ADVANCED PARSING: FROM RULES, TO P&P... AND MINIMALISM

References

- Essential references

- Extended references

From Rules to Principles and Parameters

- rules
  - Principle & Parameters (P&P)
  - Language specific
  - linguistic universals + parameters settings

- P&P aims at a better explicative adequacy (other than descriptive)

- Goal: linguistic universals capture the limited syntactic variability across languages

- Principle-based parsers (Barton 1984, Berwick & Fong 90, Stabler 92) are inspired by this intuition:
  - Grammatical principles are parser axioms
  - the parser operates as a deductive system inferring grammatical structures applying the axioms to the input
From Rules to Principles and Parameters

- **Rules**
  - passive transformation → passive sentence
  - coordination transformation → coordinated sentence
  - focalization transformation → sentence with a focalized constituent
  - ...

- **P&P**
  - P1: passive sentence
  - P2: coordinated sentence
  - P3: sentence with a focalized constituent

- Few principles + few parameters = thousands of rules!

---

P&P

- **"T" model**

  - **DS** Deep Structure
  - **SS** Surface Structure
  - **LF** Logical Form
  - **PF** Phonetic Form

---

P&P

- **Some principles**
  - **X' theory**
    - XP
    - specifier
    - head
    - complement
  - **θ - criterion**
    - every argument must receive one and only one thematic (θ) role (and every thematic role is assigned to just one argument)
  - **Case filter**
    - every lexical NP must receive case (P e V_{base} are case assigners)

---

P&P

- **Generators**
  - principles producing more structures than the ones in input:
    - Move α
    - Free indexation
    - ...

- **Filters**
  - principles selecting fewer structures than the ones received as input:
    - X' theory
    - θ - criterion
    - Case filter
Grammatical principles are expressed on a "high order" language (first order logics) then compiled in the parsed (this is a strategy for improving performance).  

- Case Filter
  - definition: "Every lexical NP must receive a case at SS"
  - (pseudo) formalization: \( \forall \text{NP}: \text{NP} \subseteq \text{CF}_{\text{SS}} \rightarrow \text{NP} \rightarrow \text{case} \)
  - Implementation in Prolog:
    ```prolog
    :- caseFilter in_all_configuration CF where
       lexicalNP(CF) then assignedCase(CF).
    lexicalNP :- cat(NP, np), +I ec(NP).
    assignedCase(X) :- X has_features case(Case), assigned (Case).
    ```

---

**PAPPI (Fong 1991)**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Extended Using</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case assignment</td>
<td>in_allConfigurations</td>
</tr>
<tr>
<td>Case marking</td>
<td>in_allConfigurations</td>
</tr>
<tr>
<td>Subject Case assignment</td>
<td>in_allConfigurations</td>
</tr>
<tr>
<td>Inherent Case assignment</td>
<td>in_allConfigurations</td>
</tr>
<tr>
<td>Structural Case assignment</td>
<td>in_allConfigurations</td>
</tr>
<tr>
<td>Selection</td>
<td>in_allConfigurations</td>
</tr>
<tr>
<td>Case filter</td>
<td>in_allConfigurations</td>
</tr>
<tr>
<td>Case condition on traces</td>
<td>in_allConfigurations</td>
</tr>
<tr>
<td>Trace Kenny (chain formation at Structure)</td>
<td>in_allConfigurations</td>
</tr>
<tr>
<td>Ph-asms movement in syntax</td>
<td>in_allConfigurations</td>
</tr>
<tr>
<td>Context Subject and INFL</td>
<td>in_allConfigurations</td>
</tr>
<tr>
<td>Free Induction</td>
<td>in_allConfigurations</td>
</tr>
<tr>
<td>Functional Determination</td>
<td>in_allConfigurations (transformed)</td>
</tr>
<tr>
<td>Condition A</td>
<td>in_allConfigurations (transformed)</td>
</tr>
<tr>
<td>Condition B</td>
<td>in_allConfigurations (transformed)</td>
</tr>
<tr>
<td>Condition C</td>
<td>in_allConfigurations (transformed)</td>
</tr>
<tr>
<td>Empty Category Principle (ECP)</td>
<td>in_allConfigurations (transformed)</td>
</tr>
<tr>
<td>Coated</td>
<td>in_allConfigurations</td>
</tr>
<tr>
<td>LF movement</td>
<td>Hand-coded</td>
</tr>
<tr>
<td>ECP at LF</td>
<td>in_allConfigurations (transformed)</td>
</tr>
<tr>
<td>License operator-variables</td>
<td>in_allConfigurations</td>
</tr>
<tr>
<td>License syntactic adjuncts</td>
<td>in_allConfigurations</td>
</tr>
<tr>
<td>WH-Comp requirement at LF</td>
<td>in_allConfigurations</td>
</tr>
</tbody>
</table>
PAPPI (Fong 1991)
Ordering Principles

- 16 principles = 16! Possible orders...
  nearly 21,922,789,888,000 possibilities (which one is the... best?)
- Each order different performance with respect to the same input
- No universal order guarantee absolute efficiency (best performance ever)
- Fong suggests using cheap cues for pre-ordering generators and filters.

Input sentence: PS → P₁ → P₂ → ... → Pₙ → LF

PAPPI (Fong 1991)
Control strategies

- Analysis-by-synthesis
  produces everything, then test the input (parallel top-down approach)
- Generate and Test
  one principle at time, generate structures, then test the input (serial top-down, depth first)
- Coroutining, freezing, clause selection, ordering
  re-organize the problem space, trying to minimize the exploration needs
- Covering
  off-line grammar compile; groups of states collapsing whenever possible, reducing time complexity (number of steps)

PAPPI (Fong 1991): Architecture

Minimalist Grammars

- Stabler’s (1997) formalization of a Minimalist Grammar, MG (Chomsky 1995) as a 4-tuple \{V, Cat, Lex, F\} such that:
  - V is a finite set of non-syntactic features, \(P \cup I\) where
    - P are phonetic features and I are semantic ones;
  - Cat is a finite set of syntactic features,
    - Cat = \{base \cup select \cup licensors \cup licensees\} where
      - base are standard categories (comp, tense, verb, noun ...),
      - select specify a selection requirement (=x | x base)
      - licensees force phrasal movement (~wh, ~case ...),
      - licensors satisfy licensee requirements (+wh, +case ...)
  - Lex is a finite set of expressions built from V and Cat (the lexicon);
  - F is a set of two partial functions from tuples of expressions to expressions : \{merge, move\};
Minimalist Grammars

\[ V = \{ {\text{what/} /\text{did/} /\text{you/} /\text{see/}} \}, \]
\[ I = \{ {\text{what/} /\text{did/} /\text{you/} /\text{see/}} \} \]

\[ \text{Cat} = \{ \text{base} = \{ D, N, V, T, C \}, \]
\[ \text{select} = \{ D = D, N = V, T = T, C = C \} \]
\[ \text{licensors} = \{ +\text{wh} \} \]
\[ \text{licensees} = \{ -\text{wh} \} \]

\[ \text{Lex} = \{ [ -\text{wh D what}], [ = V T \text{ did}], [D \text{ you}], [ = D +\text{wh C } \emptyset] \} \]

\[ F = \{ \text{merge, move} \} \text{ such that:} \]
\[ \text{merge} ([ = F X], [ F Y]) = [ X [ Y] ] \]
\[ (*\text{"simple merge" on the right, "complex merge" on the left}) \]
\[ \text{move} ([ = F X], [ W [ -\text{g} Y] ] ) = [ [ X Y] W, \text{t}] \]

\[ 1. \text{merge} ( [ = D =D V \text{ see}], [ -\text{wh D what}]) \rightarrow [ = D =D V \text{ see}, -\text{wh what}] \]
\[ 2. \text{merge} ( [D \text{ you}], [ = D V \text{ see}, -\text{wh what}]) \rightarrow [ = D \text{ you}, = D V \text{ see}, -\text{wh what}] \]
\[ 3. \text{merge} ( [ = V T \text{ did}], [ = D \text{ you}, = D V \text{ see}, -\text{wh what}]) \rightarrow [ = D \text{ did}, = D \text{ did}, = D \text{ you}, = D V \text{ see}, -\text{wh what}] \]
\[ 4. \text{merge} ( [ = T +\text{wh C } \emptyset], [ = D \text{ did}, = D \text{ did}, = D \text{ you}, = D V \text{ see}, -\text{wh what}]) \rightarrow [ = C \emptyset, = D \text{ did}, = D \text{ did}, = D \text{ you}, = D V \text{ see}, -\text{wh what}] \]
\[ 5. \text{move} ( [ = +\text{wh C } \emptyset], [ = D \text{ did}, = D \text{ did}, = D \text{ you}, = D V \text{ see}, -\text{wh what}]) \rightarrow [ = C \emptyset, = D \text{ did}, = D \text{ did}, = D \text{ you}, = D V \text{ see}, -\text{wh what}], [ = C \emptyset, = D \text{ did}, = D \text{ did}, = D \text{ you}, = D V \text{ see}, -\text{wh what}] \]
Lexicon: {
[D =DP =PP V gives], [K (N) to], [D N John], [D N children], [D N candies]}

1. merge ([D =DP =PP V gives], [D N John])

V
\[\begin{array}{c}
\text{gives} \\
\text{John}
\end{array}\]

PMGs SBO: Merge Right (Phillips 1996)

Lexicon: {
[D =DP =PP V gives], [K (N) to], [D N John], [D N children], [D N candies]}

1. merge ([D =DP =PP V gives], [D N John])
2. merge ([D =DP =PP V gives], [D N Children])

V
\[\begin{array}{c}
\text{gives} \\
\text{John} \\
\text{<gives>}
\end{array}\]

V
\[\begin{array}{c}
\text{gives} \\
\text{candies} \\
\text{<gives>}
\end{array}\]
Proposal: Phase-based MGs (PMGs, Chesi 2004-07)

- Constraints on Structure building operations
  - Top-Down (non-terminal node expansion)
    - A → B → C
  - Left-Rigth (terminal node attachment)
    - A → a b

PMG: Sample derivation of wh-question

- (default) Expand (Lex: CP_{wh} = \{+wh +T +S V\})
  - +wh +T +S V
PMG: Sample derivation of wh-question

- (default) Expand(Lex: CP_{wh} = (+wh +T +S V))
- Insert(Lex: (+wh +D N what))

PMG: Sample derivation of wh-question

- Insert(Lex: (+T did))

PMG: Sample derivation of wh-question

- Insert(Lex: (+S +D N you))
PMG: Sample derivation of wh-question

- **Insert(Lex: \( V = DP = DP \, see \))**

```
V
+wh +D N
  +T                   +S +D N V =DP =DP
    did                    see
  +D S N you
```

33

PMG: Sample derivation of wh-question

- **Expand(\( V = DP \))**
- **Move(\(+D N you\))**

```
V
+wh +D N
  +T                    +S +D N V =DP =DP
    did                    see
  +D S N you
  +D N (you)
```

34

PMG: Sample derivation of wh-question

- **Expand(\( V = DP \))**
- **Move(\(+D N what\))**

```
V
+wh +D N
  +T                    +S +D N V =DP =DP
    did                    see
  +D S N you
  +D N (what)
```

35

PMG: Sample derivation of wh-question

- **Expand(\( V = DP \))**
- **Move(\(+D N what\))**

```
V
+wh +D N
  +T                    +S +D N V =DP =DP
    did                    see
  +D S N you
  +D N (what)
```

36
**Linearization Principle (inspired by Kayne’s LCA)**

If A immediately dominates B, then either

a. `<A, B>` if A selects B as an argument, or
b. `<B, A>` if B is in a functional specification of A

**e.g.** “the boy kissed the girl”

**Success condition:** M-buffer(s) must be empty at the end of the computation

---

**PHASE (PROJECTION)**

**Move**

Two more Structure Building Operations

**Nested Phase**

**Sequential Phase**

**“Internal” Merge**

**Vs.**

Vs.

**Nested Phase**

**Vs.**

**Nested Phase**

**Parsing Algorithm with PMGs (PMG-pa)**

**Successive Cyclic A’-movement**

Who do you believe `f` with that ... John admires `f` with?
Parsing States: CFG-Earley Vs. PMG-pa

<table>
<thead>
<tr>
<th>CFG-Earley</th>
<th>PMG-pa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input position (e.g. the man runs)</td>
<td>Input position (the man runs)</td>
</tr>
<tr>
<td>Grammar rule (e.g. S → NP VP)</td>
<td>Phase inspected (e.g. Verbal)</td>
</tr>
<tr>
<td>Dot-position (e.g. S → NP • VP)</td>
<td>Constituent completion status (is the phase headed? yes are all thematic requirements satisfied? yes are non-thematic dependencies licensed? yes are non-thematic dependencies unique? yes further non-thematic dependencies available? yes status: potentially complete)</td>
</tr>
<tr>
<td>Leftmost edge of the substring this rule generates (e.g. the man runs)</td>
<td>Constituent prefix (e.g. the man runs)</td>
</tr>
<tr>
<td>Memory buffer status (e.g. one N(ominal), determined, potentially complete phase)</td>
<td></td>
</tr>
</tbody>
</table>

Parsing Operations: CFG-Earley Vs. PMG-pa

<table>
<thead>
<tr>
<th>CFG-Earley</th>
<th>PMG-pa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predict</td>
<td>Phase Projection</td>
</tr>
<tr>
<td>top-down expansion of non-terminal rules in the grammar; Result: new rules (NP, VP, PP, AP...) added</td>
<td>new constituents insertion based on selection features in the lexicon; Result: Rooted Trees (RTrees) decorated with empty NP, VP or AP</td>
</tr>
<tr>
<td>Scan</td>
<td>Move / Lexical insertion</td>
</tr>
<tr>
<td>bottom-up inspection of the lexicon given a word; Result: PoS list to be integrated in the rules</td>
<td>unselected lexicalized sub-trees (moved LTrees) are available in this list, plus lexicalized sub-trees projected from a Top-Down inspection of the processed word-token; Result: ordered list (the pending list) of lexicalized sub-trees (LTrees) to be integrated in the structure</td>
</tr>
<tr>
<td>Complete</td>
<td>Merge</td>
</tr>
<tr>
<td>top-down expansion of non-terminal rules in the grammar</td>
<td>unification algorithm among pending rooted structures (RTrees) and lexicalized sub-trees in the pending list (LTrees)</td>
</tr>
</tbody>
</table>

Getting asymmetries with PMG-pa

Subject relatives in head initial languages
a. The reporter who attacked the senator admitted the error.
b. The reporter who the senator attacked admitted the error.

1. \( w_i = \text{"the"} \)
Getting asymmetries with PMG-pa

Subject relatives in head initial languages

a. The reporter who attacked the senator admitted the error.
b. The reporter who the senator attacked admitted the error.

1. \( w_0 = "the" \)
2. `project(default): [\{VP, \}, \{NP, \}]`
   `getMoved(): nothing`
   `getPOS(\text{the}): [\{NP +D, \text{the}\}]`
3. `setAttachment([\{VP\}]): [\{VP\}]`
   `setAttachment([\{NP\}]): [\{NP\}]`
4. `merge([\{VP\}], [\{NP +D, \text{the}\}]): [\{VP\}[\{NP +D, \text{the}\}]`
   `merge([\{NP\}], [\{NP +D, \text{the}\}]): [\{NP +D, \text{the}\}]`
5. `move([\{VP\}[\{NP +D, \text{the}\}]]): nothing`
   `move([\{NP +D, \text{the}\}]): nothing`
Getting asymmetries with PMG-pa
Subject relatives in head initial languages
a. The reporter who attacked the senator admitted the error.
b. The reporter who the senator attacked admitted the error.

1. \( w_1 = \text{"reporter"} \)
2. project(\( w_1 \)): nothing
3. getPOS(\( w_1 \)): [NP N reporter]
4. setAttachment([VP [NP +D the]]): [NP +D the]
merge([NP +D the], [NP N reporter]): [VP [NP +D the N reporter]]
Getting asymmetries with PMG-pa

Subject relatives in head initial languages
a. The reporter who attacked the senator admitted the error.
b. The reporter who the senator attacked admitted the error.

1. \( w_2 = \) “reporter”
2. project(\( w_2 \)): nothing
3. getPOS(\( w_2 \)): \([NP \, N \, \text{reporter}]\)
4. setAttachment(\( [VP \, [NP +D \, \text{the}] \] \)): \( [NP +D \, \text{the}] \)
5. merge(\( [NP +D \, \text{the}] \), \( [NP N \, \text{reporter}] \)):
   \( [VP \, [NP +D \, \text{the} \, \text{reporter}] \] \)
6. move(\( [NP +D \, \text{the} \, \text{reporter}] \)): A-buffer[\( w_2 \)]

Getting asymmetries with PMG-pa

Subject relatives in head initial languages
a. The reporter who attacked the senator admitted the error.
b. The reporter who the senator attacked admitted the error.

1. \( w_2 = \) “who”
2. project(\( w_2 \)): nothing
3. getPOS(\( w_2 \)): \([NP \] \)
4. setAttachment(\( [VP +C \, \text{who}] \)): \( [NP \ldots \text{reporter}] \)
5. merge(\( [NP +D \, \text{the} \, \text{reporter}] \), \( [NP N \, \text{reporter}] \)):
   \( [VP \, [NP +D \, \text{the} \, \text{reporter}] \] \)
6. move(\( [NP +D \, \text{the} \, \text{reporter}] \)): A-buffer[\( w_2 \)]

Getting asymmetries with PMG-pa

Subject relatives in head initial languages
a. The reporter who attacked the senator admitted the error.
b. The reporter who the senator attacked admitted the error.

1. \( w_2 = \) “who”
2. project(\( w_2 \)): nothing
3. getPOS(\( w_2 \)): \([NP \] \)
4. setAttachment(\( [VP +C \, \text{who}] \)): \( [NP \ldots \text{reporter}] \)
5. move(\( [NP +D \, \text{the} \, \text{reporter}] \)): A-buffer[\( w_2 \)]

Getting asymmetries with PMG-pa

Subject relatives in head initial languages
a. The reporter who attacked the senator admitted the error.
b. The reporter who the senator attacked admitted the error.

1. \( w_2 = \) “who”
2. project(\( w_2 \)): nothing
3. getPOS(\( w_2 \)): \([NP \] \)
4. setAttachment(\( [VP +C \, \text{who}] \)): \( [NP \ldots \text{reporter}] \)

Getting asymmetries with PMG-pa

Subject relatives in head initial languages
a. The reporter who attacked the senator admitted the error.
b. The reporter who the senator attacked admitted the error.

1. \( w_2 = \) “who”
2. project(\( w_2 \)): nothing
3. getPOS(\( w_2 \)): \([NP \] \)
4. setAttachment(\( [VP +C \, \text{who}] \)): \( [NP \ldots \text{reporter}] \)

Getting asymmetries with PMG-pa

Subject relatives in head initial languages
a. The reporter who attacked the senator admitted the error.
b. The reporter who the senator attacked admitted the error.

1. \( w_2 = \) “who”
2. project(\( w_2 \)): nothing
3. getPOS(\( w_2 \)): \([NP \] \)
4. setAttachment(\( [VP +C \, \text{who}] \)): \( [NP \ldots \text{reporter}] \)

Getting asymmetries with PMG-pa

Subject relatives in head initial languages
a. The reporter who attacked the senator admitted the error.
b. The reporter who the senator attacked admitted the error.

1. \( w_2 = \) “who”
2. project(\( w_2 \)): nothing
3. getPOS(\( w_2 \)): \([NP \] \)
4. setAttachment(\( [VP +C \, \text{who}] \)): \( [NP \ldots \text{reporter}] \)
Subject relatives in head initial languages

1. `w` = "who"
2. `project(node): nothing`
3. `getMoved()`: `i = 0`
4. `setAttachment([s, reporter])`: `[s, reporter]`
5. `move([s, reporter], [s, who])`

- The reporter who attacked the senator admitted the error.
- The reporter who the senator attacked admitted the error.

Getting asymmetries with PMG-pa

1. `w` = "who"
2. `project(node): nothing`
3. `getMoved()`: `i = 0`
4. `setAttachment([s, reporter])`: `[s, reporter]`
5. `move([s, reporter], [s, who])`

- The reporter who attacked the senator admitted the error.
- The reporter who the senator attacked admitted the error.
Getting asymmetries with PMG-pa

Subject relatives in head initial languages

a. The reporter who attacked the senator admitted the error.
b. The reporter who the senator attacked admitted the error.

1. project(who): nothing
2. getMoved(): ⊥
   getPOS(attacked):
   [VP +T =NP attacked]
3. setAttachment([NP −reporter [NP who [NP the reporter]]]): ⊥
4. merge([VP +C who [NP +D the N reporter [VP +C who [NP +D the N senator]]]], [NP +D the N senator]): ⊥
5. move([VP +C who [NP +D the N reporter [VP +C who [NP +D the N senator]]]]): A-Buffer<NP who [NP the senator]>

1. project(senator): nothing
2. getMoved(): ⊥
   getPOS(attacked):
   [VP +T =NP attacked]
3. setAttachment([NP −reporter [NP who [NP the reporter]]]): ⊥
4. merge([VP +C who [NP +D the N reporter [VP +C who [NP +D the N senator]]]], [NP +D the N senator]): ⊥
5. move([VP +C who [NP +D the N reporter [VP +C who [NP +D the N senator]]]]): A-Buffer<NP who [NP the senator]>

61

Today’s key concepts

 On CFG-rules inefficiency:
- They are language specific
- Too many rules are difficult to control
- They can’t be possibly learned (explanatory adequacy flaw)

 Alternatives to CFG-rules:
- Principle and Parameters (Fong 1991, but principles are inefficient from the computational point of view)
- Minimalist Grammars (Stabler 2007, but merge and move operate in opposition to the parsing direction; deductive parsing is not psycholinguistically plausible)
- Phase-based Minimalist Grammars (Chesi 2004-15, top-down derivations are cognitively plausible and can Merge and Move can be implemented this way; locality can be captured too)