Morphological Analysis Unification Grammars

Daniel Zeman

http://ufal.mff.cuni.cz/course/npfl094

Unification Grammars

- Based on
  - context-free grammars
  - **feature structures**
  - their **unifiability**
- Feature structure
  - Sort of database record, or a variable of a structured type: **`record`** in Pascal or **`struct`** in C. Description of an object, list of features.
  - features (attributes) … names of fields
  - values
  - Examples of attribute-value pairs: [number: plural], [case: nominative].

Feature Structure

entity	
NAME	FF UK
PHONE	258562

entity	
NAME	Dan
PHONE	221914225

faculty	
NAME	MFF UK
DEAN	Netuka
PHONE	221911111

POS	noun
GEN	masculine
NUM	singular
CASE	dative

POS	adjective
GEN	masculine
NUM	plural
CASE	accusative
DEG	comparative
NEG	affirmative

Feature Structure

- In general: a partial function mapping the set of features to the set of values.

type	
$\mathbf{FEATURE_1}$	$\mathbf{VALUE_1}$
$\mathbf{FEATURE_2}$	$\mathbf{VALUE_2}$
$\mathbf{FEATURE_3}$	$\mathbf{VALUE_3}$

Unifiability

- Two feature structures are **unifiable** if their values of the features they share are identical.
- Example: structures 1 and 2 are unifiable, so are 2 and 3, while 1 and 3 are not.

1
```
GENDER    masculine
NUMBER    singular
CASE      dative
```

2
```
POS       verb
NUMBER    singular
TENSE     present
```

3
```
GENDER    masculine
NUMBER    singular
CASE      instrumental
```

Unification

- **Unification** is an operation over two unifiable feature structures. It results in a new feature structure.

$$1\begin{bmatrix}\texttt{GENDER} & \texttt{masculine}\\ \texttt{NUMBER} & \texttt{singular}\\ \texttt{CASE} & \texttt{dative}\end{bmatrix} + 2\begin{bmatrix}\texttt{PERSON} & \texttt{third}\\ \texttt{NUMBER} & \texttt{singular}\\ \texttt{TENSE} & \texttt{present}\end{bmatrix}$$

$$= 3\begin{bmatrix}\texttt{GENDER} & \texttt{masculine}\\ \texttt{NUMBER} & \texttt{singular}\\ \texttt{CASE} & \texttt{dative}\\ \texttt{PERSON} & \texttt{third}\\ \texttt{TENSE} & \texttt{present}\end{bmatrix}$$

Unification as a Tool for Morphological Generation?

- Input: feature structures “lemma” and “tag”.
- Search lexicon for all structures “entry” that are unifiable with the “lemma” structure.
- For each “entry” structure found look up a “paradigm” structure that is unifiable with both the “entry” and the “tag” structures.
- Unify the corresponding structures “entry”, “paradigm” and “tag”. The resulting structure is called “form”.
- Output: for each “form” structure, concatenate its values of features “paradigm” and “suffix”.

Unification as a Tool for Morphological Generation?

- Input: feature structures “lemma” and “tag”.

```
lemma
LEMMA      háček
```

```
tag
NUMBER     plural
CASE       nominative
```

Czech noun *háček* has two meanings and belongs to two inflection classes.

ntry” structures unifiable with

```
entry
LEMMA      háček
PARADIGM   hrad
```

```
entry
LEMMA      háček
PARADIGM   pán
```

Unification as a Tool for Morphological Generation?

- For each “entry” structure find a “paradigm” structure unifiable with both “entry” and “tag”.

```
entry
LEMMA          háček
PARADIGM       hrad
```

```
entry
LEMMA          háček
PARADIGM       pán
```

```
paradigm
PARADIGM hrad
NUMBER   plural
CASE     nominative
SUFFIX   y
```

```
paradigm
PARADIGM pán
NUMBER   plural
CASE     nominative
SUFFIX   i | ové
```

Unification as a Tool for Morphological Generation?

- Unify the corresponding structures “entry”, “paradigm” and “tag”. Call the resulting structure “form”.

```
form
LEMMA      háček
PARADIGM   hrad | pán
NUMBER     plural
CASE       nominative
SUFFIX     y | i | ové
```

Unification as a Tool for Morphological Generation?

- Unification resembles database operations.
- It does not tell how the "form" structure is to be interpreted.
- Rule: **`output = form.lemma + form.suffix`**
- The rule does not solve phonological changes (and unification cannot help us with this):

instead of

**háčeky, *háčeki, *háčekové*

we want

háčky, háčci, háčkové

Unification as a Tool for Morphological **Analysis???**

- Non-unification part: find all possible affixes recognizable in the word ⇒ set of "form" structures.
- The "paradigm" structures tell us what is the set of known affixes.
- Somehow solve phonological changes (stem-final palatalization, stem-internal ablaut etc.)
- Then take the dual procedure to the generation: unify form with paradigm, and the result with the lexicon. Entries found in the lexicon are the possible analyses.
  - e.g. *běžím ("I am running")* = *běžet*(verb:*trpět*)+person(1st), ≠ *běží*(noun:*stavení*)+case(7)

Unification as a Tool for Morphological **Analysis???**

- Non-unification part: find all possible affixes recognizable in the word ⇒ set of "form" structures.
- The "paradigm" structures tell us what is the set of known affixes.
- Somehow solve phonological changes (stem-final palatalization, stem-internal ablaut etc.)
- Then use unification…
  - In fact, this is what PC Kimmo v.2 does:
  - It combines two-level morphology with a unification grammar.

Unification Morphology Grammar (UMG)

- Jan Hajič: *Unification Morphology Grammar (PhD thesis)*. Univerzita Karlova, Praha, 1994
- Stuart Shieber: *An Introduction to Unification-based Approaches to Grammar*. CSLI Lecture Notes No. 4, Stanford, California, USA, 1986

- Based on a context-free grammar.
- A feature structure is attached to each constituent (label + span).
- Rule: left-hand side (LHS) → right-hand side (RHS) := operation over feature structures.
- Operations can block a rule by requiring unifiability.
- Unification-based chart parser, PATR-II (Shieber).
- Similarly to CFGs, unification grammars were originally designed for sentence syntax analysis and subsequently applied to word analysis as well.

UMG Syntax

- LHS → RHS **:=** operation over feature structures
  - grammar rule
- **<X>**
  - non-terminal symbol X. Terminals are written without angle brackets
- **#**
  - unification operator (it also places requirement on unifiability)
- **^**
  - reference operator (it delimits non-terminals / parts of paths to the feature structure we are referencing)
- **+**
  - concatenation operator
- **|**
  - disjunction operator. A disjunction of feature structures contains all structures that fulfill the constraints (are unifiable). A disjunction can represent alternate analyses of the same string.

Example of UMG Rule

<N> → <L> := [l = <L>^l, umlaut = <L>^umlaut # no]

- Interpretation:
  - If:
    - we recognized constituent **<L>** and
    - value of the **umlaut** attribute in the feature structure attached to this constituent is "no"
  - Then:
    - we also recognized constituent **<N>** with the same span
    - we must copy the attributes **l** and **umlaut** from the feature structure of **<L>** to the feature structure of **<N>**

Theoretical View of the Lexicon

- A rule that generates the empty string but it provides a gigantic feature structure with the entire lexicon in it.
  - **<LEX> → "" :=**
    [stem=mat, hw=matka, pos=N, x=zn6e] |
    [stem=atom, hw=atom, pos=N, x=hd1] |
    [stem=nov, hw=nový, pos=A, x=reg] |
    [stem=prac, hw=pracovat, pos=V,
    x=ovatn] |
    …;

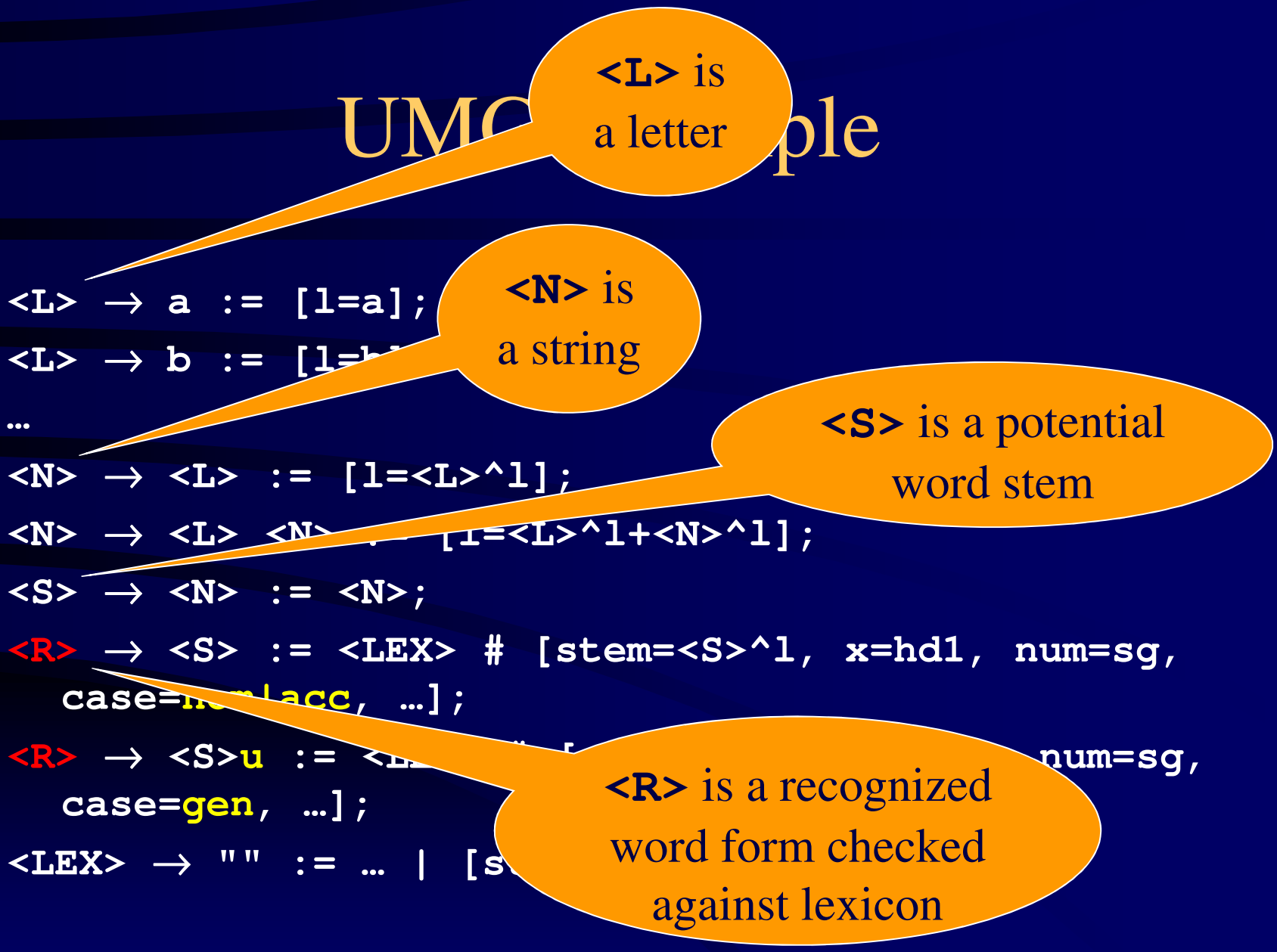
Theoretical View of the Lexicon

- How the lexicon is bound to the rest of the grammar:
  - **<R> → <S>u <LEX> := <LEX> # [x=hd1, stem=<S>, case=gen|dat|loc, num=sg]**
    - The rule describes formation of singular noun genitive, dative and locative according to the Czech masculine paradigm **hd1** *(hrad)*.
    - **R** represents a word unified with the lexicon.
    - **S** is the part of the input that corresponds to the stem of the word. The suffix is shown literally, the **LEX** that follows it corresponds to the empty string.
    - The operation after **:=** says that we are interested in those structures from **LEX** whose stem corresponds to **S** and which encode genitive, dative or locative according to the paradigm **hd1**.
    - Those lexicon entries that pass this filter will form the set of feature structures bound to the non-terminal **R**. In addition these feature structures will contain information on number and case.

UMG Example

```
<L> → a := [l=a];
<L> → b := [l=b];
…
<N> → <L> := [l=<L>^l];
<N> → <L> <N> := [l=<L>^l+<N>^l];
<S> → <N> := <N>;
<R> → <S> := <LEX> # [stem=<S>^l, x=hd1, num=sg,
  case=nom|acc, …];
<R> → <S>u := <LEX> # [stem=<S>^l, x=hd1, num=sg,
  case=gen, …];
<LEX> → "" := … | [stem=hrad, x=hd1, …] | …
```

UMC ple

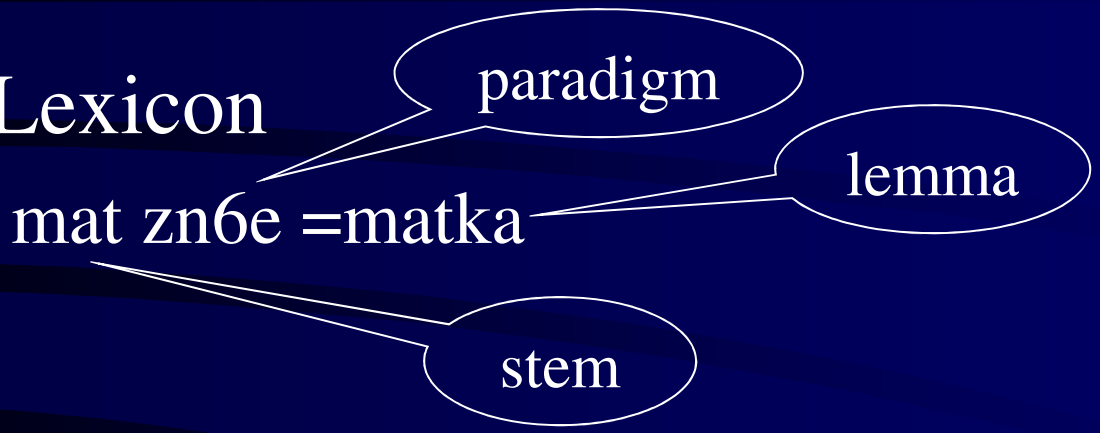

<L> is a letter
<N> is a string
<S> is a potential word stem
<R> is a recognized word form checked against lexicon
<L> → a := [l=a];
<L> → b := [l=
…
<N> → <L> := [l=<L>^l];
<N> → <L> <N> := [l=<L>^l+<N>^l];
<S> → <N> := <N>;
<R> → <S> := <LEX> # [stem=<S>^l, x=hd1, num=sg,
case= acc, …];
<R> → <S>u := <L num=sg,
case=gen, …];
<LEX> → "" := … | [s

The Lexicon in Practice

- It is not efficient to treat the lexicon as part of grammar.
- Real implementation looks different:
  - Store the lexicon in a separate data structure with fast search access.
  - Cover rules containing **<LEX>** by the algorithm accessing the data structure.
  - Use the unifying chart parser to process the rest of the grammar.

UMG Example

- Lexicon



Typical system with many paradigms, e.g. 44 distinct paradigms correspond to the "school" paradigm *žena* (yet the paradigms don't solve the shortening of the stem-internal vowel).

UMG Example

{ paradigm = *stavení;* omitting LHS, always the same }

```
<_><í>$ := [key=<_>í, x=(st|rž), cat=[pos=n],
  morf=[infl=[pf=([gnd=n, num=sg,
  case=(nom|gen|dat|acc|voc|loc)] | [gnd=n,
  num=pl, case=(nom|gen|acc|voc)])]]];
<_><í><m>$ := [key=<_>í, x=(st|rž), cat=[pos=n],
  morf=[infl=[pf=([gnd=n, num=sg, case=ins] |
  [gnd=n, num=pl, case=dat])]]];
<_><í><c><h>$ := [key=<_>í, x=(st|rž),
  cat=[pos=n], morf=[infl=[pf=[gnd=n, num=pl,
  case=loc]]]];
<_><í><m><i>$ := [key=<_>í, x=(st|rž),
  cat=[pos=n], morf=[infl=[pf=[gnd=n, num=pl,
  case=ins]]]];
```

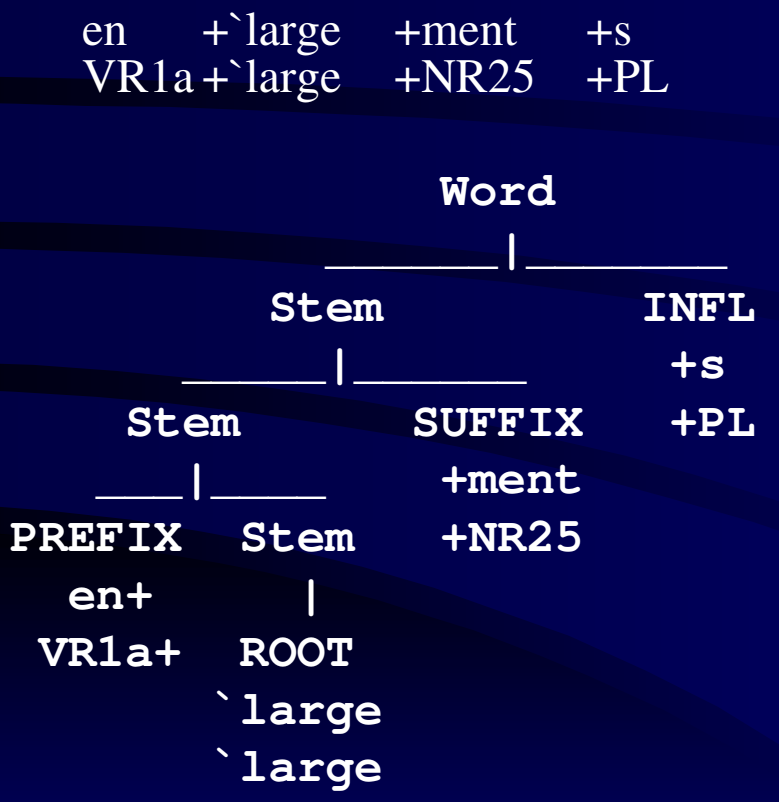
Comparison of UMG and CFG

- The feature structure contains the required output (tag) ⇒ no need for supplementary non-terminal naming convention
- At the same time, the features and their unifiability constrains rule application ⇒ no need to split non-terminals
- Disjunction of structures represents homonymous analyses
- Phonology is still an issue. Either combinatorial explosion of paradigms (UMG) or use in tandem with two-level rules (see below)

PC-Kimmo Word Grammar

- Unification grammar by Stuart Shieber. Rule syntax somewhat different from UMG, application is similar.
- lexicon
  - recognize possible morphemes in the word
- rules
  - phonological changes, esp. on morpheme boundary
- grammar
  - analysis of inter-morpheme relations
  - derivation of word features from morpheme features
  - constraints on morphotactics (what morphemes can combine and in what order)

PC-Kimmo Word Grammar




```
Word:
[ cat:    Word
  head:     [ agr:
                  [ 3sg: - ]
              number:PL
              pos:    N ]
  root:   `large
  root_pos:AJ
  clitic:-
  drvstem:- ]
```

PC-Kimmo Word Grammar

- First the old part of PC-Kimmo segments the word into morphemes.
- Then the new part parses the sequence of morphemes according to the grammar.
  - Grammar can reject some morpheme sequences.
  - It assigns an interpretation (a feature structure) to the sequences it accepts. The old PC-Kimmo was able to gloss morphemes but it could not tell how to combine the morpheme glosses into the interpretation of the whole word (e.g. that the suffix *–able* changes a verb to an adjective).
- A grammar rule looks like this:
  - Word –> Stem INFL
    <Stem head pos> = <INFL from_pos>
    <Word head> = <INFL head>

Grammar Rule

- **`Word -> Stem INFL`**
  `<Stem head pos> = <INFL from_pos>`
  `<Word head> = <INFL head>`

- The rule cannot be used if the feature **`pos`** of the substructure **`head`** of the morpheme **`Stem`** is not equal to the feature **`from_pos`** of the morpheme **`INFL`**.
  - The morpheme symbols are pre-terminals and they correspond to the names of sublexicons where the morphemes were found.

- If the rule is used it shall copy the value of the **`head`** feature from the **`INFL`** constituent to the **`head`** feature of the **`Word`** constituent.

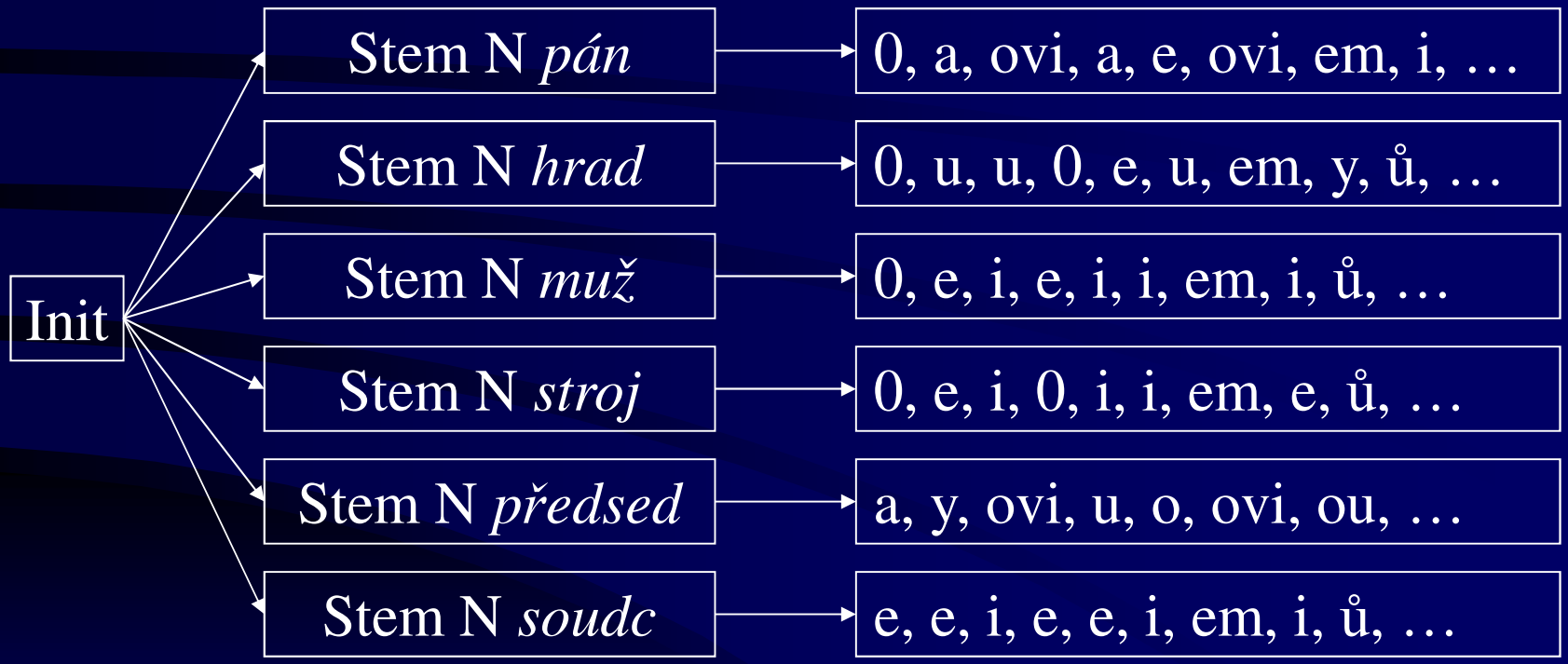
Grammar Rule

- **RULE** <rule>
  <rule constraints>
- Left-hand side is separated from the right-hand side by **->** or **=**.
- **RULE Stem_1 = Stem_2 SUFFIX**
- **X** represents any terminal or non-terminal.
- Characters **()[]{}<>=:/** are special.
  - Underscore is used only to append an index to a symbol.
- Left-hand side of the first rule is the start symbol of the grammar.
- **N = Nstem {Sing / Plural}**

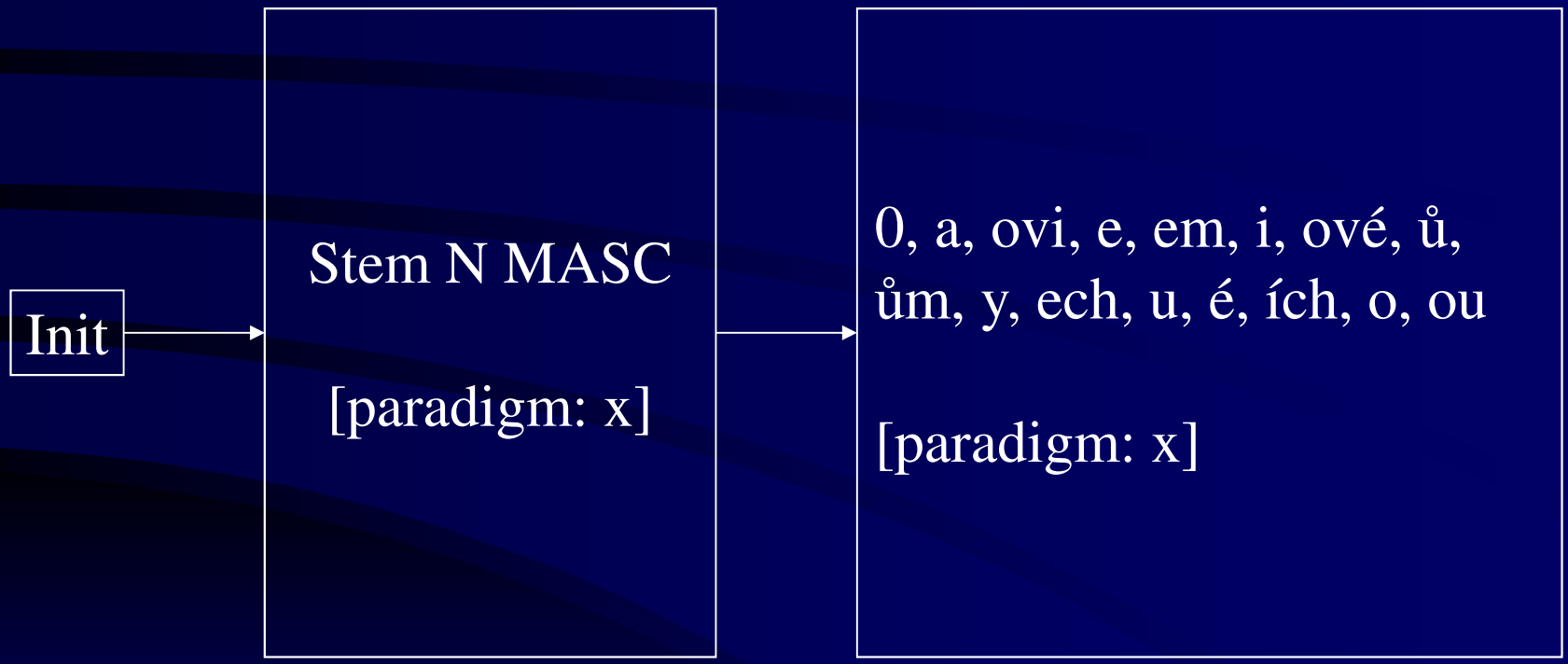
Advantages of the Grammar

- Czech examples:
  - Grammar blocks combination of a stem belonging to the *žena* paradigm with a suffix belonging to the *růže* paradigm.
  - It can even check long-distance dependencies such as:
    - nejchytřejší
  - Take feminine noun *žena (woman)*. Derive from it the possessive adjective *ženin (woman's)*. The gender changed from feminine to masculine (the suffix says that the possessed object is masculine). The grammar can store the original gender as the lexical possessor's gender.

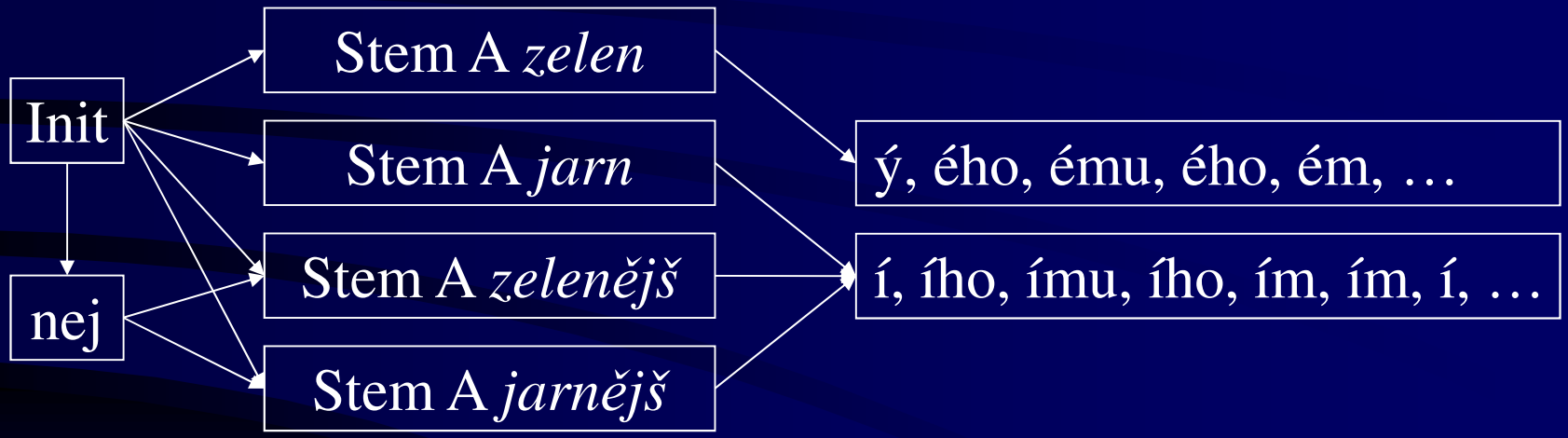
Without Grammar



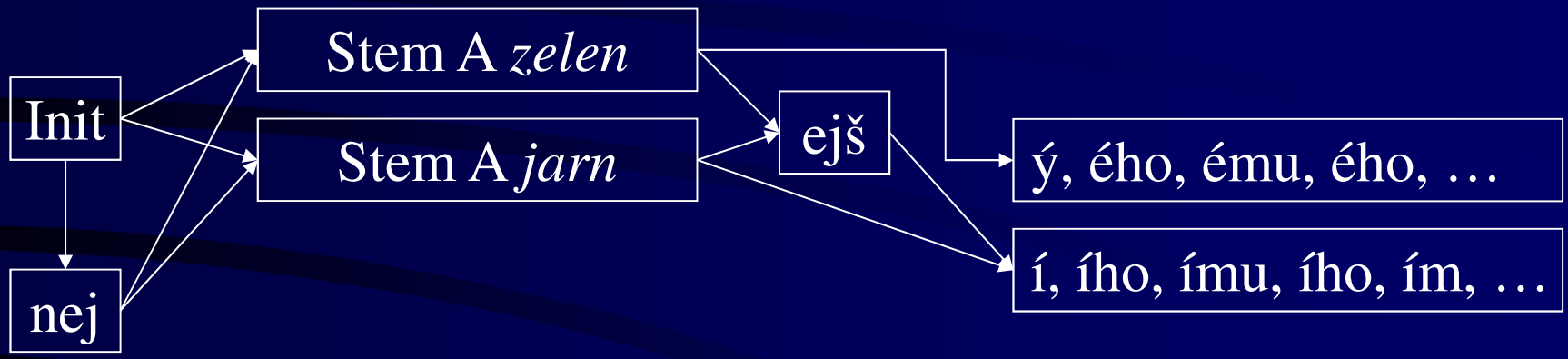
With Grammar



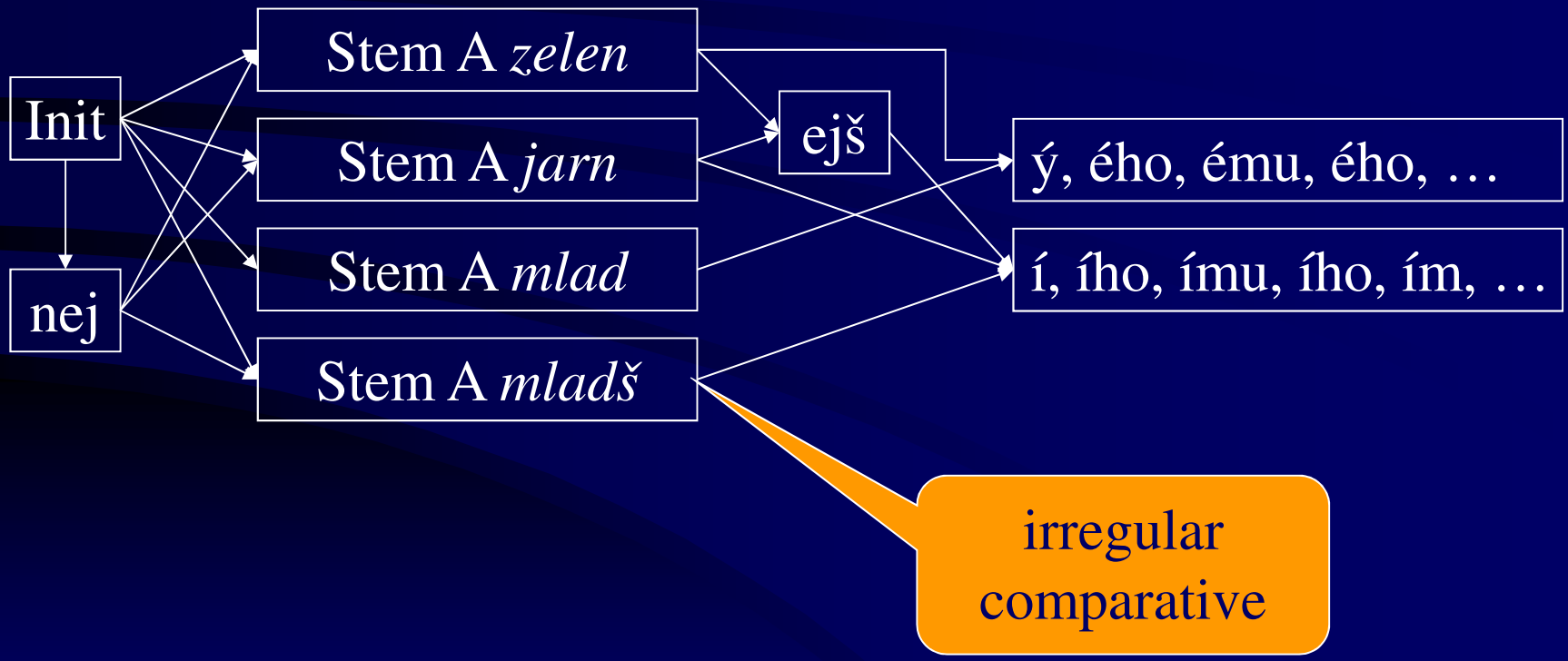
Without Grammar



With Grammar



With Grammar



Grammar Cannot Interact with Phonology

- Phonological rule of consonant softening in Czech imperative:
  - meteš (“you sweep”) → met’(me,te) (“sweep!”)
  - t:t’ ⇔ _ +:0 λ:0 *or* m:m e:e *or* t:t e:e
- The rule must not apply in genitive plural form of feminine nouns:
  - kóta (“spot elevation”) → *kót’
- Phonological rules cannot read the feature structures to constrain their application.
- There have been extensions other than PC-Kimmo that combined phonological rules with feature structures, e.g. Trost (1990).

Automatic Features

- Every lexicon entry automatically receives the following features:
  - cat = name of sublexicon (\lx)
  - lex = morpheme, lexical string (\lf)
  - gloss = gloss from the lexicon entry (\gl)

Assigning Values to Features

- Abbreviations of feature assignments
  - If we are going to assign a value to thousands of lexicon entries we want it to be as short as possible.
  - **LET** <shortcut | category> **be** <definition>
  - e.g.
  - **Let pl be [number: PL]**
  - **Let pl be <number> = PL**
  - **Let 3sg be [tense: PRES**
    agr: 3SG]
- Disjunction:
  - **Let sg/pl be {[number:SG][number:PL]}**
  - **Let sg/pl be <number> = {SG PL}**
- Default values:
  - **Let N be <number> = !SG**
  - Unless someone explicitly assigns the value of number to a noun, the noun is assumed to be in singular.

Lexical Rules

- Not shortcuts but systematic transformation of features for groups of lexicon entries. They transform a feature structure to another one.
- **`DEFINE`** <lexical rule name> **`as`** <mapping>
- The example in the on-line documentation is invalid.
- When the analysis is done and the feature structure for the whole word is ready, we can apply a lexical rule that will modify the structure.

Parameter Setting

- **PARAMETER** <name> **is** <value>
  - **Parameter Start symbol is Word**
  - **Parameter Attribute order is cat head root**
    - In which order shall PC-Kimmo display the features?
  - **Category feature** (default: **cat**)
  - **Lexical feature** (default: **lex**)
  - **Gloss feature** (default: **gloss**)
    - What are the names of important features with special meaning?

PC Kimmo Demonstration

- `r ženě`
- Synthesis (new in PCK v. 2, but grammar is still not necessary)
- `l synthesis-lexicon cs.lex`
- `s N(žena) +SG+LOC`
- If a grammar is available it may block synthesis of invalid combinations
- Unfortunately we cannot generate from a feature structure