System Description: ACGtk

A Toolkit for Developing and Running Abstract Categorial Grammars

Maxime Guillaume[1,2] Sylvain Pogodalla[1] Vincent Tourneur[1]

[1]Inria, LORIA, Nancy, France

[2]Yseop, Lyon, France

FLOPS 2024, May 15th–17th
Kumamoto, Japan

Overview

1. Abstract Categorial Grammars (ACGs)
   - Motivations
   - Definitions
2. ACGtk
   - Examples
   - Composition
3. Implementation Features
   - Parsing as Datalog Querying
   - Shared Forests Contexts (Zippers)
   - Magic Set Rewritting

Abstract Categorial Grammars (de Groote 2001)

Main Features

- ACG is a (grammatical) framework
- Dedicated to the modeling of natural language syntax and semantics
- An ACG $\mathscr{G}$ generates two languages:
  - The abstract language $\mathcal{A}(\mathscr{G})$
  - The object language $\mathcal{O}(\mathscr{G})$

Abstract Categorial Grammars (de Groote 2001)

Main Features

- ACG is a (grammatical) framework
- Dedicated to the modeling of natural language syntax and semantics
- An ACG $\mathscr{G}$ generates two languages:
  - The abstract language $\mathcal{A}(\mathscr{G})$
  - The object language $\mathcal{O}(\mathscr{G})$

Abstract language admissible *structures*
Object language *realization* of the admissible structures

Abstract Categorial Grammars (de Groote 2001)

Main Features

- ACG is a (grammatical) framework
- Dedicated to the modeling of natural language syntax and semantics
- An ACG $\mathscr{G}$ generates two languages:
  - The abstract language $\mathcal{A}(\mathscr{G})$
  - The object language $\mathcal{O}(\mathscr{G})$

Abstract language admissible *structures*
Object language *realization* of the admissible structures

- Both languages are the same objects: sets of (almost linear) λ-terms

Abstract Categorial Grammars (de Groote 2001)

Main Features

- ACG is a (grammatical) framework
- Dedicated to the modeling of natural language syntax and semantics
- An ACG $\mathscr{G}$ generates two languages:
  - The abstract language $\mathcal{A}(\mathscr{G})$
  - The object language $\mathcal{O}(\mathscr{G})$

Abstract language admissible *structures*
Object language *realization* of the admissible structures

- Both languages are the same objects: sets of (almost linear) λ-terms
- ACGtk provides:
  - a special-purpose and functional programming language to design ACGs
  - development tools for ACGs:
    - acgc: ACG compiler
    - acg: top-level command interpreter

Types and Signatures

Types and Signatures

Definition (Implicative types)

Implicative types built upon A are $\mathscr{T}_A ::=$

Types and Signatures

Definition (Implicative types)

Implicative types built upon A are $\mathscr{T}_A ::= A$

Types and Signatures

Definition (Implicative types)

Implicative types built upon A are $\mathscr{T}_A ::= A \mid \mathscr{T}_A \rightarrow \mathscr{T}_A$

Types and Signatures

Definition (Implicative types)

Implicative types built upon A are $\mathscr{T}_A ::= A \mid \mathscr{T}_A \to \mathscr{T}_A \mid \mathscr{T}_A \Rightarrow \mathscr{T}_A$

Types and Signatures

Definition (Implicative types)

Implicative types built upon A are $\mathscr{T}_A ::= A \mid \mathscr{T}_A \to \mathscr{T}_A \mid \mathscr{T}_A \Rightarrow \mathscr{T}_A$

Definition (Higher-Order Signatures)

A *higher-order signature* Σ is a triple $\Sigma = \langle A, C, \tau \rangle$ where:

- A is a finite set of atomic types;
- C is a finite set of constants;
- $\tau : C \to \mathscr{T}_A$ is a function assigning types to constants.

Types and Signatures

Definition (Implicative types)

Implicative types built upon A are $\mathscr{T}_A ::= A \mid \mathscr{T}_A \rightarrow \mathscr{T}_A \mid \mathscr{T}_A \Rightarrow \mathscr{T}_A$

Definition (Higher-Order Signatures)

A *higher-order signature* Σ is a triple $\Sigma = \langle A, C, \tau \rangle$ where:

- A is a finite set of atomic types;
- C is a finite set of constants;
- $\tau : C \rightarrow \mathscr{T}_A$ is a function assigning types to constants.

Definition (λ-Terms over a signature Σ: $\Lambda(\Sigma)$)

$\Lambda(\Sigma) ::= \quad c \quad$ with $c \in C$

Types and Signatures

Definition (Implicative types)

Implicative types built upon A are $\mathscr{T}_A ::= A \mid \mathscr{T}_A \rightarrow \mathscr{T}_A \mid \mathscr{T}_A \Rightarrow \mathscr{T}_A$

Definition (Higher-Order Signatures)

A *higher-order signature* Σ is a triple $\Sigma = \langle A, C, \tau \rangle$ where:

- A is a finite set of atomic types;
- C is a finite set of constants;
- $\tau : C \rightarrow \mathscr{T}_A$ is a function assigning types to constants.

Definition (λ-Terms over a signature Σ: $\Lambda(\Sigma)$)

$$\begin{array}{rcll} \Lambda(\Sigma) ::= & & c & \text{with } c \in C \\ & \mid & x & \text{with } x \in X \end{array}$$

Types and Signatures

Definition (Implicative types)

Implicative types built upon A are $\mathscr{T}_A ::= A \mid \mathscr{T}_A \rightarrow \mathscr{T}_A \mid \mathscr{T}_A \Rightarrow \mathscr{T}_A$

Definition (Higher-Order Signatures)

A *higher-order signature* Σ is a triple $\Sigma = \langle A, C, \tau \rangle$ where:

- A is a finite set of atomic types;
- C is a finite set of constants;
- $\tau : C \rightarrow \mathscr{T}_A$ is a function assigning types to constants.

Definition (λ-Terms over a signature Σ: $\Lambda(\Sigma)$)

$$\begin{array}{llll} \Lambda(\Sigma) ::= & & c & \text{with } c \in C \\ & \mid & x & \text{with } x \in X \\ & \mid & \lambda^{o} x.t & \text{where } x \in X \text{ occurs free in } t \in \Lambda(\Sigma) \text{ exactly once} \end{array}$$

Types and Signatures

Definition (Implicative types)

Implicative types built upon A are $\mathscr{T}_A ::= A \mid \mathscr{T}_A \rightarrow \mathscr{T}_A \mid \mathscr{T}_A \Rightarrow \mathscr{T}_A$

Definition (Higher-Order Signatures)

A *higher-order signature* Σ is a triple $\Sigma = \langle A, C, \tau \rangle$ where:

- A is a finite set of atomic types;
- C is a finite set of constants;
- $\tau : C \rightarrow \mathscr{T}_A$ is a function assigning types to constants.

Definition (λ-Terms over a signature Σ: $\Lambda(\Sigma)$)

$$
\begin{array}{rcll}
\Lambda(\Sigma) ::= & & c & \text{with } c \in C \\
& \mid & x & \text{with } x \in X \\
& \mid & \lambda^{o} x.t & \text{where } x \in X \text{ occurs free in } t \in \Lambda(\Sigma) \text{ exactly once} \\
& \mid & \lambda x.t & \text{where } x \in X \text{ occurs free in } t \in \Lambda(\Sigma) \text{ at least once}
\end{array}
$$

Types and Signatures

Definition (Implicative types)

Implicative types built upon A are $\mathscr{T}_A ::= A \mid \mathscr{T}_A \rightarrow \mathscr{T}_A \mid \mathscr{T}_A \Rightarrow \mathscr{T}_A$

Definition (Higher-Order Signatures)

A *higher-order signature* Σ is a triple $\Sigma = \langle A, C, \tau \rangle$ where:

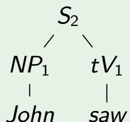
- A is a finite set of atomic types;
- C is a finite set of constants;
- $\tau : C \rightarrow \mathscr{T}_A$ is a function assigning types to constants.

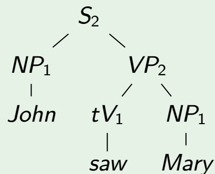
Definition (λ-Terms over a signature Σ: $\Lambda(\Sigma)$)

$$
\begin{array}{rcll}
\Lambda(\Sigma) ::= & & c & \text{with } c \in C \\
 & \mid & x & \text{with } x \in X \\
 & \mid & \lambda^{o}x.t & \text{where } x \in X \text{ occurs free in } t \in \Lambda(\Sigma) \text{ exactly once} \\
 & \mid & \lambda x.t & \text{where } x \in X \text{ occurs free in } t \in \Lambda(\Sigma) \text{ at least once} \\
 & \mid & (t\, u) & \text{with } t, u \in \Lambda(\Sigma)
\end{array}
$$

A Signature for a Tree Language

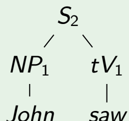
Example (Trees build over a ranked alphabet)

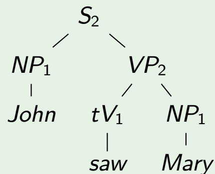




A Signature for a Tree Language

Example (Trees build over a ranked alphabet)





Example (Trees signature)

A Signature for a Tree Language

Example (Trees build over a ranked alphabet)

Example (Trees signature)

$$A_{\text{Trees}} = \{\tau\}$$

A Signature for a Tree Language

Example (Trees build over a ranked alphabet)

Example (Trees signature)

$$A_{\text{Trees}} = \{\tau\}$$
$$C_{\text{Trees}} = \{John, Mary, saw, everyone$$

A Signature for a Tree Language

Example (Trees build over a ranked alphabet)

Example (Trees signature)

$$
\begin{aligned}
A_{\text{Trees}} &= \{\tau\} \\
C_{\text{Trees}} &= \{John, Mary, saw, everyone \\
\tau_{\text{Trees}} &= \begin{cases} John & : \tau \\ Mary & : \tau \\ saw & : \tau \\ everyone & : \tau \end{cases}
\end{aligned}
$$

A Signature for a Tree Language

Example (Trees build over a ranked alphabet)

Example (Trees signature)

$$A_{\text{Trees}} = \{\tau\}$$
$$C_{\text{Trees}} = \{\mathit{John}, \mathit{Mary}, \mathit{saw}, \mathit{everyone}, \mathrm{NP}_1, \mathrm{tV}_1$$
$$\tau_{\text{Trees}} = \begin{cases} \mathit{John} & : \tau \\ \mathit{Mary} & : \tau \\ \mathit{saw} & : \tau \\ \mathit{everyone} & : \tau \end{cases}$$

A Signature for a Tree Language

Example (Trees build over a ranked alphabet)

Example (Trees signature)

$$
\begin{array}{lll}
A_{\text{Trees}} & = \{\tau\} \\
C_{\text{Trees}} & = \{\mathit{John}, \mathit{Mary}, \mathit{saw}, \mathit{everyone}, \mathrm{NP}_1, \mathrm{tV}_1 \\
\tau_{\text{Trees}} & = \left\{ \begin{array}{rlrl}
\mathit{John} & : \tau & \mathrm{NP}_1 & : \tau \to \tau \\
\mathit{Mary} & : \tau & \mathrm{tV}_1 & : \tau \to \tau \\
\mathit{saw} & : \tau & & \\
\mathit{everyone} & : \tau & &
\end{array} \right.
\end{array}
$$

A Signature for a Tree Language

Example (Trees build over a ranked alphabet)

Example (Trees signature)

$$
\begin{aligned}
A_{\text{Trees}} &= \{\tau\}\\
C_{\text{Trees}} &= \{\mathit{John}, \mathit{Mary}, \mathit{saw}, \mathit{everyone}, \mathrm{NP}_1, \mathrm{tV}_1, \mathrm{S}_2, \mathrm{VP}_2\}\\
\tau_{\text{Trees}} &= \left\{\begin{array}{rlrl}
\mathit{John} & : \tau & \mathrm{NP}_1 & : \tau \to \tau\\
\mathit{Mary} & : \tau & \mathrm{tV}_1 & : \tau \to \tau\\
\mathit{saw} & : \tau & & \\
\mathit{everyone} & : \tau & &
\end{array}\right.
\end{aligned}
$$

A Signature for a Tree Language

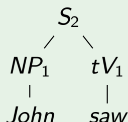
Example (Trees build over a ranked alphabet)

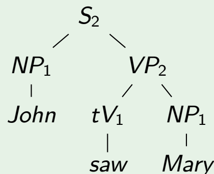
Example (Trees signature)

$$
\begin{aligned}
A_{\text{Trees}} &= \{\tau\} \\
C_{\text{Trees}} &= \{\mathit{John}, \mathit{Mary}, \mathit{saw}, \mathit{everyone}, \mathrm{NP}_1, \mathrm{tV}_1, \mathrm{S}_2, \mathrm{VP}_2\} \\
\tau_{\text{Trees}} &= \left\{ \begin{array}{rlrl}
\mathit{John} & : \tau & \mathrm{NP}_1 & : \tau \rightarrow \tau \\
\mathit{Mary} & : \tau & \mathrm{tV}_1 & : \tau \rightarrow \tau \\
\mathit{saw} & : \tau & \mathrm{VP}_2 & : \tau \rightarrow \tau \rightarrow \tau \\
\mathit{everyone} & : \tau & \mathrm{S}_2 & : \tau \rightarrow \tau \rightarrow \tau
\end{array} \right.
\end{aligned}
$$

A Signature for a Tree Language

Example (Trees build over a ranked alphabet)





$\mathrm{S}_2\,(\mathrm{NP}_1\,\mathit{John})\,(\mathrm{tV}_1\,\mathit{saw}) : \tau$

Example (Trees signature)

$$
\begin{aligned}
A_{\text{Trees}} &= \{\tau\} \\
C_{\text{Trees}} &= \{\mathit{John}, \mathit{Mary}, \mathit{saw}, \mathit{everyone}, \mathrm{NP}_1, \mathrm{tV}_1, \mathrm{S}_2, \mathrm{VP}_2\} \\
\tau_{\text{Trees}} &= \left\{
\begin{array}{rlrl}
\mathit{John} & : \tau & \mathrm{NP}_1 & : \tau \to \tau \\
\mathit{Mary} & : \tau & \mathrm{tV}_1 & : \tau \to \tau \\
\mathit{saw} & : \tau & \mathrm{VP}_2 & : \tau \to \tau \to \tau \\
\mathit{everyone} & : \tau & \mathrm{S}_2 & : \tau \to \tau \to \tau
\end{array}
\right.
\end{aligned}
$$

A Signature for a Tree Language

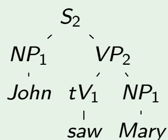
Example (Trees build over a ranked alphabet)

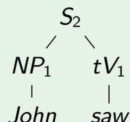
$\mathrm{S}_2\,(\mathrm{NP}_1\ \mathit{John})\,(\mathrm{tV}_1\ \mathit{saw}) : \tau$ $\quad$ $\mathrm{S}_2\,(\mathrm{NP}_1\ \mathit{John})\,(\mathrm{VP}_2\,(\mathrm{tV}_1\ \mathit{saw})\,(\mathrm{NP}_1\ \mathit{Mary})) : \tau$

Example (Trees signature)

$$
\begin{aligned}
A_{\text{Trees}} &= \{\tau\} \\
C_{\text{Trees}} &= \{\mathit{John}, \mathit{Mary}, \mathit{saw}, \mathit{everyone}, \mathrm{NP}_1, \mathrm{tV}_1, \mathrm{S}_2, \mathrm{VP}_2\} \\
\tau_{\text{Trees}} &= \left\{ \begin{array}{rlrl}
\mathit{John} & : \tau & \mathrm{NP}_1 & : \tau \rightarrow \tau \\
\mathit{Mary} & : \tau & \mathrm{tV}_1 & : \tau \rightarrow \tau \\
\mathit{saw} & : \tau & \mathrm{VP}_2 & : \tau \rightarrow \tau \rightarrow \tau \\
\mathit{everyone} & : \tau & \mathrm{S}_2 & : \tau \rightarrow \tau \rightarrow \tau
\end{array} \right.
\end{aligned}
$$

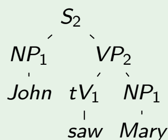
A Signature for a Tree Language: The ACGtk Way

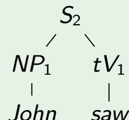




$\mathrm{S}_2\,(\mathrm{NP}_1\ \mathit{John})\,(\mathrm{VP}_2\,(\mathrm{tV}_1\ \mathit{saw})\,(\mathrm{NP}_1\ \mathit{Mary})) : \tau$ $\mathrm{S}_2\,(\mathrm{NP}_1\ \mathit{John})\,(\mathrm{tV}_1\ \mathit{saw}) : \tau$

A Signature for a Tree Language: The ACGtk Way



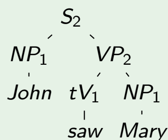


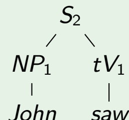
$S_2\,(NP_1\,John)\,(VP_2\,(tV_1\,saw)\,(NP_1\,Mary)) : \tau$ $S_2\,(NP_1\,John)\,(tV_1\,saw) : \tau$

cfg.acg

```
signature Trees =

end
```

A Signature for a Tree Language: The ACGtk Way



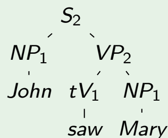


$S_2\,(NP_1\,\textit{John})\,(VP_2\,(tV_1\,\textit{saw})\,(NP_1\,\textit{Mary})) : \tau$ $\quad S_2\,(NP_1\,\textit{John})\,(tV_1\,\textit{saw}) : \tau$

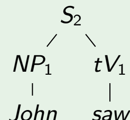
cfg.acg

```
signature Trees =
  τ                              : type;



end
```

A Signature for a Tree Language: The ACGtk Way



$S_2\,(NP_1\ \mathit{John})\,(VP_2\,(tV_1\ \mathit{saw})\,(NP_1\ \mathit{Mary})) : \tau$

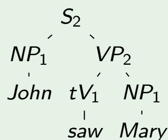


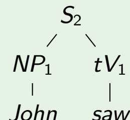
$S_2\,(NP_1\ \mathit{John})\,(tV_1\ \mathit{saw}) : \tau$

cfg.acg

```
signature Trees =
  τ                            : type;
  John, Mary, saw, everyone    : τ;


end
```

A Signature for a Tree Language: The ACGtk Way



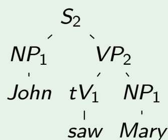


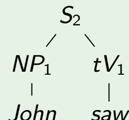
$\mathrm{S}_2\,(\mathrm{NP}_1\,\textit{John})\,(\mathrm{VP}_2\,(\mathrm{tV}_1\,\textit{saw})\,(\mathrm{NP}_1\,\textit{Mary})) : \tau$ $\quad$ $\mathrm{S}_2\,(\mathrm{NP}_1\,\textit{John})\,(\mathrm{tV}_1\,\textit{saw}) : \tau$

cfg.acg

```
signature Trees =
  τ                              : type;
  John, Mary, saw, everyone      : τ;
  NP₁, tV₁                       : τ → τ;

end
```

A Signature for a Tree Language: The ACGtk Way



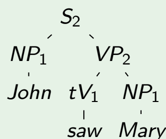


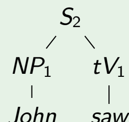
$\mathrm{S}_2\,(\mathrm{NP}_1\,\mathit{John})\,(\mathrm{VP}_2\,(\mathrm{tV}_1\,\mathit{saw})\,(\mathrm{NP}_1\,\mathit{Mary})) : \tau$ $\quad\mathrm{S}_2\,(\mathrm{NP}_1\,\mathit{John})\,(\mathrm{tV}_1\,\mathit{saw}) : \tau$

cfg.acg

```
signature Trees =
  τ                              : type;
  John, Mary, saw, everyone      : τ;
  NP1, tV1                       : τ → τ;
  S2, VP2                        : τ → τ → τ;
end
```

A Signature for a Tree Language: The ACGtk Way





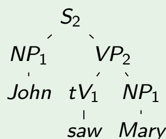
$\mathrm{S}_2\,(\mathrm{NP}_1\ \mathit{John})\,(\mathrm{VP}_2\,(\mathrm{tV}_1\ \mathit{saw})\,(\mathrm{NP}_1\ \mathit{Mary})) : \tau$ $\quad \mathrm{S}_2\,(\mathrm{NP}_1\ \mathit{John})\,(\mathrm{tV}_1\ \mathit{saw}) : \tau$

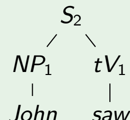
cfg.acg

```
signature Trees =
   τ                             : type;
   John, Mary, saw, everyone     : τ;
   NP1, tV1                      : τ → τ;
   S2, VP2                       : τ → τ → τ;
end
```

```
ACGtk>"S2 (NP1 John) (tV1 saw) : τ" | check signature = Trees
```

A Signature for a Tree Language: The ACGtk Way





$\mathrm{S}_2\,(\mathrm{NP}_1\,\mathit{John})\,(\mathrm{VP}_2\,(\mathrm{tV}_1\,\mathit{saw})\,(\mathrm{NP}_1\,\mathit{Mary})) : \tau$ $\quad$ $\mathrm{S}_2\,(\mathrm{NP}_1\,\mathit{John})\,(\mathrm{tV}_1\,\mathit{saw}) : \tau$

cfg.acg

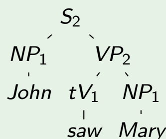
`signature` Trees $=$

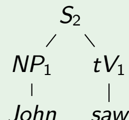
τ	: `type`;
John, Mary, saw, everyone	: τ;
$\mathrm{NP}_1, \mathrm{tV}_1$	: $\tau \to \tau$;
$\mathrm{S}_2, \mathrm{VP}_2$	: $\tau \to \tau \to \tau$;

`end`

`ACGtk>`"$\mathrm{S}_2\,(\mathrm{NP}_1\,\mathrm{John})\,(\mathrm{tV}_1\,\mathrm{saw}) : \tau$" | `check signature` = Trees
1 term computed.

A Signature for a Tree Language: The ACGtk Way





$\mathrm{S}_2\,(\mathrm{NP}_1\,\mathit{John})\,(\mathrm{VP}_2\,(\mathrm{tV}_1\,\mathit{saw})\,(\mathrm{NP}_1\,\mathit{Mary})) : \tau$ $\mathrm{S}_2\,(\mathrm{NP}_1\,\mathit{John})\,(\mathrm{tV}_1\,\mathit{saw}) : \tau$

cfg.acg

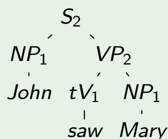
signature Trees =

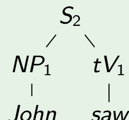
τ	: type;
John, Mary, saw, everyone	: τ;
$\mathrm{NP}_1, \mathrm{tV}_1$	: $\tau \rightarrow \tau$;
$\mathrm{S}_2, \mathrm{VP}_2$	: $\tau \rightarrow \tau \rightarrow \tau$;

end

ACGtk>"$\mathrm{S}_2\,(\mathrm{NP}_1\,\mathrm{John})\,(\mathrm{VP}_2\,(\mathrm{tV}_1\,\mathrm{saw})) : \tau$" | check signature = Trees

A Signature for a Tree Language: The ACGtk Way





$\mathrm{S}_2\,(\mathrm{NP}_1\,\mathit{John})\,(\mathrm{VP}_2\,(\mathrm{tV}_1\,\mathit{saw})\,(\mathrm{NP}_1\,\mathit{Mary})) : \tau$ $\quad$ $\mathrm{S}_2\,(\mathrm{NP}_1\,\mathit{John})\,(\mathrm{tV}_1\,\mathit{saw}) : \tau$

cfg.acg

```
signature Trees =
  τ                                  : type;
  John, Mary, saw, everyone          : τ;
  NP₁, tV₁                           : τ → τ;
  S₂, VP₂                            : τ → τ → τ;
end
```

ACGtk>"$\mathrm{S}_2\,(\mathrm{NP}_1$ John) ($\mathrm{VP}_2\,(\mathrm{tV}_1$ saw)) : τ" | check signature = Trees
[ERROR] Type error: line 1, characters 16-28:
The type of this expression is "τ" but is used with type "$\tau \rightarrow \tau$".

A Signature for a String Language

cfg.acg

```
signature Strings =

end
```

A Signature for a String Language

cfg.acg

```
signature Strings =
  o : type;

end
```

A Signature for a String Language

cfg.acg

signature Strings =
o : type;
$\sigma = o \to o$: type;

end

A Signature for a String Language

cfg.acg

signature Strings =
o : type;
$\sigma = o \rightarrow o$: type;
John, Mary, saw, everyone : σ;

end

A Signature for a String Language

cfg.acg

signature Strings =

o : type;

$\sigma = o \rightarrow o$: type;

John, Mary, saw, everyone : σ;

$\epsilon = \lambda^{o}x.\,x$: σ;

end

A Signature for a String Language

cfg.acg

$$
\begin{array}{ll}
\texttt{signature}\ \mathsf{Strings} = & \\
\quad o & : \texttt{type}; \\
\quad \sigma = o \rightarrow o & : \texttt{type}; \\
\quad \mathsf{John}, \mathsf{Mary}, \mathsf{saw}, \mathsf{everyone} & : \sigma; \\
\quad \epsilon = \lambda^{o} x.\, x & : \sigma; \\
\quad \texttt{infix}+ = \lambda x\, y.\, \lambda^{o} z.\, x\,(y\, z) & : \sigma \rightarrow \sigma \rightarrow \sigma; \\
\texttt{end} &
\end{array}
$$

A Signature for a String Language

cfg.acg

```
signature Strings =
  o                                  : type;
  σ = o → o                          : type;
  John, Mary, saw, everyone          : σ;
  ϵ = λ°x. x                         : σ;
  infix+ = λx y. λ°z. x (y z)        : σ → σ → σ;
end
```

Remark

- The string $John + saw + Mary$ is represented by the term $\lambda^{\circ} z.\, John\,(saw\,(Mary\, z))$
- $w + \epsilon =_{\beta} w =_{\beta} \epsilon + w$
- $w_1 + (w_2 + w_3) =_{\beta} (w_1 + w_2) + w_3$

A Signature for a String Language

cfg.acg

$$
\begin{array}{ll}
\texttt{signature Strings} = & \\
\quad o & : \texttt{type}; \\
\quad \sigma = o \rightarrow o & : \texttt{type}; \\
\quad \text{John}, \text{Mary}, \text{saw}, \text{everyone} & : \sigma; \\
\quad \epsilon = \lambda^{o} x.\, x & : \sigma; \\
\quad \texttt{infix}+ = \lambda x\, y.\, \lambda^{o} z.\, x\,(y\, z) & : \sigma \rightarrow \sigma \rightarrow \sigma; \\
\texttt{end} &
\end{array}
$$

Remark

- The string $John + saw + Mary$ is represented by the term $\lambda^{o} z.\, John\,(saw\,(Mary\, z))$
- $w + \epsilon =_{\beta} w =_{\beta} \epsilon + w$
- $w_1 + (w_2 + w_3) =_{\beta} (w_1 + w_2) + w_3$

How to relate the two signatures?

Relating Signature: Lexicons

Definition (Lexicon)

Let $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ be two higher-order signatures, a lexicon $\mathscr{L} = \langle F, G \rangle$ from Σ_1 to Σ_2 is such that:

Relating Signature: Lexicons

Definition (Lexicon)

Let $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ be two higher-order signatures, a lexicon $\mathscr{L} = \langle F, G \rangle$ from Σ_1 to Σ_2 is such that:

- $F : \mathscr{T}_{A_1} \to \mathscr{T}_{A_2}$ is a morphism;

Relating Signature: Lexicons

Definition (Lexicon)

Let $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ be two higher-order signatures, a lexicon $\mathscr{L} = \langle F, G \rangle$ from Σ_1 to Σ_2 is such that:

- $F : \mathscr{T}_{A_1} \to \mathscr{T}_{A_2}$ is a morphism;
- $G : \Lambda(\Sigma_1) \to \Lambda(\Sigma_2)$ is a morphism;

Relating Signature: Lexicons

Definition (Lexicon)

Let $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ be two higher-order signatures, a lexicon $\mathscr{L} = \langle F, G \rangle$ from Σ_1 to Σ_2 is such that:

- $F : \mathscr{T}_{A_1} \rightarrow \mathscr{T}_{A_2}$ is a morphism;
- $G : \Lambda(\Sigma_1) \rightarrow \Lambda(\Sigma_2)$ is a morphism;
- F and G are such that for all $c \in C_1$, $\vdash_{\Sigma_2} G(c) : F(\tau_1(c))$ is provable.

Relating Signature: Lexicons

Definition (Lexicon)

Let $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ be two higher-order signatures, a lexicon $\mathscr{L} = \langle F, G \rangle$ from Σ_1 to Σ_2 is such that:

- $F : \mathscr{T}_{A_1} \to \mathscr{T}_{A_2}$ is a morphism;
- $G : \Lambda(\Sigma_1) \to \Lambda(\Sigma_2)$ is a morphism;
- F and G are such that for all $c \in C_1$, $\vdash_{\Sigma_2} G(c) : F(\tau_1(c))$ is provable.

Notation: we also use $\mathscr{L}$ instead of F or G.

Relating Signature: Lexicons

Definition (Lexicon)

Let $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ be two higher-order signatures, a lexicon $\mathscr{L} = \langle F, G \rangle$ from Σ_1 to Σ_2 is such that:

- $F : \mathscr{T}_{A_1} \to \mathscr{T}_{A_2}$ is a morphism;
- $G : \Lambda(\Sigma_1) \to \Lambda(\Sigma_2)$ is a morphism;
- F and G are such that for all $c \in C_1$, $\vdash_{\Sigma_2} G(c) : F(\tau_1(c))$ is provable.

Notation: we also use $\mathscr{L}$ instead of F or G.

Interpretation (Realization)

If $\mathscr{L}(t) = u$, also denoted by $t := u$, we say that u is the *interpretation* or the *realization* of t.

Interpreting Trees as Strings

Interpreting Trees as Strings

cfg.acg

```
lexicon Yield (Trees) : Strings =

end
```

Interpreting Trees as Strings

cfg.acg

$\texttt{lexicon}$ Yield (Trees) : Strings =

$\tau \quad := \sigma;$

end

Interpreting Trees as Strings

cfg.acg

```
lexicon Yield (Trees) : Strings =
  τ         := σ;
  John      := John;
  Mary      := Mary;
  saw       := saw;
  everyone  := everyone;


end
```

Interpreting Trees as Strings

cfg.acg

```
lexicon Yield (Trees) : Strings =
  τ           := σ;
  John        := John;
  Mary        := Mary;
  saw         := saw;
  everyone    := everyone;
  NP_1, tV_1  := λ°x. x;

end
```

Interpreting Trees as Strings

cfg.acg

`lexicon` Yield (Trees) : Strings $=$

τ	$:= \sigma$;
John	$:=$ John;
Mary	$:=$ Mary;
saw	$:=$ saw;
everyone	$:=$ everyone;
$\mathrm{NP}_1, \mathrm{tV}_1$	$:= \lambda^{o} x.\, x$;
$\mathrm{S}_2, \mathrm{VP}_2$	$:= \lambda^{o} x\, y.\, x + y$;

end

Interpreting Trees as Strings

cfg.acg

```
lexicon Yield (Trees) : Strings =
  τ           := σ;
  John        := John;
  Mary        := Mary;
  saw         := saw;
  everyone    := everyone;
  NP₁, tV₁    := λ°x. x;
  S₂, VP₂     := λ°x y. x + y;
end
```

```
ACGtk>"S₂ (NP₁ John) (VP₂ (tV₁ saw)(NP₁ Mary)) : τ" | realize lexicons = Yield
```

Interpreting Trees as Strings

cfg.acg

$\texttt{lexicon}$ Yield (Trees) : Strings =

τ	$:= \sigma;$
John	:= John;
Mary	:= Mary;
saw	:= saw;
everyone	:= everyone;
NP_1, tV_1	$:= \lambda^o x. x;$
S_2, VP_2	$:= \lambda^o x\, y. x + y;$

end

```
ACGtk>"S_2 (NP_1 John) (VP_2 (tV_1 saw)(NP_1 Mary)) : τ" | realize lexicons = Yield
Term: λ° z. John (saw (Mary z)) : o → o
1 term computed.
```

Abstract Categorial Grammars

Definition (Abstract Categorial Grammar and vocabulary)

An *abstract categorial grammar* is a quadruple $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, S \rangle$ where:

- $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ are two higher-order signatures (the *abstract vocabulary* and the *object vocabulary*, resp.)

Abstract Categorial Grammars

Definition (Abstract Categorial Grammar and vocabulary)

An *abstract categorial grammar* is a quadruple $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, S \rangle$ where:

- $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ are two higher-order signatures (the *abstract vocabulary* and the *object vocabulary*, resp.)
- $\mathscr{L} : \Sigma_1 \to \Sigma_2$ is a lexicon.

Abstract Categorial Grammars

Definition (Abstract Categorial Grammar and vocabulary)

An *abstract categorial grammar* is a quadruple $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, S \rangle$ where:

- $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ are two higher-order signatures (the *abstract vocabulary* and the *object vocabulary*, resp.)
- $\mathscr{L} : \Sigma_1 \to \Sigma_2$ is a lexicon.
- $S \in \mathscr{T}_{A_1}$ is the *distinguished type* of the grammar.

Abstract Categorial Grammars

Definition (Abstract Categorial Grammar and vocabulary)

An *abstract categorial grammar* is a quadruple $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, S \rangle$ where:

- $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ are two higher-order signatures (the *abstract vocabulary* and the *object vocabulary*, resp.)
- $\mathscr{L} : \Sigma_1 \rightarrow \Sigma_2$ is a lexicon.
- $S \in \mathscr{T}_{A_1}$ is the *distinguished type* of the grammar.

Definition (Abstract and Object Languages)

Abstract Categorial Grammars

Definition (Abstract Categorial Grammar and vocabulary)

An *abstract categorial grammar* is a quadruple $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, S \rangle$ where:

- $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ are two higher-order signatures (the *abstract vocabulary* and the *object vocabulary*, resp.)
- $\mathscr{L} : \Sigma_1 \to \Sigma_2$ is a lexicon.
- $S \in \mathscr{T}_{A_1}$ is the *distinguished type* of the grammar.

Definition (Abstract and Object Languages)

Abstract language $\mathcal{A}(\mathscr{G}) = \{t \in \Lambda(\Sigma_1) \mid \vdash_{\Sigma_1} t : S \text{ is derivable}\}$

Abstract Categorial Grammars

Definition (Abstract Categorial Grammar and vocabulary)

An *abstract categorial grammar* is a quadruple $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, S \rangle$ where:

- $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ are two higher-order signatures (the *abstract vocabulary* and the *object vocabulary*, resp.)
- $\mathscr{L} : \Sigma_1 \to \Sigma_2$ is a lexicon.
- $S \in \mathscr{T}_{A_1}$ is the *distinguished type* of the grammar.

Definition (Abstract and Object Languages)

Abstract language	$\mathcal{A}(\mathscr{G}) = \{t \in \Lambda(\Sigma_1) \mid \vdash_{\Sigma_1} t : S \text{ is derivable}\}$
Object language	$\mathcal{O}(\mathscr{G}) = \{u \in \Lambda(\Sigma_2) \mid \exists t \in \mathcal{A}(\mathscr{G}) \text{ such that } u = \mathscr{L}(t)\}$

Abstract Categorial Grammars

Definition (Abstract Categorial Grammar and vocabulary)

An *abstract categorial grammar* is a quadruple $\mathscr{G} = \langle \Sigma_1, \Sigma_2, \mathscr{L}, S \rangle$ where:

- $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$ and $\Sigma_2 = \langle A_2, C_2, \tau_2 \rangle$ are two higher-order signatures (the *abstract vocabulary* and the *object vocabulary*, resp.)
- $\mathscr{L} : \Sigma_1 \to \Sigma_2$ is a lexicon.
- $S \in \mathscr{T}_{A_1}$ is the *distinguished type* of the grammar.

Definition (Abstract and Object Languages)

Abstract language $\quad \mathcal{A}(\mathscr{G}) = \{t \in \Lambda(\Sigma_1) \mid \vdash_{\Sigma_1} t : S \text{ is derivable}\}$

Object language $\quad \mathcal{O}(\mathscr{G}) = \{u \in \Lambda(\Sigma_2) \mid \exists t \in \mathcal{A}(\mathscr{G}) \text{ such that } u = \mathscr{L}(t)\}$

To parse u:

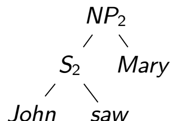
- to find $t \in \mathcal{A}(\mathscr{G})$ such that $u = \mathscr{L}(t)$
- ACG parsing is *morphism inversion*

Example: Parsing Strings

ACGtk>"λ^o z. John (saw (Mary z)) : o $\to$ o" | parse lexicon = Yield type = τ

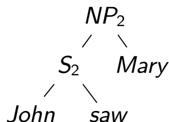
Example: Parsing Strings

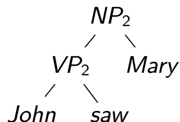
ACGtk>" λ^o z. John (saw (Mary z)) : o $\rightarrow$ o" | parse lexicon = Yield type = τ
Parsing time: 90.6μs
Parse forest building time: 16μs
Term (depth = 3, size = 5): NP_2 (S_2 John saw) Mary : τ



Example: Parsing Strings

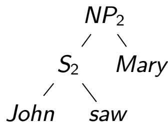
ACGtk>"λ^o z. John (saw (Mary z)) : o $\rightarrow$ o" | parse lexicon = Yield type = τ
Parsing time: 90.6μs
Parse forest building time: 16μs
Term (depth = 3, size = 5): NP_2 (S_2 John saw) Mary : τ
Term (depth = 3, size = 5): NP_2 (VP_2 John saw) Mary : τ

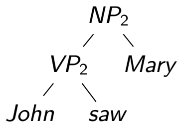


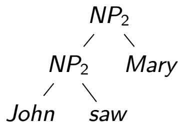


Example: Parsing Strings

`ACGtk>`" λ^o z. John (saw (Mary z)) : o $\rightarrow$ o" | `parse` `lexicon` = Yield `type` = τ
Parsing time: 90.6μs
Parse forest building time: 16μs
Term (depth = 3, size = 5): NP_2 (S_2 John saw) Mary : τ
Term (depth = 3, size = 5): NP_2 (VP_2 John saw) Mary : τ
Term (depth = 3, size = 5): NP_2 (NP_2 John saw) Mary : τ

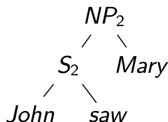


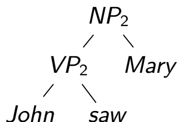


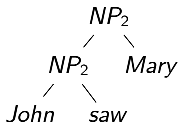


Example: Parsing Strings

ACGtk>" λ^o z. John (saw (Mary z)) : o $\rightarrow$ o" | parse lexicon = Yield type = τ
Parsing time: 90.6μs
Parse forest building time: 16μs
Term (depth = 3, size = 5): NP_2 (S_2 John saw) Mary : τ
Term (depth = 3, size = 5): NP_2 (VP_2 John saw) Mary : τ
Term (depth = 3, size = 5): NP_2 (NP_2 John saw) Mary : τ
...







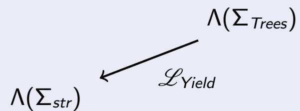
Controlling Abstract Structures

Controlling Abstract Structures

ACG Composition

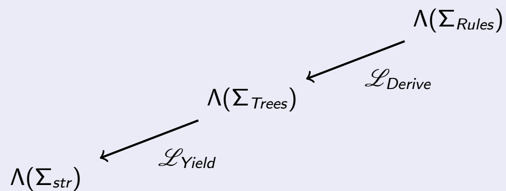
Controlling Abstract Structures

ACG Composition



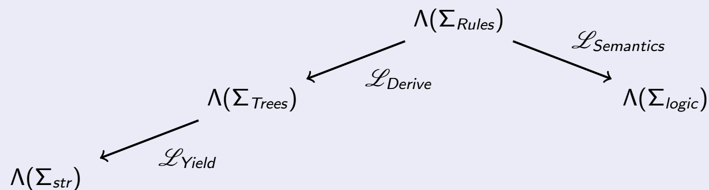
Controlling Abstract Structures

ACG Composition



Controlling Abstract Structures

ACG Composition



Encoding a Context-Free Grammar

Encoding a Context-Free Grammar

Example (CFG)

$$\begin{aligned} \rho_0 &= & \mathsf{S}' &\rightarrow \mathsf{S} \\ \rho_1 &= & \mathsf{S} &\rightarrow \mathsf{NP}\ \mathsf{VP} \\ \rho_2 &= & \mathsf{VP} &\rightarrow \mathsf{tV}\ \mathsf{NP} \\ \rho_3 &= & \mathsf{NP} &\rightarrow \textit{John} \end{aligned}$$

$$\begin{aligned} \rho_4 &= & \mathsf{NP} &\rightarrow \textit{Mary} \\ \rho_5 &= & \mathsf{NP} &\rightarrow \textit{everyone} \\ \rho_6 &= & \mathsf{tV} &\rightarrow \textit{saw} \end{aligned}$$

Encoding a Context-Free Grammar

Example (CFG)

$\rho_0 = \quad S' \rightarrow S$

$\rho_1 = \quad S \rightarrow NP\ VP$

$\rho_2 = \quad VP \rightarrow tV\ NP$

$\rho_3 = \quad NP \rightarrow \textit{John}$

$\rho_4 = \quad NP \rightarrow \textit{Mary}$

$\rho_5 = \quad NP \rightarrow \textit{everyone}$

$\rho_6 = \quad tV \rightarrow \textit{saw}$

cfg.acg

```
signature Rules =

end
```

Encoding a Context-Free Grammar

Example (CFG)

$$
\begin{array}{llll}
\rho_0 = & S' & \rightarrow S \\
\rho_1 = & S & \rightarrow NP\ VP \\
\rho_2 = & VP & \rightarrow tV\ NP \\
\rho_3 = & NP & \rightarrow \textit{John}
\end{array}
\qquad
\begin{array}{llll}
\rho_4 = & NP & \rightarrow \textit{Mary} \\
\rho_5 = & NP & \rightarrow \textit{everyone} \\
\rho_6 = & tV & \rightarrow \textit{saw}
\end{array}
$$

cfg.acg

```
signature Rules =
  S, S', NP, VP, tV    : type;

end
```

Encoding a Context-Free Grammar

Example (CFG)

$\rho_0 = \; S' \rightarrow S$

$\rho_1 = \; S \rightarrow NP\ VP$

$\rho_2 = \; VP \rightarrow tV\ NP$

$\rho_3 = \; NP \rightarrow \textit{John}$

$\rho_4 = \; NP \rightarrow \textit{Mary}$

$\rho_5 = \; NP \rightarrow \textit{everyone}$

$\rho_6 = \; tV \rightarrow \textit{saw}$

cfg.acg

```
signature Rules =
  S, S', NP, VP, tV   : type;
  ρ0                  : S → S';




end
```

Encoding a Context-Free Grammar

Example (CFG)

$$\begin{aligned}
\rho_0 &= & S' &\rightarrow S & \qquad \rho_4 &= & NP &\rightarrow \textit{Mary}\\
\rho_1 &= & S &\rightarrow NP\ VP & \rho_5 &= & NP &\rightarrow \textit{everyone}\\
\rho_2 &= & VP &\rightarrow tV\ NP & \rho_6 &= & tV &\rightarrow \textit{saw}\\
\rho_3 &= & NP &\rightarrow \textit{John} & & & &
\end{aligned}$$

cfg.acg

$$\begin{aligned}
&\texttt{signature}\ \text{Rules} =\\
&\quad S, S', NP, VP, tV &&: \texttt{type};\\
&\quad \rho_0 &&: S \rightarrow S';\\
&\quad \rho_1 &&: NP \rightarrow VP \rightarrow S;\\
&\texttt{end}
\end{aligned}$$

Encoding a Context-Free Grammar

Example (CFG)

$$\begin{aligned} \rho_0 &= & S' &\to S \\ \rho_1 &= & S &\to \text{NP VP} \\ \rho_2 &= & \text{VP} &\to \text{tV NP} \\ \rho_3 &= & \text{NP} &\to \textit{John} \end{aligned} \qquad \begin{aligned} \rho_4 &= & \text{NP} &\to \textit{Mary} \\ \rho_5 &= & \text{NP} &\to \textit{everyone} \\ \rho_6 &= & \text{tV} &\to \textit{saw} \end{aligned}$$

cfg.acg

signature Rules =

S, S', NP, VP, tV	: type;
ρ_0	: $S \to S'$;
ρ_1	: $NP \to VP \to S$;
ρ_2	: $tV \to NP \to VP$;

end

Encoding a Context-Free Grammar

Example (CFG)

$$\begin{aligned}
\rho_0 &= S' \rightarrow S & \rho_4 &= NP \rightarrow \textit{Mary}\\
\rho_1 &= S \rightarrow NP\ VP & \rho_5 &= NP \rightarrow \textit{everyone}\\
\rho_2 &= VP \rightarrow tV\ NP & \rho_6 &= tV \rightarrow \textit{saw}\\
\rho_3 &= NP \rightarrow \textit{John} & &
\end{aligned}$$

cfg.acg

```
signature Rules =
   S, S′, NP, VP, tV    : type;
   ρ0                   : S → S′;
   ρ1                   : NP → VP → S;
   ρ2                   : tV → NP → VP;
   ρ3, ρ4, ρ5           : NP;

end
```

Encoding a Context-Free Grammar

Example (CFG)

$$\begin{aligned} \rho_0 &= S' \rightarrow S & \rho_4 &= NP \rightarrow \textit{Mary} \\ \rho_1 &= S \rightarrow NP\ VP & \rho_5 &= NP \rightarrow \textit{everyone} \\ \rho_2 &= VP \rightarrow tV\ NP & \rho_6 &= tV \rightarrow \textit{saw} \\ \rho_3 &= NP \rightarrow \textit{John} \end{aligned}$$

cfg.acg

```
signature Rules =
    S, S', NP, VP, tV   : type;
    ρ0                  : S → S';
    ρ1                  : NP → VP → S;
    ρ2                  : tV → NP → VP;
    ρ3, ρ4, ρ5          : NP;
    ρ6                  : tV;
end
```

Encoding a Context-Free Grammar

Example (CFG)

$$\begin{aligned}
\rho_0 &= & S' &\rightarrow S \\
\rho_1 &= & S &\rightarrow NP\ VP \\
\rho_2 &= & VP &\rightarrow tV\ NP \\
\rho_3 &= & NP &\rightarrow \textit{John}
\end{aligned}
\qquad
\begin{aligned}
\rho_4 &= & NP &\rightarrow \textit{Mary} \\
\rho_5 &= & NP &\rightarrow \textit{everyone} \\
\rho_6 &= & tV &\rightarrow \textit{saw}
\end{aligned}$$

cfg.acg

```
signature Rules =
  S, S', NP, VP, tV   : type;
  ρ0                  : S → S';
  ρ1                  : NP → VP → S;
  ρ2                  : tV → NP → VP;
  ρ3, ρ4, ρ5          : NP;
  ρ6                  : tV;
end
```

```
lexicon Derive (Rules) : Trees =



end
```

Encoding a Context-Free Grammar

Example (CFG)

$\rho_0 = \mathsf{S}' \rightarrow \mathsf{S}$

$\rho_1 = \mathsf{S} \rightarrow \mathsf{NP}\ \mathsf{VP}$

$\rho_2 = \mathsf{VP} \rightarrow \mathsf{tV}\ \mathsf{NP}$

$\rho_3 = \mathsf{NP} \rightarrow \textit{John}$

$\rho_4 = \mathsf{NP} \rightarrow \textit{Mary}$

$\rho_5 = \mathsf{NP} \rightarrow \textit{everyone}$

$\rho_6 = \mathsf{tV} \rightarrow \textit{saw}$

cfg.acg

signature Rules =

- $\mathsf{S}, \mathsf{S}', \mathsf{NP}, \mathsf{VP}, \mathsf{tV}$: type;
- ρ_0 : $\mathsf{S} \rightarrow \mathsf{S}'$;
- ρ_1 : $\mathsf{NP} \rightarrow \mathsf{VP} \rightarrow \mathsf{S}$;
- ρ_2 : $\mathsf{tV} \rightarrow \mathsf{NP} \rightarrow \mathsf{VP}$;
- ρ_3, ρ_4, ρ_5 : NP;
- ρ_6 : tV;

end

lexicon Derive (Rules) : Trees =

- $\mathsf{S}, \mathsf{S}', \mathsf{NP}, \mathsf{VP}, \mathsf{tV} := \tau$;

end

Encoding a Context-Free Grammar

Example (CFG)

$$\begin{array}{lll} \rho_0 = & S' & \to S \\ \rho_1 = & S & \to NP\ VP \\ \rho_2 = & VP & \to tV\ NP \\ \rho_3 = & NP & \to \textit{John} \end{array} \qquad \begin{array}{lll} \rho_4 = & NP & \to \textit{Mary} \\ \rho_5 = & NP & \to \textit{everyone} \\ \rho_6 = & tV & \to \textit{saw} \end{array}$$

cfg.acg

$$\begin{array}{ll} \texttt{signature}\ \text{Rules} = & \\ \quad S, S', NP, VP, tV & : \texttt{type}; \\ \quad \rho_0 & : S \to S'; \\ \quad \rho_1 & : NP \to VP \to S; \\ \quad \rho_2 & : tV \to NP \to VP; \\ \quad \rho_3, \rho_4, \rho_5 & : NP; \\ \quad \rho_6 & : tV; \\ \texttt{end} & \end{array}$$

$$\begin{array}{ll} \texttt{lexicon}\ \text{Derive}\,(\text{Rules}) : \text{Trees} = & \\ \quad S, S', NP, VP, tV & := \tau; \\ \quad \rho_0 & := \lambda^{o} x.\, x; \\ \texttt{end} & \end{array}$$

Encoding a Context-Free Grammar

Example (CFG)

$\rho_0 = S' \rightarrow S$

$\rho_1 = S \rightarrow NP\ VP$

$\rho_2 = VP \rightarrow tV\ NP$

$\rho_3 = NP \rightarrow \textit{John}$

$\rho_4 = NP \rightarrow \textit{Mary}$

$\rho_5 = NP \rightarrow \textit{everyone}$

$\rho_6 = tV \rightarrow \textit{saw}$

cfg.acg

$\texttt{signature}$ Rules =

- S, S', NP, VP, tV : type;
- ρ_0 : $S \rightarrow S'$;
- ρ_1 : $NP \rightarrow VP \rightarrow S$;
- ρ_2 : $tV \rightarrow NP \rightarrow VP$;
- ρ_3, ρ_4, ρ_5 : NP;
- ρ_6 : tV;

end

$\texttt{lexicon}$ Derive (Rules) : Trees =

- S, S', NP, VP, tV $:= \tau$;
- ρ_0 $:= \lambda^o x. x$;
- ρ_1 $:= \lambda^o l\, r. S_2\, l\, r$;

end

Encoding a Context-Free Grammar

Example (CFG)

$$\rho_0 = S' \rightarrow S$$
$$\rho_1 = S \rightarrow NP\ VP$$
$$\rho_2 = VP \rightarrow tV\ NP$$
$$\rho_3 = NP \rightarrow \textit{John}$$
$$\rho_4 = NP \rightarrow \textit{Mary}$$
$$\rho_5 = NP \rightarrow \textit{everyone}$$
$$\rho_6 = tV \rightarrow \textit{saw}$$

cfg.acg

```
signature Rules =
    S, S′, NP, VP, tV    : type;
    ρ0                   : S → S′;
    ρ1                   : NP → VP → S;
    ρ2                   : tV → NP → VP;
    ρ3, ρ4, ρ5           : NP;
    ρ6                   : tV;
end
```

```
lexicon Derive (Rules) : Trees =
    S, S′, NP, VP, tV    := τ;
    ρ0                   := λ°x. x;
    ρ1                   := λ°l r. S2 l r;
    ρ2                   := λ°l r. VP2 l r;
end
```

Encoding a Context-Free Grammar

Example (CFG)

$\rho_0 = S' \rightarrow S$

$\rho_1 = S \rightarrow NP\ VP$

$\rho_2 = VP \rightarrow tV\ NP$

$\rho_3 = NP \rightarrow \textit{John}$

$\rho_4 = NP \rightarrow \textit{Mary}$

$\rho_5 = NP \rightarrow \textit{everyone}$

$\rho_6 = tV \rightarrow \textit{saw}$

cfg.acg

signature Rules =

- S, S', NP, VP, tV : type;
- ρ_0 : $S \rightarrow S'$;
- ρ_1 : $NP \rightarrow VP \rightarrow S$;
- ρ_2 : $tV \rightarrow NP \rightarrow VP$;
- ρ_3, ρ_4, ρ_5 : NP;
- ρ_6 : tV;

end

lexicon Derive (Rules) : Trees =

- S, S', NP, VP, tV := τ;
- ρ_0 := $\lambda^o x. x$;
- ρ_1 := $\lambda^o l\, r. S_2\, l\, r$;
- ρ_2 := $\lambda^o l\, r. VP_2\, l\, r$;
- ρ_3 := NP_1 John
- ρ_4 := NP_1 Mary
- ρ_5 := NP_1 everyone

end

Encoding a Context-Free Grammar

Example (CFG)

$\rho_0 = S' \rightarrow S$

$\rho_1 = S \rightarrow NP\ VP$

$\rho_2 = VP \rightarrow tV\ NP$

$\rho_3 = NP \rightarrow \textit{John}$

$\rho_4 = NP \rightarrow \textit{Mary}$

$\rho_5 = NP \rightarrow \textit{everyone}$

$\rho_6 = tV \rightarrow \textit{saw}$

cfg.acg

$\texttt{signature}$ Rules $=$

S, S', NP, VP, tV	$: \texttt{type};$
ρ_0	$: S \rightarrow S';$
ρ_1	$: NP \rightarrow VP \rightarrow S;$
ρ_2	$: tV \rightarrow NP \rightarrow VP;$
ρ_3, ρ_4, ρ_5	$: NP;$
ρ_6	$: tV;$

$\texttt{end}$

$\texttt{lexicon}$ Derive (Rules) : Trees $=$

S, S', NP, VP, tV	$:= \tau;$
ρ_0	$:= \lambda^o x. x;$
ρ_1	$:= \lambda^o l\, r. S_2\, l\, r;$
ρ_2	$:= \lambda^o l\, r. VP_2\, l\, r;$
ρ_3	$:= NP_1$ John
ρ_4	$:= NP_1$ Mary
ρ_5	$:= NP_1$ everyone
ρ_6	$:= tV_1$ saw

$\texttt{end}$

Encoding a Context-Free Grammar

Example (CFG)

$$\begin{array}{llll} \rho_0 = & S' & \to S & \\ \rho_1 = & S & \to NP\ VP & \\ \rho_2 = & VP & \to tV\ NP & \\ \rho_3 = & NP & \to \mathit{John} & \end{array} \qquad \begin{array}{lll} \rho_4 = & NP & \to \mathit{Mary} \\ \rho_5 = & NP & \to \mathit{everyone} \\ \rho_6 = & tV & \to \mathit{saw} \end{array}$$

cfg.acg

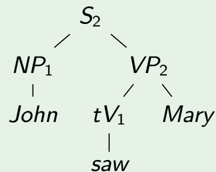
$$\begin{array}{ll} \texttt{signature}\ \text{Rules} = & \\ \quad S, S', NP, VP, tV & : \texttt{type}; \\ \quad \rho_0 & : S \to S'; \\ \quad \rho_1 & : NP \to VP \to S; \\ \quad \rho_2 & : tV \to NP \to VP; \\ \quad \rho_3, \rho_4, \rho_5 & : NP; \\ \quad \rho_6 & : tV; \\ \texttt{end} & \end{array}$$

$$\begin{array}{ll} \texttt{lexicon}\ \text{Derive}\,(\text{Rules}) : \text{Trees} = & \\ \quad S, S', NP, VP, tV & := \tau; \\ \quad \rho_0 & := \lambda^o x. x; \\ \quad \rho_1 & := \lambda^o l\, r. S_2\ l\ r; \\ \quad \rho_2 & := \lambda^o l\, r. VP_2\ l\ r; \\ \quad \rho_3 & := NP_1\ \text{John} \\ \quad \rho_4 & := NP_1\ \text{Mary} \\ \quad \rho_5 & := NP_1\ \text{everyone} \\ \quad \rho_6 & := tV_1\ \text{saw} \\ \texttt{end} & \\ \texttt{lexicon}\ \text{CFG} = \text{Yield} << \text{Derive} & \end{array}$$

Example (continued): Parsing Strings

Example

$$\rho_0 = \quad S' \rightarrow S$$
$$\rho_1 = \quad S \rightarrow NP\ VP$$
$$\rho_2 = \quad VP \rightarrow tV\ NP$$
$$\rho_3 = \quad NP \rightarrow \textit{John}$$
$$\rho_4 = \quad NP \rightarrow \textit{Mary}$$
$$\rho_5 = \quad NP \rightarrow \textit{everyone}$$
$$\rho_6 = \quad tV \rightarrow \textit{saw}$$



Example (continued): Parsing Strings

Example

$\rho_0 = \quad S' \rightarrow S$

$\rho_1 = \quad S \rightarrow NP\ VP$

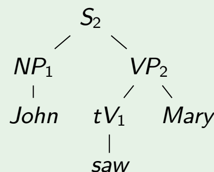
$\rho_2 = \quad VP \rightarrow tV\ NP$

$\rho_3 = \quad NP \rightarrow \mathit{John}$

$\rho_4 = \quad NP \rightarrow \mathit{Mary}$

$\rho_5 = \quad NP \rightarrow \mathit{everyone}$

$\rho_6 = \quad tV \rightarrow \mathit{saw}$



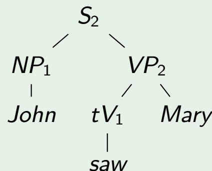
ACGtk>" λ^o z. John (saw (Mary z)) : o $\rightarrow$ o" | parse lexicon = CFG type = S

Example (continued): Parsing Strings

Example

$\rho_0 = \quad S' \rightarrow S$
$\rho_1 = \quad S \rightarrow NP\ VP$
$\rho_2 = \quad VP \rightarrow tV\ NP$
$\rho_3 = \quad NP \rightarrow \textit{John}$

$\rho_4 = \quad NP \rightarrow \textit{Mary}$
$\rho_5 = \quad NP \rightarrow \textit{everyone}$
$\rho_6 = \quad tV \rightarrow \textit{saw}$




```
ACGtk>"λ° z. John (saw (Mary z)) : o → o" | parse lexicon = CFG type = S
```

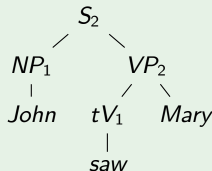
Parsing time: $75.1\mu s$
Parse forest building time: $3.96\mu s$
Term (depth = 3, size = 5): $\rho_1\ \rho_3\ (\rho_2\ \rho_6\ \rho_4) : S$
1 term computed

Example (continued): Parsing Strings

Example

$\rho_0 = \quad S' \rightarrow S$
$\rho_1 = \quad S \rightarrow NP\ VP$
$\rho_2 = \quad VP \rightarrow tV\ NP$
$\rho_3 = \quad NP \rightarrow \textit{John}$

$\rho_4 = \quad NP \rightarrow \textit{Mary}$
$\rho_5 = \quad NP \rightarrow \textit{everyone}$
$\rho_6 = \quad tV \rightarrow \textit{saw}$



`ACGtk>`"λ^o z. John (saw (Mary z)) : o → o" | `parse lexicon` = CFG `type` = S
Parsing time: 75.1μs
Parse forest building time: 3.96μs
Term (depth = 3, size = 5): $\rho_1\ \rho_3\ (\rho_2\ \rho_6\ \rho_4)$: S
1 term computed
`ACGtk>`"$\rho_1\ \rho_3\ (\rho_2\ \rho_6\ \rho_4)$: S | `realize lexicons` = Derive

Example (continued): Parsing Strings

Example

$\rho_0 = \quad S' \rightarrow S$

$\rho_1 = \quad S \rightarrow NP\ VP$

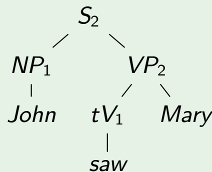
$\rho_2 = \quad VP \rightarrow tV\ NP$

$\rho_3 = \quad NP \rightarrow \textit{John}$

$\rho_4 = \quad NP \rightarrow \textit{Mary}$

$\rho_5 = \quad NP \rightarrow \textit{everyone}$

$\rho_6 = \quad tV \rightarrow \textit{saw}$



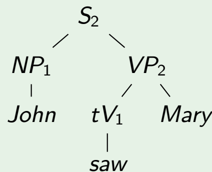
ACGtk>" λ^o z. John (saw (Mary z)) : o $\rightarrow$ o" | **parse** lexicon = CFG type = S
Parsing time: 75.1μs
Parse forest building time: 3.96μs
Term (depth = 3, size = 5): $\rho_1\ \rho_3\ (\rho_2\ \rho_6\ \rho_4)$: S
1 term computed
ACGtk>" $\rho_1\ \rho_3\ (\rho_2\ \rho_6\ \rho_4)$: S | **realize** lexicons = Derive
Term: $S_2\ (NP_1\ John)\ (VP_2\ (tV_1\ saw)\ (NP_1\ Mary))$: τ

Example (continued): Parsing Strings

Example

$\rho_0 = \quad S' \rightarrow S$
$\rho_1 = \quad S \rightarrow NP\ VP$
$\rho_2 = \quad VP \rightarrow tV\ NP$
$\rho_3 = \quad NP \rightarrow \textit{John}$

$\rho_4 = \quad NP \rightarrow \textit{Mary}$
$\rho_5 = \quad NP \rightarrow \textit{everyone}$
$\rho_6 = \quad tV \rightarrow \textit{saw}$



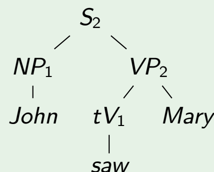
ACGtk>"λ^o z. John (saw (Mary z)) : o → o" | parse lexicon = CFG type = S
Parsing time: 75.1μs
Parse forest building time: 3.96μs
Term (depth = 3, size = 5): $\rho_1\ \rho_3\ (\rho_2\ \rho_6\ \rho_4)$: S
1 term computed
ACGtk>"$\rho_1\ \rho_3\ (\rho_2\ \rho_6\ \rho_4)$: S | realize lexicons = Derive
Term: S_2 (NP_1 John) (VP_2 (tV_1 saw) (NP_1 Mary)) : τ
ACGtk>"$\rho_1\ \rho_3\ (\rho_2\ \rho_6\ \rho_4)$: S | realize lexicons = Derive | realize lexicons = Yield

Example (continued): Parsing Strings

Example

$\rho_0 = \quad S' \rightarrow S$
$\rho_1 = \quad S \rightarrow NP\ VP$
$\rho_2 = \quad VP \rightarrow tV\ NP$
$\rho_3 = \quad NP \rightarrow \textit{John}$

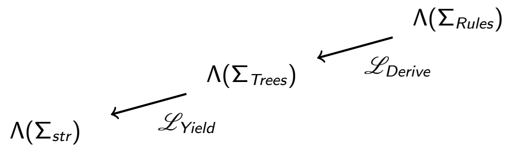
$\rho_4 = \quad NP \rightarrow \textit{Mary}$
$\rho_5 = \quad NP \rightarrow \textit{everyone}$
$\rho_6 = \quad tV \rightarrow \textit{saw}$



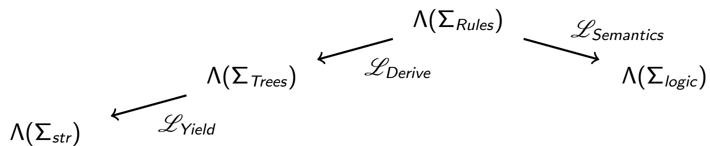
ACGtk>"λ^o z. John (saw (Mary z)) : o $\rightarrow$ o" | parse lexicon = CFG type = S
Parsing time: 75.1μs
Parse forest building time: 3.96μs
Term (depth = 3, size = 5): $\rho_1\ \rho_3\ (\rho_2\ \rho_6\ \rho_4)$: S
1 term computed
ACGtk>"$\rho_1\ \rho_3\ (\rho_2\ \rho_6\ \rho_4)$: S | realize lexicons = Derive
Term: S_2 (NP_1 John) (VP_2 (tV_1 saw) (NP_1 Mary)) : τ
ACGtk>"$\rho_1\ \rho_3\ (\rho_2\ \rho_6\ \rho_4)$: S | realize lexicons = Derive | realize lexicons = Yield
Term: λ^o z. John (saw (Mary z)) : o $\rightarrow$ o
1 term computed.

Transduction

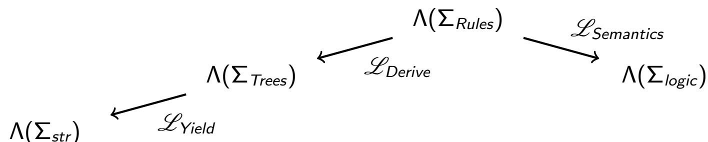
Transduction



Transduction



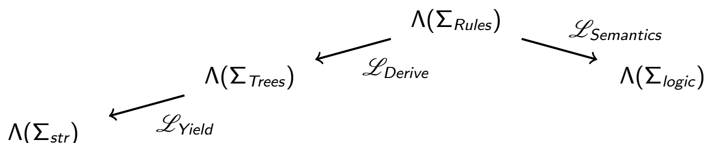
Transduction



cfg.acg

```
signature Logic =
  e, t   : type;                 j, m       : e;
  see    : t → e → t;            binder ∃   : (e ⇒ t) → t;
end
```

Transduction



cfg.acg

signature Logic =

e, t : type; j, m : e;

see : $t \to e \to t$; binder $\exists$: $(e \Rightarrow t) \to t$;

end

lexicon Semantics (Rules) : Logic =

$S' := t$; $\rho_0 := \lambda^o P.\, P\,(\lambda^o p.\, p)$; $\rho_3 := \lambda^o P.\, P\,\mathrm{j}$;

$S := (t \to t) \to t$; $\rho_4 := \lambda^o P.\, P\,\mathrm{m}$; $\rho_5 := \lambda^o P.\, \forall x.\, P\,x$;

$NP := (e \to t) \to t$; $\rho_6 := \lambda^o R.\, R\,\mathrm{saw}$;

$VP := ((e \to t) \to t) \to t$; $tV := ((e \to e \to t) \to t) \to t$;

$\rho_1 := \lambda^o s\,v.\, \lambda^o p.\, v\,(\lambda^o P.\, s\,(\lambda^o x.\, p\,(P\,x)))$;

$\rho_2 := \lambda^o v\,o.\, \lambda^o s.\, v\,(\lambda^o R.\, o\,(\lambda^o y.\, s\,(\lambda^o x.\, R\,x\,y)))$;

end

ACG Parsing and Reversibility

ACG Parsing and Reversibility

ACGtk>"ρ_1 ρ_3 (ρ_2 ρ_6 ρ_4) : S" | realize lexicons = CFG,Semantics

ACG Parsing and Reversibility

ACGtk>"$\rho_1\ \rho_3\ (\rho_2\ \rho_6\ \rho_4)$: S" | realize lexicons = CFG,Semantics
Term: λ^{o} z. John (saw (Mary z)) : o $\rightarrow$ o
Term: see j m : t
2 terms computed.

ACG Parsing and Reversibility

ACGtk>"ρ_1 ρ_3 (ρ_2 ρ_6 ρ_4) : S" | realize lexicons = CFG,Semantics
Term: λ^o z. John (saw (Mary z)) : o $\rightarrow$ o
Term: see j m : t
2 terms computed.
ACGtk>"John + saw + everyone : σ" | parse lexicon = CFG type = "S"'
| realize lexicons = Semantics

ACG Parsing and Reversibility

ACGtk>"$\rho_1\ \rho_3\ (\rho_2\ \rho_6\ \rho_4)$: S" | realize lexicons = CFG,Semantics
Term: λ^o z. John (saw (Mary z)) : o $\rightarrow$ o
Term: see j m : t
2 terms computed.
ACGtk>"John + saw + everyone : σ" | parse lexicon = CFG type = "S"'
| realize lexicons = Semantics
Parsing time: 65.2μs
Parse forest building time: 6.7μs
Term: $\forall$ x. see j x : t
1 term computed.

ACG Parsing and Reversibility

```
ACGtk>"ρ1 ρ3 (ρ2 ρ6 ρ4) : S" | realize lexicons = CFG,Semantics
Term: λ° z. John (saw (Mary z)) : o → o
Term: see j m : t
2 terms computed.
ACGtk>"John + saw + everyone : σ" | parse lexicon = CFG type = "S"
        | realize lexicons = Semantics
Parsing time: 65.2μs
Parse forest building time: 6.7μs
Term: ∀ x. see j x : t
1 term computed.
ACGtk>"∀ x. see j x : t" | parse lexicon = Semantics type = "S"
        | realize lexicons = CFG
```

ACG Parsing and Reversibility

```
ACGtk>"ρ1 ρ3 (ρ2 ρ6 ρ4) : S" | realize lexicons = CFG,Semantics
Term: λ° z. John (saw (Mary z)) : o → o
Term: see j m : t
2 terms computed.
ACGtk>"John + saw + everyone : σ" | parse lexicon = CFG type = "S"
        | realize lexicons = Semantics
Parsing time: 65.2μs
Parse forest building time: 6.7μs
Term: ∀ x. see j x : t
1 term computed.
ACGtk>"∀ x. see j x : t" | parse lexicon = Semantics type = "S"
        | realize lexicons = CFG
Parsing time: 93.4μs
Parse forest building time: 6.64μs
Term: λ° z. John (saw (everyone z)) : o → o
1 term computed.
```

ACG Parsing and Reduction to Datalog Querying

ACG Parsing and Reduction to Datalog Querying

Datalog Reduction (Kanazawa 2007)

ACG Parsing and Reduction to Datalog Querying

Datalog Reduction (Kanazawa 2007)

$\rho_1 \quad : \mathsf{NP} \rightarrow \mathsf{VP} \rightarrow \mathsf{S}$

ACG Parsing and Reduction to Datalog Querying

Datalog Reduction (Kanazawa 2007)

$$\begin{array}{lll} \rho_1 & : \mathsf{NP} \rightarrow \mathsf{VP} \rightarrow \mathsf{S} \\ M & : ((e \rightarrow t) \rightarrow t) \rightarrow (((e \rightarrow t) \rightarrow t) \rightarrow t) \rightarrow (t \rightarrow t) \rightarrow t \end{array}$$

(with $M = \lambda^{o} s\, v.\, \lambda^{o} p.\, v\, (\lambda^{o} P.\, s\, (\lambda^{o} x.\, p\, (P\, x))))$

ACG Parsing and Reduction to Datalog Querying

Datalog Reduction (Kanazawa 2007)

$$\rho_1 \quad : \mathsf{NP} \rightarrow \mathsf{VP} \rightarrow \mathsf{S}$$

$$M \quad : ((e \rightarrow t) \rightarrow t) \rightarrow (((e \rightarrow t) \rightarrow t) \rightarrow t) \rightarrow (t \rightarrow t) \rightarrow t$$

(with $M = \lambda^{o} s\, v.\, \lambda^{o} p.\, v\, (\lambda^{o} P.\, s\, (\lambda^{o} x.\, p\, (P\, x))))$

$$M \quad : ((\alpha_5 \rightarrow \alpha_7) \rightarrow \alpha_8) \rightarrow (((\alpha_5 \rightarrow \alpha_6) \rightarrow \alpha_8) \rightarrow \alpha_4) \rightarrow (\alpha_6 \rightarrow \alpha_7) \rightarrow \alpha_4$$

principal type of (the η-long form of) M

ACG Parsing and Reduction to Datalog Querying

Datalog Reduction (Kanazawa 2007)

$\rho_1 \quad : \mathsf{NP} \to \mathsf{VP} \to \mathsf{S}$

$M \quad : ((e \to t) \to t) \to (((e \to t) \to t) \to t) \to (t \to t) \to t$

(with $M = \lambda^{o} s\, v.\, \lambda^{o} p.\, v\, (\lambda^{o} P.\, s\, (\lambda^{o} x.\, p\, (P\, x)))$)

$M \quad : ((\alpha_5 \to \alpha_7) \to \alpha_8) \to (((\alpha_5 \to \alpha_6) \to \alpha_8) \to \alpha_4) \to (\alpha_6 \to \alpha_7) \to \alpha_4$

principal type of (the η-long form of) M

$\mathsf{S}(\quad) \leftarrow \mathsf{NP}(\quad), \mathsf{VP}(\quad)$

ACG Parsing and Reduction to Datalog Querying

Datalog Reduction (Kanazawa 2007)

$$\rho_1 \quad : \mathrm{NP} \quad \to \mathrm{VP} \quad \to \mathrm{S}$$

$$M \quad : ((e \to t) \to t) \quad \to (((e \to t) \to t) \to t) \quad \to (t \to t) \to t$$

(with $M = \lambda^o s\, v.\, \lambda^o p.\, v\, (\lambda^o P.\, s\, (\lambda^o x.\, p\, (P\, x))))$

$$M \quad : ((\alpha_5 \to \alpha_7) \to \alpha_8) \to (((\alpha_5 \to \alpha_6) \to \alpha_8) \to \alpha_4) \to (\alpha_6 \to \alpha_7) \to \alpha_4$$

principal type of (the η-long form of) M

$$\mathrm{S}(\alpha_6, \alpha_7, \alpha_4) \leftarrow \mathrm{NP}(\alpha_5, \alpha_7, \alpha_8), \mathrm{VP}(\alpha_5, \alpha_6, \alpha_8, \alpha_4)$$

ACG Parsing and Reduction to Datalog Querying

Datalog Reduction (Kanazawa 2007)

$$\begin{array}{lllll}\rho_1 & : \mathsf{NP} & \to \mathsf{VP} & \to \mathsf{S} \\ M & : ((e \to t) \to t) & \to (((e \to t) \to t) \to t) & \to (t \to t) \to t\end{array}$$

(with $M = \lambda^{o} s\, v.\, \lambda^{o} p.\, v\, (\lambda^{o} P.\, s\, (\lambda^{o} x.\, p\, (P\, x))))$

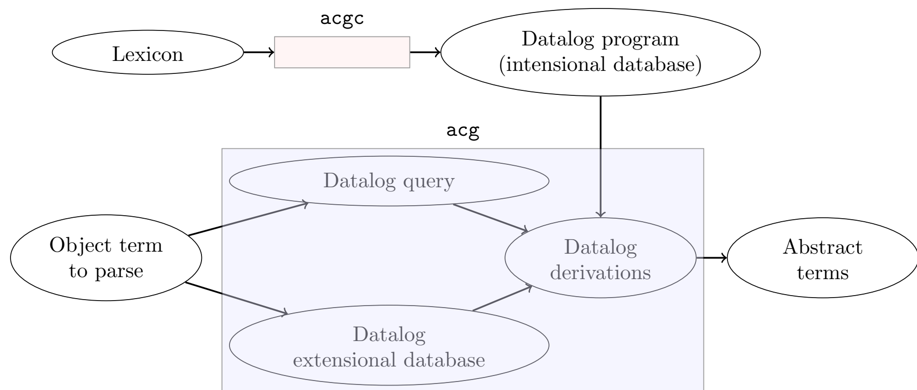
$$M \quad : ((\alpha_5 \to \alpha_7) \to \alpha_8) \to (((\alpha_5 \to \alpha_6) \to \alpha_8) \to \alpha_4) \to (\alpha_6 \to \alpha_7) \to \alpha_4$$

principal type of (the η-long form of) M

$$\mathsf{S}(\alpha_6, \alpha_7, \alpha_4) \leftarrow \mathsf{NP}(\alpha_5, \alpha_7, \alpha_8), \mathsf{VP}(\alpha_5, \alpha_6, \alpha_8, \alpha_4)$$

- Parsing algorithms and optimization techniques based on the well-established fields of database and logic programming
- General method for getting efficient tabular parsing algorithms

Overall Architecture



Selected Implementation Features

Selected Implementation Features

Datalog Evaluation

- Implementation of a Datalog prover
- Bottom-up semi-naive evaluation (Abiteboul, Hull, and Vianu 1995) and CYK-like chart parsing algorithm

Selected Implementation Features

Datalog Evaluation

- Implementation of a Datalog prover
- Bottom-up semi-naive evaluation (Abiteboul, Hull, and Vianu 1995) and CYK-like chart parsing algorithm

Shared Forests

- Parse structures extracted from charts stored as *shared forest*

Selected Implementation Features

Datalog Evaluation

- Implementation of a Datalog prover
- Bottom-up semi-naive evaluation (Abiteboul, Hull, and Vianu 1995) and CYK-like chart parsing algorithm

Shared Forests

- Parse structures extracted from charts stored as *shared forest*
  - Forest: to take ambiguity into account
  - Sharing: to keep track of memoization and account for possibly finite representation of an infinite set of solutions

Selected Implementation Features

Datalog Evaluation

- Implementation of a Datalog prover
- Bottom-up semi-naive evaluation (Abiteboul, Hull, and Vianu 1995) and CYK-like chart parsing algorithm

Shared Forests

- Parse structures extracted from charts stored as *shared forest*
  - Forest: to take ambiguity into account
  - Sharing: to keep track of memoization and account for possibly finite representation of an infinite set of solutions
- Ordered enumeration of solutions

Selected Implementation Features

Datalog Evaluation

- Implementation of a Datalog prover
- Bottom-up semi-naive evaluation (Abiteboul, Hull, and Vianu 1995) and CYK-like chart parsing algorithm

Shared Forests

- Parse structures extracted from charts stored as *shared forest*
  - Forest: to take ambiguity into account
  - Sharing: to keep track of memoization and account for possibly finite representation of an infinite set of solutions
- Ordered enumeration of solutions
  - Parametrized by a weighting scheme module
  - Current walk through the shared forest can be suspendend (and resumed later)
  - Applicative *forest context* data structure (similar to zippers, Huet 1997)

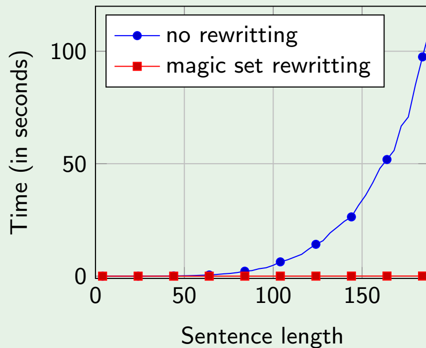
Selected Implementation Features (cont'd)

Magic Set Rewritting

Selected Implementation Features (cont'd)

Magic Set Rewritting

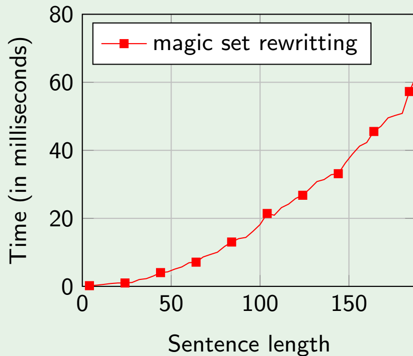
Example (Strings of the language $a^n b^n c^n d^n$)



Selected Implementation Features (cont'd)

Magic Set Rewritting

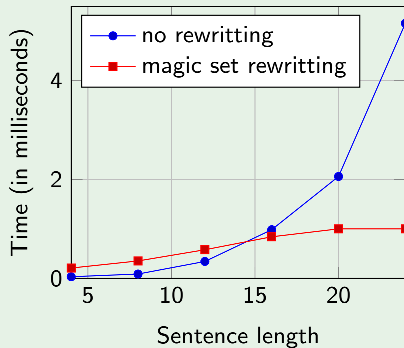
Example (Strings of the language $a^n b^n c^n d^n$)



Selected Implementation Features (cont'd)

Magic Set Rewritting

Example (Strings of the language $a^n b^n c^n d^n$)



Conclusion and Perspectives

ACGtk

- An environment to develop, test, and use ACGs
- Implements a Datalog prover and optimization related techniques
- Zipper-like data structure for shared forests

Conclusion and Perspectives

ACGtk

- An environment to develop, test, and use ACGs
- Implements a Datalog prover and optimization related techniques
- Zipper-like data structure for shared forests

Directions

- To integrate recent theoretical work on ACGs (mainly feature structures and weighted grammars).
- To improve the grammatical engineering facilities (for instance introducing functors).
- To use different optimization techniques

Bibliographie I

Abiteboul, Serge, Richard Hull, and Victor Vianu (1995). *Foundations of Databases*. Assison-Wesley. URL: http://webdam.inria.fr/Alice/pdfs/all.pdf.

de Groote, Philippe (July 2001). "Towards Abstract Categorial Grammars". In: *Proceedings of 39th Annual Meeting of the Association for Computational Linguistics*, pp. 148–155. DOI: 10.3115/1073012.1073045. ACL anthology: P01-1033.

Huet, Gérard (1997). "The Zipper". In: *Journal of Functional Programming* 7.5, pp. 549–554. DOI: 10.1017/S0956796897002864.

Kanazawa, Makoto (June 2007). "Parsing and Generation as Datalog Queries". In: *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL 2007)*. Prague, Czech Republic: Association for Computational Linguistics, pp. 176–183. ACL anthology: P07-1023.