Interfacing morphophonology and morphosyntax in constraint-based processing

Ann Copestake
University of Cambridge

Thanks to Emily Bender and other colleagues on DELPH-IN for many useful discussions and Ben Waldron for integration efforts.

Introduction

Morphology in constraint-based grammar-engineering systems

- Efficiency of processing and grammar development
- Range of languages
- Built-in morphology and external components

Overview:

1. Morphophonology, morphosyntax and syntax
2. Linguistic adequacy, efficiency and modularity
3. Three (or four) approaches to morphology
4. Morphology in the DELPH-IN LKB system

Components

- Morphophonology: what morphemes correspond to the input token?
- Morphosyntax: what is the syntactic (and semantic) structure corresponding to the morphemes?

Interface between morphophonology and morphosyntax:

- Nature of morpheme (e.g., +ed vs PAST)
- Nature of interaction between morphophonology and morphosyntax: morphotactics (constraints on morpheme sequences) etc
- Morphophonology and the lexicon

Morphology and syntax

Traditional discussions ignore tokenisation and morphology:

```
S -> NP VP
VP -> V
V -> sleep, sleeps, dream, dreams ...
NP -> Kim
```

1. Implicit assumption of separator (space) between daughters.
2. Morphology ignored or ‘simply’ to build full-form lexicon.
   - Processing complexity but not complexity of learning.
   - Identifying codes and abbreviations require analysis: *DTDs*, *401ks*, *N-ethoxycarbonylmethylation*
   - Indefinite application of affixation: *antidisestablishmentarianism*, *antiantidisestablishmentarianism*

Morphology and tokenisation in a CFG?

```
S -> NP sep VP
VP -> V
V -> Vstem
V -> Vstem 3PSG
Vstem -> "sleep", "dream",  ...
3PSG -> "s"
sep -> " "
```

But not concatenative, even for English:

```
V -> Vstem-y 3PSG-y
Vstem-y -> "carr"
3PSG-y -> "ies"
V -> Vstem-y non3PSG-y
non3PSG-y -> "y"
```

Standard NLP parsing and morphology

1. Simple tokeniser splits input into tokens ($t_1 \dots t_n$).
2. Morphology module (e.g., finite state transducer, string unification) produces morphemes from tokens (morphophonology) and produces some representation(s) of the whole token based on morphemes (morphosyntax).
3. Syntax operates on morphology output using a different formalism (concatenative).

Morphotactics: in morphophonology or morphosyntax?
Two-level morphology: continuation lexicon.
FSA for affix sequences, but the *(((en)joy)able)* problem.

Considerations for NLP morphology

- Bidirectional
- Tokenisation must allow for ambiguity.
- Splitting the tokens into morphemes has to be done without knowledge of the stem.
- Lexical rules with no morphological effect have to be allowed, potentially in between affixation operations for derivational effects.
- Multiword entries of various types have to be allowed for.
- Compounds without spaces (multi-stem tokens).

Assumptions for this talk

1. Constraint-based (feature structure) formalism for syntax (and semantics).
2. Tokenisation is a preliminary step involving no lexical resources (for written text in languages with appropriate conventions) or by an external module (e.g., Chasen). Tokenisation may return a lattice.
3. Morphophonology requires different formalisation from morphosyntax.
4. Stems can (generally) be looked up in a lexicon.
5. Derivational morphology must be handled as well as inflectional.
6. Morphosyntax is more than specialisation.
7. Derivational and inflectional morphology use same formalism.

Options for morphology for a constraint-based approach

1. Standalone morphology: set of signs for each original token (processor handles morphophonology and morphosyntax).
2. Word syntax: retokenisation, with one token/sign per morpheme (possibly with some bracketing).
3. Rule-based: affixes correspond to rules. Morphophonology specifies rule applications.

Standalone morphology

- Traditional NLP assumption.
- Highly modular solution from a processing perspective: $t_1, t_2 \ldots t_n$ to $\{FS_{1i} \ldots FS_{1m}\}$, $\{FS_{2i} \ldots FS_{2m}\}$ $\ldots \{FS_{ni} \ldots FS_{nm}\}$
- Bad for grammar development in a constraint-based framework because:
  - complexity of signs makes interface difficult
  - constraint-based approach desirable for morphosyntax
  - similarity between morphologically marked operations and unmarked ones (e.g. noun-verb conversion)
- Formally this can be treated as the full-form lexicon solution

Word syntax

- Tokenisation $t_1, t_2 \ldots t_n$ is converted to $t'_1, t'_2 \ldots t'_m$ where t' tokens could be strings (to be looked up in a lexicon) or feature structures.
  ‘I walk’ $\rightarrow$ ‘I’ ‘walk’ ‘+ed’
- Affixes are lexical items, morphosyntax like normal syntax.
- No inherent linkage between the original tokenisation and the (re)construction of the structure.
  ‘transformational’ ‘grammar’ ‘+ian’ could be bracketed (‘transformational’ ‘grammar’) ‘+ian’.
- No constraints on splitting possibilities (morphotactics) from morphosyntax.
- Formally: tokenisation T' corresponds to input T iff there is some valid transduction from T to T' etc. Syntax defined in terms of T'

Affixes as constraints on rule application

- Range of possible approaches including the Pollard and Sag sketch and all variants of the LKB built-in morphology.
- Formally, morphophonology constrains application of rules/constraints.
- Morphotactics mostly arises from morphosyntax rules.

$$\left[\begin{array}{l} \textbf{base} \\ \text{PHON}\ \boxed{1} \\ \text{3RDSNG}\ \boxed{2} \\ \text{SYN}\left[\text{LOC}\left[\text{SUBCAT}\ \boxed{3}\right]\right] \\ \text{SEM}\left[\text{CONT}\ \boxed{4}\right] \end{array}\right] \longmapsto \left[\begin{array}{l} \textbf{3rdsng} \\ \text{PHON}\ \mathit{f3rdsng}(\boxed{1}, \boxed{2}) \\ \text{SYN}\left[\text{LOC}\left[\text{SUBCAT}\ \boxed{3}\right]\right] \\ \text{SEM}\left[\text{CONT}\ \boxed{4}\right] \end{array}\right]$$

Third Singular Verb Formation, from Pollard & Sag, 1987:210-213

- output PHON/ORTH cannot always be determined from PHON/ORTH of stem: *lie*/*lay*/*lied*

- not a function (at least in practical parsing systems): *dreamed/dreamt*

Affixes as constraints in the LKB

- Morphophonology associated with morphosyntax rules.
- Implementation of interaction: morphophonology specifying partial derivation trees. Schematically: (past-rule ... ‘tango’) Morphosyntax rules (i.e., rules with morphophonological effects) only applied as licenced by a partial tree.
- Partial tree to allow for zero-morph rules: (past-rule (noun-to-verb-rule ‘tango’))
- Tree representation allows for compounding.

Current LKB morphophonology

Built-in LKB morphophonology is a simple variant of string unification (e.g., Calder, 1987), originally developed and implemented by Bernie Jones (1991), reimplemented by AAC 2005.

```
%(letter-set (!t bcdfghjklmnpqrstvwxz))

past-v_irule :=
%suffix (* ed) (!ty !tied) (e ed)
past-verb.
```

Patterns specified as affix type, followed by series of subpatterns (underlying surface).

* matches anything

!char is a letter set / bound value

Alternative: various forms of external morphophonology are supported, including standalone and word syntax approaches. Emily Bender and students have experimented with interface to XFST.

LKB approach: Implementation details

Pre-2005: morphophonology produces whole partial tree in one go. But then morphosyntax doesn't constrain morphophonology possibilities.

Current: morphophonology interleaved with morphosyntax rule filter. e.g., exclude `(3sg ... (-er ...` if there is no noun-to-verb rule applying to output of -er rule.

- Many times faster than old approach for Norwegian and similar languages, since constraints in morphosyntax rules immediately constrain morphophonology.
- Pre-computed rule filter table: can morphosyntax rule A feed rule B via 0 or more zero-morph rules?
- Extension of the syntax rule filter originally in PAGE.

- Possible extension: allow lexical types to be plugged in, to filter partial trees immediately a stem is discovered.

Interaction of morphophonology and the lexicon/orthography

Option 1: Morphophonology depends on the rule and the spelling of input entirely.

Option 1A: Morphophonology directly specified by rules, e.g. LKB % mechanism.

```
past-v_irule :=
%suffix (* ed) (!ty !tied) (e ed)
past-verb.
```

% specification

But this leads to redundancy: e.g., same pattern for 3sg and plural.

Interaction of morphophonology and the lexicon/orthography: 2

Option 1B: Morphophonology patterns reified as types which are specified in rules (see morph-example grammar)

```
+ed := suffix &
[ PATTERN "(* ed) (!ty !tied) (e ed)" ].
past-v_irule := past-verb &
[ AFFIXATION +ed ].
```

Allows rules to share patterns, avoiding redundancy.

Gives a level of indirection for multiple affixation possibilities.

Just implemented in LKB.

Interaction of morphophonology and the lexicon/orthography: 3

Morphophonology depends on rule, spelling and the sign (*lie*/*lay*/*lied*)
Allows for lexically-specified paradigms directly (rather than via subcases of rules).

```
past-v_irule := past-verb &
[ AFFIXATION #1,
  ARGS < [ PARADIGMS.PAST #1 ] > ].

verb-paradigms := *top* &
[ PAST /l +ed,
  PSP /l +ed,
  3PSG /l +s,
  PRES-PART /l +ing ].
```

```
lie_1 := trans-verb &
[ ORTH.LIST.FIRST "lie",
  KEY.PRED "lie_1_rel",
  PARADIGMS.PAST "lay" ].

lie_2 := trans-verb &
[ ORTH.LIST.FIRST "lie",
  KEY.PRED "lie_2_rel" ].
```

Not yet implemented in LKB (requires precompilation step to associate rules with possible affixation patterns).

Irregular spellings

- In current LKB, formally (almost) equivalent to additional subpatterns, but specified in a separate file.
- Paradigm approach: lexically specified morphophonology overriding inherited morphophonology.
  Everything is in TDL, with special interpretation for patterns etc.

Alternatives

Bender and Good (2005):

- Morphophonology component maps token into new token with regular morphemes (e.g., ‘carries’ to ‘carry+s’)
- Then trivial morphophonology plus morphosyntax as LKB
- Might be implemented by XFST
- Doesn’t allow morphosyntax to constrain morphophonology, so (presumably) separate morphotactics