well-nested": syntactic category.
- u and v: constituents.
- uv: operation on constituents.
- Quantifier over u and v: logical operation.

"if w is well-nested, then so is awb" (2nd rule)

Form of syntactic desriptions

"if u and v are well-nested, then so is uv"

- "well-nested": syntactic category.
- u and v: constituents.
- uv: operation on constituents.
- Quantifier over u and v: logical operation.

"if w is well-nested, then so is awb" (2nd rule)

- Disjunction between rules: another logical operation.

Form of syntactic desriptions

"if u and v are well-nested, then so is uv"

- "well-nested": syntactic category.
- u and v: constituents.
- uv: operation on constituents.
- Quantifier over u and v: logical operation.

"if w is well-nested, then so is awb" (2nd rule)

- Disjunction between rules: another logical operation.

Formal grammar: a logic for describing syntax.

Grammar families

A family of grammars is characterized by three elements.

Grammar families

A family of grammars is characterized by three elements.

1. **Constituents.**
   - In ordinary grammars: substrings.

Grammar families

A family of grammars is characterized by three elements.

1. **Constituents.**
   - In ordinary grammars: substrings.
2. **Operations on constituents.**
   - In ordinary grammars: concatenation.

Grammar families

A family of grammars is characterized by three elements.

1. **Constituents.**
   - In ordinary grammars: substrings.
2. **Operations on constituents.**
   - In ordinary grammars: concatenation.
3. **Logical operations.**
   - In ordinary grammars: "$\exists$" as part of concatenation; disjunction.

Grammar families

A family of grammars is characterized by three elements.

1. **Constituents.**
   - In ordinary grammars: substrings.
2. **Operations on constituents.**
   - In ordinary grammars: concatenation.
3. **Logical operations.**
   - In ordinary grammars: "$\exists$" as part of concatenation; disjunction.

- Each element can be modified.

Grammar families

A family of grammars is characterized by three elements.

1. **Constituents.**
   - In ordinary grammars: substrings.
2. **Operations on constituents.**
   - In ordinary grammars: concatenation.
3. **Logical operations.**
   - In ordinary grammars: "$\exists$" as part of concatenation; disjunction.

- Each element can be modified.
- Restrictions and extensions of ordinary grammars.

Restricting ordinary grammars

Linear grammars

Operations on constituents: aw, wa, with $a \in \Sigma$.

Restricting ordinary grammars

Linear grammars

Operations on constituents: aw, wa, with $a \in \Sigma$.

Unambiguous grammars

Logical operations: disjoint "$\vee$"; "$\exists$" with $\leqslant 1$ matches.

Restricting ordinary grammars

Linear grammars

Operations on constituents: aw, wa, with $a \in \Sigma$.

Unambiguous grammars

Logical operations: disjoint "$\vee$"; "$\exists$" with $\leqslant 1$ matches.

- Input-driven automata (Mehlhorn, 1980).

Restricting ordinary grammars

Linear grammars

Operations on constituents: aw, wa, with $a \in \Sigma$.

Unambiguous grammars

Logical operations: disjoint "$\vee$"; "$\exists$" with $\leqslant 1$ matches.

- Input-driven automata (Mehlhorn, 1980).
  - "visibly pushdown automata" (Alur, Madhusudan, 2004)

Restricting ordinary grammars

Linear grammars

Operations on constituents: aw, wa, with $a \in \Sigma$.

Unambiguous grammars

Logical operations: disjoint "$\vee$"; "$\exists$" with $\leqslant 1$ matches.

- Input-driven automata (Mehlhorn, 1980).
  - "visibly pushdown automata" (Alur, Madhusudan, 2004)
  - Corresponding "bracketed" grammars.

Restricting ordinary grammars

Linear grammars

Operations on constituents: aw, wa, with $a \in \Sigma$.

Unambiguous grammars

Logical operations: disjoint "$\vee$"; "$\exists$" with $\leqslant 1$ matches.

- Input-driven automata (Mehlhorn, 1980).
  - "visibly pushdown automata" (Alur, Madhusudan, 2004)
  - Corresponding "bracketed" grammars.

Several bracketed grammar models

Constituents: well-nested strings;
operation on constituents: $<u>v$.

Extension with conjunction

- Add another logical operation: conjunction.

Extension with conjunction

- Add another logical operation: conjunction.

Definition (Okhotin, 2001)

Conjunctive grammar: $G = (\Sigma, N, R, S)$, with rules

$$A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_n \quad (n \geqslant 1,\ \alpha_1, \ldots, \alpha_n \in (\Sigma \cup N)^*)$$

Extension with conjunction

- Add another logical operation: conjunction.

Definition (Okhotin, 2001)

Conjunctive grammar: $G = (\Sigma, N, R, S)$, with rules

$$A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_n \quad (n \geqslant 1,\ \alpha_1, \ldots, \alpha_n \in (\Sigma \cup N)^*)$$

- "If w is represented according to each conjunct α_i, then w has the property A".

Extension with conjunction

- Add another logical operation: conjunction.

Definition (Okhotin, 2001)

Conjunctive grammar: $G = (\Sigma, N, R, S)$, with rules

$$A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_n \quad (n \geqslant 1,\ \alpha_1, \ldots, \alpha_n \in (\Sigma \cup N)^*)$$

- "If w is represented according to each conjunct α_i, then w has the property A".

Grammar for $\{a^n b^n c^n \mid n \geqslant 0\}$

Extension with conjunction

- Add another logical operation: conjunction.

Definition (Okhotin, 2001)

Conjunctive grammar: $G = (\Sigma, N, R, S)$, with rules

$$A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_n \quad (n \geqslant 1,\ \alpha_1, \ldots, \alpha_n \in (\Sigma \cup N)^*)$$

- "If w is represented according to each conjunct α_i, then w has the property A".

Grammar for $\{a^n b^n c^n \mid n \geqslant 0\}$

$$\begin{aligned} S &\to AB \,\&\, \ldots \\ A &\to aA \mid \varepsilon \\ B &\to bBc \mid \varepsilon \end{aligned}$$

Extension with conjunction

- Add another logical operation: conjunction.

Definition (Okhotin, 2001)

Conjunctive grammar: $G = (\Sigma, N, R, S)$, with rules

$$A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_n \quad (n \geqslant 1,\ \alpha_1, \ldots, \alpha_n \in (\Sigma \cup N)^*)$$

- "If w is represented according to each conjunct α_i, then w has the property A".

Grammar for $\{a^n b^n c^n \mid n \geqslant 0\}$

$$\begin{array}{lcl} S & \to & AB \,\&\, DC \\ A & \to & aA \mid \varepsilon \\ B & \to & bBc \mid \varepsilon \end{array} \qquad \begin{array}{lcl} D & \to & aDb \mid \varepsilon \\ C & \to & cC \mid \varepsilon \end{array}$$

Extension with conjunction and negation

Definition (Okhotin, 2004)

Boolean grammar: $G = (\Sigma, N, R, S)$, with rules

$$A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_m \,\&\, \neg\beta_1 \,\&\, \ldots \,\&\, \neg\beta_n$$

Extension with conjunction and negation

Definition (Okhotin, 2004)

Boolean grammar: $G = (\Sigma, N, R, S)$, with rules

$$A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_m \,\&\, \neg\beta_1 \,\&\, \ldots \,\&\, \neg\beta_n$$

- "If w is represented according to each conjunct α_i, and is not represented according to any β_j, then w has the property A".

Extension with conjunction and negation

Definition (Okhotin, 2004)

Boolean grammar: $G = (\Sigma, N, R, S)$, with rules

$$A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_m \,\&\, \neg\beta_1 \,\&\, \ldots \,\&\, \neg\beta_n$$

- "If w is represented according to each conjunct α_i, and is not represented according to any β_j, then w has the property A".

Grammar for $\{a^m b^n c^n \mid m \neq n\}$

Extension with conjunction and negation

Definition (Okhotin, 2004)

Boolean grammar: $G = (\Sigma, N, R, S)$, with rules

$$A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_m \,\&\, \neg\beta_1 \,\&\, \ldots \,\&\, \neg\beta_n$$

- "If w is represented according to each conjunct α_i, and is not represented according to any β_j, then w has the property A".

Grammar for $\{a^m b^n c^n \mid m \neq n\}$

$$\begin{array}{lcl} S & \to & AB \,\&\, \neg DC \\ A & \to & aA \mid \varepsilon \\ B & \to & bBc \mid \varepsilon \end{array} \qquad \begin{array}{lcl} D & \to & aDb \mid \varepsilon \\ C & \to & cC \mid \varepsilon \end{array}$$

Extension using substrings with gaps

- Constituents u_1-GAP-u_2-GAP- . . . -GAP-u_k.

Extension using substrings with gaps

- Constituents u_1-GAP-u_2-GAP- . . . -GAP-u_k.

Definition (Joshi et al., 1987; Seki et al., 1991)

Multi-component gr.: $G = (\Sigma, N, \dim, R, S)$, with each $A \in N$ having *dimension* $k = \dim A$, and with rules

$$A(\alpha_1, \ldots, \alpha_k) \to B_1(x_{1,1}, \ldots, x_{1,m_1}), \ldots, B_\ell(x_{\ell,1}, \ldots, x_{\ell,m_\ell})$$

Extension using substrings with gaps

- Constituents u_1-GAP-u_2-GAP- . . . -GAP-u_k.

Definition (Joshi et al., 1987; Seki et al., 1991)

Multi-component gr.: $G = (\Sigma, N, \dim, R, S)$, with each $A \in N$ having *dimension* $k = \dim A$, and with rules

$$A(\alpha_1, \ldots, \alpha_k) \to B_1(x_{1,1}, \ldots, x_{1,m_1}), \ldots, B_\ell(x_{\ell,1}, \ldots, x_{\ell,m_\ell})$$

- m_1-tuple, . . . , m_ℓ-tuple combined into a k-tuple.

Extension using substrings with gaps

- Constituents u_1-GAP-u_2-GAP- . . . -GAP-u_k.

Definition (Joshi et al., 1987; Seki et al., 1991)

Multi-component gr.: $G = (\Sigma, N, \dim, R, S)$, with each $A \in N$ having *dimension* $k = \dim A$, and with rules

$$A(\alpha_1, \ldots, \alpha_k) \to B_1(x_{1,1}, \ldots, x_{1,m_1}), \ldots, B_\ell(x_{\ell,1}, \ldots, x_{\ell,m_\ell})$$

- m_1-tuple, . . . , m_ℓ-tuple combined into a k-tuple.

Grammar for $\{a^n b^n c^n d^n \mid n \geqslant 0\}$

Extension using substrings with gaps

- Constituents u_1-GAP-u_2-GAP- . . . -GAP-u_k.

Definition (Joshi et al., 1987; Seki et al., 1991)

Multi-component gr.: $G = (\Sigma, N, \dim, R, S)$, with each $A \in N$ having *dimension* $k = \dim A$, and with rules

$$A(\alpha_1, \ldots, \alpha_k) \to B_1(x_{1,1}, \ldots, x_{1,m_1}), \ldots, B_\ell(x_{\ell,1}, \ldots, x_{\ell,m_\ell})$$

- m_1-tuple, . . . , m_ℓ-tuple combined into a k-tuple.

Grammar for $\{a^n b^n c^n d^n \mid n \geqslant 0\}$

$$\begin{aligned} S(xy) &\to A(x,y) \\ A(axb, cyd) &\to A(x,y) \\ A(\varepsilon,\varepsilon) &\to \text{TRUE} \end{aligned}$$

Well-nested multi-component grammars

- Constituents u_1-GAP-u_2-GAP- . . . -GAP-u_k.

Well-nested multi-component grammars

- Constituents u_1-GAP-u_2-GAP- . . . -GAP-u_k.

$$A(\alpha_1, \ldots, \alpha_k) \to B_1(x_{1,1}, \ldots, x_{1,m_1}), \ldots, B_\ell(x_{\ell,1}, \ldots, x_{\ell,m_\ell})$$

Well-nested multi-component grammars

- Constituents u_1-GAP-u_2-GAP- . . . -GAP-u_k.

$$A(\alpha_1, \ldots, \alpha_k) \to B_1(x_{1,1}, \ldots, x_{1,m_1}), \ldots, B_\ell(x_{\ell,1}, \ldots, x_{\ell,m_\ell})$$

- $\alpha_1 \ldots \alpha_k$: all $x_{i,j}$ shuffled.

Well-nested multi-component grammars

- Constituents u_1-GAP-u_2-GAP- . . . -GAP-u_k.

$$A(\alpha_1, \ldots, \alpha_k) \to B_1(x_{1,1}, \ldots, x_{1,m_1}), \ldots, B_\ell(x_{\ell,1}, \ldots, x_{\ell,m_\ell})$$

- $\alpha_1 \ldots \alpha_k$: all $x_{i,j}$ shuffled.
- Well-nested: in $\alpha_1 \ldots \alpha_k$, no crossings

  $\ldots x_{i,j} \ldots x_{i',m} \ldots x_{i,k} \ldots x_{i',n} \ldots$.

Well-nested multi-component grammars

- Constituents u_1-GAP-u_2-GAP- . . . -GAP-u_k.

$$A(\alpha_1, \ldots, \alpha_k) \to B_1(x_{1,1}, \ldots, x_{1,m_1}), \ldots, B_\ell(x_{\ell,1}, \ldots, x_{\ell,m_\ell})$$

- $\alpha_1 \ldots \alpha_k$: all $x_{i,j}$ shuffled.
- Well-nested: in $\alpha_1 \ldots \alpha_k$, no crossings
  $\ldots x_{i,j} \ldots x_{i',m} \ldots x_{i,k} \ldots x_{i',n} \ldots$.
- Well-nested 2-component: tree-adjoining, also head.

Well-nested multi-component grammars

- Constituents u_1-GAP-u_2-GAP- . . . -GAP-u_k.

$$A(\alpha_1, \ldots, \alpha_k) \to B_1(x_{1,1}, \ldots, x_{1,m_1}), \ldots, B_\ell(x_{\ell,1}, \ldots, x_{\ell,m_\ell})$$

- $\alpha_1 \ldots \alpha_k$: all $x_{i,j}$ shuffled.
- Well-nested: in $\alpha_1 \ldots \alpha_k$, no crossings

  $\ldots x_{i,j} \ldots x_{i',m} \ldots x_{i,k} \ldots x_{i',n} \ldots$.
- Well-nested 2-component: tree-adjoining, also head.
  - Constituents: substrings u, pairs (u, v).

Well-nested multi-component grammars

- Constituents u_1-GAP-u_2-GAP- . . . -GAP-u_k.

$$A(\alpha_1, \ldots, \alpha_k) \to B_1(x_{1,1}, \ldots, x_{1,m_1}), \ldots, B_\ell(x_{\ell,1}, \ldots, x_{\ell,m_\ell})$$

- $\alpha_1 \ldots \alpha_k$: all $x_{i,j}$ shuffled.
- Well-nested: in $\alpha_1 \ldots \alpha_k$, no crossings

  $\ldots x_{i,j} \ldots x_{i',m} \ldots x_{i,k} \ldots x_{i',n} \ldots$.
- Well-nested 2-component: tree-adjoining, also head.
  - Constituents: substrings u, pairs (u, v).
  - Main operation: wrap (u, v) around (x, y), get (ux, yv).

Well-nested multi-component grammars

- Constituents u_1-GAP-u_2-GAP- . . . -GAP-u_k.

$$A(\alpha_1, \ldots, \alpha_k) \to B_1(x_{1,1}, \ldots, x_{1,m_1}), \ldots, B_\ell(x_{\ell,1}, \ldots, x_{\ell,m_\ell})$$

- $\alpha_1 \ldots \alpha_k$: all $x_{i,j}$ shuffled.
- Well-nested: in $\alpha_1 \ldots \alpha_k$, no crossings

  $\ldots x_{i,j} \ldots x_{i',m} \ldots x_{i,k} \ldots x_{i',n} \ldots$.
- Well-nested 2-component: tree-adjoining, also head.
  - Constituents: substrings u, pairs (u, v).
  - Main operation: wrap (u, v) around (x, y), get (ux, yv).
- Grammar models combining these ideas, e.g.,

Well-nested multi-component grammars

- Constituents u_1-GAP-u_2-GAP- . . . -GAP-u_k.

$$A(\alpha_1, \ldots, \alpha_k) \to B_1(x_{1,1}, \ldots, x_{1,m_1}), \ldots, B_\ell(x_{\ell,1}, \ldots, x_{\ell,m_\ell})$$

- $\alpha_1 \ldots \alpha_k$: all $x_{i,j}$ shuffled.
- Well-nested: in $\alpha_1 \ldots \alpha_k$, no crossings

  $\ldots x_{i,j} \ldots x_{i',m} \ldots x_{i,k} \ldots x_{i',n} \ldots$.
- Well-nested 2-component: tree-adjoining, also head.
  - Constituents: substrings u, pairs (u, v).
  - Main operation: wrap (u, v) around (x, y), get (ux, yv).
- Grammar models combining these ideas, e.g.,
  - unambiguous tree-adjoining,

Well-nested multi-component grammars

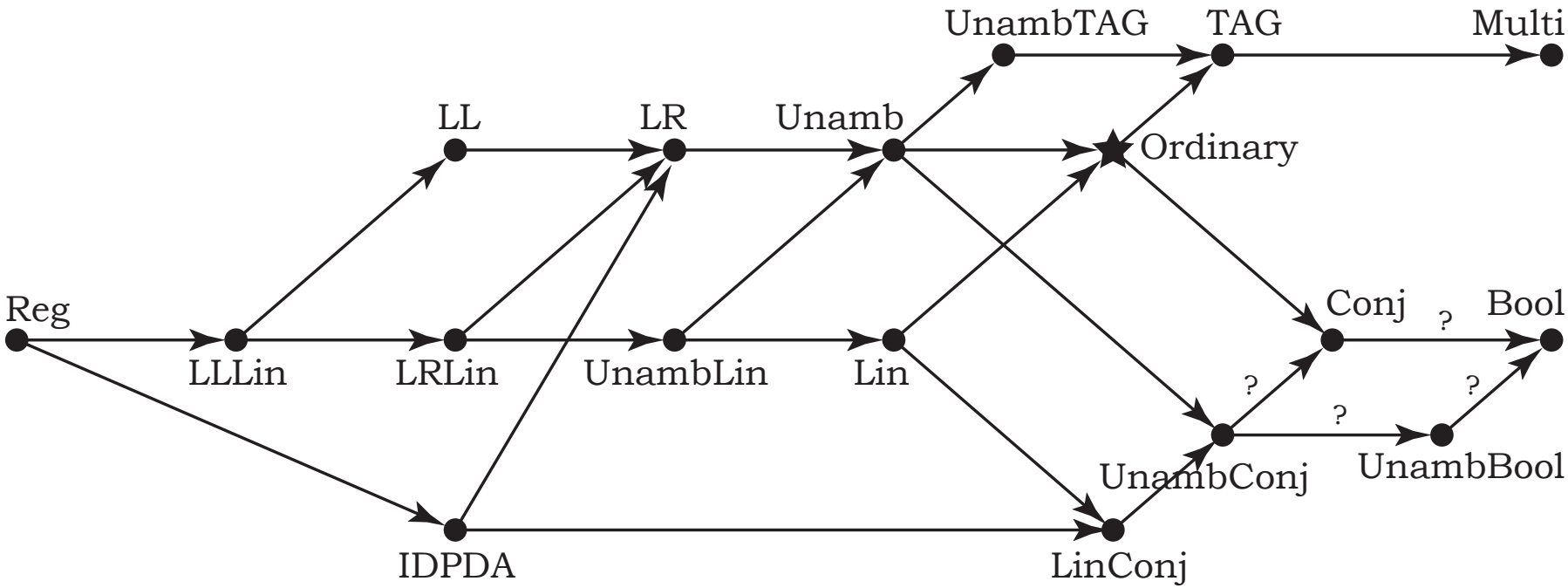
- Constituents u_1-GAP-u_2-GAP- . . . -GAP-u_k.

$$A(\alpha_1, \ldots, \alpha_k) \to B_1(x_{1,1}, \ldots, x_{1,m_1}), \ldots, B_\ell(x_{\ell,1}, \ldots, x_{\ell,m_\ell})$$

- $\alpha_1 \ldots \alpha_k$: all $x_{i,j}$ shuffled.
- Well-nested: in $\alpha_1 \ldots \alpha_k$, no crossings

  $\ldots x_{i,j} \ldots x_{i',m} \ldots x_{i,k} \ldots x_{i',n} \ldots$.
- Well-nested 2-component: tree-adjoining, also head.
  - Constituents: substrings u, pairs (u, v).
  - Main operation: wrap (u, v) around (x, y), get (ux, yv).
- Grammar models combining these ideas, e.g.,
  - unambiguous tree-adjoining,
  - linear conjunctive, etc.

The hierarchy



Part III

Formal definitions

Definition by logical derivation

- $S \rightarrow \mathrm{NP\ VP}$

Definition by logical derivation

- $S \to \mathrm{NP}\ \mathrm{VP}$

"If u is NP and v is VP, then uv is a sentence".

Definition by logical derivation

- $S \to \mathrm{NP}\ \mathrm{VP}$

"If u is NP and v is VP, then uv is a sentence".

- Propositions $A(w)$, meaning "w has the property A".

Definition by logical derivation

- $S \to \mathrm{NP}\ \mathrm{VP}$

"If u is NP and v is VP, then uv is a sentence".

- Propositions $A(w)$, meaning "w has the property A".

Definition by logical derivation (folklore)

$$\frac{\mathrm{NP}(\texttt{Every man}) \quad \mathrm{VP}(\texttt{is mortal})}{S(\texttt{Every man is mortal})}\ (S \to \mathrm{NP}\ \mathrm{VP})$$

Definition by logical derivation

- $S \to \mathrm{NP}\ \mathrm{VP}$

"If u is NP and v is VP, then uv is a sentence".

- Propositions $A(w)$, meaning "w has the property A".

Definition by logical derivation (folklore)

$$\frac{\mathrm{NP}(\texttt{Every man}) \quad \mathrm{VP}(\texttt{is mortal})}{S(\texttt{Every man is mortal})}\ (S \to \mathrm{NP}\ \mathrm{VP})$$

- A proof tree is a parse tree.

Definition by logical derivation

- $S \to \mathrm{NP}\ \mathrm{VP}$

"If u is NP and v is VP, then uv is a sentence".

- Propositions $A(w)$, meaning "w has the property A".

Definition by logical derivation (folklore)

$$\frac{\mathrm{NP}(\texttt{Every man}) \quad \mathrm{VP}(\texttt{is mortal})}{S(\texttt{Every man is mortal})}\ (S \to \mathrm{NP}\ \mathrm{VP})$$

- A proof tree is a parse tree.

✓ Directly formalizes the intuition.

Logical derivation for generalizations

For conjunctive grammars

$$\frac{A(a^n) \quad B(b^n c^n) \quad D(a^n b^n) \quad C(c^n)}{S(a^n b^n c^n)} \; (S \to AB \,\&\, DC)$$

Logical derivation for generalizations

For conjunctive grammars

$$\frac{A(a^n) \quad B(b^n c^n) \quad D(a^n b^n) \quad C(c^n)}{S(a^n b^n c^n)} \; (S \to AB \,\&\, DC)$$

For multi-component grammars

$$\frac{A(ab, cd)}{A(aabb, ccdd)} \; (A(axb, cyd) \to A(x, y))$$

Logical derivation for generalizations

For conjunctive grammars

$$\frac{A(a^n) \quad B(b^n c^n) \quad D(a^n b^n) \quad C(c^n)}{S(a^n b^n c^n)} \; (S \to AB \,\&\, DC)$$

For multi-component grammars

$$\frac{A(ab, cd)}{A(aabb, ccdd)} \; (A(axb, cyd) \to A(x, y))$$

Works for all kinds of grammars without negation.

Definition by language equations

- $S \to \mathrm{NP}\ \mathrm{VP} \mid \ldots$

Definition by language equations

- $S \to \mathrm{NP}\ \mathrm{VP} \mid \ldots$

"w is a sentence if and only if
$w = uv$, with u an NP and v a VP, or…".

Definition by language equations

- $S \to \mathrm{NP}\ \mathrm{VP} \mid \ldots$

"w is a sentence if and only if
$w = uv$, with u an NP and v a VP, or…".

- $S, \mathrm{NP}, \mathrm{VP}$: unknown languages.

Definition by language equations

- $S \to \mathrm{NP}\ \mathrm{VP} \mid \ldots$

"w is a sentence if and only if
$w = uv$, with u an NP and v a VP, or…".

- $S, \mathrm{NP}, \mathrm{VP}$: unknown languages.
- Equation $S = (\mathrm{NP} \cdot \mathrm{VP}) \cup \ldots$

Definition by language equations

- $S \to \mathrm{NP}\ \mathrm{VP} \mid \ldots$

"w is a sentence if and only if
$w = uv$, with u an NP and v a VP, or…".

- $S, \mathrm{NP}, \mathrm{VP}$: unknown languages.
- Equation $S = (\mathrm{NP} \cdot \mathrm{VP}) \cup \ldots$

Definition (Ginsburg, Rice, 1962)

Grammar $G = (\Sigma, N, R, S)$ as a least solution of:

$$A = \bigcup_{A \to X_1 \ldots X_\ell \in R} X_1 \cdot \ldots \cdot X_\ell \qquad \text{(for each } A \in N\text{)}$$

Language equations for extensions

Conjunctive: conjunction as intersection.

Language equations for extensions

Conjunctive: conjunction as intersection.

- $S \to AB \,\&\, DC$ as $S = (A \cdot B) \cap (D \cdot C)$.

Language equations for extensions

Conjunctive: conjunction as intersection.

- $S \to AB \,\&\, DC$ as $S = (A \cdot B) \cap (D \cdot C)$.

Boolean: negation as complementation.

Language equations for extensions

Conjunctive: conjunction as intersection.

- $S \to AB \,\&\, DC$ as $S = (A \cdot B) \cap (D \cdot C)$.

Boolean: negation as complementation.

- $S \to AB \,\&\, \neg DC$ as $S = (A \cdot B) \cap \overline{D \cdot C}$.

Language equations for extensions

Conjunctive: conjunction as intersection.

- $S \to AB \,\&\, DC$ as $S = (A \cdot B) \cap (D \cdot C)$.

Boolean: negation as complementation.

- $S \to AB \,\&\, \neg DC$ as $S = (A \cdot B) \cap \overline{D \cdot C}$.
- Can express contradiction: $S \to \neg S$.

Language equations for extensions

Conjunctive: conjunction as intersection.

- $S \to AB \,\&\, DC$ as $S = (A \cdot B) \cap (D \cdot C)$.

Boolean: negation as complementation.

- $S \to AB \,\&\, \neg DC$ as $S = (A \cdot B) \cap \overline{D \cdot C}$.
- Can express contradiction: $S \to \neg S$.
  - Same problem with negation as in logic programming.

Language equations for extensions

Conjunctive: conjunction as intersection.

- $S \to AB \,\&\, DC$ as $S = (A \cdot B) \cap (D \cdot C)$.

Boolean: negation as complementation.

- $S \to AB \,\&\, \neg DC$ as $S = (A \cdot B) \cap \overline{D \cdot C}$.
- Can express contradiction: $S \to \neg S$.
  - Same problem with negation as in logic programming.
  - Resolved, e.g., by using 3-valued logic.

Language equations for extensions

Conjunctive: conjunction as intersection.

- $S \to AB \,\&\, DC$ as $S = (A \cdot B) \cap (D \cdot C)$.

Boolean: negation as complementation.

- $S \to AB \,\&\, \neg DC$ as $S = (A \cdot B) \cap \overline{D \cdot C}$.
- Can express contradiction: $S \to \neg S$.
  - Same problem with negation as in logic programming.
  - Resolved, e.g., by using 3-valued logic.

Multi-component: unknown sets of k-tuples.

Language equations for extensions

Conjunctive: conjunction as intersection.

- $S \to AB \,\&\, DC$ as $S = (A \cdot B) \cap (D \cdot C)$.

Boolean: negation as complementation.

- $S \to AB \,\&\, \neg DC$ as $S = (A \cdot B) \cap \overline{D \cdot C}$.
- Can express contradiction: $S \to \neg S$.
  - Same problem with negation as in logic programming.
  - Resolved, e.g., by using 3-valued logic.

Multi-component: unknown sets of k-tuples.

Applies to all grammar families.

Oh yes, that definition by rewriting...

By string rewriting (Chomsky, 1956)

$S \Longrightarrow \mathrm{NP\ VP} \Longrightarrow \ldots \Longrightarrow \mathrm{NP}\ \texttt{is mortal} \Longrightarrow \ldots \Longrightarrow$
`Every man is mortal`

Oh yes, that definition by rewriting. . .

By string rewriting (Chomsky, 1956)

$S \Longrightarrow \mathrm{NP\ VP} \Longrightarrow \ldots \Longrightarrow \mathrm{NP}\ \texttt{is mortal} \Longrightarrow \ldots \Longrightarrow$
`Every man is mortal`

- Equivalent to logical derivation, therefore correct.

Oh yes, that definition by rewriting...

By string rewriting (Chomsky, 1956)

$S \Longrightarrow \mathrm{NP\ VP} \Longrightarrow \ldots \Longrightarrow \mathrm{NP}\ \texttt{is mortal} \Longrightarrow \ldots \Longrightarrow$
`Every man is mortal`

- Equivalent to logical derivation, therefore correct.
- May give a totally wrong impression of grammars!

Oh yes, that definition by rewriting...

By string rewriting (Chomsky, 1956)

$S \Longrightarrow \mathrm{NP\ VP} \Longrightarrow \ldots \Longrightarrow \mathrm{NP}\ \texttt{is mortal} \Longrightarrow \ldots \Longrightarrow$
`Every man is mortal`

- Equivalent to logical derivation, therefore correct.
- May give a totally wrong impression of grammars!
- Attempts to generalize this definition lead to nonsense.

Oh yes, that definition by rewriting...

By string rewriting (Chomsky, 1956)

$S \Longrightarrow \mathrm{NP\ VP} \Longrightarrow \ldots \Longrightarrow \mathrm{NP}\ \texttt{is mortal} \Longrightarrow \ldots \Longrightarrow$
`Every man is mortal`

- Equivalent to logical derivation, therefore correct.
- May give a totally wrong impression of grammars!
- Attempts to generalize this definition lead to nonsense.

Works for ordinary grammars, gives wrong ideas.

Part IV

Recurring ideas

Expressibility of operations

- A family of grammars.

Expressibility of operations

- A family of grammars.

Closure under operation f

If $L_1, \ldots L_n$ are in the family, then so is $f(L_1, \ldots L_n)$.

Expressibility of operations

- A family of grammars.

Closure under operation f

If $L_1, \ldots L_n$ are in the family, then so is $f(L_1, \ldots L_n)$.

- Practical value: a way of constructing grammars.

Expressibility of operations

- A family of grammars.

Closure under operation f

If $L_1, \ldots L_n$ are in the family, then so is $f(L_1, \ldots L_n)$.

- Practical value: a way of constructing grammars.
- Explicit: $\{\cdot, \cup\}$ in ordinary grammars,
  $\{\cdot, \cup, \cap\}$ in conjunctive, etc.

Expressibility of operations

- A family of grammars.

Closure under operation f

If $L_1, \ldots L_n$ are in the family, then so is $f(L_1, \ldots L_n)$.

- Practical value: a way of constructing grammars.
- Explicit: $\{\cdot, \cup\}$ in ordinary grammars, $\{\cdot, \cup, \cap\}$ in conjunctive, etc.
- Easily expressed: Kleene star, reversal.

Expressibility of operations

- A family of grammars.

Closure under operation f

If $L_1, \ldots L_n$ are in the family, then so is $f(L_1, \ldots L_n)$.

- Practical value: a way of constructing grammars.
- Explicit: $\{\cdot, \cup\}$ in ordinary grammars, $\{\cdot, \cup, \cap\}$ in conjunctive, etc.
- Easily expressed: Kleene star, reversal.
- Any other operations expressible?

Expressibility of operations, II

Theorem (Bar-Hillel et al., 1962)

Ordinary grammars closed under $\cap Reg$.

Expressibility of operations, II

Theorem (Bar-Hillel et al., 1962)

Ordinary grammars closed under $\cap Reg$.

- ✓ Embed DFA's computation into a parse tree.

Expressibility of operations, II

Theorem (Bar-Hillel et al., 1962)

Ordinary grammars closed under $\cap Reg$.

- ✓ Embed DFA's computation into a parse tree.

Ordinary grammars closed under finite transductions.

Expressibility of operations, II

Theorem (Bar-Hillel et al., 1962)

Ordinary grammars closed under $\cap Reg$.

- ✓ Embed DFA's computation into a parse tree.

Ordinary grammars closed under finite transductions.

- Applies to linear; multi-component.

Expressibility of operations, II

Theorem (Bar-Hillel et al., 1962)

Ordinary grammars closed under $\cap$*Reg*.

✓ Embed DFA's computation into a parse tree.

Ordinary grammars closed under finite transductions.

- Applies to linear; multi-component.
- Non-closure for unambiguous; conjunctive.

Expressibility of operations, II

Theorem (Bar-Hillel et al., 1962)

Ordinary grammars closed under $\cap Reg$.

✓ Embed DFA's computation into a parse tree.

Ordinary grammars closed under finite transductions.

- Applies to linear; multi-component.
- Non-closure for unambiguous; conjunctive.
  - Closed under inverse transductions.

Expressibility of operations, II

Theorem (Bar-Hillel et al., 1962)

Ordinary grammars closed under $\cap$*Reg*.

✓ Embed DFA's computation into a parse tree.

Ordinary grammars closed under finite transductions.

- Applies to linear; multi-component.
- Non-closure for unambiguous; conjunctive.
  - Closed under inverse transductions.

Idea and the limits of its applicability.

Negative results using iteration

Ordinary grammars: pumping lemma (Bar-Hillel et al., 1962).

Negative results using iteration

Ordinary grammars: pumping lemma (Bar-Hillel et al., 1962).

- ✓ Large parse trees: insert repetitive structure.

Negative results using iteration

Ordinary grammars: pumping lemma (Bar-Hillel et al., 1962).

- ✓ Large parse trees: insert repetitive structure.
- Extensions: Ogden (1968); Bader–Moura (1982).

Negative results using iteration

Ordinary grammars: pumping lemma (Bar-Hillel et al., 1962).

- ✓ Large parse trees: insert repetitive structure.
- Extensions: Ogden (1968); Bader–Moura (1982).
- Multi-component: restricted variants.

Negative results using iteration

Ordinary grammars: pumping lemma (Bar-Hillel et al., 1962).

- ✓ Large parse trees: insert repetitive structure.
- Extensions: Ogden (1968); Bader–Moura (1982).
- Multi-component: restricted variants.

Ordinary grammars: Parikh's theorem.

Negative results using iteration

Ordinary grammars: pumping lemma (Bar-Hillel et al., 1962).

- ✓ Large parse trees: insert repetitive structure.
- Extensions: Ogden (1968); Bader–Moura (1982).
- Multi-component: restricted variants.

Ordinary grammars: Parikh's theorem.

- Applies to multi-component.

Negative results using iteration

Ordinary grammars: pumping lemma (Bar-Hillel et al., 1962).

- ✓ Large parse trees: insert repetitive structure.
- Extensions: Ogden (1968); Bader–Moura (1982).
- Multi-component: restricted variants.

Ordinary grammars: Parikh's theorem.

- Applies to multi-component.

Methods based on iteration for grammars with disjunction. Nothing known for conjunctive!

Normal forms

Chomsky normal form: $A \to BC$, $A \to a$.

Normal forms

Chomsky normal form: $A \to BC$, $A \to a$.

- Applies to unambiguous, LL, LR.

Normal forms

Chomsky normal form: $A \to BC$, $A \to a$.

- Applies to unambiguous, LL, LR.
- Applies to conjunctive: $A \to B_1 C_1 \,\&\, \ldots \,\&\, B_n C_n$.

Normal forms

Chomsky normal form: $A \to BC$, $A \to a$.

- Applies to unambiguous, LL, LR.
- Applies to conjunctive: $A \to B_1 C_1 \,\&\, \ldots \,\&\, B_n C_n$.
- Not to multi-component grammars.

Normal forms

Chomsky normal form: $A \to BC$, $A \to a$.

- Applies to unambiguous, LL, LR.
- Applies to conjunctive: $A \to B_1 C_1 \,\&\, \ldots \,\&\, B_n C_n$.
- Not to multi-component grammars.

Greibach normal form: $A \to a\alpha$.
Rosenkrantz normal form: $A \to a\alpha b$.

Normal forms

Chomsky normal form: $A \to BC$, $A \to a$.

- Applies to unambiguous, LL, LR.
- Applies to conjunctive: $A \to B_1 C_1 \,\&\, \ldots \,\&\, B_n C_n$.
- Not to multi-component grammars.

Greibach normal form: $A \to a\alpha$.
Rosenkrantz normal form: $A \to a\alpha b$.

- Applies to unambiguous, linear, LL, LR.

Normal forms

Chomsky normal form: $A \to BC$, $A \to a$.

- Applies to unambiguous, LL, LR.
- Applies to conjunctive: $A \to B_1 C_1 \,\&\, \ldots \,\&\, B_n C_n$.
- Not to multi-component grammars.

Greibach normal form: $A \to a\alpha$.
Rosenkrantz normal form: $A \to a\alpha b$.

- Applies to unambiguous, linear, LL, LR.
- Conjunctive, Boolean: not known.

Parsing time

Ordinary grammars: Cocke–Kasami–Younger

G in CNF, given $w = a_1 \dots a_n$, construct

$$T_{i,j} = \{A \mid a_{i+1} \dots a_j \in L_G(A)\}$$

Parsing time

Ordinary grammars: Cocke–Kasami–Younger

G in CNF, given $w = a_1 \dots a_n$, construct

$$T_{i,j} = \{A \mid a_{i+1} \dots a_j \in L_G(A)\}$$

- Time $\Theta(n^3)$, space $\Theta(n^2)$.

Parsing time

Ordinary grammars: Cocke–Kasami–Younger

G in CNF, given $w = a_1 \dots a_n$, construct

$$T_{i,j} = \{A \mid a_{i+1} \dots a_j \in L_G(A)\}$$

- Time $\Theta(n^3)$, space $\Theta(n^2)$.
- By matrix multiplication (Valiant, 1975): time $\Theta(n^\omega)$.

Parsing time

Ordinary grammars: Cocke–Kasami–Younger

G in CNF, given $w = a_1 \dots a_n$, construct

$$T_{i,j} = \{A \mid a_{i+1} \dots a_j \in L_G(A)\}$$

- Time $\Theta(n^3)$, space $\Theta(n^2)$.
- By matrix multiplication (Valiant, 1975): time $\Theta(n^\omega)$.
- Conjunctive, Boolean: same algorithms.

Parsing time

Ordinary grammars: Cocke–Kasami–Younger

G in CNF, given $w = a_1 \dots a_n$, construct

$$T_{i,j} = \{A \mid a_{i+1} \dots a_j \in L_G(A)\}$$

- Time $\Theta(n^3)$, space $\Theta(n^2)$.
- By matrix multiplication (Valiant, 1975): time $\Theta(n^\omega)$.
- Conjunctive, Boolean: same algorithms.
- Unambiguous: time $O(n^2)$.

Parsing time

Ordinary grammars: Cocke–Kasami–Younger

G in CNF, given $w = a_1 \dots a_n$, construct

$$T_{i,j} = \{A \mid a_{i+1} \dots a_j \in L_G(A)\}$$

- Time $\Theta(n^3)$, space $\Theta(n^2)$.
- By matrix multiplication (Valiant, 1975): time $\Theta(n^\omega)$.
- Conjunctive, Boolean: same algorithms.
- Unambiguous: time $O(n^2)$.
- Tree-adjoining: time $O(n^6)$, can do in time $O(n^{2\omega})$.

Parsing time

Ordinary grammars: Cocke–Kasami–Younger

G in CNF, given $w = a_1 \dots a_n$, construct

$$T_{i,j} = \{A \mid a_{i+1} \dots a_j \in L_G(A)\}$$

- Time $\Theta(n^3)$, space $\Theta(n^2)$.
- By matrix multiplication (Valiant, 1975): time $\Theta(n^\omega)$.
- Conjunctive, Boolean: same algorithms.
- Unambiguous: time $O(n^2)$.
- Tree-adjoining: time $O(n^6)$, can do in time $O(n^{2\omega})$.
- k-component: time $O(n^{3k})$, can do in time $O(n^{\omega k})$.

Parsing time

Ordinary grammars: Cocke–Kasami–Younger

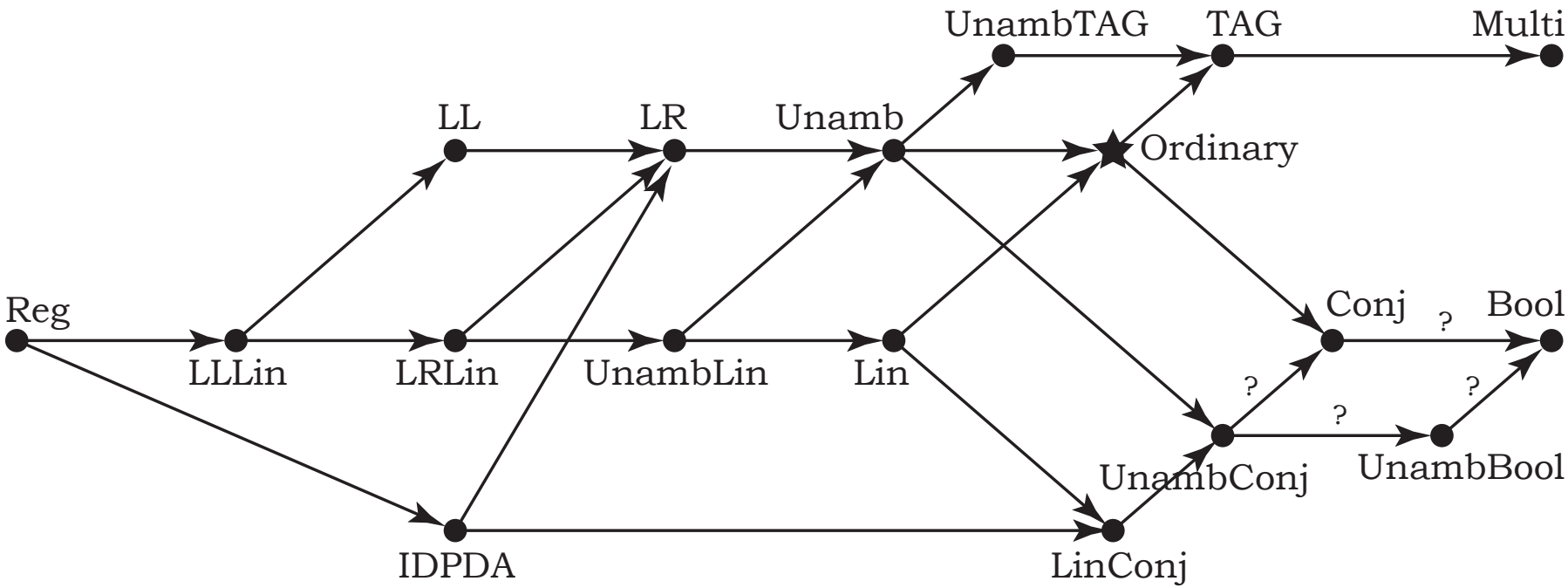
G in CNF, given $w = a_1 \dots a_n$, construct

$$T_{i,j} = \{A \mid a_{i+1} \dots a_j \in L_G(A)\}$$

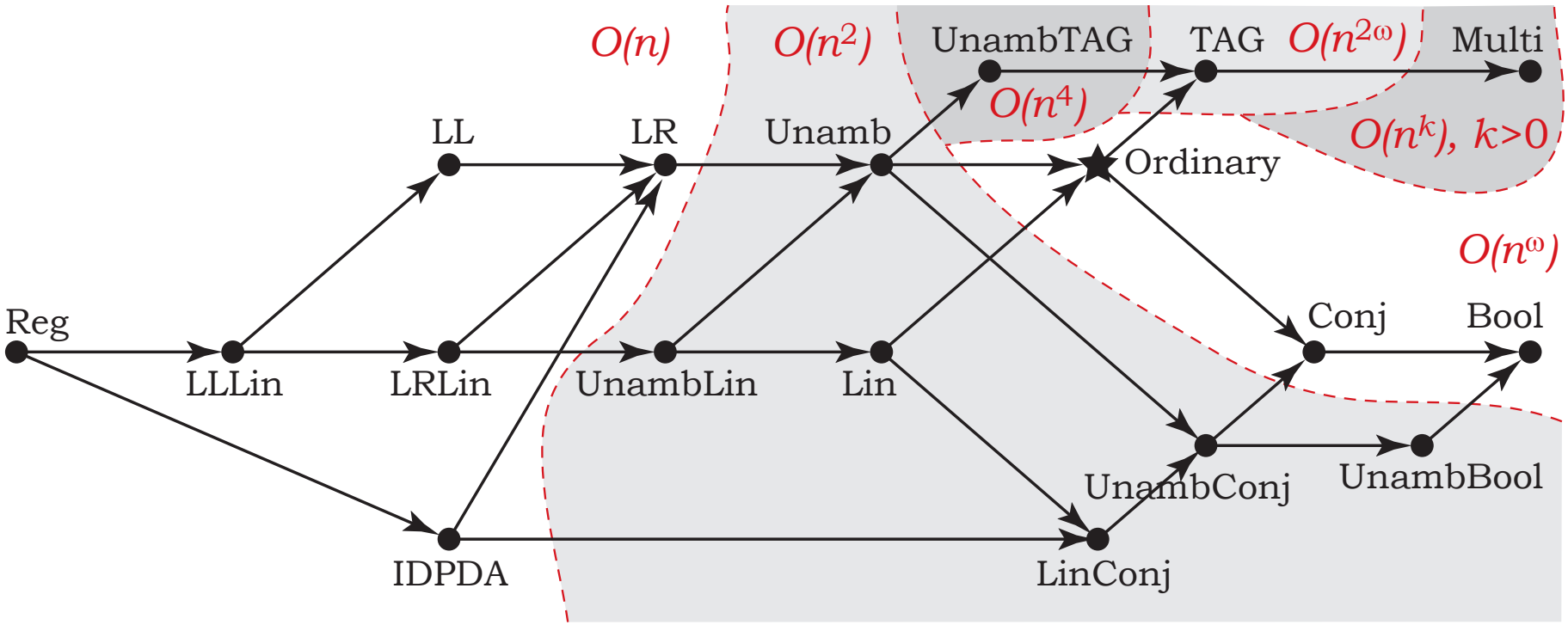
- Time $\Theta(n^3)$, space $\Theta(n^2)$.
- By matrix multiplication (Valiant, 1975): time $\Theta(n^\omega)$.
- Conjunctive, Boolean: same algorithms.
- Unambiguous: time $O(n^2)$.
- Tree-adjoining: time $O(n^6)$, can do in time $O(n^{2\omega})$.
- k-component: time $O(n^{3k})$, can do in time $O(n^{\omega k})$.

Polynomial-time parsing for each family.

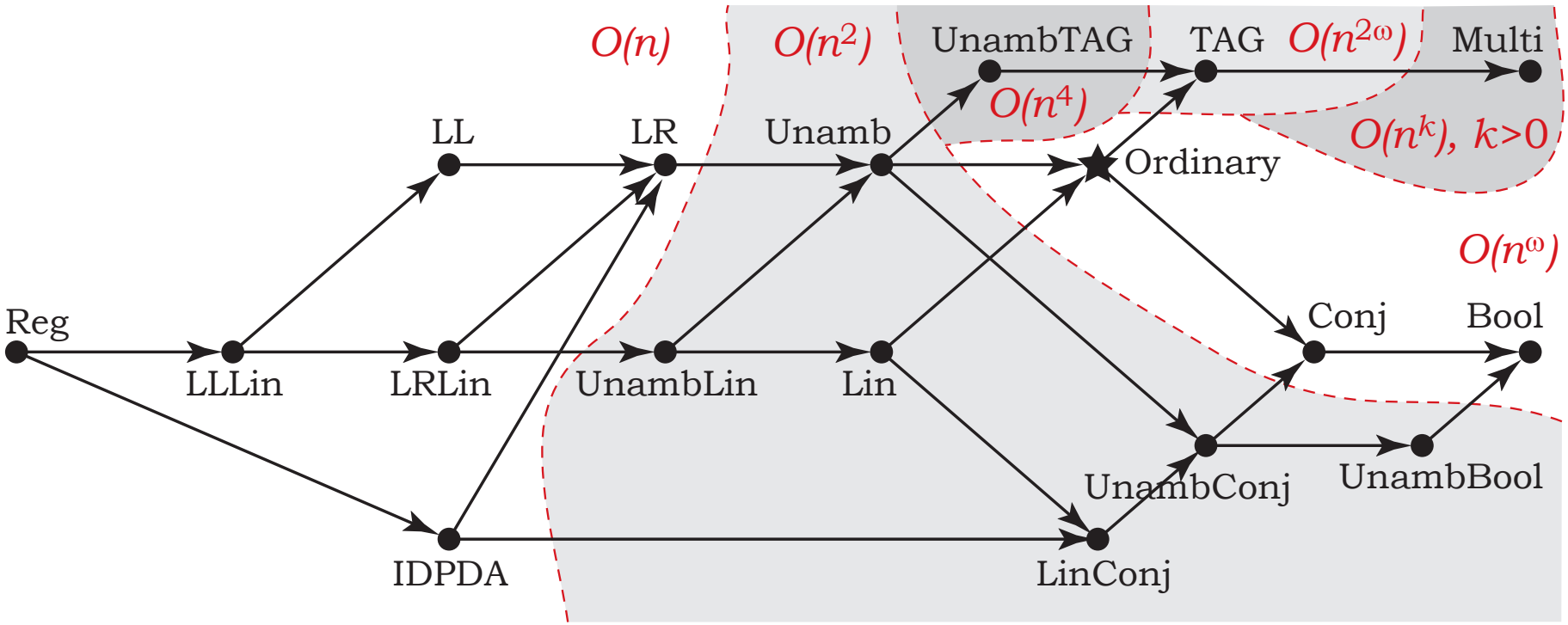
The hierarchy: parsing algorithms



The hierarchy: parsing algorithms



The hierarchy: parsing algorithms



- Why is there always a polynomial-time algorithm?..

Part V

General model: FO(LFP) logic

Logical contents of grammars

- In 1957: grammars as a special case of TM.

Logical contents of grammars

- In 1957: grammars as a special case of TM.
- Not useful at all!

Logical contents of grammars

- In 1957: grammars as a special case of TM.
- Not useful at all!

Correct general model: FO(LFP) logic (Rounds, 1988).

Logical contents of grammars

- In 1957: grammars as a special case of TM.
- Not useful at all!

Correct general model: FO(LFP) logic (Rounds, 1988).

- Variables ranging over positions in a string.

Logical contents of grammars

- In 1957: grammars as a special case of TM.
- Not useful at all!

Correct general model: FO(LFP) logic (Rounds, 1988).

- Variables ranging over positions in a string.
- Recursive definitions of predicates.

Logical contents of grammars

- In 1957: grammars as a special case of TM.
- Not useful at all!

Correct general model: FO(LFP) logic (Rounds, 1988).

- Variables ranging over positions in a string.
- Recursive definitions of predicates.

Example ($S \to SS \mid aSb \mid \varepsilon$)

$S(x,y) =$

$\bigvee$ $\bigvee$

Logical contents of grammars

- In 1957: grammars as a special case of TM.
- Not useful at all!

Correct general model: FO(LFP) logic (Rounds, 1988).

- Variables ranging over positions in a string.
- Recursive definitions of predicates.

Example ($S \to SS \mid aSb \mid \varepsilon$)

$$S(x,y) = \Big((\exists z)(S(x,z) \wedge S(z,y))\Big)$$

$\vee$ $\vee$

Logical contents of grammars

- In 1957: grammars as a special case of TM.
- Not useful at all!

Correct general model: FO(LFP) logic (Rounds, 1988).

- Variables ranging over positions in a string.
- Recursive definitions of predicates.

Example ($S \to SS \mid aSb \mid \varepsilon$)

$$S(x,y) = \big((\exists z)(S(x,z) \wedge S(z,y))\big) \vee \big(a(x+1) \wedge S(x+1, y-1) \wedge b(y)\big) \vee$$

Logical contents of grammars

- In 1957: grammars as a special case of TM.
- Not useful at all!

Correct general model: FO(LFP) logic (Rounds, 1988).

- Variables ranging over positions in a string.
- Recursive definitions of predicates.

Example ($S \to SS \mid aSb \mid \varepsilon$)

$$S(x,y) = \big((\exists z)(S(x,z) \wedge S(z,y))\big) \\ \vee \big(a(x+1) \wedge S(x+1, y-1) \wedge b(y)\big) \vee x = y$$

Logical contents of grammars

- In 1957: grammars as a special case of TM.
- Not useful at all!

Correct general model: FO(LFP) logic (Rounds, 1988).

- Variables ranging over positions in a string.
- Recursive definitions of predicates.

Example ($S \to SS \mid aSb \mid \varepsilon$)

$$S(x,y) = \big((\exists z)(S(x,z) \wedge S(z,y))\big) \vee \big(a(x+1) \wedge S(x+1, y-1) \wedge b(y)\big) \vee x = y$$

$$\sigma = S(\underline{\text{begin}}, \underline{\text{end}})$$

Grammar families as fragments of FO(LFP)

- Ordinary: free $\vee$, restricted $\{\exists, \wedge\}$.

Grammar families as fragments of FO(LFP)

- Ordinary: free $\vee$, restricted $\{\exists, \wedge\}$.
  - $\{\exists, \wedge\}$: in one particular way to express concatenation.

Grammar families as fragments of FO(LFP)

- Ordinary: free $\vee$, restricted $\{\exists, \wedge\}$.
  - $\{\exists, \wedge\}$: in one particular way to express concatenation.
- Linear: no quantification.

Grammar families as fragments of FO(LFP)

- Ordinary: free $\vee$, restricted $\{\exists, \wedge\}$.
  - $\{\exists, \wedge\}$: in one particular way to express concatenation.
- Linear: no quantification.
- Unambiguous: one true value for $\{\vee, \exists\}$.

Grammar families as fragments of FO(LFP)

- Ordinary: free $\vee$, restricted $\{\exists, \wedge\}$.
  - $\{\exists, \wedge\}$: in one particular way to express concatenation.
- Linear: no quantification.
- Unambiguous: one true value for $\{\vee, \exists\}$.
- LL: $A(i, j)$ uses i to determine j.

Grammar families as fragments of FO(LFP)

- Ordinary: free $\vee$, restricted $\{\exists, \wedge\}$.
  - $\{\exists, \wedge\}$: in one particular way to express concatenation.
- Linear: no quantification.
- Unambiguous: one true value for $\{\vee, \exists\}$.
- LL: $A(i, j)$ uses i to determine j.
  - $j = A(i)$, grammar is a program.

Grammar families as fragments of FO(LFP)

- Ordinary: free $\vee$, restricted $\{\exists, \wedge\}$.
  - $\{\exists, \wedge\}$: in one particular way to express concatenation.
- Linear: no quantification.
- Unambiguous: one true value for $\{\vee, \exists\}$.
- LL: $A(i, j)$ uses i to determine j.
  - $j = A(i)$, grammar is a program.
- Conjunctive: free use of $\wedge$.

Grammar families as fragments of FO(LFP)

- Ordinary: free $\vee$, restricted $\{\exists, \wedge\}$.
  - $\{\exists, \wedge\}$: in one particular way to express concatenation.
- Linear: no quantification.
- Unambiguous: one true value for $\{\vee, \exists\}$.
- LL: $A(i, j)$ uses i to determine j.
  - $j = A(i)$, grammar is a program.
- Conjunctive: free use of $\wedge$.
- Boolean: eliminate negation through duality.

Grammar families as fragments of FO(LFP)

- Ordinary: free $\vee$, restricted $\{\exists, \wedge\}$.
  - $\{\exists, \wedge\}$: in one particular way to express concatenation.
- Linear: no quantification.
- Unambiguous: one true value for $\{\vee, \exists\}$.
- LL: $A(i, j)$ uses i to determine j.
  - $j = A(i)$, grammar is a program.
- Conjunctive: free use of $\wedge$.
- Boolean: eliminate negation through duality.
- k-component: predicates $A(x_1, y_1, \ldots x_k, y_k)$

The FO(LFP) logic

Definition (Chandra/Harel, 1980)

FO(LFP) definition: $G = (\Sigma, N, \dim, \langle \varphi_A \rangle_{A \in N}, \sigma)$, where

The FO(LFP) logic

Definition (Chandra/Harel, 1980)

FO(LFP) definition: $G = (\Sigma, N, \dim, \langle \varphi_A \rangle_{A \in N}, \sigma)$, where

- $\varphi_A(x_1, \ldots, x_{\dim A})$ is a formula defining A,

The FO(LFP) logic

Definition (Chandra/Harel, 1980)

FO(LFP) definition: $G = (\Sigma, N, \dim, \langle \varphi_A \rangle_{A \in N}, \sigma)$, where

- $\varphi_A(x_1, \ldots, x_{\dim A})$ is a formula defining A,
- σ is a formula with no free variables.

The FO(LFP) logic

Definition (Chandra/Harel, 1980)

FO(LFP) definition: $G = (\Sigma, N, \dim, \langle \varphi_A \rangle_{A \in N}, \sigma)$, where

- $\varphi_A(x_1, \ldots, x_{\dim A})$ is a formula defining A,
- σ is a formula with no free variables.

- Semantics: same as for language equations.

The FO(LFP) logic

Definition (Chandra/Harel, 1980)

FO(LFP) definition: $G = (\Sigma, N, \dim, \langle \varphi_A \rangle_{A \in N}, \sigma)$, where

- $\varphi_A(x_1, \ldots, x_{\dim A})$ is a formula defining A,
- σ is a formula with no free variables.

- Semantics: same as for language equations.

Theorem (Immerman, 1986; Vardi, 1982)

L defined in FO(LFP)

$$\Updownarrow$$

The FO(LFP) logic

Definition (Chandra/Harel, 1980)

FO(LFP) definition: $G = (\Sigma, N, \dim, \langle \varphi_A \rangle_{A \in N}, \sigma)$, where

- $\varphi_A(x_1, \ldots, x_{\dim A})$ is a formula defining A,
- σ is a formula with no free variables.

- Semantics: same as for language equations.

Theorem (Immerman, 1986; Vardi, 1982)

L defined in FO(LFP)

$$\Updownarrow$$

L recognized in polynomial time.

The FO(LFP) logic

Definition (Chandra/Harel, 1980)

FO(LFP) definition: $G = (\Sigma, N, \dim, \langle \varphi_A \rangle_{A \in N}, \sigma)$, where

- $\varphi_A(x_1, \ldots, x_{\dim A})$ is a formula defining A,
- σ is a formula with no free variables.

- Semantics: same as for language equations.

Theorem (Immerman, 1986; Vardi, 1982)

L defined in FO(LFP)

$$\Updownarrow$$

L recognized in polynomial time.

- Membership in P: almost Cocke–Kasami–Younger!

Part VI

Common results

Equivalent representations

Ordinary by pushdown automata (NPDA).

Equivalent representations

Ordinary by pushdown automata (NPDA).

- (special cases) LR: DPDA; linear: one-turn.

Equivalent representations

Ordinary by pushdown automata (NPDA).

- (special cases) LR: DPDA; linear: one-turn.
- Conjunctive: PDA with tree-like stack (Aizikowitz, Kaminski).

Equivalent representations

Ordinary by pushdown automata (NPDA).

- (special cases) LR: DPDA; linear: one-turn.
- Conjunctive: PDA with tree-like stack (Aizikowitz, Kaminski).
- Bracketed: input-driven automata.

Equivalent representations

Ordinary by pushdown automata (NPDA).

- (special cases) LR: DPDA; linear: one-turn.
- Conjunctive: PDA with tree-like stack (Aizikowitz, Kaminski).
- Bracketed: input-driven automata.
- Linear conjunctive: 1W RT cellular automata.

Equivalent representations

Ordinary by pushdown automata (NPDA).

- (special cases) LR: DPDA; linear: one-turn.
- Conjunctive: PDA with tree-like stack (Aizikowitz, Kaminski).
- Bracketed: input-driven automata.
- Linear conjunctive: 1W RT cellular automata.

Ordinary by categorial grammars (Bar-Hillel et al.)

Equivalent representations

Ordinary by pushdown automata (NPDA).

- (special cases) LR: DPDA; linear: one-turn.
- Conjunctive: PDA with tree-like stack (Aizikowitz, Kaminski).
- Bracketed: input-driven automata.
- Linear conjunctive: 1W RT cellular automata.

Ordinary by categorial grammars (Bar-Hillel et al.)

- $$\frac{n(\texttt{John}) \quad n\backslash s(\texttt{works})}{s(\texttt{John works})}$$

Equivalent representations

Ordinary by pushdown automata (NPDA).

- (special cases) LR: DPDA; linear: one-turn.
- Conjunctive: PDA with tree-like stack (Aizikowitz, Kaminski).
- Bracketed: input-driven automata.
- Linear conjunctive: 1W RT cellular automata.

Ordinary by categorial grammars (Bar-Hillel et al.)

- $$\frac{n(\texttt{John}) \quad n\backslash s(\texttt{works})}{s(\texttt{John works})}$$
- Extends to conjunctive and to multi-component.

Homomorphic characterizations

Theorem (Chomsky, Schützenberger, 1963)

Ordinary as $h(D_k \cap R)$, for a homomorphism h, Dyck language D_k, regular R.

Homomorphic characterizations

Theorem (Chomsky, Schützenberger, 1963)

Ordinary as $h(D_k \cap R)$, for a homomorphism h, Dyck language D_k, regular R.

- Extends to multi-component.

Homomorphic characterizations

Theorem (Chomsky, Schützenberger, 1963)

Ordinary as $h(D_k \cap R)$, for a homomorphism h, Dyck language D_k, regular R.

- Extends to multi-component.
- Textbook proofs use erasing h.

Homomorphic characterizations

Theorem (Chomsky, Schützenberger, 1963)

Ordinary as $h(D_k \cap R)$, for a homomorphism h, Dyck language D_k, regular R.

- Extends to multi-component.
- Textbook proofs use erasing h.
- May use non-erasing (Okhotin, 2012).

Homomorphic characterizations

Theorem (Chomsky, Schützenberger, 1963)

Ordinary as $h(D_k \cap R)$, for a homomorphism h, Dyck language D_k, regular R.

- Extends to multi-component.
- Textbook proofs use erasing h.
- May use non-erasing (Okhotin, 2012).
- k depends only on $|\Sigma|$ (Crespi-Reghizzi, San Pietro, 2016).

Homomorphic characterizations

Theorem (Chomsky, Schützenberger, 1963)

Ordinary as $h(D_k \cap R)$, for a homomorphism h, Dyck language D_k, regular R.

- Extends to multi-component.
- Textbook proofs use erasing h.
- May use non-erasing (Okhotin, 2012).
- k depends only on $|\Sigma|$ (Crespi-Reghizzi, San Pietro, 2016).

Theorem

$L \subseteq (\Sigma^2)^*$ as $h(D_k \cap R)$, with length-preserving h.

Hardest languages

Theorem (Greibach, 1973)

$\exists$ *"hardest" ordinary grammar* $G_0 = (\Sigma_0, N_0, R_0, S_0)$,
s.t. for every G *there is* h*:*
$w \in L(G)$ *iff* $h(w) \in L(G_0)$.

Hardest languages

Theorem (Greibach, 1973)

$\exists$ *"hardest" ordinary grammar* $G_0 = (\Sigma_0, N_0, R_0, S_0)$,
s.t. for every G *there is* h*:*
$w \in L(G)$ *iff* $h(w) \in L(G_0)$.

- Does not extend to LR (Greibach).

Hardest languages

Theorem (Greibach, 1973)

$\exists$ *"hardest" ordinary grammar* $G_0 = (\Sigma_0, N_0, R_0, S_0)$,
s.t. for every G *there is* h*:*
$w \in L(G)$ *iff* $h(w) \in L(G_0)$.

- Does not extend to LR (Greibach).
- Does not extend to linear (Boasson/Nivat).

Hardest languages

Theorem (Greibach, 1973)

$\exists$ *"hardest" ordinary grammar* $G_0 = (\Sigma_0, N_0, R_0, S_0)$,
s.t. for every G *there is* h*:*
$w \in L(G)$ *iff* $h(w) \in L(G_0)$.

- Does not extend to LR (Greibach).
- Does not extend to linear (Boasson/Nivat).
- Extends to conjunctive, Boolean.

Hardest languages

Theorem (Greibach, 1973)

$\exists$ *"hardest" ordinary grammar* $G_0 = (\Sigma_0, N_0, R_0, S_0)$,
s.t. for every G *there is* h*:*
$w \in L(G)$ *iff* $h(w) \in L(G_0)$.

- Does not extend to LR (Greibach).
- Does not extend to linear (Boasson/Nivat).
- Extends to conjunctive, Boolean.
- Does not extend to linear conjunctive.

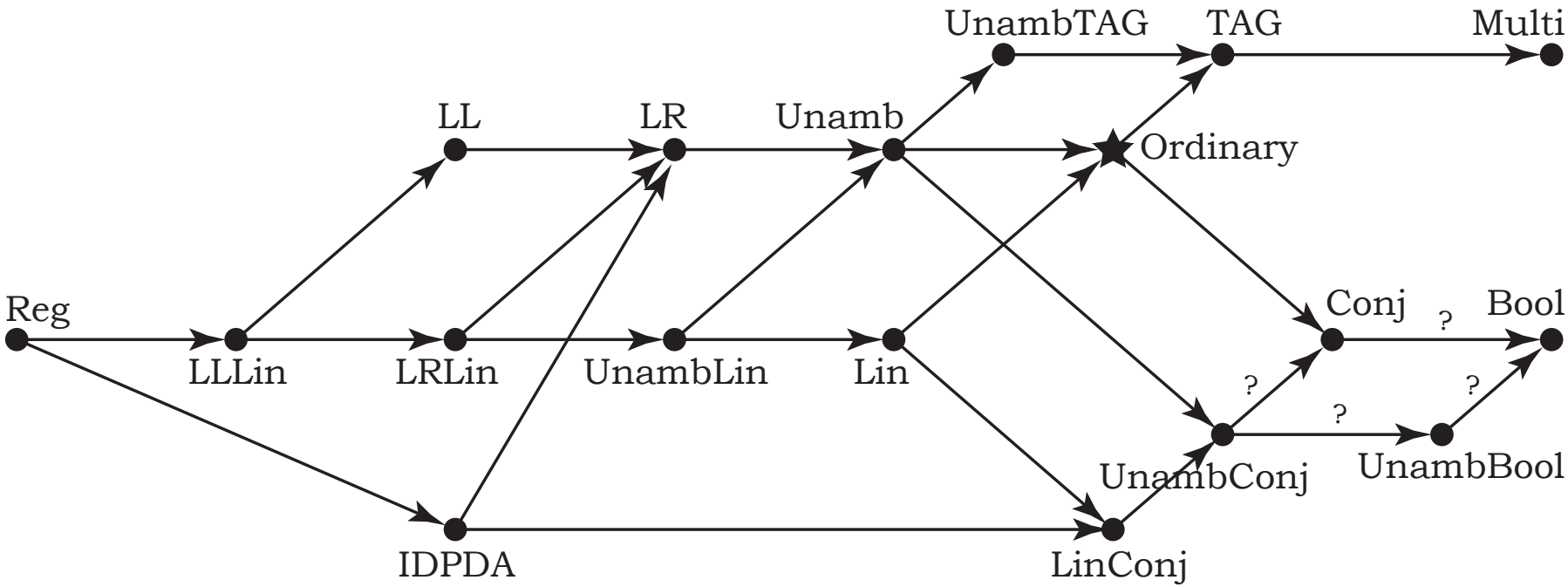
Hardest languages

Theorem (Greibach, 1973)

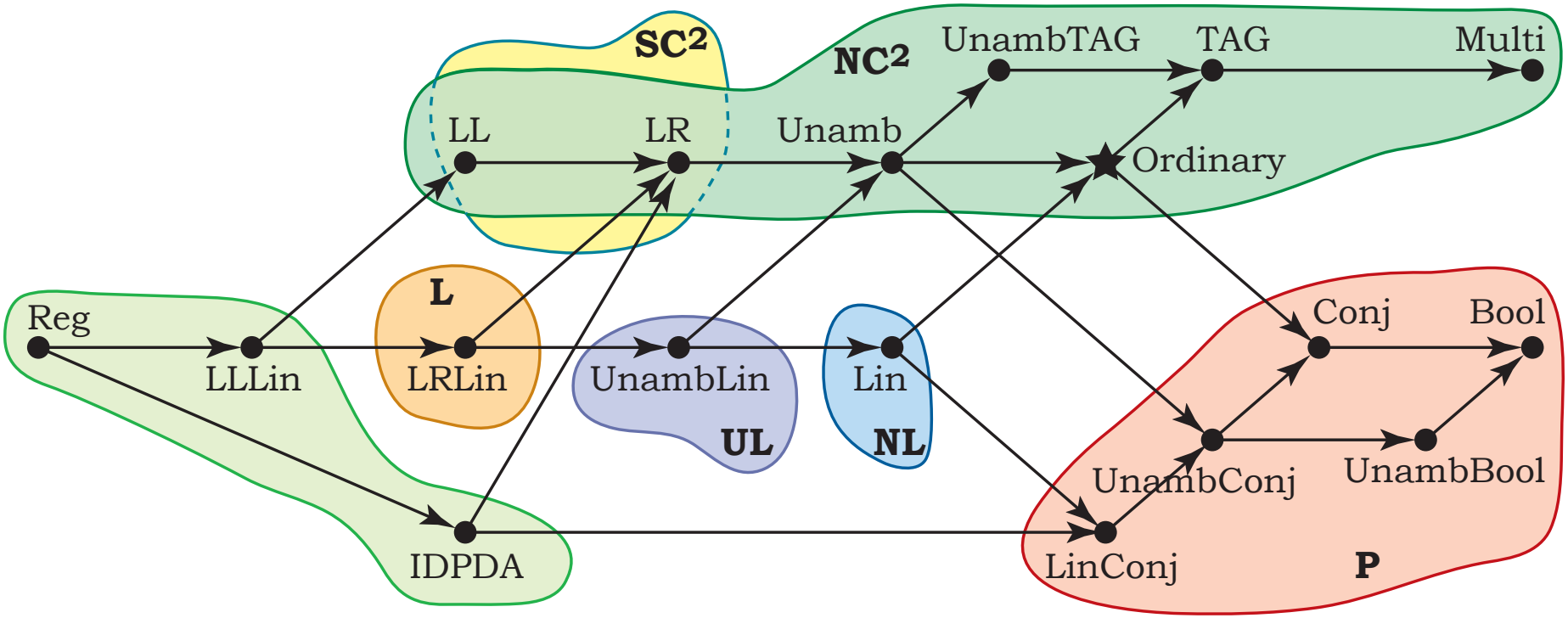
$\exists$ *"hardest" ordinary grammar* $G_0 = (\Sigma_0, N_0, R_0, S_0)$,
s.t. for every G *there is* h*:*
$w \in L(G)$ *iff* $h(w) \in L(G_0)$.

- Does not extend to LR (Greibach).
- Does not extend to linear (Boasson/Nivat).
- Extends to conjunctive, Boolean.
- Does not extend to linear conjunctive.
- To unambiguous?

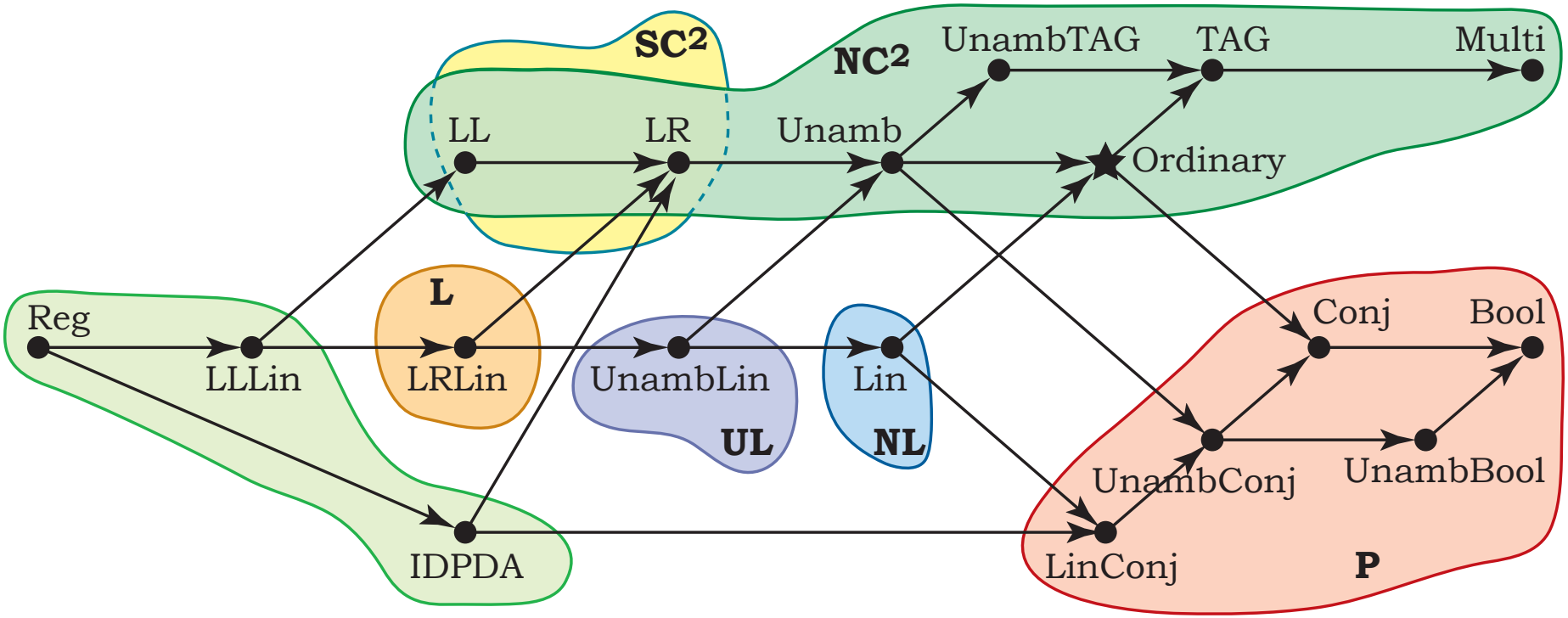
The hierarchy: complexity



The hierarchy: complexity

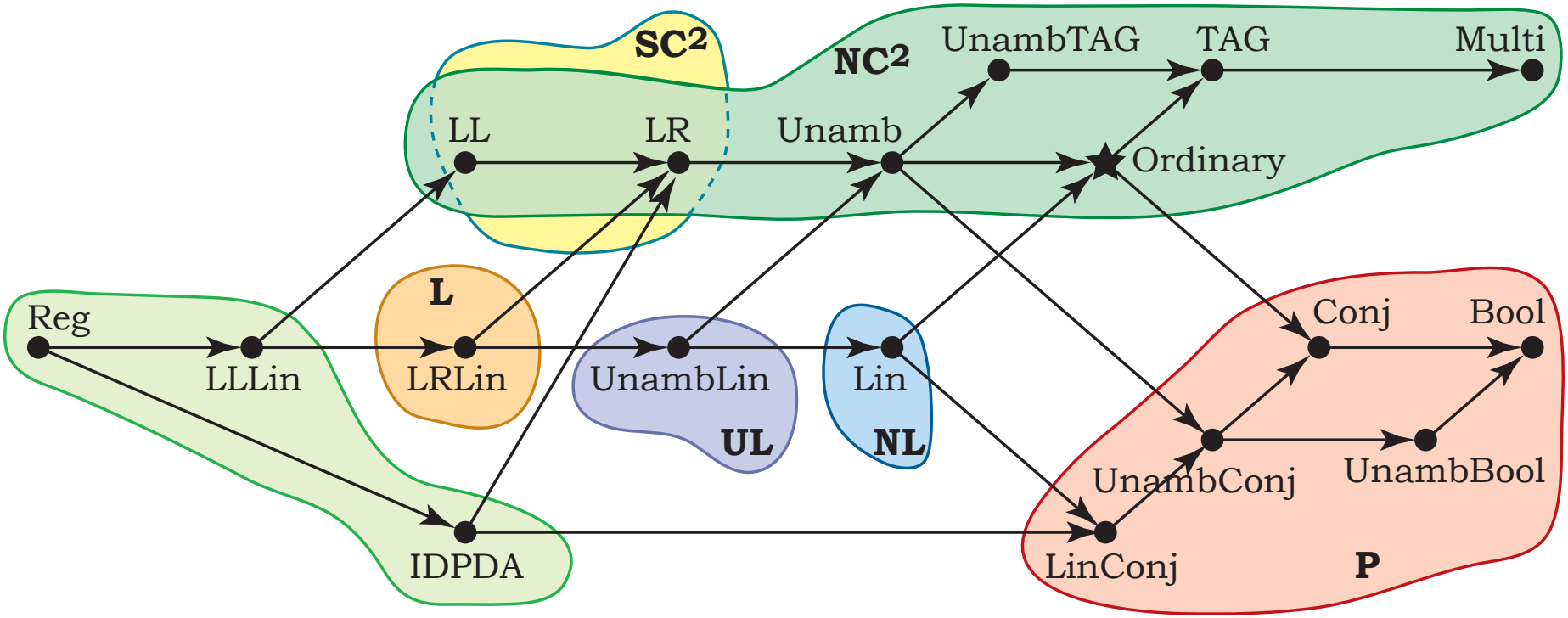


The hierarchy: complexity



- Complete sets for L, NL, P.

The hierarchy: complexity



- Complete sets for L, NL, P.
- Ordinary in NC^2:
  depth $O((\log n)^2)$, with $O(n^6)$ gates.

Part VII

Conclusion

Towards further models

Desired properties

Towards further models

Desired properties

1. Necessary syntactic structures expressed.

Towards further models

Desired properties

1. Necessary syntactic structures expressed.
2. Expressed in an intuitive way!

Towards further models

Desired properties

1. Necessary syntactic structures expressed.
2. Expressed in an intuitive way!
3. Algorithms for efficient processing.

Towards further models

Desired properties

1. Necessary syntactic structures expressed.
2. Expressed in an intuitive way!
3. Algorithms for efficient processing.

- Further fragments of FO(LFP)?

Towards further models

Desired properties

1. Necessary syntactic structures expressed.
2. Expressed in an intuitive way!
3. Algorithms for efficient processing.

- Further fragments of FO(LFP)?
  - E.g., "grammars with contexts" (Barash, Okhotin, 2014), rules such as $A \to BC \,\&\, {\lhd}D$.

Towards further models

Desired properties

1. Necessary syntactic structures expressed.
2. Expressed in an intuitive way!
3. Algorithms for efficient processing.

- Further fragments of FO(LFP)?
  - E.g., "grammars with contexts" (Barash, Okhotin, 2014), rules such as $A \to BC \,\&\, {\lhd}D$.
- New logical foundation?

Towards further models

Desired properties

1. Necessary syntactic structures expressed.
2. Expressed in an intuitive way!
3. Algorithms for efficient processing.

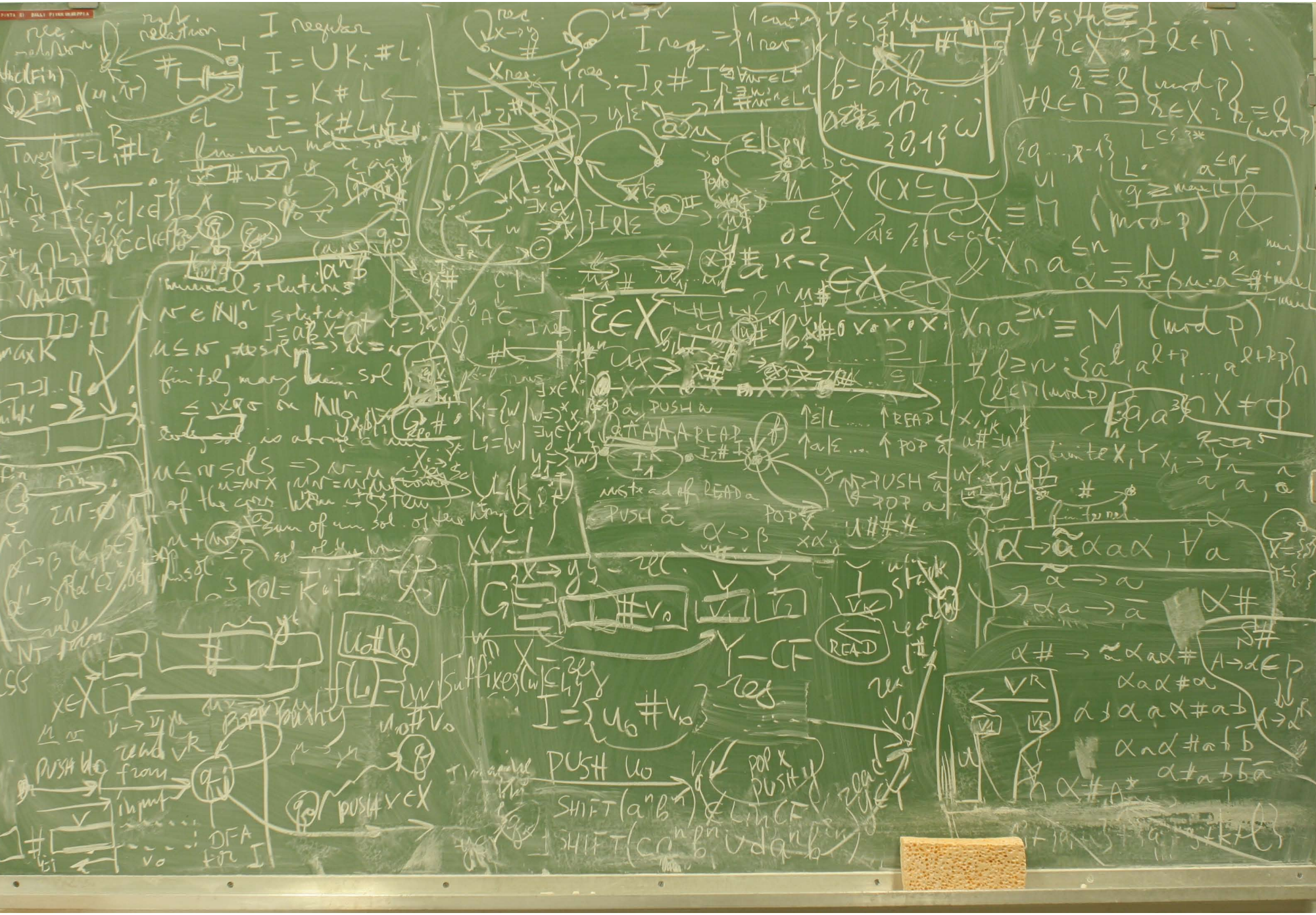
- Further fragments of FO(LFP)?
  - E.g., "grammars with contexts" (Barash, Okhotin, 2014), rules such as $A \to BC \,\&\, {\lhd}D$.
- New logical foundation?
  - Models from *descriptive complexity* (Immerman).

Towards further models

Desired properties

1. Necessary syntactic structures expressed.
2. Expressed in an intuitive way!
3. Algorithms for efficient processing.

- Further fragments of FO(LFP)?
  - E.g., "grammars with contexts" (Barash, Okhotin, 2014), rules such as $A \to BC \,\&\, {\lhd}D$.
- New logical foundation?
  - Models from *descriptive complexity* (Immerman).
- Understanding the existing models:
  any negative methods for conjunctive grammars?


I regular
I = UK_i # L_i
I = K # L
I = K # L
PUSH u
READ
POP
PUSH u0
SHIFT(a^n b^n)
SHIFT(ca^n b^n ∪ da^n b^n)
DFA for I