# Querying Linguistic Databases

Steven Bird

Department of Computer Science
University of Melbourne, AUSTRALIA

Linguistic Data Consortium
University of Pennsylvania, USA

International Conference on
Linguistic Evidence

# Linguistic Description

- require large databases of annotated text and speech
- language documentation, linguistic analysis
- tasks: collection, curation, annotation, analysis
- size: $10^3$ – $10^6$ words
- annotations: phonetics, prosody, orthography, syntax, dialog, and gesture

# Linguistic Description

- require large databases of annotated text and speech
- language documentation, linguistic analysis
- tasks: collection, curation, annotation, analysis
- size: $10^3$ – $10^6$ words
- annotations: phonetics, prosody, orthography, syntax, dialog, and gesture

# Linguistic Description

- require large databases of annotated text and speech
- language documentation, linguistic analysis
- tasks: collection, curation, annotation, analysis
- size: $10^3$ – $10^6$ words
- annotations: phonetics, prosody, orthography, syntax, dialog, and gesture

# Linguistic Description

- require large databases of annotated text and speech
- language documentation, linguistic analysis
- tasks: collection, curation, annotation, analysis
- size: $10^3 - 10^6$ words
- annotations: phonetics, prosody, orthography, syntax, dialog, and gesture

# Linguistic Description

- require large databases of annotated text and speech
- language documentation, linguistic analysis
- tasks: collection, curation, annotation, analysis
- size: $10^3 - 10^6$ words
- annotations: phonetics, prosody, orthography, syntax, dialog, and gesture

# Language Technology

- require large databases of annotated text and speech
- QA, MT, TDT, SLDS, TIDES, ...
- tasks: collection, curation, annotation, analysis
- size: $10^6$ – $10^9$ words
- annotations: phonetics, prosody, orthography, syntax, dialog, and gesture

# Language Technology

- require large databases of annotated text and speech
- QA, MT, TDT, SLDS, TIDES, ...
- tasks: collection, curation, annotation, analysis
- size: $10^6 - 10^9$ words
- annotations: phonetics, prosody, orthography, syntax, dialog, and gesture

# Language Technology

- require large databases of annotated text and speech
- QA, MT, TDT, SLDS, TIDES, ...
- tasks: collection, curation, annotation, analysis
- size: $10^6$ – $10^9$ words
- annotations: phonetics, prosody, orthography, syntax, dialog, and gesture

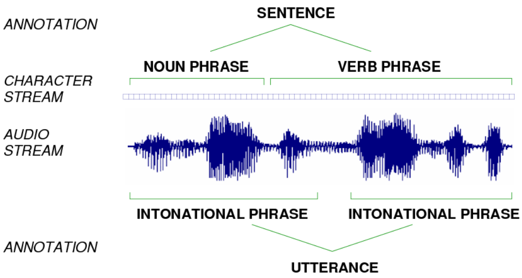
# Language Technology

- require large databases of annotated text and speech
- QA, MT, TDT, SLDS, TIDES, ...
- tasks: collection, curation, annotation, analysis
- size: $10^6$ – $10^9$ words
- annotations: phonetics, prosody, orthography, syntax, dialog, and gesture

# Language Technology

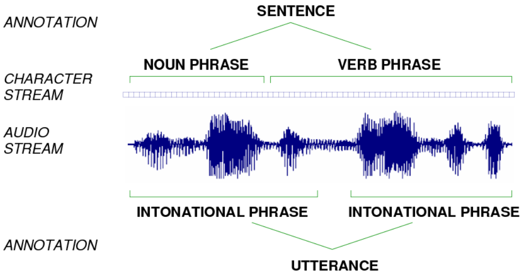
- require large databases of annotated text and speech
- QA, MT, TDT, SLDS, TIDES, ...
- tasks: collection, curation, annotation, analysis
- size: $10^6$ – $10^9$ words
- annotations: phonetics, prosody, orthography, syntax, dialog, and gesture

# Linguistic Annotation



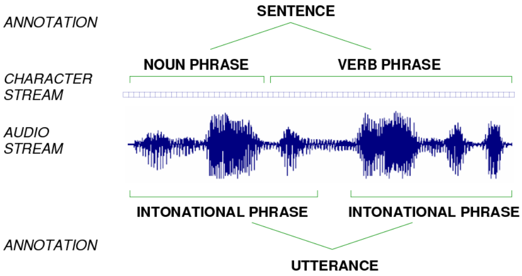

- primary data: immutable, “signal”, supports incoming references
- annotations: coding scheme, structure, hierarchy: trees
- created: manually, or automatically in large quantity

# Linguistic Annotation




- primary data: immutable, “signal”, supports incoming references
- annotations: coding scheme, structure, hierarchy: trees
- created: manually, or automatically in large quantity

# Linguistic Annotation




- primary data: immutable, “signal”, supports incoming references
- annotations: coding scheme, structure, hierarchy: trees
- created: manually, or automatically in large quantity

# Linguistic Databases

- corpus: source, balance, curation, critical judgement
- corpus $\neq$ database!
- collection of records, schema
- managed: integrity, quality
- access: shared, controlled
- language: query, update
- optimisation, indexing
- three-level model

# Linguistic Databases

- corpus: source, balance, curation, critical judgement
- corpus $\neq$ database!
- collection of records, schema
- managed: integrity, quality
- access: shared, controlled
- language: query, update
- optimisation, indexing
- three-level model

# Linguistic Databases

- corpus: source, balance, curation, critical judgement
- corpus $\neq$ database!
- collection of records, schema
- managed: integrity, quality
- access: shared, controlled
- language: query, update
- optimisation, indexing
- three-level model

# Linguistic Databases

- corpus: source, balance, curation, critical judgement
- corpus $\neq$ database!
- collection of records, schema
- managed: integrity, quality
- access: shared, controlled
- language: query, update
- optimisation, indexing
- three-level model

# Linguistic Databases

- corpus: source, balance, curation, critical judgement
- corpus $\neq$ database!
- collection of records, schema
- managed: integrity, quality
- access: shared, controlled
- language: query, update
- optimisation, indexing
- three-level model

# Linguistic Databases

- corpus: source, balance, curation, critical judgement
- corpus $\neq$ database!
- collection of records, schema
- managed: integrity, quality
- access: shared, controlled
- language: query, update
- optimisation, indexing
- three-level model

# Linguistic Databases

- corpus: source, balance, curation, critical judgement
- corpus $\neq$ database!
- collection of records, schema
- managed: integrity, quality
- access: shared, controlled
- language: query, update
- optimisation, indexing
- three-level model

# Linguistic Databases

- corpus: source, balance, curation, critical judgement
- corpus $\neq$ database!
- collection of records, schema
- managed: integrity, quality
- access: shared, controlled
- language: query, update
- optimisation, indexing
- three-level model

# Semistructured Databases

- relational model vs XML
- optional, repeatable elements
- tree model
- schema-later
- XPath

# Semistructured Databases

- relational model vs XML
- optional, repeatable elements
- tree model
- schema-later
- XPath

# Semistructured Databases

- relational model vs XML
- optional, repeatable elements
- tree model
- schema-later
- XPath

# Semistructured Databases

- relational model vs XML
- optional, repeatable elements
- tree model
- schema-later
- XPath

# Semistructured Databases

- relational model vs XML
- optional, repeatable elements
- tree model
- schema-later
- XPath

# Motivations

- benefitting from database theory and technology
- 2005 vs 1965: one-level model: flat files, ad hoc QLs
- general data model: Annotation Graphs

  *Bird & Liberman (2001), Speech Communication 33, pp 23-60*
- existing QLs inadequate:
  - relational and semistructured QLs: expressiveness
  - linguistic QLs: efficiency

# Motivations

- benefitting from database theory and technology
- 2005 vs 1965: one-level model: flat files, ad hoc QLs
- general data model: Annotation Graphs

  *Bird & Liberman (2001), Speech Communication 33, pp 23-60*
- existing QLs inadequate:
  - relational and semistructured QLs: expressiveness
  - linguistic QLs: efficiency

# Motivations

- benefitting from database theory and technology
- 2005 vs 1965: one-level model: flat files, ad hoc QLs
- general data model: Annotation Graphs

  *Bird & Liberman (2001), Speech Communication 33, pp 23-60*
- existing QLs inadequate:
  - relational and semistructured QLs: expressiveness
  - linguistic QLs: efficiency

# Motivations

- benefitting from database theory and technology
- 2005 vs 1965: one-level model: flat files, ad hoc QLs
- general data model: Annotation Graphs

  *Bird & Liberman (2001), Speech Communication 33, pp 23-60*
- existing QLs inadequate:
  - relational and semistructured QLs: expressiveness
  - linguistic QLs: efficiency

# Motivations

- benefitting from database theory and technology
- 2005 vs 1965: one-level model: flat files, ad hoc QLs
- general data model: Annotation Graphs

  *Bird & Liberman (2001), Speech Communication 33, pp 23-60*
- existing QLs inadequate:
  - relational and semistructured QLs: expressiveness
  - linguistic QLs: efficiency

# Motivations

- benefitting from database theory and technology
- 2005 vs 1965: one-level model: flat files, ad hoc QLs
- general data model: Annotation Graphs

  *Bird & Liberman (2001), Speech Communication 33, pp 23-60*
- existing QLs inadequate:
  - relational and semistructured QLs: expressiveness
  - linguistic QLs: efficiency

# First Order Logic over Trees

- universal language for describing structures
- equivalent to SQL
- two linguistic tasks on trees: finding, relating
- both require exactly two free variables

# First Order Logic over Trees

- universal language for describing structures
- equivalent to SQL
- two linguistic tasks on trees: finding, relating
- both require exactly two free variables

# First Order Logic over Trees

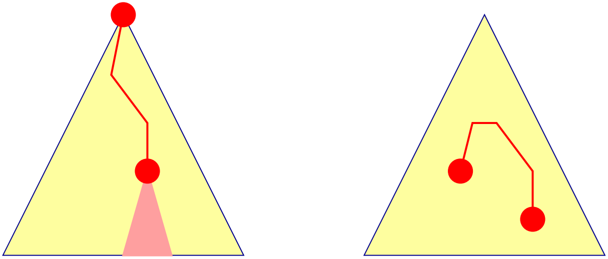
- universal language for describing structures
- equivalent to SQL
- two linguistic tasks on trees: finding, relating
- both require exactly two free variables

# First Order Logic over Trees

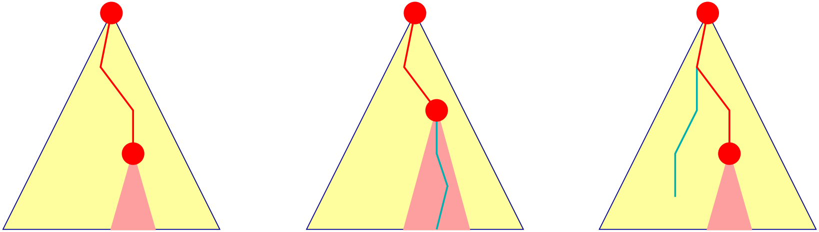
- universal language for describing structures
- equivalent to SQL
- two linguistic tasks on trees: finding, relating
- both require exactly two free variables

# Formulas with Two Free Variables

- binary relations on trees
- $\{(x, y) \mid \exists z \cdots\}$



# Filters



# Axiomatisation of $FO^2_{\text{Tree}}$

- signature: desc , fs
- c.f. Relational tables
- derived relations:
- child $=_{\text{def}} \{x, z \mid \text{desc}(x, z) \wedge \neg\exists y(\text{desc}(x, y) \wedge \text{desc}(y, z))\}$
- f $=_{\text{def}} \{w, z \mid \text{fs}(w, z) \vee \exists x, y(\text{desc}(x, w) \wedge \text{fs}(x, y) \wedge \text{desc}(y, z))\}$

# Goals

- **Create a scalable and reusable model for linguistic query and relate it to well-understood semistructured (XML) and relational (SQL) languages.**
- **Support applications to: exploration, validation, transformation, update, and query-replace**

# Goals

- **Create a scalable and reusable model for linguistic query and relate it to well-understood semistructured (XML) and relational (SQL) languages.**
- **Support applications to: exploration, validation, transformation, update, and query-replace**

# Outline

1. Background and Overview
   - Linguistic Description and Language Technology
   - Annotated Linguistic Databases
   - First Order Logic over Trees
   - Goal, Outline, Acknowledgements
2. Data Model and Query Language
   - Linguistic Trees
   - Linguistic Query Languages
   - Query Requirements
   - Query Language
3. Query Translation and Experiments
   - Relational Storage
   - Translation
   - Experiments
   - Discussion
4. Ongoing Work and Conclusions
   - Graphical Query
   - Update
   - Dependency Trees
   - Conclusions

# Acknowledgements

1. US National Science Foundation:
   Grant No. 0317826 *Querying Linguistic Databases.*
2. Co-Principal Investigators:
   Susan Davidson, Mark Liberman
3. Research Associates:
   Yi Chen, Catherine Lai, Haejoong Lee, Yifeng Zheng
4. For more information:
   `http://www.ldc.upenn.edu/Projects/QLDB/`

# Acknowledgements

1. US National Science Foundation:
   Grant No. 0317826 *Querying Linguistic Databases.*
2. Co-Principal Investigators:
   Susan Davidson, Mark Liberman
3. Research Associates:
   Yi Chen, Catherine Lai, Haejoong Lee, Yifeng Zheng
4. For more information:
   `http://www.ldc.upenn.edu/Projects/QLDB/`

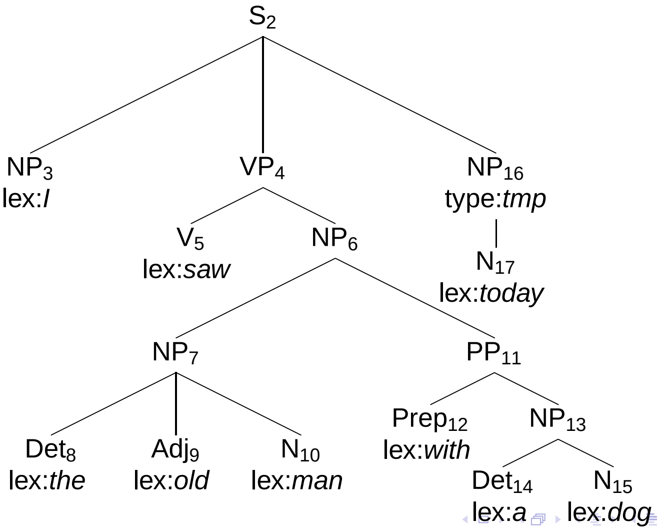
# Acknowledgements

1. US National Science Foundation:
   Grant No. 0317826 *Querying Linguistic Databases.*
2. Co-Principal Investigators:
   Susan Davidson, Mark Liberman
3. Research Associates:
   Yi Chen, Catherine Lai, Haejoong Lee, Yifeng Zheng
4. For more information:
   `http://www.ldc.upenn.edu/Projects/QLDB/`

# Acknowledgements

1. US National Science Foundation:
   Grant No. 0317826 *Querying Linguistic Databases.*
2. Co-Principal Investigators:
   Susan Davidson, Mark Liberman
3. Research Associates:
   Yi Chen, Catherine Lai, Haejoong Lee, Yifeng Zheng
4. For more information:
   `http://www.ldc.upenn.edu/Projects/QLDB/`

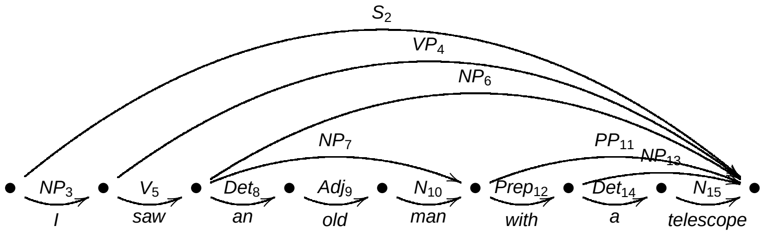
# Linguistic Trees 1



# Linguistic Trees 2: Proper Analyses

| CFG Productions | | | Some Proper Analyses |
|---|---|---|---|
| S | $\longrightarrow$ | NP VP (NP) | *I saw the old N with NP today* |
| VP | $\longrightarrow$ | V NP (NP) | *I V the Adj man PP today* |
| NP | $\longrightarrow$ | NP PP | *NP saw NP with a telescope NP* |
| NP | $\longrightarrow$ | Det Adj$^*$ N | *I VP NP* |
| PP | $\longrightarrow$ | Prep NP | *I saw NP today* |

# Linguistic Trees 3: Charts



# Existing Query Languages

- Linguistic QLs: Issues with efficiency, reusability
- CorpusSearch (U Penn); EMU (Macquarie U); Gsearch (U Edinburgh); Linguists' Search Engine (U Maryland); NetGraph (Charles U, Prague); NXT Search, Q4M, TIGERSearch (U Stuttgart); TGrep2 (MIT); VIQTORYA (Tübingen, Paris);
- Semistructured QLs: problems with expressiveness (XPath) or efficiency (XQuery)

# Existing Query Languages

- Linguistic QLs: Issues with efficiency, reusability
- CorpusSearch (U Penn); EMU (Macquarie U); Gsearch (U Edinburgh); Linguists' Search Engine (U Maryland); NetGraph (Charles U, Prague); NXT Search, Q4M, TIGERSearch (U Stuttgart); TGrep2 (MIT); VIQTORYA (Tübingen, Paris);
- Semistructured QLs: problems with expressiveness (XPath) or efficiency (XQuery)

# Existing Query Languages

- Linguistic QLs: Issues with efficiency, reusability
- CorpusSearch (U Penn); EMU (Macquarie U); Gsearch (U Edinburgh); Linguists' Search Engine (U Maryland); NetGraph (Charles U, Prague); NXT Search, Q4M, TIGERSearch (U Stuttgart); TGrep2 (MIT); VIQTORYA (Tübingen, Paris);
- Semistructured QLs: problems with expressiveness (XPath) or efficiency (XQuery)

# E.g.: NPs whose rightmost child is N

- TGrep2: `NP <- N`
- EMU: `end(Syntax=NP, Syntax=N)=1`
- TIGERSearch

```
[cat="NP"] & #n2:[pos="N"]
    & (#n1 >* #n2) & (#n1 >@r #n3)
    & (#n2 >* #n4)
```

- CorpusSearch

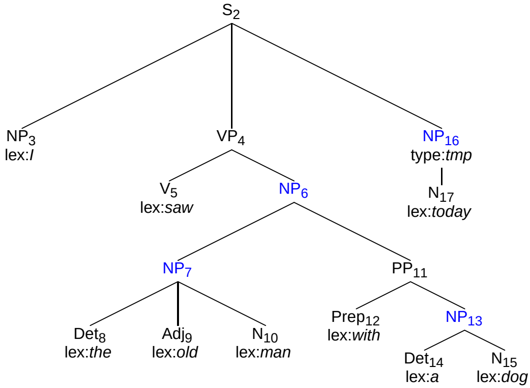
```
node: NP
query: NP iDomsLast1 N
```

- NXT Search

```
($np cat) ($w word) :
    ($np@cat=="NP") && ($w@pos=="N")
    && ($np ^1[-1] $w)
```

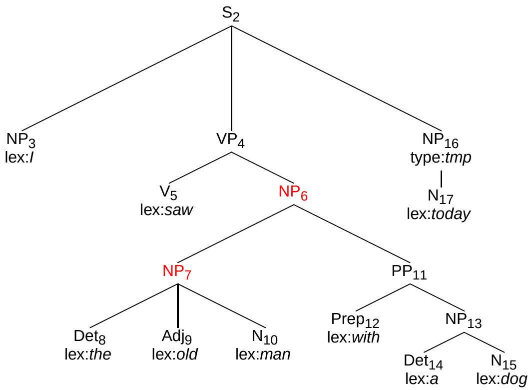
# Sample Queries: Immediate Following

$Q_1$: Find noun phrases that follow a verb `//V/following::NP`



# Sample Queries: Immediate Following

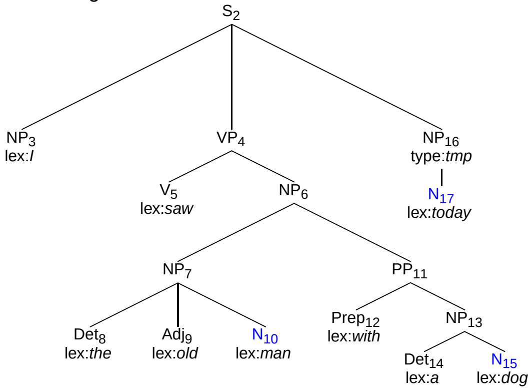
$Q_1$: Find noun phrases that follow a verb `//V/following::NP`
$Q_2$: Find noun phrases that immediately follow a verb



# Sample Queries: Subtree Scoping

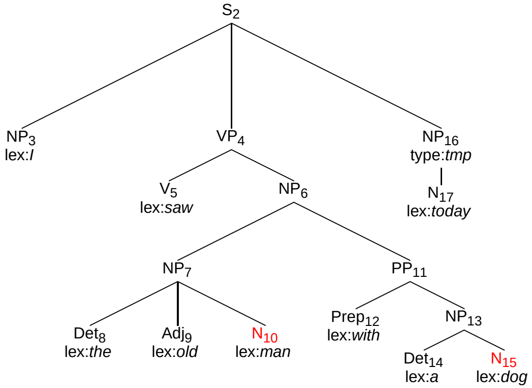
$Q_3$: Find nouns that follow a verb which is a child of a verb phrase

```
//VP/V/following::N
```



# Sample Queries: Subtree Scoping

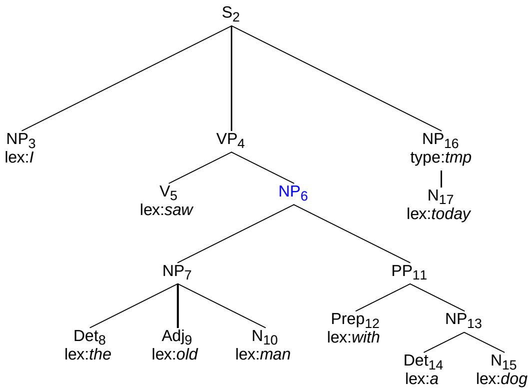
$Q_3$: Find nouns that follow a verb which is a child of a verb phrase
`//VP/V/following::N`
$Q_4$: Within a verb phrase, find nouns that follow a verb which is a child of the verb phrase



# Sample Queries: Edge Alignment

$Q_5$: Find noun phrases which are the rightmost child of a verb phrase

```
//VP/*[last()][self::NP]
```

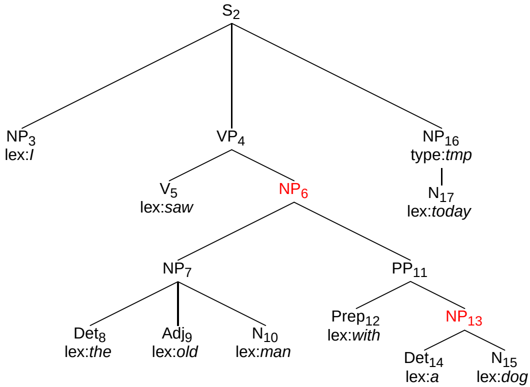


# Sample Queries: Edge Alignment

$Q_5$: Find noun phrases which are the rightmost child of a verb phrase

```
//VP/*[last()][self::NP]
```

$Q_6$: Find noun phrases which are the rightmost descendants of a verb phrase



# LPath

- Path language for linguistic trees

  *B*ird, S, Chen, Y, Davidson, S, Lee, H and Zheng Y (2006) Designing and evaluating an XPath dialect for linguistic queries, 22nd International Conference on Data Engineering
- Extended with "zero-star height" closures (LPath$^+$)

# LPath

- Path language for linguistic trees

  Bird, S, Chen, Y, Davidson, S, Lee, H and Zheng Y (2006) Designing and evaluating an XPath dialect for linguistic queries, 22nd International Conference on Data Engineering
- Extended with "zero-star height" closures (LPath$^+$)

# LPath$^+$ Syntax

```
locpath   :=  abspath | abspath ’{’ locpath ’}’ |
              locpath ’|’ locpath
abspath   :=          | locstep abspath
locstep   :=  axis test (’[’fexpr’]’) (closure)
fexpr     :=  locpath | fexpr ’and’ fexpr | fexpr ’or’
              | ’not’ fexpr | ’(’ fexpr ’)’
axis      :=  ’\’ | ’\\’ | ’\\*’ | ’.’ | ’/’ | ’//’ | ’
              ’->’ | ’<-’ | ’-->’ | ’<--’ |
              ’=>’ | ’<=’ | ’==>’ | ’<==’
closure   :=  ’?’ | ’*’ | ’+’
test      :=  p | _ | ’^’p | p’$’
```

# Navigation Axes in LPath

| | | | |
|---|---|---|---|
| `\` | parent | `/` | child |
| `\\` | ancestor | `//` | descendant |
| `\\*` | ancestor or self | `//*` | descendant or self |
| `->` | immediate following | `<-` | immediate preceding |
| `-->` | following | `<--` | preceding |
| `=>` | immediate following sibling | `<=` | immediate preceding sibling |
| `==>` | following sibling | `<==` | preceding sibling |
| `.` | self | | |

# Subtree Scoping

- XPath doesn't support tree queries that circumscribe navigations to remain inside a subtree
- necessity for compositional queries
  - $Q_4$: Within a verb phrase, find nouns that follow a verb which is a child of the verb phrase.
- LPath uses braces to represent subtree scoping
  - $Q_3$: Find nouns that follow a verb which is a child of a verb phrase `//VP/V-->N`
  - $Q_4$: `//VP{/V-->N}`.

# Subtree Scoping

- XPath doesn't support tree queries that circumscribe navigations to remain inside a subtree
- necessity for compositional queries
  - $Q_4$: Within a verb phrase, find nouns that follow a verb which is a child of the verb phrase.
- LPath uses braces to represent subtree scoping
  - $Q_3$: Find nouns that follow a verb which is a child of a verb phrase `//VP/V-->N`
  - $Q_4$: `//VP{/V-->N}`.

# Subtree Scoping

- XPath doesn't support tree queries that circumscribe navigations to remain inside a subtree
- necessity for compositional queries
  - $Q_4$: Within a verb phrase, find nouns that follow a verb which is a child of the verb phrase.
- LPath uses braces to represent subtree scoping
  - $Q_3$: Find nouns that follow a verb which is a child of a verb phrase `//VP/V-->N`
  - $Q_4$: `//VP{/V-->N}`.

# Subtree Scoping

- XPath doesn't support tree queries that circumscribe navigations to remain inside a subtree
- necessity for compositional queries
  - $Q_4$: Within a verb phrase, find nouns that follow a verb which is a child of the verb phrase.
- LPath uses braces to represent subtree scoping
  - $Q_3$: Find nouns that follow a verb which is a child of a verb phrase `//VP/V-->N`
  - $Q_4$: `//VP{/V-->N}`.

# Subtree Scoping

- XPath doesn't support tree queries that circumscribe navigations to remain inside a subtree
- necessity for compositional queries
  - $Q_4$: Within a verb phrase, find nouns that follow a verb which is a child of the verb phrase.
- LPath uses braces to represent subtree scoping
  - $Q_3$: Find nouns that follow a verb which is a child of a verb phrase `//VP/V-->N`
  - $Q_4$: `//VP{/V-->N}`.

# Subtree Scoping

- XPath doesn't support tree queries that circumscribe navigations to remain inside a subtree
- necessity for compositional queries
  - $Q_4$: Within a verb phrase, find nouns that follow a verb which is a child of the verb phrase.
- LPath uses braces to represent subtree scoping
  - $Q_3$: Find nouns that follow a verb which is a child of a verb phrase `//VP/V-->N`
  - $Q_4$: `//VP{/V-->N}`.

# Edge Alignment

- XPath doesn't fully support alignment on the tree
  - $Q_6$: Find noun phrases which are the rightmost descendants of a verb phrase.
- LPath provides the following grep-like syntactic sugar
  - Leftmost descendant of A: ^A
  - Rightmost descendant of A: A$
- $Q_6$: `//VP{//NP$}`

# Edge Alignment

- XPath doesn't fully support alignment on the tree
  - $Q_6$: Find noun phrases which are the rightmost descendants of a verb phrase.
- LPath provides the following grep-like syntactic sugar
  - Leftmost descendant of A: ^A
  - Rightmost descendant of A: A$
- $Q_6$: `//VP{//NP$}`

# Edge Alignment

- XPath doesn't fully support alignment on the tree
  - $Q_6$: Find noun phrases which are the rightmost descendants of a verb phrase.
- LPath provides the following grep-like syntactic sugar
  - Leftmost descendant of A: ^A
  - Rightmost descendant of A: A$
- $Q_6$: `//VP{//NP$}`

# Edge Alignment

- XPath doesn't fully support alignment on the tree
  - $Q_6$: Find noun phrases which are the rightmost descendants of a verb phrase.
- LPath provides the following grep-like syntactic sugar
  - Leftmost descendant of A: ^A
  - Rightmost descendant of A: A$
- $Q_6$: `//VP{//NP$}`

# Edge Alignment

- XPath doesn't fully support alignment on the tree
  - $Q_6$: Find noun phrases which are the rightmost descendants of a verb phrase.
- LPath provides the following grep-like syntactic sugar
  - Leftmost descendant of A: ^A
  - Rightmost descendant of A: A$
- $Q_6$: `//VP{//NP$}`

# Edge Alignment

- XPath doesn't fully support alignment on the tree
  - $Q_6$: Find noun phrases which are the rightmost descendants of a verb phrase.
- LPath provides the following grep-like syntactic sugar
  - Leftmost descendant of A: ^A
  - Rightmost descendant of A: A$
- $Q_6$: `//VP{//NP$}`

# Sample Queries in LPath

$Q_1$ Find noun phrases that follow a verb
`//V-->NP`

$Q_2$ Find noun phrases that are immediately following a verb.
`//V->NP`

$Q_3$ Find nouns that follow a verb which is a child of a verb phrase.
`//VP/V-->N`

$Q_4$ Within a verb phrase, find nouns that follow a verb which is a child of a verb phrase. `//VP{/V-->N}`

$Q_5$ Find noun phrases which are the rightmost child of a verb phrase.
`//VP{/NP$}`

$Q_6$ Find noun phrases which are rightmost descendants of a verb phrase.
`//VP{//NP$}`

# Sample Queries in LPath

$Q_1$ Find noun phrases that follow a verb
`//V-->NP`

$Q_2$ Find noun phrases that are immediately following a verb.
`//V->NP`

$Q_3$ Find nouns that follow a verb which is a child of a verb phrase.
`//VP/V-->N`

$Q_4$ Within a verb phrase, find nouns that follow a verb which is a child of a verb phrase. `//VP{/V-->N}`

$Q_5$ Find noun phrases which are the rightmost child of a verb phrase.
`//VP{/NP$}`

$Q_6$ Find noun phrases which are rightmost descendants of a verb phrase.
`//VP{//NP$}`

# Sample Queries in LPath

$Q_1$ Find noun phrases that follow a verb
`//V-->NP`

$Q_2$ Find noun phrases that are immediately following a verb.
`//V->NP`

$Q_3$ Find nouns that follow a verb which is a child of a verb phrase.
`//VP/V-->N`

$Q_4$ Within a verb phrase, find nouns that follow a verb which is a child of a verb phrase. `//VP{/V-->N}`

$Q_5$ Find noun phrases which are the rightmost child of a verb phrase.
`//VP{/NP$}`

$Q_6$ Find noun phrases which are rightmost descendants of a verb phrase.
`//VP{//NP$}`

# Sample Queries in LPath

$Q_1$ Find noun phrases that follow a verb
`//V-->NP`

$Q_2$ Find noun phrases that are immediately following a verb.
`//V->NP`

$Q_3$ Find nouns that follow a verb which is a child of a verb phrase.
`//VP/V-->N`

$Q_4$ Within a verb phrase, find nouns that follow a verb which is a child of a verb phrase. `//VP{/V-->N}`

$Q_5$ Find noun phrases which are the rightmost child of a verb phrase.
`//VP{/NP$}`

$Q_6$ Find noun phrases which are rightmost descendants of a verb phrase.
`//VP{//NP$}`

# Sample Queries in LPath

- $Q_1$ Find noun phrases that follow a verb
  `//V-->NP`
- $Q_2$ Find noun phrases that are immediately following a verb.
  `//V->NP`
- $Q_3$ Find nouns that follow a verb which is a child of a verb phrase.
  `//VP/V-->N`
- $Q_4$ Within a verb phrase, find nouns that follow a verb which is a child of a verb phrase. `//VP{/V-->N}`
- $Q_5$ Find noun phrases which are the rightmost child of a verb phrase.
  `//VP{/NP$}`
- $Q_6$ Find noun phrases which are rightmost descendants of a verb phrase.
  `//VP{//NP$}`

# Sample Queries in LPath

$Q_1$ Find noun phrases that follow a verb
`//V-->NP`

$Q_2$ Find noun phrases that are immediately following a verb.
`//V->NP`

$Q_3$ Find nouns that follow a verb which is a child of a verb phrase.
`//VP/V-->N`

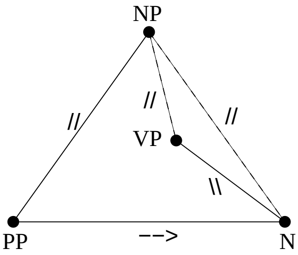
$Q_4$ Within a verb phrase, find nouns that follow a verb which is a child of a verb phrase. `//VP{/V-->N}`

$Q_5$ Find noun phrases which are the rightmost child of a verb phrase.
`//VP{/NP$}`

$Q_6$ Find noun phrases which are rightmost descendants of a verb phrase.
`//VP{//NP$}`

# Dealing with Cyclic Queries

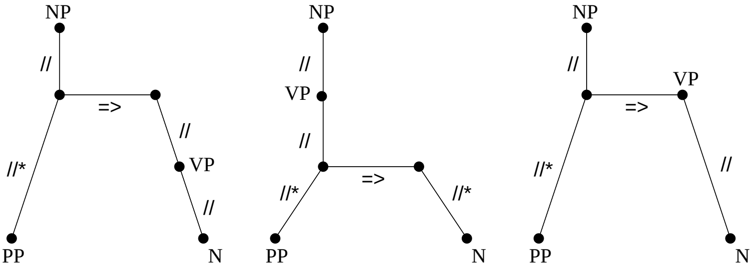
- Apparent mismatch...
- Path queries are acyclic, $FO^2_{Tree}$ formulas may be cyclic
- Scoping syntax leads to cycles
- E.g. `//NP{//PP-->N\\VP}`



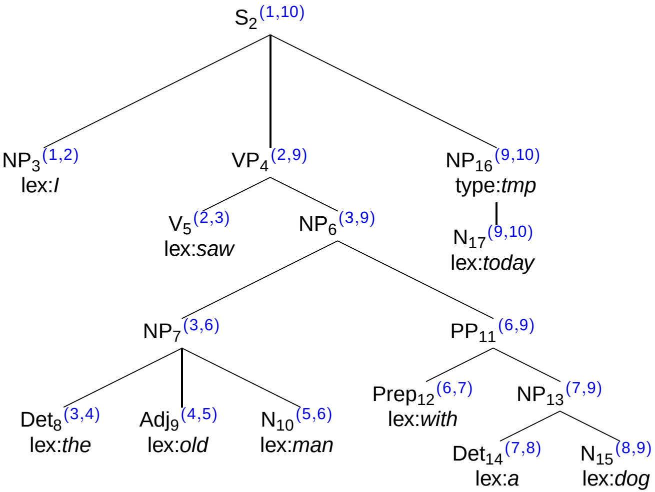
# Solution 1: Re-Introduce Variables

$$\{(w, z) \mid \exists x, y : \mathrm{NP}(w) \wedge \mathrm{PP}(x) \wedge \mathrm{N}(y) \wedge \mathrm{VP}(z)$$
$$\wedge\, \mathrm{desc}(w, x) \wedge \mathrm{f}(x, y) \wedge \mathrm{desc}(y, z)$$
$$\wedge\, \mathrm{desc}(w, y) \wedge \mathrm{desc}(w, y)\}$$

# Solution 2: Factor out Cycles



# Interval Labeling Scheme



# Evaluation: Storage

**T**

| left | right | depth | id | pid | name | value |
|---|---|---|---|---|---|---|
| 1 | 9 | 1 | 1 | 0 | root | |
| 1 | 9 | 1 | 2 | 1 | S | |
| 1 | 2 | 2 | 3 | 2 | NP | |
| 1 | 2 | 3 | 4 | 3 | @lex | I |
| 2 | 9 | 2 | 5 | 2 | VP | |
| 2 | 3 | 3 | 6 | 5 | V | |
| 2 | 3 | 4 | 7 | 6 | @lex | saw |
| 3 | 9 | 3 | 8 | 5 | NP | |
| 3 | 6 | 4 | 9 | 8 | NP | |
| 3 | 4 | 5 | 10 | 9 | Det | |
| 3 | 4 | 6 | 11 | 10 | @lex | the |
| . | . | . | . | . | . | . |
| . | . | . | . | . | . | . |

# LPath Axes and Label Conditions

| | |
|---|---|
| **Vertical Navigation** | |
| child($m, n$) | $n.id = m.pid$ |
| descendent($m, n$) | $m.l \geq n.l, m.r \leq n.r, m.d > n.d$ |
| parent($m, n$) | $m.id = n.pid$ |
| ancestor($m, n$) | $m.l \leq n.l, m.r \geq n.r, m.d < n.d$ |
| **Horizontal Navigation** | |
| immediate-following($m, n$) | $m.l = n.r$ |
| following($m, n$) | $m.l \geq n.r$ |
| immediate-preceding($m, n$) | $m.r = n.l$ |
| preceding($m, n$) | $m.r \leq n.l$ |
| **Sibling Navigation** | |
| immediate-following-sibling($m, n$) | $m.l = n.r, m.pid = n.pid$ |
| following-sibling($m, n$) | $m.l \geq n.r, m.pid = n.pid$ |
| immediate-preceding-sibling($m, n$) | $m.r = n.l, m.pid = n.pid$ |
| preceding-sibling($m, n$) | $m.r \leq n.l, m.pid = n.pid$ |

Interval labels:
(l,r,d,id,pid)
l - left
r - right
d - depth

# Evaluation: Axes and Join Constraints

| | |
|---|---|
| **Vertical Navigation** | |
| child($m, n$) | $n.id = m.pid$ |
| descendent($m, n$) | $m.l \geq n.l, m.r \leq n.r, m.d > n.d$ |
| parent($m, n$) | $m.id = n.pid$ |
| ancestor($m, n$) | $m.l \leq n.l, m.r \geq n.r, m.d < n.d$ |
| **Horizontal Navigation** | |
| immediate-following($m, n$) | $m.l = n.r$ |
| following($m, n$) | $m.l \geq n.r$ |
| immediate-preceding($m, n$) | $m.r = n.l$ |
| preceding($m, n$) | $m.r \leq n.l$ |
| **Sibling Navigation** | |
| immediate-following-sibling($m, n$) | $m.l = n.r, m.pid = n.pid$ |
| following-sibling($m, n$) | $m.l \geq n.r, m.pid = n.pid$ |
| immediate-preceding-sibling($m, n$) | $m.r = n.l, m.pid = n.pid$ |
| preceding-sibling($m, n$) | $m.r \leq n.l, m.pid = n.pid$ |

# SQL Translation

- `//VP/V-->N`

```
select T2.* from T T0, T T1, T T2
where T0.tid=T1.tid and T0.id=T1.pid
and T0.tag='VP' and T1.tid=T2.tid
and T1."right"<=T2."left"
and T1.tag='V' and T2.tag='N'
```

- `//VP{/V-->N}`

```
select T2.* from T T0, T T1, T T2
where T0.tid=T1.tid and T0.id=T1.pid
 and T0."left"<T1."right" and T0."left"<T2."right"
and T0.tag='VP' and T1.tid=T2.tid
 and T1."left"<T0."right" and T1."right"<=T2."left"
and T1.tag='V' and  T2."left"<T0."right" and T2.tag='N'
```

# Experiments

- Load test data
- Translate LPath to SQL using yacc
- Compare with two existing linguistic query engines: CorpusSearch, TGrep2
- Evaluate LPath queries on two data sets
  - Wall Street Journal (WSJ)
  - Switchboard (SWB)

# Experiments

- Load test data
- Translate LPath to SQL using yacc
- Compare with two existing linguistic query engines: CorpusSearch, TGrep2
- Evaluate LPath queries on two data sets
  - Wall Street Journal (WSJ)
  - Switchboard (SWB)

# Experiments

- Load test data
- Translate LPath to SQL using yacc
- Compare with two existing linguistic query engines: CorpusSearch, TGrep2
- Evaluate LPath queries on two data sets
  - Wall Street Journal (WSJ)
  - Switchboard (SWB)

# Experiments

- Load test data
- Translate LPath to SQL using yacc
- Compare with two existing linguistic query engines: CorpusSearch, TGrep2
- Evaluate LPath queries on two data sets
    - Wall Street Journal (WSJ)
    - Switchboard (SWB)

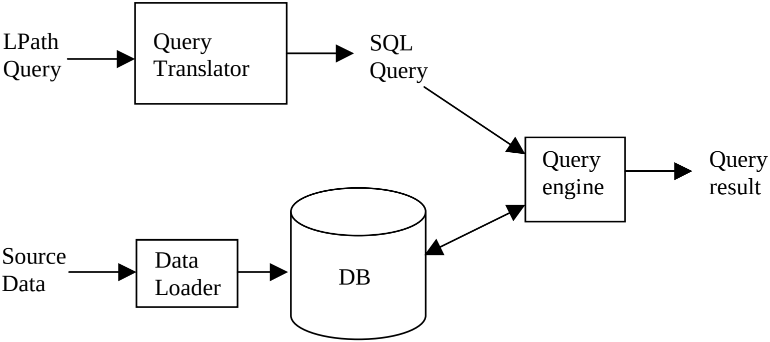
# Experiments

- Load test data
- Translate LPath to SQL using yacc
- Compare with two existing linguistic query engines: CorpusSearch, TGrep2
- Evaluate LPath queries on two data sets
  - Wall Street Journal (WSJ)
  - Switchboard (SWB)

# Experiments

- Load test data
- Translate LPath to SQL using yacc
- Compare with two existing linguistic query engines: CorpusSearch, TGrep2
- Evaluate LPath queries on two data sets
  - Wall Street Journal (WSJ)
  - Switchboard (SWB)

# System Architecture



# Test Data Sets

- Statistics of data sets

| | WSJ | SWB |
|---|---:|---:|
| File Size | 35983kB | 35880kB |
| Tree Nodes | 3484899 | 3972148 |
| Unique Tags | 1274 | 715 |
| Maximum Depth | 36 | 36 |

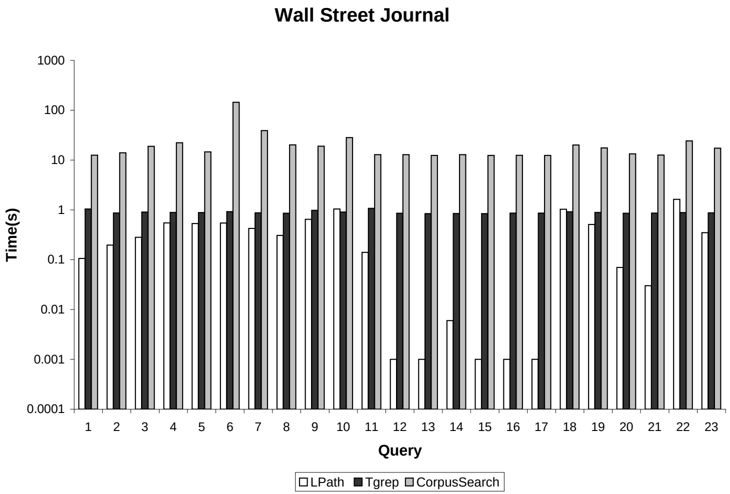
# Queries (1)

| | LPath Query | Size of WSJ Result | Size of SWB result |
|---|---|---|---|
| $Q_1$ | `//S[//_[@lex=saw]]` | 153 | 339 |
| $Q_2$ | `//VB->NP` | 23618 | 16557 |
| $Q_3$ | `//VP/VB-->NN` | 63857 | 32386 |
| $Q_4$ | `//VP{/VB-->NN}` | 46116 | 25305 |
| $Q_5$ | `//VP{/NP$}` | 29923 | 22554 |
| $Q_6$ | `//VP{//NP$}` | 215104 | 112159 |
| $Q_7$ | `//VP[{//^VB->NP->PP$}]` | 2831 | 1963 |
| $Q_8$ | `//S[//NP/ADJP]` | 7832 | 2900 |
| $Q_9$ | `//NP[not(//JJ)]` | 211392 | 109311 |
| $Q_{10}$ | `//NP[->PP[//IN[@lex=of]]=>VP]` | 192 | 31 |
| $Q_{11}$ | `//S[{//_[@lex=what]-->_[@lex=building]}]` | 2 | 5 |

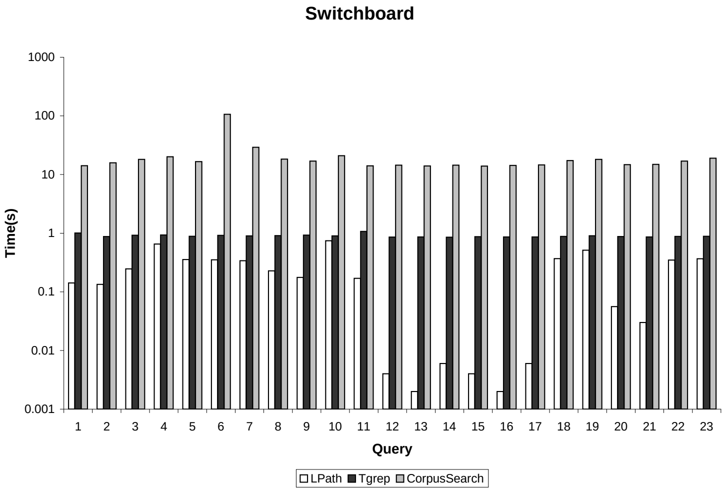
# Queries (2)

| | LPath Query | Size of WSJ Result | Size of SWB result |
|---|---|---|---|
| $Q_{12}$ | `//_[@lex=rapprochement]` | 1 | 0 |
| $Q_{13}$ | `//_[@lex=1929]` | 14 | 0 |
| $Q_{14}$ | `//ADVP-LOC-CLR` | 60 | 0 |
| $Q_{15}$ | `//WHPP` | 87 | 20 |
| $Q_{16}$ | `//RRC/PP-TMP` | 8 | 3 |
| $Q_{17}$ | `//UCP-PRD/ADJP-PRD` | 17 | 4 |
| $Q_{18}$ | `//NP/NP/NP/NP/NP` | 254 | 12 |
| $Q_{19}$ | `//VP/VP/VP` | 8769 | 6093 |
| $Q_{20}$ | `//PP=>SBAR` | 640 | 651 |
| $Q_{21}$ | `//ADVP=>ADJP` | 15 | 37 |
| $Q_{22}$ | `//NP=>NP=>NP` | 7 | 7 |
| $Q_{23}$ | `//VP=>VP` | 20 | 72 |

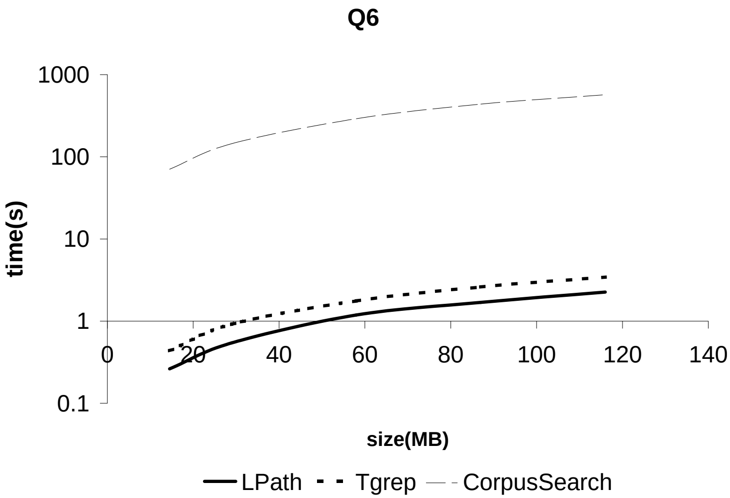
# Query Execution Time



# Query Execution Time



# Scalability as Data Size(WSJ) Increases



# Discussion

- LPath engine works well in most cases
- LPath is slower than TGrep2 in low selectivity queries
  - $Q_{10}$:`//NP[->PP[//IN[@lex=of]]=>VP]`
  - $Q_{18}$:`//NP/NP/NP/NP/NP`
  - $Q_{22}$:`//NP=>NP=>NP`

Most frequent tags in data sets

| | WSJ | | SWB | |
|---|---|---|---|---|
| | Tag | Frequency | Tag | Frequency |
| 1 | NP | 292430 | -DFL- | 193708 |
| 2 | VP | 180405 | VP | 185259 |
| 3 | NN | 163935 | NP-SBJ | 135867 |
| 4 | IN | 121903 | . | 135753 |
| 5 | NNP | 114053 | , | 133528 |
| 6 | S | 107570 | S | 132336 |

# Discussion

- LPath engine works well in most cases
- LPath is slower than TGrep2 in low selectivity queries
  - $Q_{10}$:`//NP[->PP[//IN[@lex=of]]=>VP]`
  - $Q_{18}$:`//NP/NP/NP/NP/NP`
  - $Q_{22}$:`//NP=>NP=>NP`

Most frequent tags in data sets

| | WSJ | | SWB | |
|---|---|---|---|---|
| | Tag | Frequency | Tag | Frequency |
| 1 | NP | 292430 | -DFL- | 193708 |
| 2 | VP | 180405 | VP | 185259 |
| 3 | NN | 163935 | NP-SBJ | 135867 |
| 4 | IN | 121903 | . | 135753 |
| 5 | NNP | 114053 | , | 133528 |
| 6 | S | 107570 | S | 132336 |

# Discussion

- LPath engine works well in most cases
- LPath is slower than TGrep2 in low selectivity queries
  - $Q_{10}$:`//NP[->PP[//IN[@lex=of]]=>VP]`
  - $Q_{18}$:`//NP/NP/NP/NP/NP`
  - $Q_{22}$:`//NP=>NP=>NP`

Most frequent tags in data sets

| | WSJ | | SWB | |
|---|---|---|---|---|
| | Tag | Frequency | Tag | Frequency |
| 1 | NP | 292430 | -DFL- | 193708 |
| 2 | VP | 180405 | VP | 185259 |
| 3 | NN | 163935 | NP-SBJ | 135867 |
| 4 | IN | 121903 | . | 135753 |
| 5 | NNP | 114053 | , | 133528 |
| 6 | S | 107570 | S | 132336 |

# Discussion

- LPath engine works well in most cases
- LPath is slower than TGrep2 in low selectivity queries
  - $Q_{10}$:`//NP[->PP[//IN[@lex=of]]=>VP]`
  - $Q_{18}$:`//NP/NP/NP/NP/NP`
  - $Q_{22}$:`//NP=>NP=>NP`

Most frequent tags in data sets

| | WSJ | | SWB | |
|---|---|---|---|---|
| | Tag | Frequency | Tag | Frequency |
| 1 | NP | 292430 | -DFL- | 193708 |
| 2 | VP | 180405 | VP | 185259 |
| 3 | NN | 163935 | NP-SBJ | 135867 |
| 4 | IN | 121903 | . | 135753 |
| 5 | NNP | 114053 | , | 133528 |
| 6 | S | 107570 | S | 132336 |

# Discussion

- LPath engine works well in most cases
- LPath is slower than TGrep2 in low selectivity queries
  - $Q_{10}$:`//NP[->PP[//IN[@lex=of]]=>VP]`
  - $Q_{18}$:`//NP/NP/NP/NP/NP`
  - $Q_{22}$:`//NP=>NP=>NP`

Most frequent tags in data sets

| | WSJ | | SWB | |
|---|---|---|---|---|
| | Tag | Frequency | Tag | Frequency |
| 1 | NP | 292430 | -DFL- | 193708 |
| 2 | VP | 180405 | VP | 185259 |
| 3 | NN | 163935 | NP-SBJ | 135867 |
| 4 | IN | 121903 | . | 135753 |
| 5 | NNP | 114053 | , | 133528 |
| 6 | S | 107570 | S | 132336 |

# Ongoing Work and Conclusions

- Graphical query interface
- Update
- Dependency trees

# Ongoing Work and Conclusions

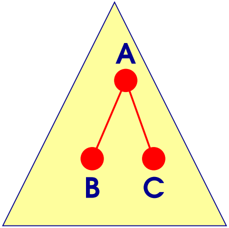
- Graphical query interface
- Update
- Dependency trees

# Ongoing Work and Conclusions

- Graphical query interface
- Update
- Dependency trees

# Graphical Query Language

- Tree subgraphs as a graphical query language
- Ambiguity...



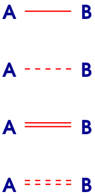

- relative ordering of B and C?

# Graphical Query Language: Path-based Language

- acyclic graph
- edges correspond to LPath$^+$ axes


A / B

A//B




A->B

A-->B

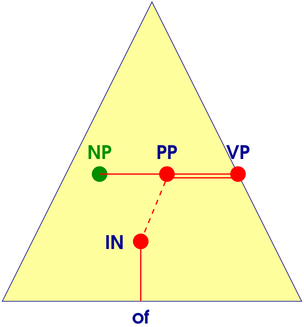
A => B

A==>B

- target node to be returned by query: highlighted
- scopes: triangular regions
- edge alignment: nodes placed on scope boundaries

# Graphical Query Language: Example

- `//NP[->PP[//IN[@lex=of]]=>VP]$`




- overlay query graph on a result tree: query-by-example

# Graphical Query Language: Translation

1. first case: no scopes:
   - distinguished result node R is starting point
   - `//R[context-outwards,content-inwards]`
   - context and content are paths
   - forking points: `...F[fork1,fork2]`
2. second case: scopes, but result node outside them all:
   - distinguished result node R is starting point
   - at the root of each new scope: `//R{...}`
3. third case: result node is inside a scope
   - identify unique path from outermost scope containing R
   - this scope node S is the starting point: `//S`
   - forking points: `...F[fork]/path-to-R`

# Graphical Query Language: Translation

1. first case: no scopes:
   - distinguished result node R is starting point
   - `//R[context-outwards,content-inwards]`
   - context and content are paths
   - forking points: `...F[fork1,fork2]`
2. second case: scopes, but result node outside them all:
   - distinguished result node R is starting point
   - at the root of each new scope: `//R{...}`
3. third case: result node is inside a scope
   - identify unique path from outermost scope containing R
   - this scope node S is the starting point: `//S`
   - forking points: `...F[fork]/path-to-R`

# Graphical Query Language: Translation

1. first case: no scopes:
   - distinguished result node R is starting point
   - `//R[context-outwards,content-inwards]`
   - context and content are paths
   - forking points: `...F[fork1,fork2]`
2. second case: scopes, but result node outside them all:
   - distinguished result node R is starting point
   - at the root of each new scope: `//R{...}`
3. third case: result node is inside a scope
   - identify unique path from outermost scope containing R
   - this scope node S is the starting point: `//S`
   - forking points: `...F[fork]/path-to-R`

# Graphical Query Language: Translation

1. first case: no scopes:
   - distinguished result node R is starting point
   - `//R[context-outwards,content-inwards]`
   - context and content are paths
   - forking points: `...F[fork1,fork2]`
2. second case: scopes, but result node outside them all:
   - distinguished result node R is starting point
   - at the root of each new scope: `//R{...}`
3. third case: result node is inside a scope
   - identify unique path from outermost scope containing R
   - this scope node S is the starting point: `//S`
   - forking points: `...F[fork]/path-to-R`

# Graphical Query Language: Translation

1. first case: no scopes:
   - distinguished result node `R` is starting point
   - `//R[context-outwards,content-inwards]`
   - context and content are paths
   - forking points: `...F[fork1,fork2]`
2. second case: scopes, but result node outside them all:
   - distinguished result node `R` is starting point
   - at the root of each new scope: `//R{...}`
3. third case: result node is inside a scope
   - identify unique path from outermost scope containing `R`
   - this scope node `S` is the starting point: `//S`
   - forking points: `...F[fork]/path-to-R`

# Graphical Query Language: Translation

1. first case: no scopes:
   - distinguished result node `R` is starting point
   - `//R[context-outwards,content-inwards]`
   - context and content are paths
   - forking points: `...F[fork1,fork2]`
2. second case: scopes, but result node outside them all:
   - distinguished result node `R` is starting point
   - at the root of each new scope: `//R{...}`
3. third case: result node is inside a scope
   - identify unique path from outermost scope containing `R`
   - this scope node `S` is the starting point: `//S`
   - forking points: `...F[fork]/path-to-R`

# Graphical Query Language: Translation

1. first case: no scopes:
   - distinguished result node `R` is starting point
   - `//R[context-outwards,content-inwards]`
   - context and content are paths
   - forking points: `...F[fork1,fork2]`
2. second case: scopes, but result node outside them all:
   - distinguished result node `R` is starting point
   - at the root of each new scope: `//R{...}`
3. third case: result node is inside a scope
   - identify unique path from outermost scope containing `R`
   - this scope node `S` is the starting point: `//S`
   - forking points: `...F[fork]/path-to-R`

# Graphical Query Language: Translation

1. first case: no scopes:
   - distinguished result node `R` is starting point
   - `//R[context-outwards,content-inwards]`
   - context and content are paths
   - forking points: `...F[fork1,fork2]`
2. second case: scopes, but result node outside them all:
   - distinguished result node `R` is starting point
   - at the root of each new scope: `//R{...}`
3. third case: result node is inside a scope
   - identify unique path from outermost scope containing `R`
   - this scope node `S` is the starting point: `//S`
   - forking points: `...F[fork]/path-to-R`

# Graphical Query Language: Translation

1. first case: no scopes:
   - distinguished result node R is starting point
   - `//R[context-outwards,content-inwards]`
   - context and content are paths
   - forking points: `...F[fork1,fork2]`
2. second case: scopes, but result node outside them all:
   - distinguished result node R is starting point
   - at the root of each new scope: `//R{...}`
3. third case: result node is inside a scope
   - identify unique path from outermost scope containing R
   - this scope node S is the starting point: `//S`
   - forking points: `...F[fork]/path-to-R`

# Graphical Query Language: Translation

1. first case: no scopes:
   - distinguished result node `R` is starting point
   - `//R[context-outwards,content-inwards]`
   - context and content are paths
   - forking points: `...F[fork1,fork2]`
2. second case: scopes, but result node outside them all:
   - distinguished result node `R` is starting point
   - at the root of each new scope: `//R{...}`
3. third case: result node is inside a scope
   - identify unique path from outermost scope containing `R`
   - this scope node `S` is the starting point: `//S`
   - forking points: `...F[fork]/path-to-R`

# Graphical Query Language: Translation

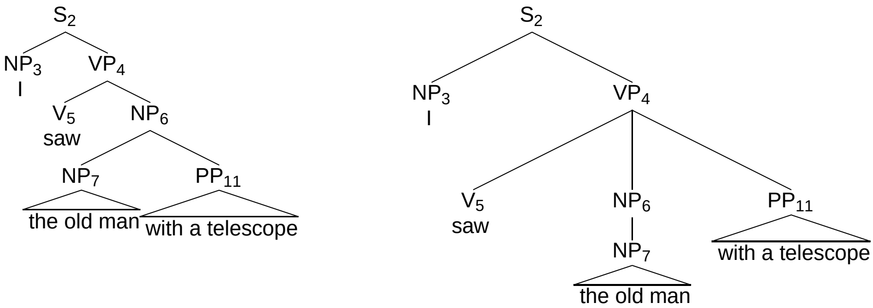
1. first case: no scopes:
   - distinguished result node `R` is starting point
   - `//R[context-outwards,content-inwards]`
   - context and content are paths
   - forking points: `...F[fork1,fork2]`
2. second case: scopes, but result node outside them all:
   - distinguished result node `R` is starting point
   - at the root of each new scope: `//R{...}`
3. third case: result node is inside a scope
   - identify unique path from outermost scope containing `R`
   - this scope node `S` is the starting point: `//S`
   - forking points: `...F[fork]/path-to-R`

# Graphical Query Language: Translation

1. first case: no scopes:
   - distinguished result node `R` is starting point
   - `//R[context-outwards,content-inwards]`
   - context and content are paths
   - forking points: `...F[fork1,fork2]`
2. second case: scopes, but result node outside them all:
   - distinguished result node `R` is starting point
   - at the root of each new scope: `//R{...}`
3. third case: result node is inside a scope
   - identify unique path from outermost scope containing `R`
   - this scope node `S` is the starting point: `//S`
   - forking points: `...F[fork]/path-to-R`

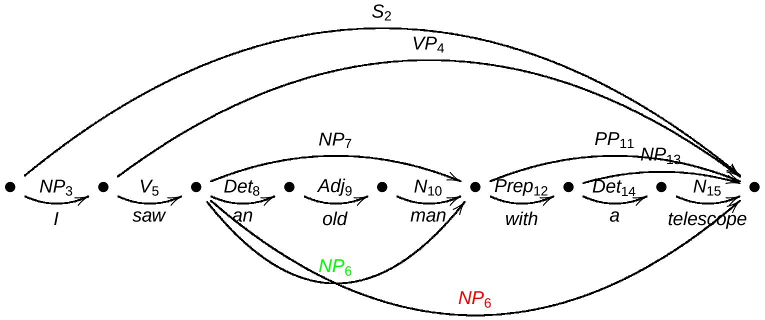
# Update

MOVE //NP/PP RIGHT




Cotton, S and Bird, S (2002) An Integrated Framework for Treebanks and Multilayer Annotations, Proc LREC

# Update: Interpretation



# Dependency Trees

- Popular model for treebanks
- $FO^2_{Tree}$ signature: $\rightarrow$, $\uparrow$
- relational storage: (word-id, parent-id, span, projection, label-data)
- query axes: same as LPath, slight differences of interpretation
- interpretation in $FO^2_{Tree}$ and SQL
- challenge: generalize across tree types

# Dependency Trees

- Popular model for treebanks
- $FO^2_{Tree}$ signature: $\rightarrow$, $\uparrow$
- relational storage: (word-id, parent-id, span, projection, label-data)
- query axes: same as LPath, slight differences of interpretation
- interpretation in $FO^2_{Tree}$ and SQL
- challenge: generalize across tree types

# Dependency Trees

- Popular model for treebanks
- $\mathrm{FO}^2_{\mathrm{Tree}}$ signature: $\rightarrow$, $\uparrow$
- relational storage: (word-id, parent-id, span, projection, label-data)
- query axes: same as LPath, slight differences of interpretation
- interpretation in $\mathrm{FO}^2_{\mathrm{Tree}}$ and SQL
- challenge: generalize across tree types

# Dependency Trees

- Popular model for treebanks
- $\mathrm{FO}^2_{\mathrm{Tree}}$ signature: $\rightarrow, \uparrow$
- relational storage: (word-id, parent-id, span, projection, label-data)
- query axes: same as LPath, slight differences of interpretation
- interpretation in $\mathrm{FO}^2_{\mathrm{Tree}}$ and SQL
- challenge: generalize across tree types

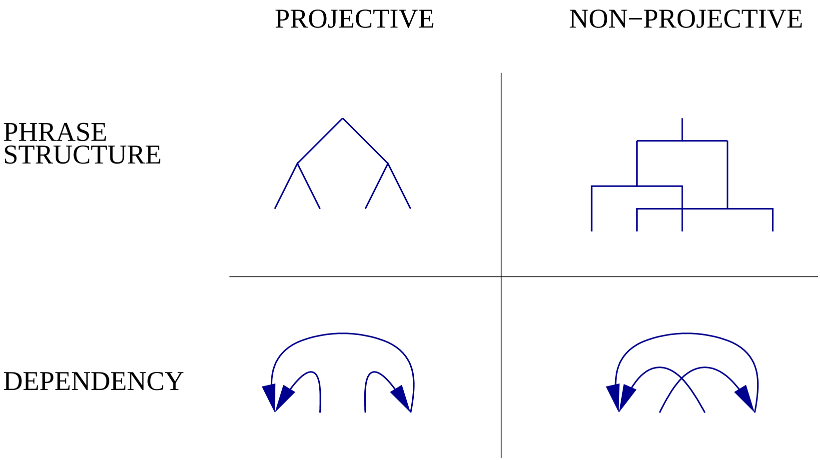
# Dependency Trees

- Popular model for treebanks
- $\mathrm{FO}^2_{\mathrm{Tree}}$ signature: $\rightarrow, \uparrow$
- relational storage: (word-id, parent-id, span, projection, label-data)
- query axes: same as LPath, slight differences of interpretation
- interpretation in $\mathrm{FO}^2_{\mathrm{Tree}}$ and SQL
- challenge: generalize across tree types

# Dependency Trees

- Popular model for treebanks
- $\mathrm{FO}^2_{\mathrm{Tree}}$ signature: $\rightarrow$, $\uparrow$
- relational storage: (word-id, parent-id, span, projection, label-data)
- query axes: same as LPath, slight differences of interpretation
- interpretation in $\mathrm{FO}^2_{\mathrm{Tree}}$ and SQL
- challenge: generalize across tree types

# Tree Types



# First-Order Formulation

- $T$ (terminals), $N$ (non-terminals), $R$ (special root node)
- two binary relations:
  - $\rightarrow$: transitive, total on $T$
  - $\uparrow$: transitive, maps $T \cup N \rightarrow T \cup N \cup \{R\}$
- derived relations:
  - $\dot{\uparrow} : \{\langle m, n \rangle \mid m \uparrow n \wedge \neg \exists p : m \rightarrow p \wedge p \uparrow n\}$
  - symmetry: $\dot{\uparrow}, \downarrow, \dot{\downarrow}, \dot{\downarrow}, \leftarrow$
- parameters:
  - dependency vs phrase-structure: $N = \emptyset, N \neq \emptyset$
  - projective vs non-projective:
    $\forall pqrs \in T \cup N : (p \uparrow r \wedge q \uparrow r \wedge p \rightarrow s \rightarrow q) \longrightarrow s \uparrow r \vee s = r$

# First-Order Formulation

- $T$ (terminals), $N$ (non-terminals), $R$ (special root node)
- two binary relations:
  - $\rightarrow$: transitive, total on $T$
  - $\uparrow$: transitive, maps $T \cup N \rightarrow T \cup N \cup \{R\}$
- derived relations:
  - $\uparrow\cdot : \{\langle m, n\rangle \mid m \uparrow n \wedge \neg\exists p : m \rightarrow p \wedge p \uparrow n\}$
  - symmetry: $\cdot\uparrow, \downarrow, \downarrow\cdot, \cdot\downarrow, \leftarrow$
- parameters:
  - dependency vs phrase-structure: $N = \emptyset, N \neq \emptyset$
  - projective vs non-projective:
    $\forall pqrs \in T \cup N : (p \uparrow r \wedge q \uparrow r \wedge p \rightarrow s \rightarrow q) \longrightarrow s \uparrow r \vee s = r$

# First-Order Formulation

- $T$ (terminals), $N$ (non-terminals), $R$ (special root node)
- two binary relations:
  - $\rightarrow$: transitive, total on $T$
  - $\uparrow$: transitive, maps $T \cup N \rightarrow T \cup N \cup \{R\}$
- derived relations:
  - $\uparrow\!\cdot : \{\langle m, n\rangle \mid m \uparrow n \wedge \neg\exists p : m \rightarrow p \wedge p \uparrow n\}$
  - symmetry: $\cdot\!\uparrow, \downarrow, \downarrow\!\cdot, \cdot\!\downarrow, \leftarrow$
- parameters:
  - dependency vs phrase-structure: $N = \emptyset, N \neq \emptyset$
  - projective vs non-projective: $\forall pqrs \in T \cup N : (p \uparrow r \wedge q \uparrow r \wedge p \rightarrow s \rightarrow q) \longrightarrow s \uparrow r \vee s = r$

# First-Order Formulation

- $T$ (terminals), $N$ (non-terminals), $R$ (special root node)
- two binary relations:
  - $\rightarrow$: transitive, total on $T$
  - $\uparrow$: transitive, maps $T \cup N \rightarrow T \cup N \cup \{R\}$
- derived relations:
  - $\uparrow\!\cdot : \{\langle m, n\rangle \mid m \uparrow n \wedge \neg\exists p : m \rightarrow p \wedge p \uparrow n\}$
  - symmetry: $\cdot\!\uparrow, \downarrow, \downarrow\!\cdot, \cdot\!\downarrow, \leftarrow$
- parameters:
  - dependency vs phrase-structure: $N = \emptyset, N \neq \emptyset$
  - projective vs non-projective: $\forall pqrs \in T \cup N : (p \uparrow r \wedge q \uparrow r \wedge p \rightarrow s \rightarrow q) \longrightarrow s \uparrow r \vee s = r$

# First-Order Formulation

- $T$ (terminals), $N$ (non-terminals), $R$ (special root node)
- two binary relations:
  - $\rightarrow$: transitive, total on $T$
  - $\uparrow$: transitive, maps $T \cup N \rightarrow T \cup N \cup \{R\}$
- derived relations:
  - $\uparrow\!\cdot : \{\langle m, n\rangle \mid m \uparrow n \wedge \neg\exists p : m \rightarrow p \wedge p \uparrow n\}$
  - symmetry: $\cdot\!\uparrow, \downarrow, \downarrow\!\cdot, \cdot\!\downarrow, \leftarrow$
- parameters:
  - dependency vs phrase-structure: $N = \emptyset, N \neq \emptyset$
  - projective vs non-projective: $\forall pqrs \in T \cup N : (p \uparrow r \wedge q \uparrow r \wedge p \rightarrow s \rightarrow q) \longrightarrow s \uparrow r \vee s = r$

# First-Order Formulation

- $T$ (terminals), $N$ (non-terminals), $R$ (special root node)
- two binary relations:
  - $\rightarrow$: transitive, total on $T$
  - $\uparrow$: transitive, maps $T \cup N \rightarrow T \cup N \cup \{R\}$
- derived relations:
  - $\uparrow\!\cdot : \{\langle m, n\rangle \mid m \uparrow n \wedge \neg\exists p : m \rightarrow p \wedge p \uparrow n\}$
  - symmetry: $\cdot\!\uparrow, \downarrow, \downarrow\!\cdot, \cdot\!\downarrow, \leftarrow$
- parameters:
  - dependency vs phrase-structure: $N = \emptyset, N \neq \emptyset$
  - projective vs non-projective: $\forall pqrs \in T \cup N : (p \uparrow r \wedge q \uparrow r \wedge p \rightarrow s \rightarrow q) \longrightarrow s \uparrow r \vee s = r$

# First-Order Formulation

- $T$ (terminals), $N$ (non-terminals), $R$ (special root node)
- two binary relations:
  - $\rightarrow$: transitive, total on $T$
  - $\uparrow$: transitive, maps $T \cup N \rightarrow T \cup N \cup \{R\}$
- derived relations:
  - $\dot{\uparrow} : \{\langle m, n\rangle \mid m \uparrow n \wedge \neg\exists p : m \rightarrow p \wedge p \uparrow n\}$
  - symmetry: $\dot{\uparrow}, \downarrow, \dot{\downarrow}, \dot{\downarrow}, \leftarrow$
- parameters:
  - dependency vs phrase-structure: $N = \emptyset, N \neq \emptyset$
  - projective vs non-projective: $\forall pqrs \in T \cup N : (p \uparrow r \wedge q \uparrow r \wedge p \rightarrow s \rightarrow q) \longrightarrow s \uparrow r \vee s = r$

# First-Order Formulation

- $T$ (terminals), $N$ (non-terminals), $R$ (special root node)
- two binary relations:
  - $\rightarrow$: transitive, total on $T$
  - $\uparrow$: transitive, maps $T \cup N \rightarrow T \cup N \cup \{R\}$
- derived relations:
  - $\uparrow\!\cdot : \{\langle m, n\rangle \mid m \uparrow n \wedge \neg\exists p : m \rightarrow p \wedge p \uparrow n\}$
  - symmetry: $\cdot\!\uparrow, \downarrow, \downarrow\!\cdot, \cdot\!\downarrow, \leftarrow$
- parameters:
  - dependency vs phrase-structure: $N = \emptyset, N \neq \emptyset$
  - projective vs non-projective: $\forall pqrs \in T \cup N : (p \uparrow r \wedge q \uparrow r \wedge p \rightarrow s \rightarrow q) \longrightarrow s \uparrow r \vee s = r$

# First-Order Formulation

- $T$ (terminals), $N$ (non-terminals), $R$ (special root node)
- two binary relations:
  - $\rightarrow$: transitive, total on $T$
  - $\uparrow$: transitive, maps $T \cup N \rightarrow T \cup N \cup \{R\}$
- derived relations:
  - $\uparrow\!\cdot : \{\langle m, n\rangle \mid m \uparrow n \wedge \neg\exists p : m \rightarrow p \wedge p \uparrow n\}$
  - symmetry: $\cdot\!\uparrow, \downarrow, \downarrow\!\cdot, \cdot\!\downarrow, \leftarrow$
- parameters:
  - dependency vs phrase-structure: $N = \emptyset, N \neq \emptyset$
  - projective vs non-projective:
    $\forall pqrs \in T \cup N : (p \uparrow r \wedge q \uparrow r \wedge p \rightarrow s \rightarrow q) \longrightarrow s \uparrow r \vee s = r$

# First-Order Formulation

- $T$ (terminals), $N$ (non-terminals), $R$ (special root node)
- two binary relations:
  - $\rightarrow$: transitive, total on $T$
  - $\uparrow$: transitive, maps $T \cup N \rightarrow T \cup N \cup \{R\}$
- derived relations:
  - $\uparrow\!\cdot : \{\langle m, n \rangle \mid m \uparrow n \wedge \neg \exists p : m \rightarrow p \wedge p \uparrow n\}$
  - symmetry: $\cdot\!\uparrow, \downarrow, \downarrow\!\cdot, \cdot\!\downarrow, \leftarrow$
- parameters:
  - dependency vs phrase-structure: $N = \emptyset, N \neq \emptyset$
  - projective vs non-projective:
    $\forall pqrs \in T \cup N : (p \uparrow r \wedge q \uparrow r \wedge p \rightarrow s \rightarrow q) \longrightarrow s \uparrow r \vee s = r$

# Axes

| | | |
|---|---|---|
| `M//N` | $\{\langle m,n\rangle \mid m \downarrow n\}$ | $m.pl \geq n.pl, m.pr \leq n.pr, m.d$ |
| `M-->N` | $\{\langle m,n\rangle \mid m \rightarrow n\}$ | $m.wr \leq n.wl$ |
| `M==>N` | `M-->N` $\cap$ `M\_/N` | ? |
| `M~~>N` | $\{\langle m,n\rangle \mid \exists p,q : m \uparrow\!\cdot p$ `==>` $q\cdot\!\downarrow n\}$ | $m.pr \leq n.pl$ |
| `M/N` | `M//N` $-$ `M//_//N` | $n.id = m.pid$ |
| `M->N` | `M-->N` $-$ `M-->_-->N` | $m.wl = n.wr$ |
| `M=>N` | `M==>N` $-$ `M==>_==>N` | ? |
| `M~>N` | `M~~>N` $-$ `M~~>_~~>N` | $m.pr = n.pl$ |

# Conclusions

- Path languages are very suitable for linguistics
- Proposed LPath language which extends XPath to support immediate precedence, subtree scoping, edge alignment
- Designed a labeling scheme to speed up LPath navigations
- Implemented an LPath interpreter on top of SQL
- General purpose, scalable

# Conclusions

- Path languages are very suitable for linguistics
- Proposed LPath language which extends XPath to support immediate precedence, subtree scoping, edge alignment
- Designed a labeling scheme to speed up LPath navigations
- Implemented an LPath interpreter on top of SQL
- General purpose, scalable

# Conclusions

- Path languages are very suitable for linguistics
- Proposed LPath language which extends XPath to support immediate precedence, subtree scoping, edge alignment
- Designed a labeling scheme to speed up LPath navigations
- Implemented an LPath interpreter on top of SQL
- General purpose, scalable

# Conclusions

- Path languages are very suitable for linguistics
- Proposed LPath language which extends XPath to support immediate precedence, subtree scoping, edge alignment
- Designed a labeling scheme to speed up LPath navigations
- Implemented an LPath interpreter on top of SQL
- General purpose, scalable

# Conclusions

- Path languages are very suitable for linguistics
- Proposed LPath language which extends XPath to support immediate precedence, subtree scoping, edge alignment
- Designed a labeling scheme to speed up LPath navigations
- Implemented an LPath interpreter on top of SQL
- General purpose, scalable

# Postscript

Some thoughts on the agenda...

1. **Expressiveness:** first-order is sufficient
2. **Approximation:** linguistic exploration does not require exact query
3. **Scalability:** linguistic query systems will only scale if they are built on scalable technology
4. **Limits to variation:** treebanks are not as diverse as they seem: need a parametric approach

# Postscript

Some thoughts on the agenda...

1. **Expressiveness:** first-order is sufficient
2. **Approximation:** linguistic exploration does not require exact query
3. **Scalability:** linguistic query systems will only scale if they are built on scalable technology
4. **Limits to variation:** treebanks are not as diverse as they seem: need a parametric approach

# Postscript

Some thoughts on the agenda...

1. **Expressiveness:** first-order is sufficient
2. **Approximation:** linguistic exploration does not require exact query
3. **Scalability:** linguistic query systems will only scale if they are built on scalable technology
4. **Limits to variation:** treebanks are not as diverse as they seem: need a parametric approach

# Postscript

Some thoughts on the agenda...

1. **Expressiveness:** first-order is sufficient
2. **Approximation:** linguistic exploration does not require exact query
3. **Scalability:** linguistic query systems will only scale if they are built on scalable technology
4. **Limits to variation:** treebanks are not as diverse as they seem: need a parametric approach

# The End

Danke schön