# *Part 1:* Presentation of my **MSc. thesis**

# *Part 2:* **Bubble Trees**: A Dependency Grammar Approach to Coordination

Pierre Lison
pierrel@coli.uni-sb.de

Department of Computational Linguistics & Phonetics
Universität des Saarlandes

# Part I: Outline

**Part 1** - *Implementation of a Semantics-Syntax Interface based on Polarized Unification Grammars*:

# Part I: Outline

**Part 1** - *Implementation of a Semantics-Syntax Interface based on Polarized Unification Grammars*:

1. Introduction
2. Linguistic and Formal Foundations
   - Meaning-Text Theory
   - Meaning-Text Unification Grammars
   - Polarized Unification Grammars

# Part I: Outline (cont'd)

3. Axiomatization of MTUG/PUG
   - Constraint Programming
   - Extensible Dependency Grammar
   - Translation of MTUG/PUG into XDG

4. Implementation and Validation
   - Developed Prototype: auGUSTe
   - Experimental Validation and Demo

5. Conclusion

# Part I: Outline (cont'd)

3. Axiomatization of MTUG/PUG
   - Constraint Programming
   - Extensible Dependency Grammar
   - Translation of MTUG/PUG into XDG

4. Implementation and Validation
   - Developed Prototype: `auGUSTe`
   - Experimental Validation and Demo

5. Conclusion

# Part I: Outline (cont'd)

3. Axiomatization of MTUG/PUG
   - Constraint Programming
   - Extensible Dependency Grammar
   - Translation of MTUG/PUG into XDG
4. Implementation and Validation
   - Developed Prototype: `auGUSTe`
   - Experimental Validation and Demo
5. Conclusion

# Part II: Outline

**Part 2** - *Bubble Trees: A DG Approach to Coordination*:

6. Dependency Grammars and Coordination: a short overview
   - Essential ideas of Dependency Grammars
   - Modelling Coordination in DG: two possible directions
     - 1. "Coordination structures do have heads"
     - 2. "Coordination structures are not usual dependency structures"
7. Bubble trees: a new syntactic representation
   - Preliminary definitions
   - Definition of bubble trees
   - Perspectives on dependency and constituency

# Part II: Outline

**Part 2** - *Bubble Trees: A DG Approach to Coordination*:

6. Dependency Grammars and Coordination: a short overview
   - Essential ideas of Dependency Grammars
   - Modelling Coordination in DG: two possible directions
     - 1. “Coordination structures do have heads”
     - 2. “Coordination structures are not usual dependency structures”
7. Bubble trees: a new syntactic representation
   - Preliminary definitions
   - Definition of bubble trees
   - Perspectives on dependency and constituency

# Part II: Outline (cont'd)

8. Handling coordination phenomena
   - Coordination bubbles
   - Shared coordination
   - Gapping and valency slot coordination
   - Agreement and coordination of unlikes
   - Constraints between coordination and extraction
     - Projectivity of a Bubble Tree
     - Handling Ross's Coordinate Structure Constraint

9. Summary and Conclusion

# Part II: Outline (cont'd)

8. Handling coordination phenomena
   - Coordination bubbles
   - Shared coordination
   - Gapping and valency slot coordination
   - Agreement and coordination of unlikes
   - Constraints between coordination and extraction
     - Projectivity of a Bubble Tree
     - Handling Ross's Coordinate Structure Constraint

9. Summary and Conclusion

# Part 1

*Implementation of Semantic-Syntax Interface based on Polarized Unification Grammars*

# Introduction

## Background Information

- The title of my thesis is:

  *"Implementation of a semantics-syntax Interface based on Polarized Unification Grammars"*.

- It was realized last year for my MSc. thesis in Computer Science at the University of Louvain ;
- it draws heavily from the work of S. Kahane on :
  - *Meaning-Text Unification Grammars* [MTUG] [Kahane and Lareau 2005] ;
  - *Polarized Unification Grammars* [PUG] [Kahane 2002].

# Introduction
## Essential parts

- Our work can be divided in three basic parts:

1. The **axiomatization** of our initial formalism, MTUG/PUG, into a *Constraint Satisfaction Problem*, and more precisely into the XDG formalism :
2. The **implementation** of a semantics-syntax interface by means of a compiler from MTUG/PUG grammars to XDG grammars called auGUSTe as well as by the integration of 8 new “principles” (ie. constraints sets) into XDG ;
3. The **application** of our compiler to a small hand-crafted grammar centered on culinary vocabulary in order to experimentally validate our work.

# Introduction
## Essential parts

- Our work can be divided in three basic parts:

1. The **axiomatization** of our initial formalism, MTUG/PUG, into a *Constraint Satisfaction Problem*, and more precisely into the XDG formalism :
2. The **implementation** of a semantics-syntax interface by means of a compiler from MTUG/PUG grammars to XDG grammars called auGUSTe as well as by the integration of 8 new “principles” (ie. constraints sets) into XDG ;
3. The **application** of our compiler to a small hand-crafted grammar centered on culinary vocabulary in order to experimentally validate our work.

# Introduction

## Essential parts

- Our work can be divided in three basic parts:

1. The **axiomatization** of our initial formalism, MTUG/PUG, into a *Constraint Satisfaction Problem*, and more precisely into the XDG formalism :
2. The **implementation** of a semantics-syntax interface by means of a compiler from MTUG/PUG grammars to XDG grammars called auGUSTe as well as by the integration of 8 new “principles” (ie. constraints sets) into XDG ;
3. The **application** of our compiler to a small hand-crafted grammar centered on culinary vocabulary in order to experimentally validate our work.

# Introduction

## Essential parts

- Our work can be divided in three basic parts:

1. The **axiomatization** of our initial formalism, MTUG/PUG, into a *Constraint Satisfaction Problem*, and more precisely into the XDG formalism :
2. The **implementation** of a semantics-syntax interface by means of a compiler from MTUG/PUG grammars to XDG grammars called auGUSTe as well as by the integration of 8 new “principles” (ie. constraints sets) into XDG ;
3. The **application** of our compiler to a small hand-crafted grammar centered on culinary vocabulary in order to experimentally validate our work.

# Meaning-Text Theory

The **Meaning-Text Theory** is the principal (linguistic) inspiration behind the PUG formalism:

- Theory initiated in the sixties in the USSR ;
- Distinguish itself by its great linguistic richness and sophistication ;
- *Multistratal* formalism: distinct semantic, syntactic, morphological and phonological representations ;
- The kernel of the semantic representation is a **graph** of *predicate-argument relations*.
- The syntactic representation is a **dependency tree**.

# Meaning-Text Unification Grammars

The **Meaning-Text Unification Grammars** [Kahane 2002] [MTUG] is a new architecture for natural language modeling:

- An *articulated*, *mathematical* model of natural language ;
- Synthesis of various frameworks (HPSG, LFG, TAG, and of course MTT) ;
- Based on **unification** (ie. structure combination) ;
- Posit four representation levels:
  1. Semantic (graph)
  2. Syntactic (DG tree)
  3. Morpho-topological (ordered tree)
  4. Phonological (linear chain)
- A MTUG **grammar** $=$ *well-formedness rules* on each level $+$ *interface rules* between levels ;

# Polarized unification Grammars

## Basic ideas

**Polarized unification Grammars** are a general *descriptive linguistic formalism*.

- Initially developped within the framework of Meaning-Text Unification Grammars.
- Can *manipulate* different kinds of structures (graph, tree, phonological chain) and *bind* them.
- Control the *saturation* of the combined objects by the explicit assignment of a **polarity** to each of them.
- Most formalisms based on structure combination (ie. unification), like TAG, LFG, dependency grammars can be easily simulated by PUGs.

# Polarized unification Grammars

## Polarities

- We first define a finite set $P$ of polarities (here $P = ($◍ , ○ , ●$)$) ;
- A commutative and associative operator denoted "$\times$" ("product") is associated to this set ;
- Moreover, we define a subset $N$ of $P$ containing the *neutral* polarities.

| $\times$ | ◍ | ○ | ● |
|---|---|---|---|
| ◍ | ◍ | ○ | ● |
| ○ | ○ | ○ | ● |
| ● | ● | ● | $\perp$ |

Table: Polarities product

# Polarized unification Grammars

## Formal definition

**Definition**

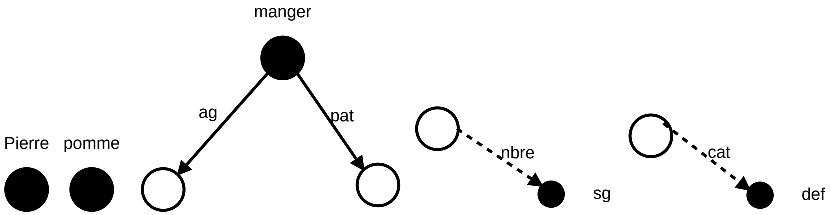
A **PUG grammar** is formally defined as a finite family $T$ of object types, a system $(P, \times)$ of polarities, a subset $N \in P$ of neutral polarities, and a finite set of elementary polarized structures, whose objects are specified by $T$ and where at least one is marqued as the initial structure.

**Definition**

The structures *generated* by this grammar are then defined as the neutral structures obtained by combining the initial structure and a finite number of elementary structures.

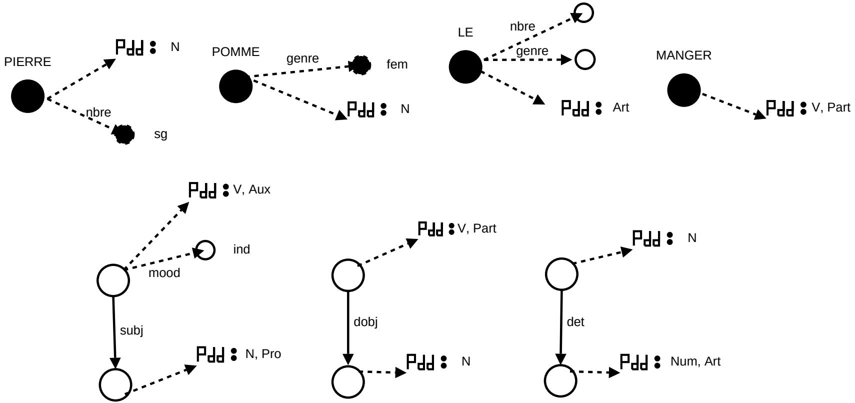
# Polarized unification Grammars

## Fragment of the semantic well-formedness grammar $\mathcal{G}_{sem}$



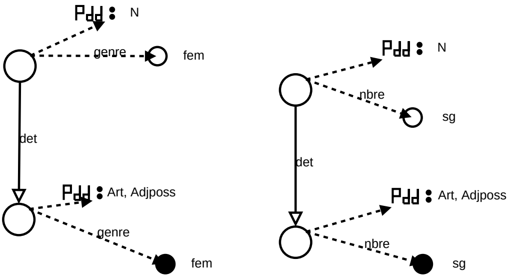
# Polarized unification Grammars

## Fragment of the semantic well-formedness grammar $\mathcal{G}_{synt}$



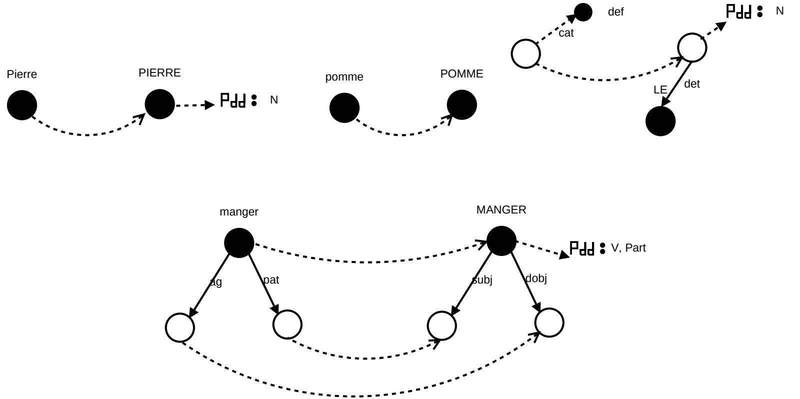
# Polarized unification Grammars

## Fragment of the semantic well-formedness grammar $\mathcal{G}_{sem}$ - con'd



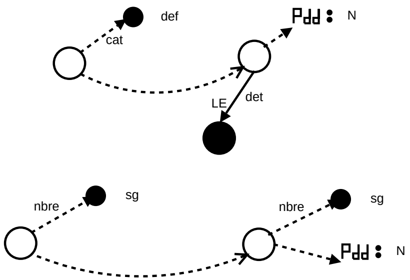
# Polarized unification Grammars

## Fragment of the interface grammar $\mathcal{I}_{sem-synt}$



# Polarized unification Grammars

## Fragment of the interface grammar $\mathcal{I}_{sem-synt}$ - cont'd



# Constraint Programming

## Basic ideas

We decided to ground the implementation of our interface on **constraint programming**:

- Very interesting computing paradigm for NLP (declarativity, monotonicity, parallelism, rather good efficiency) ;
- Analysis and generation are seen as the *enumeration* of the *well-formed models* according to the grammar ;
- Progressive *elimination* of interpretations which are not models of the grammar ;
- Two fundamental processes alternate:
  1. **propagation**: application of deterministic rules in order to reduce the search space ;
  2. **distribution**: non-deterministic choice.

# Extensible Dependency Grammar

## Short presentation

- XDG is a new **grammatical formalism**, developped by Ralph Debusmann in his Ph.D thesis [Debusmann 06] ;
- Formally defined as a *multigraph description* language ;
- Main features:
  1. Parallel architecture ;
  2. Use of Dependency Grammar ;
  3. Model-theoretic Syntax ;
  4. Based on Constraint Programming.
- Comes with a (very good) development platform and constraint solver: *XDG Development Kit* (XDK) ;

# Translation of MTUG/PUG into XDG

## Monotonicity

- A crucial property of the PUG is their **monotonicity**, that is, for a polarity system $P = \{ \bigcirc , ◐ , \bullet \}$ with the order $\bigcirc < ◐ < \bullet$ , we have:

$$\forall x, y \in P, x.y \geq max(x, y) \quad (1)$$

$\Rightarrow$ Corollary: the GUP rules can be applied in any order !

# Translation of MTUG/PUG into XDG

## Basic idea

- Baisc idea of our axiomatization: we require all the objects of each level are completely saturated, ie.
  - All the semantic objects must therefore have a polarity

$$(p_{\mathcal{G}_{sem}}, p_{\mathcal{I}_{sem-synt}}) = (\bullet, \bullet) \quad (2)$$

  - and all the syntactic objects must have a polarity

$$(p_{\mathcal{G}_{synt}}, p_{\mathcal{I}_{sem-synt}}) = (\bullet, \bullet) \quad (3)$$

$\Rightarrow$ this means **4 basic constraints** to enforce.

# Translation of MTUG/PUG into XDG

## Ways to operate the saturation

- In order to operate this saturation, a set of **rules** are specified, using particular feature structures:
  1. “Sagittal rules” (ie valency constraints) ;
  2. Agreement rules ;
  3. Interface rules.
- They can be either associed to specific lexical units, or to lexical classes ;
- They are only activated if the preconditions are met, and may require the satisfaction of additional constraints
- When several distinct saturated are possibled for the same object, we simply **distribute** on these possibilites.

# Developed Prototype: auGUSTe

Our implementation is composed of :

1. A **grammar compiler**, called auGUSTe, which translates PUG grammars into XDG ones
   - 17.000 lines of Python code;
   - Two input formats accepted: graphical (Dia file) or textual.

2. A set of 8 principles (constraints sets) integrated to the XDK.
   - Approximately 2.000 lines of Oz code ;
   - A substantial amount of time has been devoted to performance optimization, but with very mitigated success: average parsing time between 250 ms. and 10 s.

# Developed Prototype: auGUSTe

Our implementation is composed of :

1. A **grammar compiler**, called auGUSTe, which translates PUG grammars into XDG ones
   - 17.000 lines of Python code;
   - Two input formats accepted: graphical (Dia file) or textual.
2. A set of 8 principles (constraints sets) integrated to the XDK.
   - Approximately 2.000 lines of Oz code ;
   - A substantial amount of time has been devoted to performance optimization, but with very mitigated success: average parsing time between 250 ms. and 10 s.

# Developed Prototype: auGUSTe

Our implementation is composed of :

1. A **grammar compiler**, called auGUSTe, which translates PUG grammars into XDG ones
   - 17.000 lines of Python code;
   - Two input formats accepted: graphical (Dia file) or textual.
2. A set of 8 principles (constraints sets) integrated to the XDK.
   - Approximately 2.000 lines of Oz code ;
   - A substantial amount of time has been devoted to performance optimization, but with very mitigated success: average parsing time between 250 ms. and 10 s.

# Experimental validation

## Method

We used the following method to validate our implementation:

1. *Extract* a small grammar and lexicon (a few hundreds words) centered on culinary vocabulary;
2. *Create* (via our GUI) a MTUG/PUG grammar containing about 900 rules;
3. *Verify* the coherence and well-formedness of the grammar;
4. *Design* (by an external person) a test suite of 50 sentences;
5. *Encode* the semantic representation of these 50 sentences (resulting in 50 semantic graphs);
6. *Generate* the syntactic realization of these semantic graphs, and analyze results.

# Experimental validation

## Method

We used the following method to validate our implementation:

1. *Extract* a small grammar and lexicon (a few hundreds words) centered on culinary vocabulary;
2. *Create* (via our GUI) a MTUG/PUG grammar containing about 900 rules;
3. *Verify* the coherence and well-formedness of the grammar;
4. *Design* (by an external person) a test suite of 50 sentences;
5. *Encode* the semantic representation of these 50 sentences (resulting in 50 semantic graphs);
6. *Generate* the syntactic realization of these semantic graphs, and analyze results.

# Experimental validation

## Method

We used the following method to validate our implementation:

1. *Extract* a small grammar and lexicon (a few hundreds words) centered on culinary vocabulary;
2. *Create* (via our GUI) a MTUG/PUG grammar containing about 900 rules;
3. *Verify* the coherence and well-formedness of the grammar;
4. *Design* (by an external person) a test suite of 50 sentences;
5. *Encode* the semantic representation of these 50 sentences (resulting in 50 semantic graphs);
6. *Generate* the syntactic realization of these semantic graphs, and analyze results.

# Experimental validation

## Method

We used the following method to validate our implementation:

1. *Extract* a small grammar and lexicon (a few hundreds words) centered on culinary vocabulary;
2. *Create* (via our GUI) a MTUG/PUG grammar containing about 900 rules;
3. *Verify* the coherence and well-formedness of the grammar;
4. *Design* (by an external person) a test suite of 50 sentences;
5. *Encode* the semantic representation of these 50 sentences (resulting in 50 semantic graphs);
6. *Generate* the syntactic realization of these semantic graphs, and analyze results.

# Experimental validation

## Method

We used the following method to validate our implementation:

1. *Extract* a small grammar and lexicon (a few hundreds words) centered on culinary vocabulary;
2. *Create* (via our GUI) a MTUG/PUG grammar containing about 900 rules;
3. *Verify* the coherence and well-formedness of the grammar;
4. *Design* (by an external person) a test suite of 50 sentences;
5. *Encode* the semantic representation of these 50 sentences (resulting in 50 semantic graphs);
6. *Generate* the syntactic realization of these semantic graphs, and analyze results.

# Experimental validation

## Method

We used the following method to validate our implementation:

1. *Extract* a small grammar and lexicon (a few hundreds words) centered on culinary vocabulary;
2. *Create* (via our GUI) a MTUG/PUG grammar containing about 900 rules;
3. *Verify* the coherence and well-formedness of the grammar;
4. *Design* (by an external person) a test suite of 50 sentences;
5. *Encode* the semantic representation of these 50 sentences (resulting in 50 semantic graphs);
6. *Generate* the syntactic realization of these semantic graphs, and analyze results.

# Experimental validation

## Method

We used the following method to validate our implementation:

1. *Extract* a small grammar and lexicon (a few hundreds words) centered on culinary vocabulary;
2. *Create* (via our GUI) a MTUG/PUG grammar containing about 900 rules;
3. *Verify* the coherence and well-formedness of the grammar;
4. *Design* (by an external person) a test suite of 50 sentences;
5. *Encode* the semantic representation of these 50 sentences (resulting in 50 semantic graphs);
6. *Generate* the syntactic realization of these semantic graphs, and analyze results.

# Experimental validation

## Demo

# Conclusion

- Our semantics-syntax interface we developped is fully operational and has been validated with a grammar of about 900 PUG rules ;
- But it still suffers from big performance problems.
- Many rooms for improvement:
  1. Insertion of additional linguistic levels ;
  2. Finer-grained modelisation of linguistic phenomena ;
  3. Our translation PUG $\Rightarrow$ XDG lacks a real mathematical formalization ;
  4. And a lot of technical improvements (performance, robustness, ease-of-use).

# Conclusion

## One last comment about XDG

- Restricting the mapping of nodes between different levels to be 1:1 has been a major source of problems.
- The “trick” used in (Debusmann, 2004) for handling Multiword Expressions does not seem to scale up when bigger grammars are used.
- In my view, this restriction should really be lifted if XDG really wants to go beyond “toy grammars” and treat the full range of linguistic phenomena

# Part 2

*Bubble Trees: A Dependency Grammar Approach to Coordination*

# Preliminary Note

- My main sources for this work:

1. Section 1 of this talk was partly inspired by an ESSLLI course on dependency grammar by Denys Duchier and Geert-Jan Kruijff [Duchier 02], and by various other works (see bibliography for details).
2. Section 2 & 3 are essentially a summary of [Kahane 97], with a few personal additions.
3. The demo relies on the XDG Development Kit developped by Ralph Debusmann [Debusmann 06].

# Dependency Grammars (DG)

## Basic ideas

- Dependency Grammar is essentially based on **relationships between words** (instead of *groupings* - or *constituents* - as in phrase-structure trees)
- The dependency relation, noted $A \rightarrow B$, is defined as an *oriented* relation between two words, where:
  - The “source” word *A* is called the **head** or the governor ;
  - The “target” word *B* is called the **dependent** or governee.
- Dependency in language can be of different types: morphological, syntactic, semantic. In this talk, we will focus only on *syntactic* dependency.

# Dependency Grammars (DG)

## Nature of the dependency relation

- The theoretical characterization of the notion of syntactic *head* is a difficult question. [Zwicky 85] argues for the use of eight different criteria, like *subcategorization*, *morphosyntactic marking*, *concord*, etc.
- Moreover, the dependency relation must also satisfy several formal properties: *antisymetry*, *antireflexivity*, *antitransitivity*, *labelling* and *uniqueness*.

# Dependency Grammars (DG)

## Dependency structure

- The syntactic structure of a sentence thus consists of a set of pairwise relations among words.
- Depending on the chosen framework, this can lead either to a *graph* or a *tree* structure.
- In the general case, dependency structures don't directly provide a linear order (of the words in the sentence).

# Dependency Grammars (DG)

## Projectivity

- Linear order is taken into account by constraining the structure to satisfy some form of *projectivity*.
- Put simply, a dependency structure is said to be projective iff, $\forall$ words $A$ and $B$ where $A \rightarrow B$, all the words situated between $A$ and $B$ in the sentence are subordinated to $A$.
- The projectivity constraint must sometimes be substantially “relaxed” in order to handle phenomena like extraction or languages with free word order.

# Dependency Grammars (DG)

## Contemporary DG Frameworks

- Dependency is a very old concept in linguistics (8th century Arabic grammarians already used DG's core ideas).
- Modern notion of DG is usually attributed to Lucien Tesnière [Tesnière 59].
- DG comes nowadays in many different "flavors":
  - Functional Generative Description ("Prague School", [Sgall 86]) ;
  - Hudson's Word Grammar [Hudson 90] ;
  - Meaning-Text Theory [Mel'čuk 88] ;

# Modelling Coordination in DG

## The issue

- Coordination structures are usually hard to describe in terms of dependency.
- Indeed, Coordination is often described as an *orthogonal* (ie. “horizontal”) relation...
- ... whereas dependency constructions are best at formalizing *subordination* (ie. “vertical” relations).

# Modelling Coordination in DG
## Example

- Let's examine the two following examples:

  Maria and Hans went camping. (4)

  John stole and ate all the cookies. (5)

- *Question*: Where is the head ?
  1. One of the coordinate elements ? *No*: none has a higher priority than the other ;
  2. Both coordinate elements ? *No*: this would lead John in (2) to have two heads, which violates one formal property (uniqueness) of the dependency relation ;
  3. The and connective ? *No*: the connective in (1) cannot be the subject (eg. it would never be inflected).

# Modelling Coordination in DG
## Example

- Let's examine the two following examples:

Maria and Hans went camping. (4)

John stole and ate all the cookies. (5)

- *Question*: Where is the head ?
  1. One of the coordinate elements ? *No*: none has a higher priority than the other ;
  2. Both coordinate elements ? *No*: this would lead John in (2) to have two heads, which violates one formal property (uniqueness) of the dependency relation ;
  3. The and connective ? *No*: the connective in (1) cannot be the subject (eg. it would never be inflected).

# Modelling Coordination in DG
## Example

- Let's examine the two following examples:

  Maria and Hans went camping. (4)

  John stole and ate all the cookies. (5)

- *Question*: Where is the head ?
  1. One of the coordinate elements ? *No*: none has a higher priority than the other ;
  2. Both coordinate elements ? *No*: this would lead John in (2) to have two heads, which violates one formal property (uniqueness) of the dependency relation ;
  3. The and connective ? *No*: the connective in (1) cannot be the subject (eg. it would never be inflected).

# Modelling Coordination in DG
## Example

- Let's examine the two following examples:

Maria and Hans went camping. (4)

John stole and ate all the cookies. (5)

- *Question*: Where is the head ?
  1. One of the coordinate elements ? *No*: none has a higher priority than the other ;
  2. Both coordinate elements ? *No*: this would lead John in (2) to have two heads, which violates one formal property (uniqueness) of the dependency relation ;
  3. The and connective ? *No*: the connective in (1) cannot be the subject (eg. it would never be inflected).

# Modelling coordination in DG

## Possible solutions

- How can we address this difficult issue ?
- Two main directions have been explored so far:
    1. Preserve the initial framework by showing that *"coordination structures do have heads"*, and can therefore be modelled within DG without substantially altering the framework ;
    2. Or alternatively, argue that *"coordination structures are not usual dependency structures"* and thus need a particular treatment. In other words, the DG formalism will have to be extended to take some notions of constituency into account, leading to "hybrid" dependency/constituency formalisms.

# Modelling coordination in DG

## Possible solutions

- How can we address this difficult issue ?
- Two main directions have been explored so far:
  1. Preserve the initial framework by showing that *"coordination structures do have heads"*, and can therefore be modelled within DG without substantially altering the framework ;
  2. Or alternatively, argue that *"coordination structures are not usual dependency structures"* and thus need a particular treatment. In other words, the DG formalism will have to be extended to take some notions of constituency into account, leading to "hybrid" dependency/constituency formalisms.

# Modelling coordination in DG

## Possible solutions

- How can we address this difficult issue ?
- Two main directions have been explored so far:
  1. Preserve the initial framework by showing that *"coordination structures do have heads"*, and can therefore be modelled within DG without substantially altering the framework ;
  2. Or alternatively, argue that *"coordination structures are not usual dependency structures"* and thus need a particular treatment. In other words, the DG formalism will have to be extended to take some notions of constituency into account, leading to "hybrid" dependency/constituency formalisms.

# 1. “coordination structures do have heads”

## Some Evidence

- **First solution**: “coordination structures do have heads” , as argued in [Mel’čuk 88, Mel’čuk 98]:

  1. In the general case, the coordination structure is not symetrical:

     Hans slipped into his jacket and left. (6)

     ≠ Hans left and slipped into his jacket. (7)

  2. The right conjunct (connective included) is always omissible, while the left one is usually not:

     Hans, as well as Maria, came here ⇒ Hans came here. (8)

     ⇏ *As well as Maria came here. (9)

# 1. “coordination structures do have heads”

## Some Evidence

- **First solution**: “coordination structures do have heads” , as argued in [Mel’čuk 88, Mel’čuk 98]:
  1. In the general case, the coordination structure is not symetrical:

Hans slipped into his jacket and left. (6)

≠ Hans left and slipped into his jacket. (7)

  2. The right conjunct (connective included) is always omissible, while the left one is usually not:

Hans, as well as Maria, came here ⇒ Hans came here. (8)

⇏ *As well as Maria came here. (9)

# 1. “coordination structures do have heads”

## Some Evidence

- **First solution**: “coordination structures do have heads” , as argued in [Mel’čuk 88, Mel’čuk 98]:
  1. In the general case, the coordination structure is not symetrical:

     Hans slipped into his jacket and left. (6)

     ≠ Hans left and slipped into his jacket. (7)

  2. The right conjunct (connective included) is always omissible, while the left one is usually not:

Hans, as well as Maria, came here ⇒ Hans came here. (8)

⇏ *As well as Maria came here. (9)

# 1. “coordination structures do have heads”

## Mel’čuk’s approach

- For [Mel’čuk 88], the head of the coordination structure is always the **first conjoint**.
- This approach has one obvious advantage: it allows the coordinative construction to be analyzed in “pure” Dependency Grammar.
- But it also leads to various problems, notably for handling all types of “shared” constructions.

# 1. “coordination structures do have heads”

## Mel’čuk’s approach - Illustrative Example

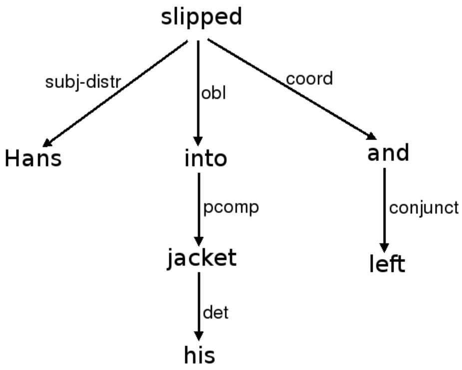



Figure: Analysis of sentence (3), in Mel’čuk’s approach

# 1. “coordination structures do have heads”

## Connectives as syntactic heads ?

- Alternatively, we could consider the *connective* as the syntactic head of the construction.
- But this is clearly not a viable solution:
  - How to characterize the “valency” of the connective ?
  - How to treat inflection and agreement ?
- More a semantic than a syntactic view (on the semantic level, connectives play the role of semantic operators).
- To my knowledge, no mainstream DG formalism still supports this approach.

# 1. “coordination structures do have heads”

## Connectives as syntactic heads ? Illustrative Example

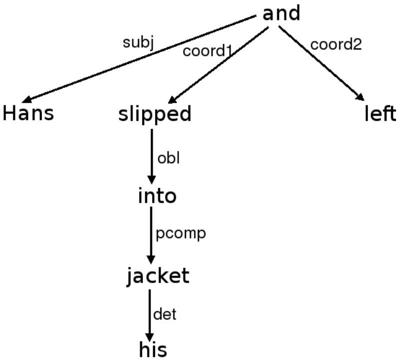



Figure: Analysis of sentence (4), w/ the connective as syntactic head

# 2. “Coordination structures are not usual dependency structures”

## Tesnière’s and FGD’s Approaches

- **Second solution**: Many other researchers argue that “pure” DG is intrinsically *insufficient* to account for all coordination phenomena, and that a radically different approach must be sought:
  1. [Tesnière 59, p.80-82] already distinguished dependency and coordinative relations with his concept of “junction” ;
  2. *Functional Generative Description* represents coordination by adding a new dimension to the tectogrammatical tree (by using special “bracketing”) [Žabokrtský 05]

# 2. "Coordination structures are not usual dependency structures"

## Tesnière's and FGD's Approaches

- **Second solution**: Many other researchers argue that "pure" DG is intrinsically *insufficient* to account for all coordination phenomena, and that a radically different approach must be sought:
  1. [Tesnière 59, p.80-82] already distinguished dependency and coordinative relations with his concept of "junction" ;
  2. *Functional Generative Description* represents coordination by adding a new dimension to the tectogrammatical tree (by using special "bracketing") [Žabokrtský 05]

# 2. “Coordination structures are not usual dependency structures”

## Tesnière’s and FGD’s Approaches

- **Second solution**: Many other researchers argue that “pure” DG is intrinsically *insufficient* to account for all coordination phenomena, and that a radically different approach must be sought:
  1. [Tesnière 59, p.80-82] already distinguished dependency and coordinative relations with his concept of “junction” ;
  2. *Functional Generative Description* represents coordination by adding a new dimension to the tectogrammatical tree (by using special “bracketing”) [Žabokrtský 05]

# 2. "Coordination structures are not usual dependency structures"

## Hudson's Approach

3. [Hudson 00] considers coordination as "a continuous string of words held together by principles other than dependency".

   The *Dependency in coordination Principle* states that

**Principle**

*"The conjuncts of a coordination must share the same dependencies to words outside the coordination"*

# 2. “Coordination structures are not usual dependency structures”

## Hudson’s Approach - Illustrative Example

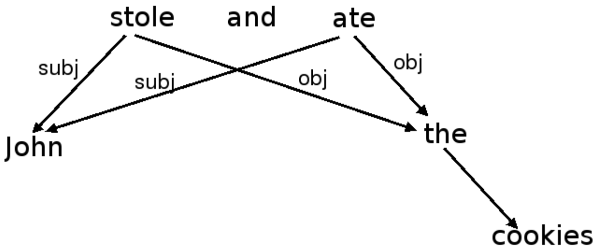



Figure: Analysis of sentence (2), with Hudson’s approach

# 2. “Coordination structures are not usual dependency structures”

## Introducing Bubble Trees

- Now that the general background of our talk is set, it’s time to get to the heart of the subject !
- We’ll now examine in more detail a new syntactic representation, **bubble trees**, which also belongs to this class of“hybrid” dependency-constituency models , and which, in our view, is particularly appropriate for the treatment of coordination (amongs others).
- Section 2 presents the mathematical structure and its formal properties, and Section 3 shows how it can be applied to the analysis of coordination phenomena.

# Preliminary definitions

## What is a tree, anyway?

### Definition

A **tree** can be viewed as:

- An oriented graph ;
- A binary relation $\triangleleft$, where $x \triangleleft y$ iff $(y, x)$ is a link in the corresponding graph, with $x$ and $y$ being 2 distinct nodes.

### Definition

Each tree induces a **dominance relation** $\preceq$ on node pairs, defined as follows: $x \preceq y$ iff $\exists x_1, x_2, ..., x_n$ such that $x = x_1 \triangleleft x_2 \triangleleft ... \triangleleft x_n = y$ $(n \geq 0)$.

# Preliminary definitions

## What is a dependency tree ?

- Let $X$ be an arbitrary set of lexical units.

**Definition**

A **dependency tree** on $X$ is simply a plain tree on $X$, defined by the couple $(X, \triangleleft)$

- In the example on the right, the tree is defined by the couple $(X, \triangleleft)$ , where

$$X = \{\text{Pierre}, \text{eats}, \text{noodles}\}$$
$$\triangleleft = \{(\text{eats}, \text{Pierre}, subj), (\text{eats}, \text{noodles}, dobj)\}$$

(NB: we added labelling of grammatical functions to the tree relations)

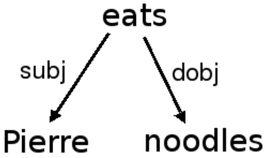



Figure: One dependency tree

# Preliminary definitions

## What is a constituency tree ?

### Definition

A **phrase-structure tree** on $X$ is a four-tuple $(X, \mathfrak{B}, \phi, \triangleleft)$, where $\mathfrak{B}$ is a set of constituents, $\triangleleft$ a tree relation defined on $\mathfrak{B}$, and $\phi$ a function (describing the "content" of the constituents) from $\mathfrak{B}$ to the non-empty subsets of $X$, so that the three following conditions are satisfied:

1. ($\mathbb{P}_1$) $\triangleleft$ is a tree relation ;
2. ($\mathbb{P}_2$) Every subset of $X$ containing only one element is the content of one and only one terminal node ;
3. ($\mathbb{P}_5$) If $\alpha \triangleleft \beta$, then $\phi(\alpha) \subseteq \phi(\beta)$.

# Preliminary definitions

## What is a constituency tree ? Illustrative example

- Don't panic ! Let's clarify this with an example:

- We specify our tree by the four-tuple $(X, \mathfrak{B}, \phi, \triangleleft)$, where:
  - $X = \{\text{Pierre}, \text{eats}, \text{noodles}\}$
  - $\mathfrak{B} = \{S, VP, NP_1, NP_2, V\}$
  - $\phi = \{ (S \rightarrow \{\text{Pierre}, \text{eats}, \text{noodles}\}), (NP_1 \rightarrow \{\text{Pierre}\}), (VP \rightarrow \{\text{eats}, \text{noodles}\}), (NP_2 \rightarrow \{\text{noodles}\}) \}$
  - $\triangleleft = \{(S, NP_1), (S, VP), (VP, V), (VP, NP_2)\}$

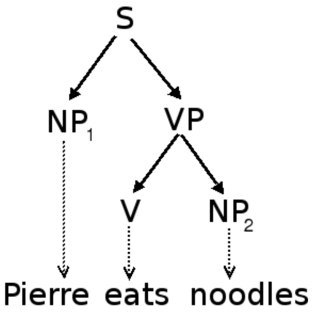



Figure: One constituency tree

# Definition of a bubble tree

## Basic idea

- Intuitively, a **bubble tree** is a tree whose nodes are *bubbles*. Each bubble can
  - Contain other bubbles or a lexical element ;
  - Form dependency relations with other bubbles.

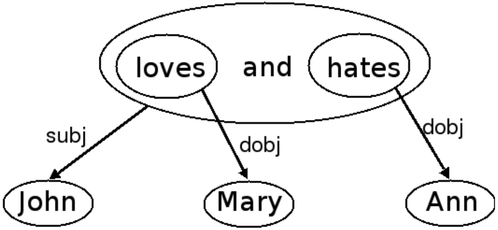



Figure: A bubble tree

# Definition of a bubble tree

## Formal definition

### Definition

A **bubble tree** is a four-tuple $(X, \mathfrak{B}, \phi, \triangleleft)$, where:

- $X$ is the set of lexical units ;
- $\mathfrak{B}$ is the set of bubbles ;
- $\phi$ is a map from $\mathfrak{B}$ to the non-empty subsets of $X$ (which describes the *content* of the bubbles) ;
- $\triangleleft$ is a relation on $\mathfrak{B}$ satisfying $\mathbb{P}_1$, $\mathbb{P}_2$, and moreover:
  1. ($\mathbb{P}_3$) If $\alpha, \beta \in \mathfrak{B}$, then $\phi(\alpha) \cap \phi(\beta) = \emptyset$
     **or** $\phi(\alpha) \subseteq \phi(\beta)$
     **or** $\phi(\beta) \subseteq \phi(\alpha)$
  2. ($\mathbb{P}_4$) If $\phi(\alpha) \subset \phi(\beta)$, then $\alpha \prec \beta$.
     If $\phi(\alpha) = \phi(\beta)$, then $\alpha \preceq \beta$ or $\alpha \preceq \beta$.

# Definition of a bubble tree

## Dependency-embedding relation

- The binary relation $\triangleleft$ is called the **dependency-embedding relation**, because it represents *both* the dependency relations between bubbles and the inclusion of bubbles in other bubbles (embedding).
- We can define two sub-relations of $\triangleleft$:
  1. The **dependency relation** $\lll$: $\alpha \lll \beta$ iff $\alpha \triangleleft \beta$ and $\phi(\alpha) \cap \phi(\beta) = \emptyset$.
  2. The **embedding relation** $\odot$: $\alpha \odot \beta$ iff $\alpha \triangleleft \beta$ and $\alpha \subseteq \beta$.
- If $\alpha \lll \beta$, we will say that $\alpha$ **depends** on $\beta$, and represent it graphically by an oriented arrow linking the two bubbles
- If $\alpha \odot \beta$, we will say that $\alpha$ is **included** in $\beta$, and represent it graphically by inserting $\alpha$ inside $\beta$'s bubble.

# Definition of a bubble tree

## Illustrative example

- The bubble tree is specified by the four-tuple $(X, \mathfrak{B}, \phi, \triangleleft)$:
  1. $X = \{\text{John}, \text{loves}, \text{Mary}, \text{hates}, \text{Ann}\}$
  2. $\mathfrak{B} = \{b_1, b_2, b_3, b_4, b_5, b_6\}$
  3. ...

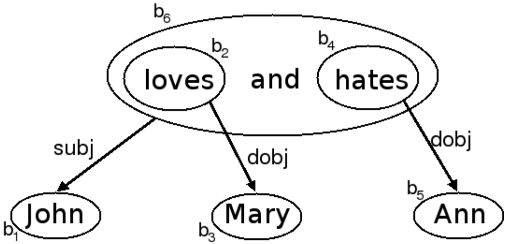



Figure: A bubble tree

# Definition of a bubble tree

## Illustrative example - cont'd

- The bubble tree is specified by the four-tuple $(X, \mathfrak{B}, \phi, \lhd)$:
  1. $X = \ldots$
  2. $\mathfrak{B} = \ldots$
  3. $\phi = \{(b_1 \rightarrow \{\text{John}\}), \quad (b_2 \rightarrow \{\text{loves}\}), \quad (b_3 \rightarrow \{\text{Mary}\}), \quad (b_4 \rightarrow \{\text{hates}\}), \quad (b_5 \rightarrow \{\text{Ann}\}), \quad (b_6 \rightarrow \{\text{loves, and, hates}\})$
  4. Concerning the $\lhd$ relation, we have:
     - As **dependency** relations: $b_1 \lll b_6$, $b_2 \lll b_3$, $b_5 \lll b_5$
     - As **embedding** relations: $b_2 \odot b_6$, $b_4 \odot b_6$

# Perspectives on dependency and constituency

## Dependency and constituency trees

- It is a well known result that any dependency tree $(X, \triangleleft_1)$ induces a constituency tree $(X, \mathfrak{B}, \phi, \triangleleft_2)$ [Gaifman 65].
- However, the reverse is not true in the general case. In order to “translate” a constituency tree into a dependency tree, we need to specify the **head(s)** of each constituent.
- By doing so, we end up with what is called a **co-headed constituency tree**, which is a very common mathematical structure in computational linguistics (LFG, HPSG, GB are notably based on them).
- A co-headed constituency tree induces a dependency tree, but the dependency relation is not explicit.

# Perspectives on dependency and constituency

## Relevance of bubble trees

- Interestingly, it can be shown that a co-headed constituency tree is also a *particular case* of a bubble tree, where every bubble contains a unique element (namely the head of the constituent).
- Bubble trees are therefore a very valuable tool to compare different syntactic models.
- **Moral of the story**: DG and PS models are much closer than they appear at first sight, and mathematical formalization can help create a common language between them, and foster “cross-fertilization” of ideas!

# Coordination bubbles

## Basic idea

- Put simply, coordination boils down to the fact that two or more elements together occupy one syntactic position. [Bloomfield 33]
- We'll group these elements in a bubble, called a **coordination bubble**, which occupies this position.
- The coordination bubble contains two types of elements :
  1. The coordinated elements ;
  2. The coordinating conjunctions (connectives).

# Coordination bubbles

## Iterativity of coordination

- The coordination bubble can be expanded in two ways:
  1. **Iterativity** of coordination: a theoretically illimited number of elements can be coordinated.

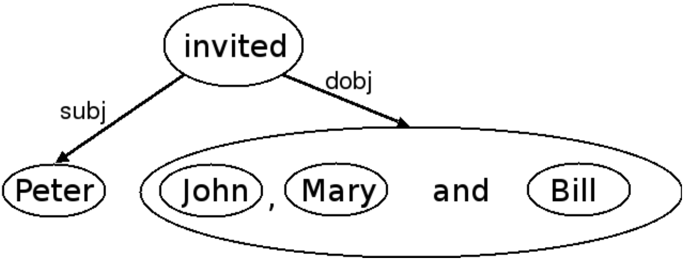



Figure: Iterativity of coordination

# Coordination bubbles

## Recursivity of coordination

3. **Recursivity** of coordination: coordination bubbles can be themselves coordinated.

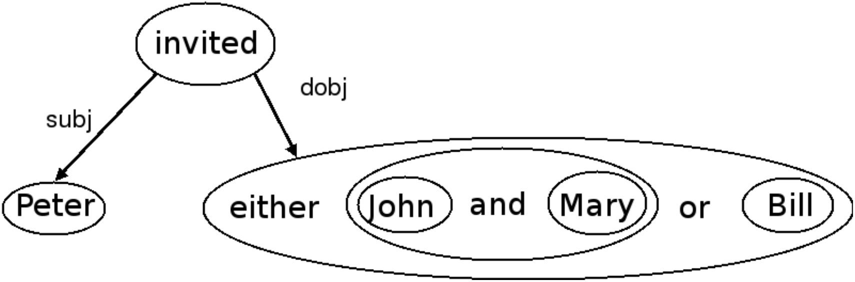



Figure: Recursivity of coordination

# Shared coordination
## Principle

- Coordinated elements *must* necessarily share their governor (if there is one).
- And they *can* share all or parts of their dependents.

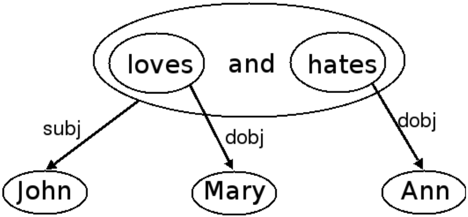



Figure: Bubble tree with a shared coordination

# Shared coordination

## Example 1: lexical coordination

- Several dependents can be shared, as detailed below
- Note this particular case is called a lexical coordination, and must obey to special constraints [Abeillé 05]

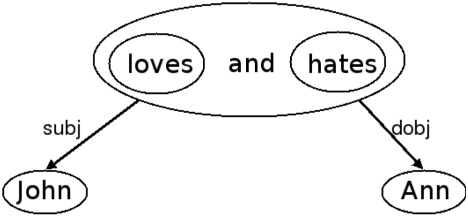



Figure: Bubble tree with two shared coordinations

# Shared coordination

## Example 2: Right Node Raising

- Our formalism can also easily account for Right Node Raising phenomena.

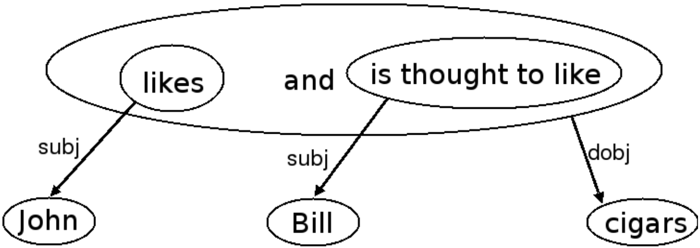



Figure: Right Node Raising

# Shared coordination

## Example 2: Right Node Raising - cont'd

Note:

- "is thought to like" is called a **verbal nucleus**, ie. a verb or a complex unit such as:
  - Auxiliary-participle ("have read"),
  - Verb-infinitive ("wants to read"),
  - Verb-conjunction-verb ("think that read"),
  - Verb-preposition ("look for"),
  - and all constructions built by transitivity from these.
- See [Gerdes 06] for details (in French).

# Shared coordination
## Valency frame

- The lexicon provides us with information about the **valency** (subcategorization) frame of each word.
- How to use this information in bubble trees ? In other words, how to *constrain* the representation such that only dependency relations explicitly licensed by the grammar/lexicon are allowed ?

**Principle**

*The valency of any coordinated element is the* **union** *of the valency of every coordination bubble containing it.*

# Shared coordination

## Valency frame - formal definition

- Formally (recursive definition) :

**Definition**

Let $\alpha$ be a bubble part of the bubble tree $(X, \mathfrak{B}, \phi, \triangleleft)$. We define the **valency** $\upsilon$ of $\alpha$ as the union of

- the set of bubbles that directly depends on $\alpha$ ;
- the union of the valency of every bubble that includes $\alpha$.

In other words:

$$\upsilon(\alpha) = \{\beta \in \mathfrak{B} : \beta \lll \alpha\} \cup \left( \bigcup_{\forall \gamma:\ \alpha \odot \gamma} \upsilon(\gamma) \right)$$

# Gapping and valency slot coordination

## Gapping coordination

- **Gapping**: If two clauses with the same main verb are coordinated, the second occurrence of the verb can be omitted (= ellipsis).

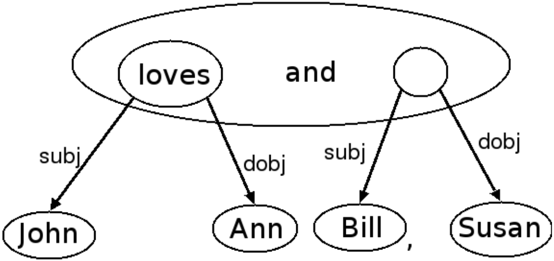



Figure: Gapping coordination

# Gapping and valency slot coordination

## Valency slot coordination (≈ Conjunction Reduction)

- We define a **valency slot bubble** as a *subset* of the valency of a governing element grouped in a bubble.
- Two valency slot bubbles can be coordinated iff they are of the same kind.

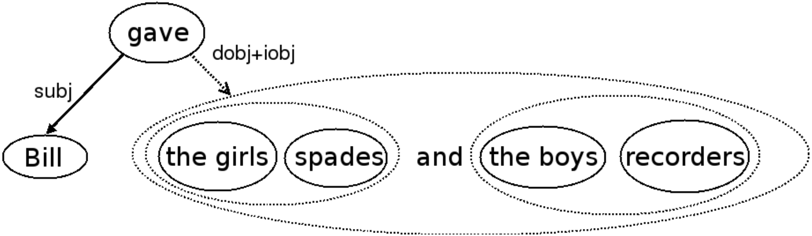



Figure: Valency slot coordination

# Gapping and valency slot coordination

## Similar or different phenomena ?

- Do gapping and CR coordination refer to the same phenomenon ?
  - *Pro:* They are formally very close (valency slot can be easily represented as gapping).
  - *Cons:* As [Crysmann 06] rightly points out, gapping is similar in many respect to true ellipsis (and hence to a semantic/pragmatic phenomenon), while CR essentially remains on syntactic grounds.
- Note that the constraint “of the same kind” in our definition of valency slot coordination is quite vague, and should be more clearly specified.

# Agreement and coordination of unlikes

## How to handle (basic) agreement ?

- As in most formalisms, a **feature structure** is associated to each element (bubble, word).
- In order to handle agreement, we have to constrain these feature structures. Let $\beta$ be a bubble containing two coordinated elements , $el_1$ and $el_2$. We would then have to enforce a set of constraints like:
  - $case(\alpha) = case(el_1) = case(el_2)$ [1]
  - $number(\alpha) = number(el_2) + number(el_2)$ [2]
  - $gender(\alpha) = min(gender(el_2) + gender(el_2))$
  - ...

[1] only for constituent coordination
[2] for coordination with the "and" connective

# Agreement and coordination of unlikes

## How to handle coordination of unlikes ? (personal attempt)

- To handle coordination of unlikes, I propose to define a feature similar to the HEAD feature in HPSG, where the part-of-speech information would be encoded, and constrain its value for a given bubble to be the **intersection** of the values in the coordinated elements.
- Formally: $cat(\alpha) = cat(el_1) \cap cat(el_2)$
- We would then be able to analyse a sentence such as

  John is a republican and proud of it (10)

  as long as the noun and the adjective share a positive value for the PRD feature, as required by the copula.

# Constraints between coordination and extraction

## Projectivity of a bubble tree - Reminder

- In order to explain how bubble trees handle the constraints between coordination and extraction, I'll first give some explanations about the **projectivity** of bubble trees.
- Recall what we said in the first part of this lecture about the projectivity of a dependency tree:

**Principle**

*"A dependency structure is said to be projective iff, $\forall$ words A and B where $A \rightarrow B$, all the words situated between A and B in the sentence are subordinated to A."*

- Ensuring the projectivity of bubble tree is not much more complicated !

# Constraints between coordination and extraction

## Projectivity of a bubble tree - Definition 1

- Informal definition:

**Definition**

A linearly ordered bubble tree is said to be **projective** iff

1. bubblinks do no cross each other and,
2. no bubblink covers an ancestor or a co-head

(where a **bubblink** is either a bubble or a link)

- Ensuring projectivity is thus a matter of verifying simple geometric properties !

# Constraints between coordination and extraction

## Projectivity of a bubble tree - Definition 2

- Or more formally (personal attempt):

**Definition**

Suppose we have

1. A bubble tree $(X, \mathfrak{B}, \phi, \triangleleft)$,
2. A linear order $<$ on $X$
3. An (arbitrary) relation (either dependency or embedding) between two bubbles $x$ and $y$ (with $x$ being the head), noted $\overrightarrow{xy}$.

# Constraints between coordination and extraction

## Projectivity of a bubble tree - Definition 2 (con'td)

### Definition (cont'd)

1. We now define the **support** of $\overrightarrow{xy}$, noted $Supp(\overrightarrow{xy})$ as the set of bubbles situated between the extremities of $\overrightarrow{xy}$.

   More precisely, we have $Supp(\overrightarrow{xy}) = \{\beta \in \mathfrak{B} : x < \beta \leq y\}$.

2. We say that the relation $\overrightarrow{xy}$ is **projective** iff, for every bubble $\beta$ in $Supp(\overrightarrow{xy})$, we have $\beta \preceq x$.

3. Finally, we define a **projective tree** as a tree for which every relation is projective.

# Constraints between coordination and extraction

## Projectivity of a bubble tree - Definition 2 (con'td)

### Definition (cont'd)

1. We now define the **support** of $\overrightarrow{xy}$, noted $Supp(\overrightarrow{xy})$ as the set of bubbles situated between the extremities of $\overrightarrow{xy}$.

   More precisely, we have $Supp(\overrightarrow{xy}) = \{\beta \in \mathfrak{B} : x < \beta \leq y\}$.

2. We say that the relation $\overrightarrow{xy}$ is **projective** iff, for every bubble $\beta$ in $Supp(\overrightarrow{xy})$, we have $\beta \preceq x$.

3. Finally, we define a **projective tree** as a tree for which every relation is projective.

# Constraints between coordination and extraction

## Projectivity of a bubble tree - Definition 2 (con'td)

**Definition (cont'd)**

1. We now define the **support** of $\overrightarrow{xy}$, noted $Supp(\overrightarrow{xy})$ as the set of bubbles situated between the extremities of $\overrightarrow{xy}$.

   More precisely, we have $Supp(\overrightarrow{xy}) = \{\beta \in \mathfrak{B} : x < \beta \leq y\}$.
2. We say that the relation $\overrightarrow{xy}$ is **projective** iff, for every bubble $\beta$ in $Supp(\overrightarrow{xy})$, we have $\beta \preceq x$.
3. Finally, we define a **projective tree** as a tree for which every relation is projective.

# Constraints between coordination and extraction

## Principle

- Recall [Ross 67]'s Coordinate Structure Constraint:

**Principle**

*In a coordinate structure:*

- *no conjunct can be moved*
- *nor may any element contained in a conjunct be moved out of the conjunct"*

- The nice thing with bubble trees is that we don't have to specify any special constraint to rule out these "movements", they are blocked by simple and visual *geometrical* properties !

# Constraints between coordination and extraction

## Example 1

- Let's examine the ungrammatical example below
- The structure is *not* licensed because we have an arc from "a student" to "whose mother" that crosses the large bubble embedding the coordination.

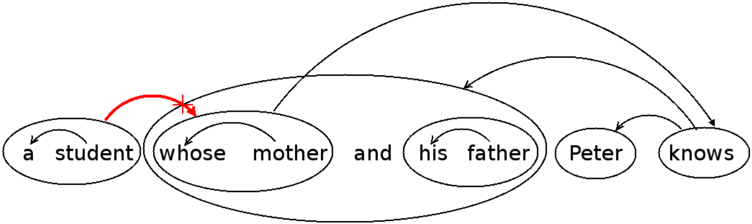



Figure: Ungrammatical sentence (unsatisfied CSC)

# Constraints between coordination and extraction
## Example 2

- On the contrary, this example is perfectly grammatical[3]
- The structure is licenced because all the bubble relations are projective.

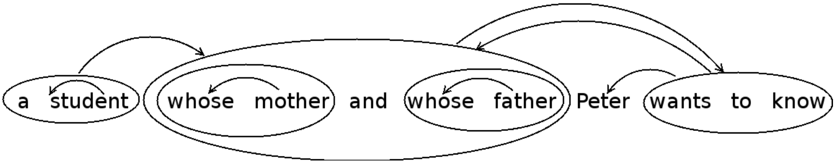



Figure: grammatical sentence

[3] Even if the sentence sounds a bit weird!

# Summary

In this talk we discussed a new syntactic representation for the treatment of coordination, namely **bubble trees**.

1. We first analyzed how various Dependency Grammars frameworks handled coordination, and we pointed out that some researchers made a point of preserving the initial dependency model, while others emphasized its intrinsic insufficiency and proposed more expressive formalisms.

2. We then presented a new syntactic representation, the bubble tree, which integrates information from dependency *and* constituency in a single, coherent framework.

# Summary

In this talk we discussed a new syntactic representation for the treatment of coordination, namely **bubble trees**.

1. We first analyzed how various Dependency Grammars frameworks handled coordination, and we pointed out that some researchers made a point of preserving the initial dependency model, while others emphasized its intrinsic insufficiency and proposed more expressive formalisms.
2. We then presented a new syntactic representation, the bubble tree, which integrates information from dependency *and* constituency in a single, coherent framework.

# Summary

In this talk we discussed a new syntactic representation for the treatment of coordination, namely **bubble trees**.

1. We first analyzed how various Dependency Grammars frameworks handled coordination, and we pointed out that some researchers made a point of preserving the initial dependency model, while others emphasized its intrinsic insufficiency and proposed more expressive formalisms.
2. We then presented a new syntactic representation, the bubble tree, which integrates information from dependency *and* constituency in a single, coherent framework.

# Summary

3. Finally, examined how the bubble trees were precisely handling various coordinations phenomenas like shared coordination, gapping, agreement, and the constraints on extraction.

# Conclusion

- Bubble trees seem to be a very promising mathematical framework for modelling difficult linguistic phenomena like *coordination* (as we have seen), but also others like *extraction* and *modification of groupings*.
- A lot of work remains to be done to characterize precisely how a “bubble grammar” would operate.
- Moreover, there are a lot of interesting questions concerning the potential use of such formalisms in existing frameworks like TAG, LFG, HPSG, and CCG.
- **Thanks for your attention ! Questions ?**

# Aknowledgements

- Many thanks to Berthold Crysmann, Ralph Debusmann and Sylvain Kahane for their help and advice.
- Section 1 of this talk was mainly inspired by [Duchier 02]. Section 2 & 3 are essentially a summary of [Kahane 97], with a few personal additions.
- Blame me for any remaining errors.

# Bibliography I


- Anne Abeillé.
  *In defense of lexical coordination*.
  In CSSP Proceedings, Paris, 2005.
- Patrick Blackburn & Maarten de Rijke.
  *Zooming in, zooming out*.
  Journal of Logic, Language and Information, 6:5-31, 1997.
- Leonard Bloomfield.
  Language.
  George Allen and Unwin, London, 1933.
  Reprinted in 1961, Holt, Reinhart, and Winston.

# Bibliography II


- Berthold Crysmann.
  Coordination.
  Elsevier, 2nd edition, 2006.
- Ralph Debusmann.
  *Extensible Dependency Grammar: A Modular Grammar Formalism Based On Multigraph Description*.
  PhD thesis, Saarland University, 2006.
- Denys Duchier & Geert-Jan Kruijff.
  *Formal and computational aspects of dependency grammar*.
  ESSLLI 2002, Trento Italy, 2002.

# Bibliography III


- H. Gaifman.
  *Dependency Systems and Phrase-Structure Systems*.
  Information and Control, vol. 8, no. 3, pages 304–37, 1965.
- Kim Gerdes & Sylvain Kahane.
  *L'amas verbal au coeur d'une modélisation topologique du français*.
  In Kim Gerdes & C. Müller, editeurs, Ordre des mots et topologie de la phrase française, volume 29, 2006.
- Sylvain Kahane & François Lareau.
  *Grammaires d'Unification Sens-Texte: Modularité et Polarisation*.
  In Actes TALN (Dourdan, 2005), p. 23-32.

# Bibliography IV


- Sylvain Kahane.
  *Grammaires d'Unification Sens-Texte: vers un modèle mathématique articulé de la langue*.
  Habilitation à diriger des recherches, Université Paris 7, 2002.

- Aleksej Gladkij.
  *On describing the syntactic structure structure of a sentence*.
  Computational Linguistics, vol. 7, pages 21–44, 1968.
  (in Russian with English summary).

# Bibliography V


R. Hudson.
English word grammar.
Basil Blackwell, Oxford, 1990.

Richard A. Hudson.
Dependency grammar.
Course notes from ESSLLI2000, Birmingham, 2000.

S. Kahane.
*Bubble trees and syntactic representations*.
In Proceedings of the 5th Meeting of the Mathematics of the Language, Saarbrücken, 1997.

# Bibliography VI


- Sylvain Kahane.
  Une grammaire de dépendance lexicalisée avec bulles pour traiter les coordinations.
  2000.
- Igor Mel'čuk.
  Dependency syntax: Theory and practice.
  State University of New York Press, Albany, NY, 1988.
- Igor Mel'čuk.
  *Dependency in Linguistic Description*.
  1998.
  (unpublished).

# Bibliography VII


John Robert Ross.
*Constraints on Variables in Syntax*.
PhD dissertation, MIT, Cambridge, MA, 1967.

P. Sgall, E. Hajičová & J. Panevová.
The meaning of the sentence and its semantic and pragmatic aspects.
Academia/Reidel Publishing Company, Prague, Czech Republic/Dordrecht, Netherlands, 1986.

Lucien Tesnière.
Eléments de syntaxe structurale.
Kincksieck, 1959.

# Bibliography VIII

Zdeněk Žabokrtský.
*Resemblances between Meaning-Text Theory and Functional Generative Description*.
In In proceedings of 2nd International Conference of Meaning-Text Theory, Moscow, 2005.

Arnold Zwicky.
*Heads*.
Journal of Linguistics, vol. 21, pages 1–30, 1985.