Error diagnosis and feedback generation for language tutoring systems

ELSNET Summer School, Lille 2003

Wolfgang Menzel
menzel@informatik.uni-hamburg.de

Fachbereich Informatik
Universität Hamburg

Content

- Overview
- Instance-based methods
- Constraint satisfaction
- Parsing with error rules
- Error-sensitive parsing
- Summing up

Overview

- Foreign language learning
- Natural language processing
- Diagnosis

Foreign language learning

- different situations
  - classroom vs. distant learning
  - within vs. outside of a language community
  - supervised vs. self-learning
- different motivations
  - extrinsic: curriculum, job requirements, ...
  - intrinsic: joy, curiosity, ...
- different goals
  - occasional communication
  - regular use

Foreign language learning

- different pedagogical methods
  - grammar-centred
  - immersive
  - pattern-drill
  - ...

Foreign language learning

- different communicative modes
  - monolingual

	spoken	written
passive	listening	reading
active	hearing	writing

  - crosslingual
    - translation / interpretation

Foreign language learning

- different aspects of language learning
  - vocabulary
  - grammar
  - usage
  - cultural background
  - spelling
  - pronunciation
  - fluency
  - ...

Foreign language learning

- different expectations for using computer in language learning
  - make language learning more efficient (faster, less expensive, ...)
  - raise motivation of the learner
  - raise awareness of language and language structure
  - ...
- technology-driven vs. goal-driven

Foreign language learning

- the big mismatch:
  didactic goals vs. potential of multimedia technology

language learning	multimedia technology
interaction	presentation
active production	passive consumption
oral interaction	written interaction
verbal expression	non-verbal gestures (mouse, joystick)

Foreign language learning

- multiple-choice exercises don't suffice

  Consequently, a session with one of the currently fashionable multimedia packages is like trying to learn to speak a foreign language from a tutor who is stone deaf . . .

  The reason is that there exists no software intelligent enough to process ill-formed sentences other than by simply refusing them.

  BRIAN FARRINGTON (Eurocall 2000)

Natural language processing

- Can NLP help closing the gap?
- A mismatch again?

language learning	NLP technology
goal-directed communication	form aspects
language use	repetition of patterns
high quality standards	approximative results
highly deficient language	normative language
controlled situations	limited coverage

Natural language processing

- basic technologies
  - morphological analysis
  - parsing
  - semantic construction
  - transfer
  - generation
  - error detection/diagnosis ?

Natural language processing

- applications
  - spell and grammar checker
  - NL query systems
  - information retrieval/extraction
  - summarization
  - machine translation
  - computer-assisted language learning ?

Diagnosis

- four steps of error diagnosis
  - error detection
    - Was something wrong with the input utterance?
  - error localisation
    - Where went something wrong?
  - error identification
    - What went wrong?
  - error evaluation
    - How serious is the error?

Diagnosis

- error explanation
  - selection of error descriptions
    - Which errors can be explained?
      economically/understandable
    - Which errors should be explained?
      first/later/never
  - generation of feedback messages
    - How can an error be explained?
      verbally/graphically/by example/ . . .

Diagnosis

- different approaches to diagnosis
  - medicine: experience-based
    - if symptom then disease
    - deductive reasoning
  - medicine: explanation-based
    - if disease then symptom
    - abductive reasoning
  - technical devices: model-based
    - if not(faulty) then normal behaviour
    - non-monotonic reasoning: retract assumptions on non-normal behaviour

Diagnosis for NL utterances

- depth of diagnosis
  - error detection
  - error localisation/identification
- exercise types
  - restricted input (structure known)
  - free-form input (structure unknown)
- level of description
  - for individual instances
  - generalized for (possibly huge) classes of similar cases

Diagnosis for NL utterances

- knowledge available
  - on errors (non-normal behaviour)
  - on correctness (normal behaviour)
- diagnostic methods
  - experience-based (error centered)
    - anticipation of erroneous utterances
    - parsing with error rules
  - model-based (positive knowledge)
    - flexible matching of sequences
    - constraint satisfaction
    - error-sensitive parsing

Error detection: instance-based

- non-monotonic reasoning:
  - compile a collection of correct utterances (solutions to a given exercise)
  - try to identify the given solution in the collection
  - if impossible, signal an error
- problem:
  - *language production* is a creative process
  - except for trivial cases no comprehensive compilation of *correct solutions* can be expected

Error detection: generalized

- generalised descriptions of NL utterances: grammars
- non-monotonic reasoning again
  - compile a grammar for generating all correct utterances (solutions to a given exercise)
  - check if the grammar covers the solution (parsing)
  - if not, signal an error
- problem:
  - making a grammar distinguishing precisely between correct and faulty utterances is difficult
  - all utterances not covered by the grammar are flagged as being wrong

Error localisation/identification

- **error anticipation** for individual cases
  - compile a collection of typical student errors together with possible explanations
  - identify the error case and give the associated explanation
- problem:
  - *making errors* is a highly creative process
  - except for trivial cases no comprehensive compilation of *error cases* can be expected

Error localisation/identification

- **error anticipation** for *classes* of similar cases (error rules)
  - compile a collection of rules describing classes of typical student errors
  - identify the error case and give the associated explanation
- problem:
  - *making errors* is a highly creative process
  - except for some cases no comprehensive compilation of *error rules* can be expected

Error localisation/identification

- **anticipation-free techniques**: constraint satisfaction
- diagnosis
  - compile a collection of conditions to be fulfilled
  - check if the solution provided by the student violates one or several constraints
  - if so, report the corresponding error(s)
- problem:
  - all constructions not violating a constraint are accepted as being correct
- works best, if the structure of the utterance is already known (e.g. restricted by the exercise)

Error localisation/identification

- **anticipation-free techniques**: constraint retraction / relaxation
  - modify a parser in a way that allows it to accept ill-formed utterances (constraint retraction) and diagnose errors in them (error sensitive parsing)
  - derive error explanations from the error handling mechanisms
- problem:
  - how to control the huge space of possible parsing and error hypotheses? (→ constraint relaxation)

Diagnosis

- the landscape

	instance-based	generalised methods	
		restricted input	"free-form" input
anticipation-based	data-base lookup	error rules	parsing with error rules
anticipation-free	flexible matching	constraint satisfaction	error-sensitive parsing

Diagnosis

- usually error detection is a byproduct of error localization/identification
- instance-based methods have little to do with NLP
  - but: NLP is only a means not an end in itself
  - deliberation necessary: Are NLP techniques worth the effort?

Diagnosis

- where parsing and diagnosis meet:
  - combining language understanding and error sensitivity
- the vision: develop a system
  - you can communicate with
  - while solving a close-to-reality task
    - shopping
    - travel planning
    - seeking information
  - and receiving hints for improving your language capabilities upon request.

Diagnosis

- a road map:
  from simple toy systems to more ambitious goals
  - allowing "free-form" input in highly restricted environments
    - fill-in-the-gap exercises
    - answering strongly focussed questions
    - produce descriptions for a pictorial stimulus
  - gradually remove the restrictions
    - lexical choice
    - syntactic flexibility
    - broader variety of speech acts

Diagnosis

- different tasks are closely tied together
  - scenario design: What to talk about?
  - lingware development: Which linguistic knowledge is required?
  - courseware development: What kind of explanations, examples, background information etc. is needed?
  - software development: Which strategies and algorithms can be used to produce the desired results?
  - user interface design: How much visual support? Speech input?

Diagnosis

- the successful system will
  - profit from the limitations of a carefully designed communication scenario
  - which are used to cover the shortcomings of the NLP technology and
  - which can motivate the inevitable limitations of system resources (dictionary, grammar, etc.)

Diagnosis

- the successful system avoids empty promises
  - better a small but reliable solution
  - quality matters!
  - quick-and-dirty systems will cause dissatisfaction
  - playing games with a student's inncocent trust is irresponsible

The course

- outline
  - instance-based methods
    - with error anticipation
    - without error anticipation
  - constraint satisfaction
  - parsing with error rules
  - error-sensitive parsing
    - constraint retraction
    - flexible parsing
    - parsing with weighted constraints

The course

- what the course is about:
  - error diagnosis
  - feed back generation
- what the course is not about:
  - courseware authoring
  - development of multimedia products
  - web technology

The course

- the practical exercises
  - goal:
    - developing small web-based tutoring demos
    - in different application areas
    - for different european languages
  - students with little programming experience
    - using existing software modules
    - develop lingware content for different applications
  - students with richer technical background
    - modify the existing modules to add more functionality

Instance-based methods

	instance-based	generalised methods	
		restricted input	”free-form” input
anticipation-based	**data-base lookup**	error rules	parsing with error rules
anticipation-free	flexible matching	constraint satisfaction	error-sensitive parsing

Instance-based methods

- Error detection
- Error anticipation
- Flexible matching
- A simple web-interface

Error detection

- store possible correct solutions in a database
- check the student solution against the entries in the database
- report an error if no matching entry found

Error detection

- the knowledge:
  - a collection of exercise items:
    - *Build a sentence from the following words: ”John, sleep”*
    - *Fill in the gap with the words given in brackets: ”John ... her yesterday.” (meet)*
  - a collection of possible correct student solutions
    - one solution per item
    - multiple solutions per item

Error detection

- Example 1.1a: agreement: single best solution

```
exercise(agreement1,'Build a sentence in the
   present tense using the following words.').
item(agreement1,1,'Mary, sleep',
  [[[mary, sleeps]]]).
item(agreement1,2,'I, drink',
  [[[i, drink]]]).
item(agreement1,3,'he, walk',
  [[[he, walks]]]).
```

- start the demo

Error detection

- Example 1.1b: agreement: multiple correct solutions

```
exercise(agreement1,'Build a sentence in the
   present tense using the following words.').
item(agreement1,1,'Mary, sleep',
  [[[mary, sleeps]],
   [[mary, is, sleeping]]]).
item(agreement1,2,'I, drink',
  [[[i, drink]],
   [[i, am, drinking]]]).
item(agreement1,3,'he, walk',
  [[[he, walks]],
   [[he, is, walking]]]).
```

- start the demo

Error detection

- Example 1.2: tense

```
exercise(tense1,'Build an appropriately tensed
   sentence using the following words.').
item(tense1,1,'he, be, at, home, yesterday',
   [[[yesterday, he, was, at, home]],
    [[he, was, at, home, yesterday]]]).
item(tense1,2,'he, be, at, school, tomorrow',
   [[[tomorrow, he, will, be, at, school]],
    [[he, will, be, at, school, tomorrow]]]).
item(tense1,3,'he, read, now',
   [[[now, he, is, reading]],
    [[he, is, reading, now]]]).
```

- start the demo

Error detection

- a simple tutoring program (shell):

```
do_exercise(Name) :-
  exercise(Name,Prompt), write_ln(Prompt),
  item(Name,_,_,Task,Solutions),
  tab(3),write_ln(Task),
  read_utt(Utterance),
  (member([Utterance],Solutions) ->
    write_ln('well done') ;
    write_ln('sorry'))),
  nl, fail.
do_exercise(_) :-
  write_ln('No more items. Bye.').
```


Error detection

- observations:
  - the knowledge can easily be contributed (low level knowledge)
  - there is plenty of redundancy in the data base
  - free-form input will be unfeasible
  - no diagnostic information available
    - only error detection
  - processing is extremely simple (database lookup)

Error detection

- from a logical perspective: non-monotonic reasoning
  - If you cannot prove the solution to be correct, assume it is wrong.
  - but: any collection of correct solutions is by definition incomplete

Error detection

- possible improvements
  - in addition to correct answers try to also anticipate student errors
  - instead of a simple database look-up try to flexibly match the student input with the model solutions

Error anticipation

- the knowledge:
  - a collection of exercise items
  - a collection of possible correct student solutions
  - a collection of possible erroneous student solutions together with individual error explanations
    - `'I drinks' -> 'agreement error'`
    - `'Now go we.' ->`
      `'linear ordering problems'`

Error anticipation

- Example 2.1a: agreement

```
item(agreement1,1,'Mary, sleep',
  [[[mary, sleeps]],
   [[mary, is, sleeping]],
   [[mary, sleep],agree],
   [[mary, sleeping],progressive],
   [[sleeps, mary],subject_first]]).
item(agreement1,2,'I, drink',
  [[[i, drink]],
   ...
   [[i, drinking],progressive],
   [[drink, i],subject_first]]).
```


Error anticipation

- Example 2.1a: agreement

```
message(agreement1,agree,
   'Subject and Predicate must agree in
    person and number').
message(agreement1,progressive,
   'The progressive tense requires a complex
    verb group containing an auxiliary.').
message(agreement1,subject_first,'The subject
    must precede the verb.').
```

- start the demo

Error anticipation

- extending the tutoring shell

```
  ...
item(Name,_,Task,Solutions),
  ...
read_utt(Utterance),
(member([Utterance,Explanation],Solutions) ->
  (message(Name,Explanation,Message),
   write_ln(Message) ;
  (member([Utt],Solutions) ->
   write_ln('well done');
   write_ln('sorry'))),
nl, fail.
  ...
```


Error anticipation

- extended to include additional levels of error explanations
  - if requested give increasingly more detailed information
  - maximum level: present a correction proposal

Error anticipation

- Example 2.1b: agreement (with multiple explanations)

```
item(agreement1,1,'Mary, sleep',
  [[[mary, sleeps]],
   [[mary, is, sleeping]],
   [[mary, sleep],[agree,names]],
   [[mary, sleeping],[progressive,no_aux]],
   [[sleeps, mary],[subject_first,swap]]]).
item(agreement1,2,'I, drink',
  [[[i, drink]],
   ...
   [[i, drinking],[progressive,no_aux]],
   [[drink, i],[subject_first,swap]]]).
```


Error anticipation

- Example 2.1b: additional explanations

```
message(agreement1,names,
   'If the subject is a name the verb needs
    to be in 3rd person singular form.').
message(agreement1,no_aux,
   'An appropriate form of the auxiliary
    "to be" will be required.').
message(agreement1,swap,
   'Swap the two words around.').
```

- start the demo

Error anticipation

- Example 2.2: tense (past/items)

```
item(tense1,1,'he, be, at, home, yesterday',
   [[[yesterday, he, was, at, home]],
    [[yesterday, at, home, he, was]],
    [[he, was, at, home, yesterday]],
    [[yesterday, he, is, at, home],
       [tense,past_tense,past_was]],
    [[yesterday, he, will, be, at, home],
       [tense,past_tense,past_was],
    [[he, is, at, home, yesterday],
       [tense,past_tense,past_was],
    [[he, will, be, at, home, yesterday],
       [tense,past_tense,past_was],
    [[he, will, yesterday, be, at, home],
       [vg_together,adv_peripheral]],
    [[he, will, yesterday, at, home, be],
       [vg_together,adv_peripheral]]]).
```


Error anticipation

- Example 2.2: tense (past/messages)

```
message(tense1,tense,
   'The tense of the auxiliary is not
    appropriate').
message(tense1,past_tense,
   'When talking about yesterday you should
    use the past tense form.').
message(tense1,past_was,
   'The past tense of ''to be'' is ''was''').
message(tense1,vg_together,
   'The members of the verb group should
    not be separated by local or temporal
    adjuncts.').
message(tense1,adv_peripheral,
   'Temporal or local adverbials have to
    be placed at the beginning or the end
    of a sentence.').
```


Error anticipation

- Example 2.2: tense (future/items)

```
item(tense1,2,'he, be, at, school, tomorrow',
   [[[tomorrow, he, will, be, at, school]],
    [[he, will, be, at, school, tomorrow]],
    [[tomorrow, he, is, at, home],
       [tense,future_tense,future_will_be]],
    [[tomorrow, he, was, at, home],
       [tense,future_tense,future_will_be],
    [[he, is, at, home, tomorrow],
       [tense,future_tense,future_will_be],
    [[he, was, at, home, tomorrow],
       [tense,future_tense,future_will_be],
    [[he, will, yesterday, be, at, home],
       [vg_together,adv_peripheral]],
    [[he, will, yesterday, at, home, be],
       [vg_together,adv_peripheral]]]).
```


Error anticipation

- Example 2.2: tense (future/messages)

```
message(tense1,future_tense,
   'When talking about tomorrows you should
    use the future tense form.').
message(tense1,future_will_be,
   'The future tense of ''to be'' is
    ''will be''').
```

Error anticipation

- Example 2.2: tense (present/items)

```
item(tense1,3,'he, read, now',
   [[[now, he, is, reading]],
    [[he, is, reading, now]],
    [[he, reads, now],
       [progressive,progressive_ing]],
    [[he, was, reading, now],
       [tense,present_tense],present_is]],
    [[he, will, read, now],
       [tense,present_tense]],
    [[he, will, be, reading, now],
       [tense,present_tense,present_is]]]).
```


Error anticipation

- Example 2.2: tense (present/messages)

```
message(tense1,present_tense,
   'When talking about 'now' you should
    use the present tense form.').
message(tense1,present_is,
   'The present tense of ''to be'' is ''is''').
message(tense1,progressive,
   'Use the progressive form for events
    currently going on').
message(tense1,progressive_ing,
   'The progressive form is constructed using
    the suffix ''-ing'' and an appropriate
    form of ''to be''').
```

- start the demo

Error anticipation

- observations
  - individually tailored explanations can be provided
  - high degree of redundancy in the database
  - combinatory explosion of data-base items as the number of variants (both correct and incorrect) grows

Error anticipation

- simple form of abstraction
  - canned text for explanations:
    - message patterns with variable components which are substituted for words from the student's input

      ```
      'The past form of ' word(3)
      ' is ' past(word(3))
      ```
    - improves the utility of explanations

Flexible matching

	instance-based	generalised methods: restricted input	generalised methods: "free-form" input
anticipation-based	data-base lookup	error rules	parsing with error rules
anticipation-free	**flexible matching**	constraint satisfaction	error-sensitive parsing

Flexible matching

- error anticipation is of limited use
  - error productivity is virtually unlimited
  - imagine a spelling checker trying to collect all possible misspellings of a word

```
spelling
apelling
dpelling
soelling
süelling
...
selling
splling
speling
spellng
spellig
```


Flexible matching

- make error assumptions:
  - e.g. substitution, omission, insertion, transposition of characters
  - simulate error hypotheses to check whether they are plausible
  - assign weights (costs) to error hypotheses
  - look for a minimum-cost hypothesis
    → optimisation problem

Flexible matching

- most simple cost scheme: LEVENSHTEIN metric
  - substitution, omission and insertion with equal costs
  - computes string edit distance

```
pattern: s p e   l l i n g
         | | |   \ \   | |
  input: z p e e l l   n k
```

  - edit distance: 4
    2 substitutions, 1 omission, 1 insertion
  - possible error description:
    `[[s,z],[nil,e],[i,nil],[g,k]]`

Flexible matching

- elastic string matching
  - strings are represented as lists

```
% match(+Pattern,+Input,+OldDist,?NewDist)

match([],[],Dist,Dist).
% both lists are empty, ready

match([],[_|R],Dist,DistNew) :-
% first list is empty, assume an insertion
  Dist1 is Dist+1,
  match([],R,Dist1,DistNew).

match([_|R],[],Dist,DistNew) :-
% second list is empty, assume an omission
  Dist1 is Dist+1,
  match(R,[],Dist1,DistNew).
```


Flexible matching

- elastic string matching (cont.)

```
match([H|R1],[H|R2],Dist,DistNew) :-
% heads are equal, assume no error
  match(R1,R2,Dist,DistNew).

match([H1|R1],[H2|R2],Dist,DistNew) :-
% heads are different, assume a substitution
  H1\=H2,
  Dist1 is Dist+1,
  match(R1,R2,Dist1,DistNew).

match([H1|R1],[H2|R2],Dist,DistNew) :-
% heads are different, assume an insertion
  H1\=H2,
  Dist1 is Dist+1,
  match([H1|R1],R2,Dist1,DistNew).
```


Flexible matching

- elastic string matching (cont.)

```
match([H1|R1],[H2|R2],Dist,DistNew) :-
% heads are different, assume an omission
  H1\=H2,
  Dist1 is Dist+1,
  match(R1,[H2|R2],Dist1,DistNew).
```

- wrapper: initialises the distance

```
% match(+Pattern,+Input,?Distance)
match(L1,L2,Dist) :-
% compute a string mapping
  match(L1,L2,0,Dist).
```

- start the demo

Flexible matching

- observation:
  - multitude of string-to-string mappings
    - go for a minimum distance
    - requires to compare different intermediate results
    - must be stored in the database:

      `flag(mindist,OldValue,NewValue).`

Flexible matching

- no error assumption: keep the old value

```
% first elements are equal: go ahead
match([H|R1],[H|R2],Dist,DistNew) :-
  match(R1,R2,Dist,DistNew).
```

- error assumption: increase the counter
  - e.g. insertion

```
% first elements are different
% assume an insertion and increment distance
match([H1|R1],[H2|R2],Dist,DistNew) :-
  H1\=H2, Dist1 is Dist+1,
  flag(mindist,MinDist,MinDist),
  Dist1<MinDist,
  match([H1|R1],R2,Dist1,DistNew).
```


Flexible matching

- wrapper: initialises the distance

```
% match(+Pattern,+Input,?Distance)

match(L1,L2,Dist) :-
% find a matching which is better than the
% current threshold
  flag(mindist,_,20000),
  match(L1,L2,0,Dist),
  fail.

match(_,_,Dist) :-
% if no better matching found
% return the best distance so far
  flag(mindist,Dist,Dist).
```

- start the demo

Flexible matching

- diagnosis: compilation of an error description
  - e.g. `[[a,b],[nil,c],[d,nil]]`
  - accumulated on an additional argument position
  - e.g. insertion-clause

```
match([H1|R1],[H2|R2],Dist,DistNew,Errors) :-
  H1\=H2,
  Dist1 is Dist+1,
  flag(mindist,MinDist,MinDist),
  Dist1<MinDist,
  match(R1,[H2|R2],Dist1,DistNew,
     [[H1,nil]|Errors]).
```


Flexible matching

- wrapper: intitialising the error description

```
match(L1,L2,Dist,_) :-
% compute a string mapping better than
% the current threshold
  flag(mindist,_,20000),
  retractall(errorlist(_)),
  match(L1,L2,0,Dist,[]),
  fail.
```

Flexible matching

- wrapper (cont.)

```
match(_,_,Dist,Errors) :-
% unable to find a better matching, return
% the error description from the data base
  flag(mindist,Dist,Dist),
  errorlist(Err),
  reverse(Err,Errors),
  retractall(errorlist(_)).
```

- start the demo

Flexible matching

- application to word sequences

  pattern: He will be at school tomorrow

  input: He will tomorrow be at school

  - edit distance: 2 (1 insertion, 1 omission)
  - error description:
    `[[ins,tomorrow],[del,tomorrow]]`
  - error explanation:
    Tomorrow has been placed wrongly. Move it to the final position.

Flexible matching

- observations:
  - search space grows considerably with input length
    - error model must (and can) be restricted
  - generation of error explanations from error descriptions is not trivial
  - `match/4` is *not* a complete implementation
    - returns the first minimal mapping which is found
  - string mapping with LEVENSHTEIN-metric is not unique
    - multiple error descriptions are possible

Flexible matching

- mapping 1:

  pattern: He will come today

  input: He came today

  - error description: `[[will,came],[come,nil]]`
- mapping 2:

  pattern: He will come today

  input: He came today

  - error description: `[[will,nil],[come,came]]`

Flexible matching

- second mapping is much more plausible
- integration of lexical knowledge into the matching procedure
  - using a richer distance metrics:
    - d_1: substitution with another form from the same word
    - d_2: substitution with a word of the same part-of-speech class
    - d_3: substitution across word classes
    - cost(d_1) $<$ cost(d_2) $<$ cost(d_3)

Flexible matching

- observations
  - error simulation adds more flexibility for error handling
  - general mechanism, no anticipation of errors is necessary
  - but: limited diagnostic precision
    - *substitution of word form "come" by "came"*
    - better explanation: *wrong tense*

Flexible matching

- summing up
  - instance-based methods are restricted to limited exercise types
    free-form input is impossible
  - a combination of anticipation-based with anticipation-free techniques is possible and desirable

A simple web-interface

- http-protocol is strictly event-driven
  - browser-request: get, post
  - server-reply: html-page (can be generated dynamically)

A simple web-interface

- requests are processed by means of handlers
  - parse the request
  - call the backend-components (i.e. diagnosis, feedback generation, ...)
  - generate html-code using the computed results
- handlers need to give back control to the web-server
  - otherwise no html-page is generated
  - recursive structure of Prolog-programs needs to be changed to a call-and-return behaviour

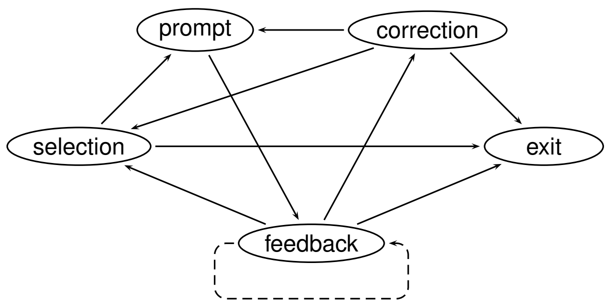
A simple web-interface

- information between successive requests can be transmitted
  - as parameters in the URI ("hidden" variables)
  - by setting cookies at the client computer

A simple web-interface

- 5 different types of html-pages
  - select an exercise
  - prompt the student with an exercise item
  - present feedback
  - present a solution proposal
  - exit the tutor

A simple web-interface



A simple web-interface

- information transmitted as URI-parameters
  - the name of the exercise (`exercise`)
  - the number of the current exercise item (`item`)
  - the utterance typed in by the student (`text`)
  - the explanation level reached (`exlevel`)

A simple web-interface

- an example: the prompt-handler

```
reply(Request) :-
  member(path('/prompt'), Request), !,
  member(search(Search),Request),
  member(exercise=Exercise,Search),
  memberchk(item=ItemId,Search),
  title_html,
  prompt_html(Exercise, ItemId),
  end_html.
```


A simple web-interface

```
prompt_html(Exercise,ItemId) :-
  get_prompt(Exercise,Prompt),
  next_item(Exercise,ItemId,Text,NewItemId),
  format('<p> ~w </p>~n',Prompt),
  format('<p> ~w </p>~n',Text),
  format('<form action="feedback">~n'),
  format('<input type="text" size="50"
     maxlength="200" name="text"> ~n'),
  format('<input type="hidden" name="item"
     value="~w"> ~n',[NewItemId]),
  format('<input type="hidden" name="exercise"
     value="~w"> ~n',[Exercise]),
  format('<input type="hidden" name="exlevel"
     value="0"> ~n'),
  format('<p><input type="submit" value="
     submit "></p>~n'),
  format('</form>~n').
```


A simple web-interface

- the interface to the backend-system
  - `get_prompt/2`: get the exercise prompt from the data base
  - `next_item/4`: get the next exercise item from the data base
  - `check_utt/6`: check the student input against the solutions in the data base and return an explanation
  - `get_expl/4`: switch to the next explanation level and return the corresponding message
- start the server

Instance-based methods

- instance-based methods are easy to implement and computationally inexpensive
- authoring requires only low-level knowledge
- high degree of redundancy in the knowledge base
- reuse of knowledge components is difficult
- no scaling-up to more complex exercises
- unconstrained communication is infeasible

Constraint Satisfaction

	instance-based	generalised methods	
		restricted input	"free-form" input
anticipation-based	data-base lookup	error rules	parsing with error rules
anticipation-free	flexible matching	**constraint satisfaction**	error-sensitive parsing

Constraint Satisfaction

- Constraints
- Constraint checking
- Error perspectives
- Structure identification
- Linear ordering constraints
- Co-occurrence constraints
- Error selection

Constraints

- the idea:
  - describe correct solutions not as isolated instances but by means of general well-formedness conditions (MENZEL 1988, 1990, 1992)
- the approach:
  - classify linguistic objects into abstract categories
    - part of speech: noun, verb, adjective, ...
    - morpho-syntactic features: number, person, case, gender, ...
  - give general requirements on categories, feature values and their combination

Constraints

- dictionary: database of category and feature assignments to word forms
- here: categories treated separately

```
category(the,det).
category(man,n).
category(is,aux).
category(cool,adj).
```


Constraints

- feature values are coded as lists

```
featurepos(number,1).

features(the,[sg]).
features(man,[sg]).
features(is,[sg]).
features(cool,[]).
```

- meaning is assigned by position

```
featurepos(number,1).
```


Constraints

- feature assignments are not necessarily unique

```
features(the,[sg]).
features(the,[pl]).
features(fish,[sg]).
features(fish,[pl]).
```

- from a logical point of view: correctness conditions will form a disjunctive theory
  - requires special care to avoid inconsistent diagnoses

Constraints

- constraints express logical conditions which hold for one or several variable assignments
- (for diagnostic purposes) an exercise is defined as a collection of variables

```
vars([det,noun,verb,adv]).
```

Constraints

- word forms from the student's input, e.g.

  The man is cool.

  are (dynamically) assigned to variables

variable	det	noun	verb	adv
word form	*the*	*man*	*is*	*cool*
category	det	n	aux	adv
position	1	2	3	4
features	[sg]	[sg]	[sg]	[ ]

Constraints

- representation of variables as facts in the Prolog-database

```
vword(det,the).
vcat(det,det).
vpos(det,1).
vfeat(det,[sg]).
```

- assignment of alternative values by means of the normal backtracking mechanism

Constraints

- constraints are logical implications

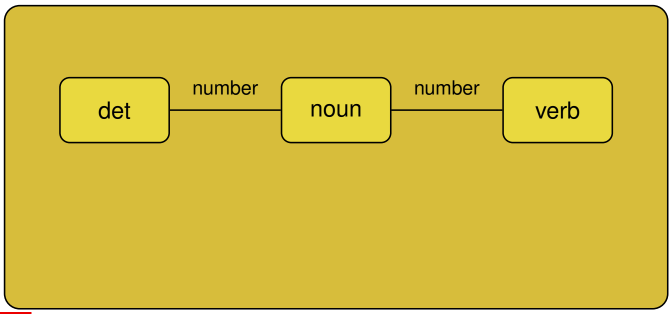
```
true ==> cat(det,det).
true ==> cat(noun,n).
true ==> cat(verb,aux).
true ==> cat(adv,adj).

true ==> agree(det,noun,number).
true ==> agree(noun,verb).
```


Constraint checking

- for all variable assignments find the constraints that are violated
- if an assignment with no constraint violations is found, signal "ok"
- else find the assignment with a minimum number of constraint violations and output the explanation(s) connected to the violated constraint(s)

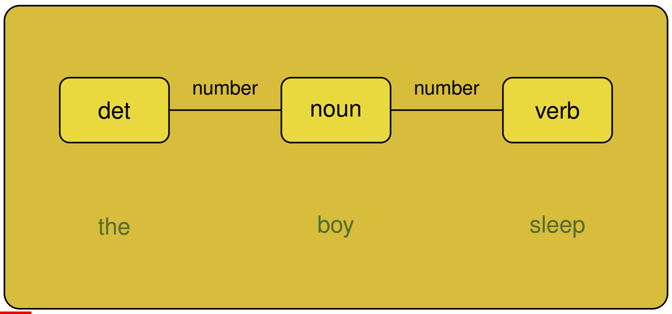
Constraint checking

- a constraint problem



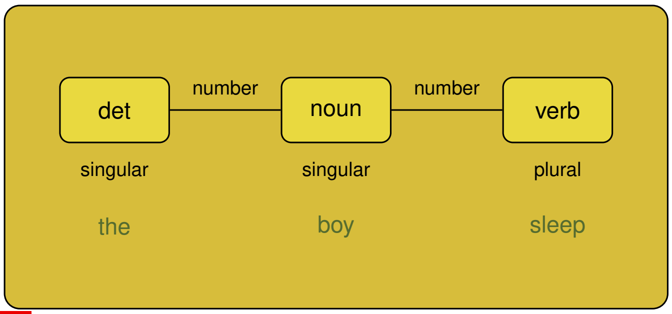
Constraint checking

- assigning word forms to variables



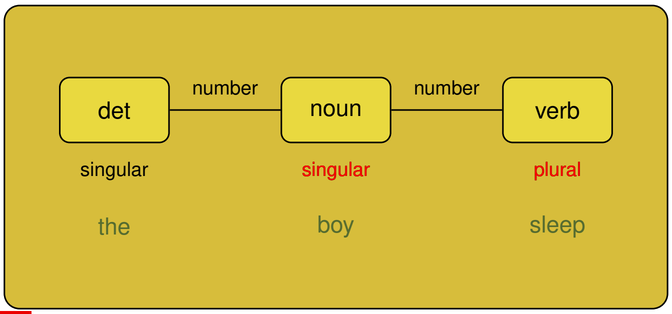
Constraint checking

- assigning features to variables



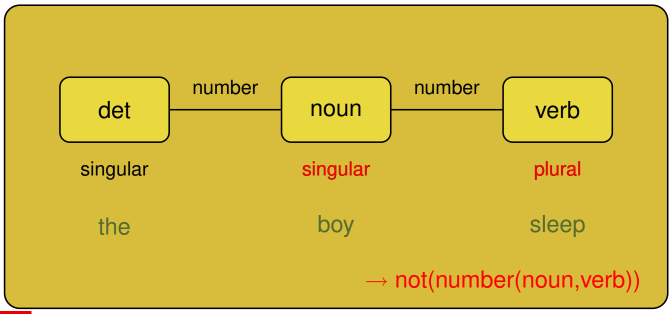
Constraint checking

- checking for constraint violations



Constraint checking

- deriving an error hypothesis



Constraint checking

- diagnosis: computes several alternative diagnoses for an input utterance

```
diagnose(WordList,Errors) :-
  vars(Vars),
  forall(member(Var,Vars),clean_var(Var)),
     % housekeeping: reset all variables
  assign(WordList,Vars,AssignErrors),
     % call the variable assignment
  check_constraints(ConstErrors),
     % call the constraint checker
  append(AssignErrors,ConstErrors,Errors).
     % combine the results
```


Constraint checking

- constraint checker: applies all but cat-constraints and returns a list of error descriptions

```
check_constraints(Err) :-
  findall(Conclusio,
      (Premise ==> Conclusio,
       \+ check_constraint(Premise,Conclusio)),
      Err).

check_constraint(_,cat(_,_)) :- !, true.
    % cat-constraints are always true
check_constraint(Prem,Concl) :-
  Prem -> Concl.
    % others must be fulfilled
```


Constraint checking

- information in the database

```
featurepos(number,1).
true ==> agree(det,noun,number).
true ==> agree(noun,verb,number).
```

- the semantics of a constraint

```
agree(Var1,Var2,Feature) :-
   featurepos(Feature,Pos),
   ((vfeat(Var1,Feats1),
     vfeat(Var2,Feats2) ->
      nth1(Pos,Feats1,Feat),
      nth1(Pos,Feats2,Feat)) ; true ).
```

- start the demo

Constraint checking

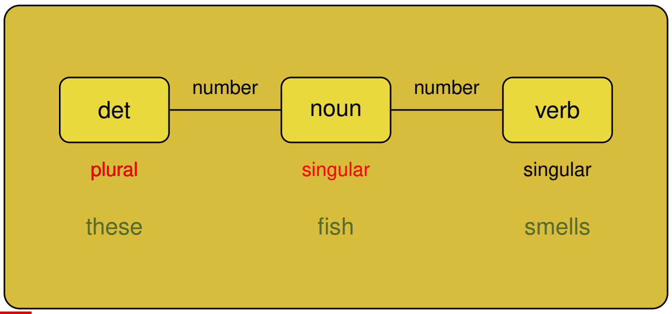
- from a logical point of view:
  - non-monotonic reasoning
    - from the conclusion to the premise

      Comp=agree(X,Y,Feature) $\wedge\neg$abnormal(Comp)
      $\rightarrow$ Feature(X) = Feature(Y)

    - if global consistency cannot be established, retract assumptions about normal behaviour

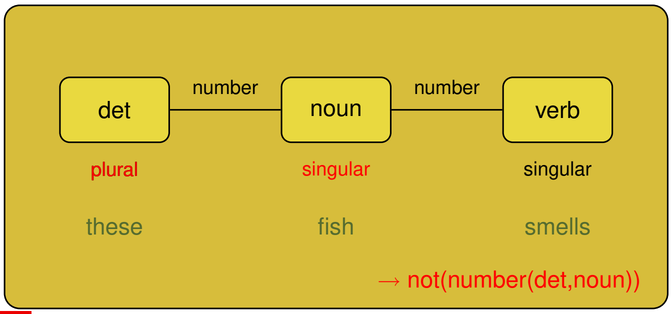
Constraint checking

- even a minimum diagnosis might be ambiguous



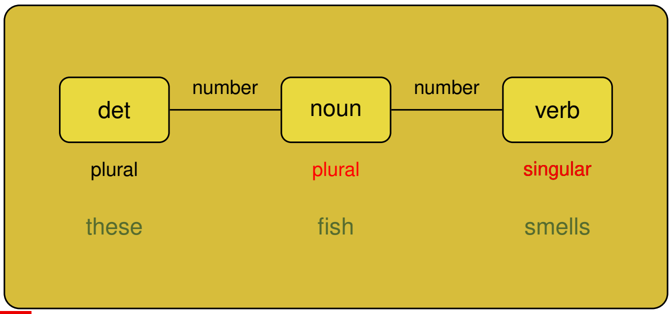
Constraint checking

- even a minimum diagnosis might be ambiguous



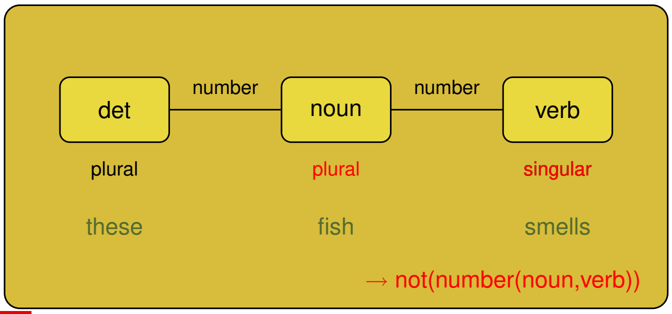
Constraint checking

- even a minimum diagnosis might be ambiguous



Constraint checking

- even a minimum diagnosis might be ambiguous



- alternative interpretations: which one to choose?

Constraint checking

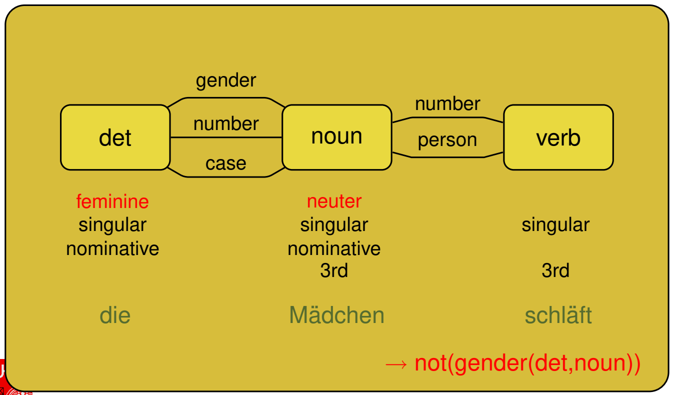
- constraint system of English is rather impoverished
- richer constraint systems in other languages
- e.g. German
  - subject-verb: person, number
  - noun phrase: number, gender, case
- e.g. Russian
  - subject-verb: person, number, gender
  - noun phrase: number, gender, case, animatedness

Constraint checking

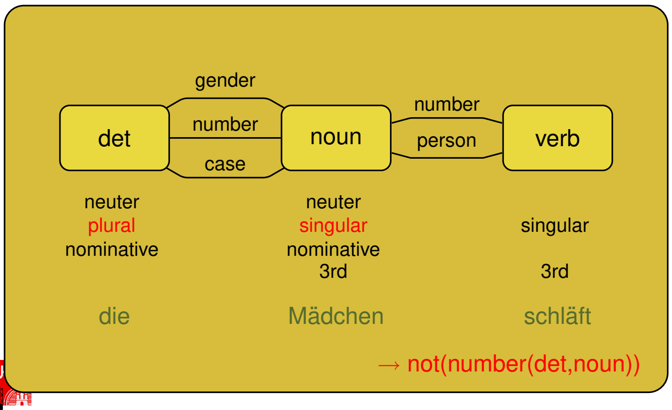
- complex combinations of features in the dictionary

```
features(die,[sg,femin,nom]).
features(die,[sg,femin,acc]).
features(die,[pl,_,nom]).
features(die,[pl,_,nom]).
features(mädchen,[sg,neutr,nom,'3rd']).
features(mädchen,[sg,neutr,dat,'3rd']).
features(mädchen,[sg,neutr,acc,'3rd']).
features(mädchen,[pl,neutr,_,'3rd']).
features(schläft,[sg,_,_,'3rd']).
```

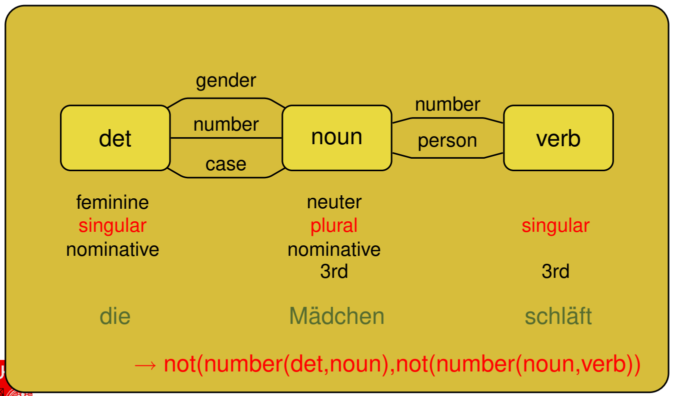

Constraint checking



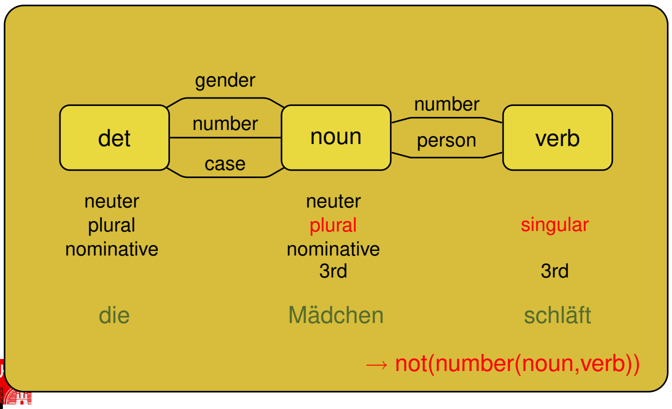
Constraint checking



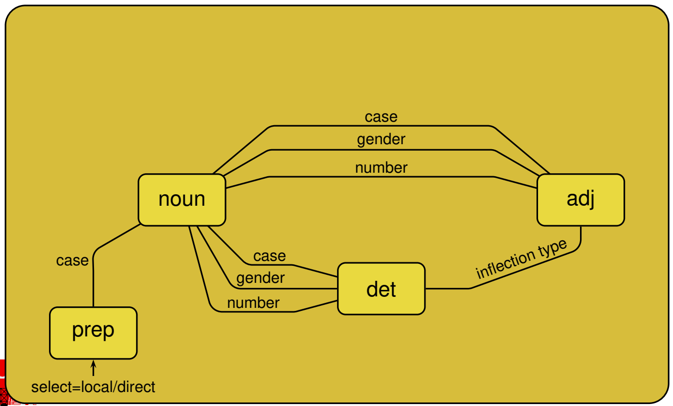
Constraint checking



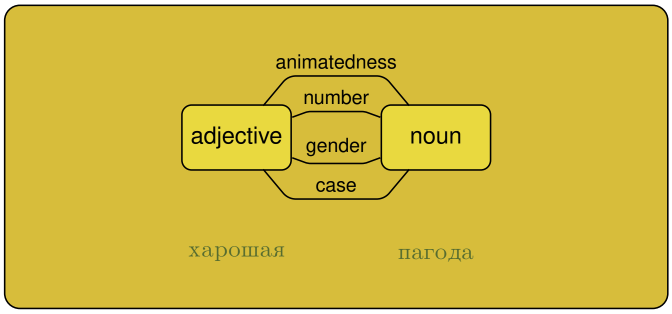
Constraint checking



Agreement in a German PP



Agreement in a Russian NP

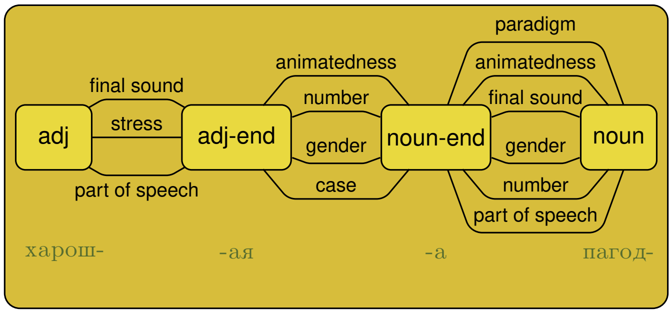


Agreement in a Russian NP

- rather simple agreement constraints
  - adjectives (and determiners) follow the same pattern:

    такая красивая женщина
    такой красивой женщины
    такой красивой женщине
    такую красивую женщину
    такой красивой женщиной
    такой красивой женщине
- do not reflect the complexity of the Russian inflectional system
- idea: separate inflection from agreement

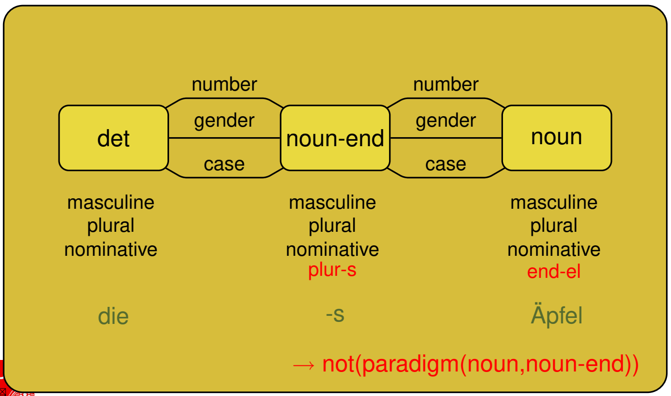
Inflection in a Russian NP



Agreement in a Russian NP

- additional diagnostic capabilities:
  detection of erroneous inflection patterns
- новый дом but большой человек
  *болшый человек → not(stress(adj,adj-end))
  - большой *has word stress on the ending which therefore changes to* ой
- новый дом but маленкий человек
  *маленкый человек → not(final-sound(adj,adj-end))
  - *the root* маленк- *ends with a hard consonant but* ый *can only be used with a soft one*

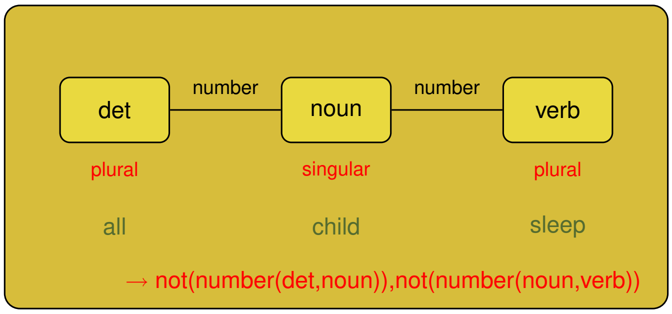
Inflection in German NPs



Inflection in German NPs

- again: wrong inflectional patterns can be diagnosed
- die Gabel ... (mit) den Gabeln
  der Apfel ... (mit) den Äpfeln
  *(mit) den Apfeln → not(number(noun,noun-end))
  - *'Apfel' requires umlaut in plural*
- die Schachtel ... die Schachteln
  die Äpfel ... die Äpfel
  *die Äpfeln → not(gender(noun,noun-end))
  - *'Apfel' is masculine, not feminine*

Error perspectives



- is this a plausible diagnosis?

Error perspectives

- alternative perspective:
  - the student might have thought that "child" is not a singular form:

    not(value(noun,singular))
- assuming the ignorance of lexical information
  - gives different view on the same error
  - sometimes yields more concise explanations

Error perspectives

- how to obtain lexical error descriptions?
  - transforming constraint violations into lexical error descriptions
    - typical patterns
    - e.g. one variable in a network of conditions is isolated from all the other variables
    - start the demo

Error perspectives

- how to obtain lexical error descriptions? (cont.)
  - performing a separate error simulation for the lexical error case (MENZEL 1988, 1992)
    - requires to generate the complement of the features specified in the dictionary
    - constraint propagation with a different propagation rule

Error perspectives

- how to obtain lexical error descriptions? (cont.)
  - HEIFT 1998 directly codes all possible features (and feature combinations!) in the dictionary of a HPSG grammar
  - dictionary entry for *er*

$$\begin{bmatrix} \text{phon} & \langle\, \text{er} \,\rangle \\ \text{synsem} \mid \text{local} & \begin{bmatrix} \text{cat} \mid \text{head} & \text{n} \\ \text{content} \mid \text{index} \mid \text{per} & \begin{bmatrix} \text{1st} & \text{error} \\ \text{2nd} & \text{error} \\ \text{3rd} & \text{correct} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Error perspectives

- dictionary entry for *gehst*

$$\begin{bmatrix} \text{phon} & \langle\, \text{gehst} \,\rangle \\ \text{synsem} \mid \text{local} & \begin{bmatrix} \text{cat} & \begin{bmatrix} \text{head} & \text{v} \\ \text{subj} \mid \text{synsem} \mid \text{local} & \begin{bmatrix} \text{cat} & \begin{bmatrix} \text{head} & \text{n} \\ \text{case} & \text{nom} \end{bmatrix} \\ \text{content} \, \boxed{1} & \begin{bmatrix} \text{index} \mid \text{per} \mid \text{2nd} \, \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{content} & \begin{bmatrix} \text{reln} & \text{gehen} \\ \text{geher} & \boxed{1} \end{bmatrix} \\ \multicolumn{2}{l}{\text{descriptor} \mid \text{main\_clause} \mid \text{vp\_per} \mid \text{2nd} \quad \boxed{2}} \end{bmatrix} \end{bmatrix}$$

Error perspectives

- unifying

$$\begin{bmatrix} \text{synsem} \mid \text{local} \mid \text{content} \mid \text{index} \mid \text{per} & \begin{bmatrix} \text{1st} & \text{error} \\ \text{2nd} & \text{error} \\ \text{3rd} & \text{correct} \end{bmatrix} \end{bmatrix}$$

- with

$$\begin{bmatrix} \text{synsem} \mid \text{local} \mid \text{content} \mid \text{index} \mid \text{per} \mid \text{2nd} & \boxed{2} \end{bmatrix}$$

- yields

$$\begin{bmatrix} \text{synsem} \mid \text{local} \mid \text{content} \mid \text{index} \mid \text{per} \mid \text{2nd} & \boxed{2} \; \text{error} \end{bmatrix}$$

- which is coreferenced into

$$\begin{bmatrix} \text{descriptor} \mid \text{main\_clause} \mid \text{vp\_per} \mid \text{2nd} & \boxed{2} \; \text{error} \end{bmatrix}$$

Structure identification

- assigning word forms from the student's input to the variables of the diagnosis problem
- different degrees of freedom
  - fixed ordering
  - flexible ordering
  - flexible matching with substitution, omission and insertion assumptions
    - start the demo
  - with morphological segmentation

Linear ordering constraints

- request a (partial) ordering of variables in the exercise and check for constraint violations (MENZEL 1990)
- linear precedence constraints

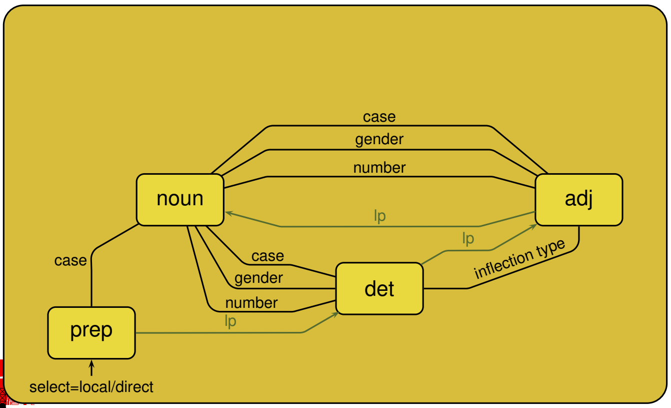
```
true ==> lp(prep,det)
true ==> lp(det,adj)
true ==> lp(adj,noun)
```

`lp(A,B)` is true, if `A` is assigned to a word left of the one assigned to `B`

Linear ordering constraints

- perform a normal constraint check
  - if omissions are allowed: the transitive closure of linear precedence constraints has to be used!
  - constraint propagation instead of constraint checking required
- use constraint violations as correction proposals e.g. on the table small → not(lp(adj,noun))
  - the adjective has to be placed before the noun
- constraint violations can be directly used as explanations
- start the demo again

Linear Precedence in a German NP



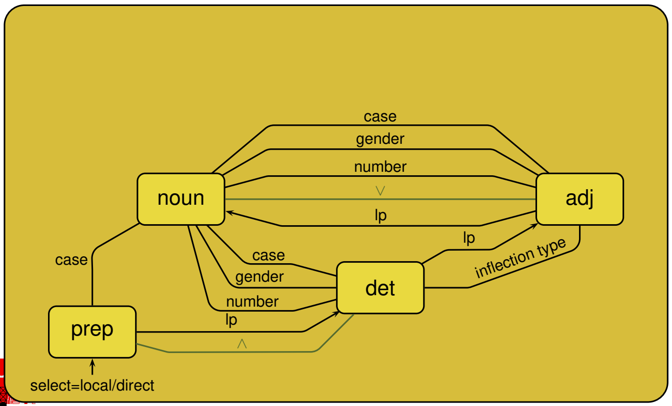
Co-occurrence constraints

- allow word form be omitted under particular conditions (MENZEL 1990)
  - so far: obligatoriness as default case
    - all variables need to receive a value assignment
  - now: optionality as default case
- explicit specification of mandatory assignments becomes necessary

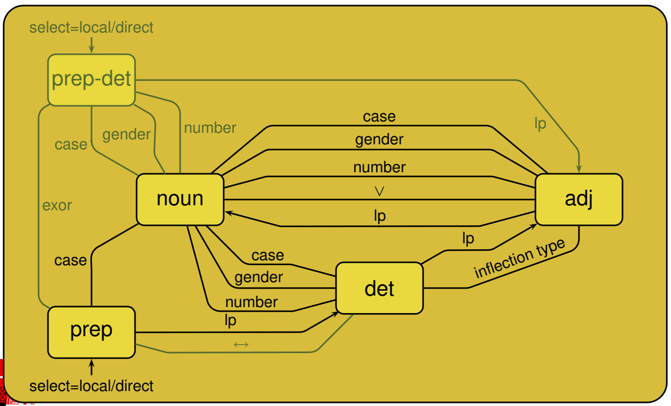
Co-occurrence constraints

- totally mandatory: conjunction (∧) of variable assignments
- partially mandatory: disjunction (∨) of variable assignments
  - at least one variable needs to receive a value assignment
- conditionally mandatory:
  - implication (→)
  - equivalence/biimplication (↔)
- mutual exclusion: alternative (exor)

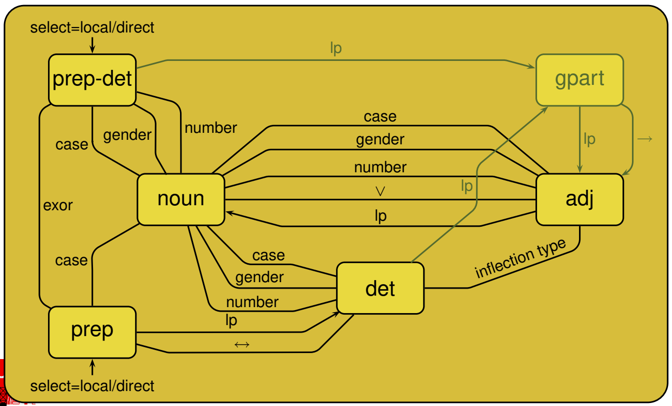
Coocurrence in a German PP



Phonological Contraction



Gradually modified adjectives



Error selection

- diagnosis provides a multitude of alternative explanation possibilities
  → selection becomes necessary
- selection heuristics
  - higher up in a syntactic structure:
    - better reflects the violated grammar regularity
  - deeper down in a syntactic structure:
    - better indicates a correction possibility
  - preference for constraint violations:
    - better reflects the violated grammar regularity

Error selection

- selection heuristics (cont.)
  - preference for lexical error descriptions:
    - better indicates a correction possibility
  - conjunctive or disjunctive combinability:
    - results in more compact error descriptions
  - citation form preference:
    - explanations referring to the default case are more plausible
  - error type preference:
    - typical errors are more likely (e.g. gender in German)

Constraint satisfaction

- summing up
  - constraint satisfaction makes more flexible exercises possible
    - free lexical choice (within the limitations of the dictionary)
  - provides a multitude of different error interpretations
    - error selection can be made sensitive to a didactic goal and / or the desires of the student

Constraint satisfaction

- summing up (cont.)
  - in limited domains close-to-perfect diagnostic results can be achieved
    - can be used to directly generate correction proposals from
    - explorative learning-by-doing experiments become possible
  - linguistic knowledge is easily accessible

Constraint satisfaction

- summing up (cont.)
  - syntactic flexibility is restricted by the power of structure identification
  - exercise needs to be embedded into an appropriate context
    - fill-in-the-gap exercises, question answering, . . .
  - simple communicative situations can be simulated
  - limitations can be motivated by an appropriate scenario
    - my home in disorder
    - shopping in a supermarket
    - . . .

Parsing with error rules

	instance-based	generalised methods	
		restricted input	”free-form” input
anticipation-based	data-base lookup	error rules	**parsing with error rules**
anticipation-free	flexible matching	constraint satisfaction	error-sensitive parsing

Parsing with error rules

- Grammars
- Features
- Error rules

Grammars

- rule-based description of sentences which belong to a language
- can be used to
  - model sets of possible solutions to an exercise
  - distinguish between correct and incorrect (error detection)
  - obtain a structural description of an utterance – essential prerequisite for language understanding
- many different models and formalisms available

Grammars

- Definite Clause Grammar (DCG)
  - simple formalism for prototyping
  - built-in feature of Prolog
  - unification grammar with a context-free backbone

Grammars

- *dictionary* (preterminal rules)

```
det --> [the].          adj --> [good].
n --> [man].            adj --> [happy].
n --> [men].            adj --> [nice].
n --> [woman].          cop --> [are].
n --> [women].          cop --> [is].
n --> [fish].           cop --> [was].
                        cop --> [were].
```

Grammars

- DCG uses difference lists to efficiently deal with grammars
  - any linguistic object (sequence of words) is identified by two pointers: `start` and `end`
- (internal) form of a dictionary entry:

```
det([the|X],X).
```

  - the atom 'the' at the beginning of a list is a determiner
  - it starts at the beginning of the list and ends after its first element
  - the difference of `[the|X]` and `X` is a determiner

Grammars

- for your convienience the internal form is produced automatically from the simpler external notation
- *grammar*: a set of (context-free) rules
  - combine simple linguistic objects into more complex ones
  - lexical items → phrases (constituents) → clauses → sentences

Grammars

- a sample grammar

```
s --> np, vp.
np --> det,nbar.
nbar --> n.
vp --> cop, advp.
advp --> adj.
```

- rules are also translated into the internal difference list representation, e.g.

```
s(X,Y) :- np(X,Z), vp(Z,Y).
```


Grammars

- grammars can be used for generating sentences e.g. producing exercise contexts

```
?- s(X).
X = [the, man, are, good] ;
X = [the, man, are, happy] ;
X = [the, man, are, nice] ;
X = [the, man, is, good] ;
X = [the, man, is, happy] ;
...
X = [the, fish, were, happy] ;
X = [the, fish, were, nice] ;
No
```

- overall 60 sentences

Grammars

- grammars can be used for checking sentences i.e. error identification

```
?- s([the, woman, is, happy]).
Yes
?- s([the, woman, is, very, happy]).
No
```

- start the demo

Grammars

- parsing aims at producing a structural description of the sentence
  - fundamental prerequisite for language understanding
- augmentation:
  add another argument position in the dictionary ...

```
det(det(the)) --> [the].
n(n(man)) --> [man].
 ...
adj(adj(good)) --> [good].
 ...
cop(cop(are)) --> [are].
 ...
```


Grammars

- ... and in the grammar

```
s(s(NP,VP)) --> np(NP), vp(VP).
np(np(D,NBAR)) --> det(D), nbar(NBAR).
nbar(nbar(N)) --> n(N).
vp(vp(C,AP)) --> cop(C), advp(AP).
advp(advp(A)) --> adj(A).
```


Grammars

- computation of the sentence structure

```
s(Sentence,Structure) :-
  s(Structure,Sentence,[ ]).

?- s([the, woman, is, happy],S).
   S = s(np(det(the),
            nbar(n(woman))),
         vp(cop(is),
    advp(adj(happy)))) ;
No
```

- start the demo

Grammars

- sentence structure can follow the rule structure but need not
- other structural descriptions can be built, e.g.
  - predicate-argument structure
  - semantic form
  - ...

Features

- grammar licenses many ungrammatical sentences
- more fine grained regularities required
- augmentation again:
  add morpho-syntactic features to linguistic categories

Features

- agreement features in the dictionary ...

```
det(det(the),_) --> [the].
n(n(man),sg) --> [man].
n(n(men),pl) --> [men].
 ...
adj(adj(good)) --> [good].
 ...
cop(cop(are),second,sg,pres) --> [are].
cop(cop(are),_,pl,pres) --> [are].
cop(cop(is),third,sg,pres) --> [is].
```


Features

- ... and in the grammar

```
s(s(NP,VP),Tense) -->
  np(NP,Person,Number),
  vp(VP,Person,Number,Tense).
np(np(D,NBAR),third,Number) -->
  det(D,Number),
  nbar(NBAR,Number).
nbar(nbar(N),Number) -->
  n(N,Number).
vp(vp(C,AP)) -->
  cop(C),
  advp(AP).
advp(advp(A)) -->
  adj(A).
```


Features

- observations:
  - more fine grained descriptions introduce additional ambiguity to the dictionary
  - sometimes ambiguity can be treated by means of underspecification

Features

- other features, e.g. subcategorisation of verbs
  - list of obligatory complements

```
verb(v(sleeps),third,sg,pres,
  []) --> [sleeps].
verb(v(loves),third,sg,pres,
  [np(_,_,_,nom)]) --> [loves].
verb(v(gives),third,sg,pres,
  [np(_,_,_,acc),np(_,_,_,nom)]) --> [gives].
verb(v(makes),third,sg,pres,
  [np(np(det(_),_),_,_,nom)]) --> [makes].
verb(v(makes),third,sg,pres,
  [np(_,_,_,acc),advp(_)]) --> [makes].
```


Features

- saturation of subcategorisation requirements as daughter nodes in appropriate grammar rules

```
vp(vp(Vbar),Pers,Num,Tense) -->
  vbar(Vbar,Pers,Num,Tense).
vbar(vbar(V0),Pers,Num,Tense) -->
  verb(V0,Pers,Num,Tense,[]).
vbar(vbar(V0,S1),Pers,Num,Tense) -->
  verb(V0,Pers,Num,Tense,[C1]),C1,
   {arg(1,C1,S1)}.
vbar(vbar(V0,S1,S2),Pers,Num,Tense) -->
  verb(V0,Pers,Num,Tense,[C1,C2]),C1,C2,
    {arg(1,C1,S1),arg(1,C2,S2)}.
```

- start the demo

Error rules

- so far: rather weak diagnostic capabilities
  - error detection only
- idea again:
  - try to anticipate typical student errors and integrate them into the grammar
    (YAZDANI 1988, SCHWIND 1995, SCHNEIDER AND MCCOY 1998)

Error rules

- the approach:
  - add rules describing expected faulty structures to the grammar
  - annotated these error rules with an error description
  - collect the error descriptions by means of an additional error feature

Error rules

- e.g. violated agreement requirements

```
s(s(NP,VP),Tense,ErrorS) -->
  np(NP,Person1,Number,ErrorN),
  vp(VP,Person2,Number,Tense,ErrorV),
    {Person1\=Person2,
     append([person_subj_obj|ErrorN],
       ErrorV,ErrorS)}.
```


Error rules

- e.g. incorrect constituent orderings

```
s(s(NP,VP),Tense,ErrorS) -->
  vp(VP,Person,Number,Tense,ErrorV),
  np(NP,Person,Number,ErrorN),
    {append([lp_obj_subj|ErrorN],
       ErrorV,ErrorS)}.
```

- e.g. omission of obligatory complements

```
s(s(NP,VP),Tense,ErrorS) -->
  vp(VP,Person,Number,Tense,ErrorV),
    {append([missing_subj|ErrorN],
       ErrorV,ErrorS)}.
```


Error rules

- summing up
  - grammars increase the flexibility of language tutoring systems considerably
  - grammars support language understanding for goal-oriented communication
  - in goal-oriented settings ”free-form” input becomes feasible
  - diagnosis based on error rules is limited to some typical student errors
  - there is a tradeoff between error coverage and efficiency

Error-sensitive parsing

	instance-based	generalised methods	
		restricted input	”free-form” input
anticipation-based	data-base lookup	error rules	parsing with error rules
anticipation-free	flexible matching	constraint satisfaction	**error-sensitive parsing**

Error-sensitive parsing

- Constraint retraction
- Flexible parsing
- Parsing with weighted constraints

Error-sensitive parsing

- grammar development is cumbersome
- error anticipation is even more cumbersome and, hence, limited
- can’t we provide a general mechanism for error handling?
- similar requirements in other areas of NLP

Error-sensitive parsing

- **robust parsing:**
  - find a structural interpretation despite of ungrammatical input
    - processing of unrestricted text
    - speech understanding
  - approximate understanding is sufficient
- CALL applications are more demanding:
  error-sensitive parsing:
  - additionally find out what is wrong with the input
  - approximate results are not good enough

Constraint retraction

- old idea: Weischedel et al. 1978
- abandon certain constraints of the grammar if a normal parse fails
- problem: constraining information to restrict the space of hypotheses is lost
- approach: retract constraints cautiously and in a controlled manner

Constraint retraction

- "weak" feature unification (e.g. Schwind 1988, 1994, Carpenter 1993, Vogel and Cooper 1995, Sågvall Hein 1998, Fouvry 2000, Reuer 2003)
- allow unification to succeed even in case of conflicting feature values
- collect feature value clashes in a special data structure

Constraint retraction

- successful unification

```
?- npagr(nom,sg,masc) = npagr(nom,sg,masc).
   Yes
?- unify(npagr(nom,sg,masc),
         npagr(nom,sg,masc),Result,Errors).
   Result = npagr(nom, sg, masc)
   Errors = []
```


Constraint retraction

- failed unification

```
?- npagr(nom,sg,masc) = npagr(gen,sg,fem).
   No
?- unify(npagr(nom,sg,masc),
         npagr(gen,sg,fem),Result,Errors).
   Result = npagr(nom, sg, masc)
   Errors = [npagr(nom, gen), npagr(masc, fem)]
```


Constraint retraction

- unification with underspecified values

```
?- npagr(gen,_,_) = npagr(nom,sg,masc).
   No
?- unify(npagr(gen,_,_),
         npagr(nom,sg,masc),Result,Errors).
   Result = npagr(gen, sg, masc)
   Errors = [npagr(gen, nom)]
```


Constraint retraction

- unification of nested terms

```
?- unify(a(b(c),d(_)),
         a(_,d(e)),Result,Errors).
   Result = a(b(c), d(e))
   Errors = []
?- unify(a(b(c),d(f)),
         a(g,d(e)),Result,Errors).
   Result = a(b(c), d(f))
   Errors = [a(b(c), g), d(f, e)]
```

- start the demo

Constraint retraction

- modified dictionary entries:
  - coding of features in a single term

```
noun(mann,npagr(nom,sg,masc)) --> [mann].
noun(männer,npagr(nom,pl,masc)) --> [männer].
...
det(der,npagr(nom,sg,masc)) --> [der].
det(die,npagr(nom,pl,_)) --> [die].
...
v(schläft,subjagr(third,sg)) --> [schläft].
v(schlafen,subjagr(third,pl)) --> [schlafen].
...
```


Constraint retraction

- modified grammar rules

```
s(s(NP,VP),ErrorsS) -->
  np(NP,_,SUBJagrNP,ErrorsNP),
  vp(VP,SUBJagrVP,ErrorsVP),
    {unify(SUBJagrVP,SUBJagrNP,_,ErrorsNew),
     append(ErrorsNP,ErrorsVP,E),
     append(ErrorsNew,E,ErrorsS)}.

np(np(Det,Noun),NPagr,SUBJagr,ErrorsNP) -->
  det(Det,NPagrD),
  noun(Noun,NPagrN),
    {NPagr=npagr(_,Num,_),
     SUBJagr=subjagr(third,Num),
     unify(NPagrN,NPagrD,NPagr,ErrorsNP)}.

vp(vp(V),SUBJagr,[]) -->
  v(V,SUBJagr).
```

Constraint retraction

```
?- s(S,E,[der,mann,schläft]).
   S = s(np(der, mann), vp(schläft))
   E = [] ;
   S = s(np(der, mann), vp(schläft))
   E = [npagr(acc, nom)] ;
   S = s(np(der, mann), vp(schläft))
   E = [npagr(dat, nom)] ;
   S = s(np(der, mann), vp(schläft))
   E = [npagr(nom, gen), npagr(masc, fem)] ;
   S = s(np(der, mann), vp(schläft))
   E = [npagr(acc, gen), npagr(masc, fem)] ;
   ...
```


Constraint retraction

```
?- s(S,E,[die,mann,schläft]).
   S = s(np(die, mann), vp(schläft))
   E = [npagr(sg, pl)] ;
   S = s(np(die, mann), vp(schläft))
   E = [npagr(acc, nom), npagr(sg, pl)] ;
   S = s(np(die, mann), vp(schläft))
   E = [npagr(dat, nom), npagr(sg, pl)] ;
   S = s(np(die, mann), vp(schläft))
   E = [npagr(nom, acc), npagr(sg, pl)] ;
   S = s(np(die, mann), vp(schläft))
   E = [npagr(sg, pl)] ;
   ...
```


Constraint retraction

```
?- s(S,E,[die,männer,schläft]).
   S = s(np(die, männer), vp(schläft))
   E = [subjagr(sg, pl)] ;
   S = s(np(die, männer), vp(schläft))
   E = [subjagr(sg, pl), npagr(gen, nom)] ;
   S = s(np(die, männer), vp(schläft))
   E = [subjagr(sg, pl), npagr(acc, nom)] ;
   S = s(np(die, männer), vp(schläft))
   E = [subjagr(sg, pl), npagr(nom, acc)] ;
   ...
```

- start the demo

Constraint retraction

- multitude of explanations reflects lexical ambiguity in the dictionary
- search space grows considerably
  - efficiency problems
  - many spurious hypotheses
- how to control retraction in a meaningful way?
  - go for minimal error descriptions
  - retraction only for carefully selected features (VANDEVENTER 2001, REUER 2003)

Constraint retraction

- summing up
  - constraint retraction is a powerful technique as long as the inflation of the search space can be kept under control
  - anticipation-free techniques are not able to distinguish between "incorrect" and "don't know"!

Constraint retraction

- summing up (cont.)
  - combinations of constraint retraction and error anticipation have been successful (SCHWIND 1988, 1995)
    - agreement and case government are sufficiently regular phenomena
    - structural regularities, which are much more difficult to describe in a "water proof" fashion are left to anticipation-based techniques

Flexible Parsing

- idea: run a parser with a modified parsing strategy
  - if no structural interpretation of a sentence can be found grammar rules are modified dynamically
  - makes possible to deal with sentences not licensed by the original grammar
  - minimum distance parsing: find the structure which requires a minimum of modifications (error hypotheses)
- mostly developed for purposes other than CALL

Flexible Parsing

- MELLISH 1989, KATO 1994:
  - modified two-pass chart-parser
  - bottom-up parsing
  - in case of failure: top-down-reconstruction with modified parsing rules
  - substitution, insertion and omission errors can only be hypothesized within a rule

Flexible Parsing

- LAVIE AND ROSÉ (2001):
  - GLR*: modified TOMITA-parser
  - skipping over words which cannot be integrated
  - LCFLEX: modified chart-parser
  - word skipping, non-terminal insertion, weak feature unification
- LEE ET AL. (1995):
  - modified Earley-parser
  - substitutions, insertions and deletions of non-terminal and terminal nodes

Flexible Parsing

- few special-purpose developments for CALL
- REUER (2003):
  - modified Earley-parser
  - heap mechanism: superfluous constituents can be placed on a heap to be later inserted where appropriate
  - misplacement errors crossing constituent boundaries can also be treated
  - no insertions, since not very frequent in learner utterances

Flexible Parsing

- VANDEVENTER (2001)
  - GB-parser with constraint retraction

Flexible Parsing

- summing up
  - full-fledged error simulation by checking out all combinations is computationally infeasible
  - carefully controlled application of rule modifications is vital
  - simulated conversations become possible while still being able to diagnose a wide range of possible learner errors
  - diagnoses are biased by syntactic similarity to the closest correct utterance
  - world knowledge and learner intentions are difficult to take into consideration

Parsing with weighted constraints

- Constraining structures
- Weighted constraints
- Constraint violations
- System design

Constraining structures

- idea: extend constraint retraction to syntactic structures
- instead of having rules to construct hierarchical representations use constraints to describe the space of possible structural descriptions
- HSPG - a constraint-based grammar formalism
  - structural description is part of the feature structure
  - complex constraints based on feature coreference define the space of admissible structures
  - when structure-building constraints are retracted no structure is built at all
- structure should exist independently of the constraints

Constraining structures

- an alternative: Constraint Dependency Grammar (MARUYAMA 1990)
- initial space of hypotheses:
  - fully underspecified structural descriptions
  - every node modifies every other with all possible labels
  - containing all possible dependency trees for an utterance

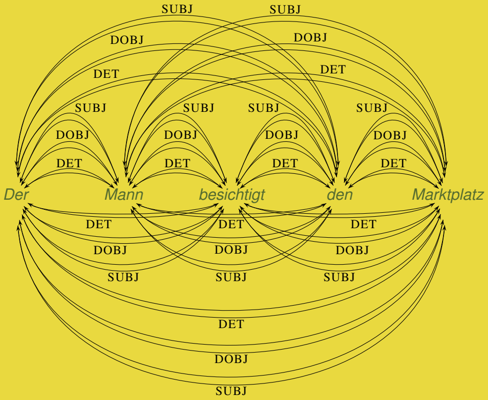
Constraining structures

- constraints license certain dependency edges or combinations thereof
  - dependency edges which violate a constraint can be removed from the space of hypotheses

Constraining structures

- parsing problem becomes a *constraint satisfaction problem*
  - variables: words in the input utterance
  - values: pairs of words and labels
  - constraints
  - solution: one or several value assignments (i.e. dependency trees
- many well-understood solution methods available (TSANG 1993)

Constraining structures

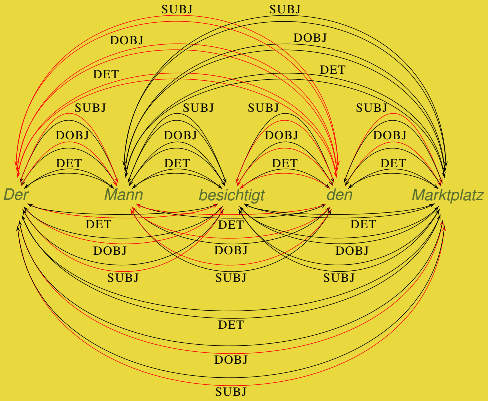


Initial state of a parsing problem with three labels (DET, SUBJ, DOBJ)

Constraining structures

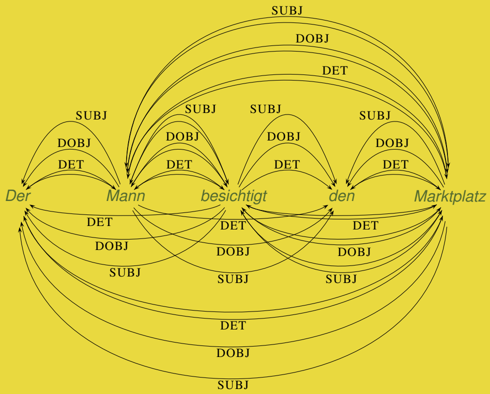
- constraints are logical conditions which *every* dependency edge must fulfill
- therefore contraints are typically implications of the form `condition -> restriction`
- constraints can be
  - unary: considering only one edge at a time or
  - binary: considering pairs of edges at a time
- constraints with higher arity are possible but usually very expensive

Constraining structures

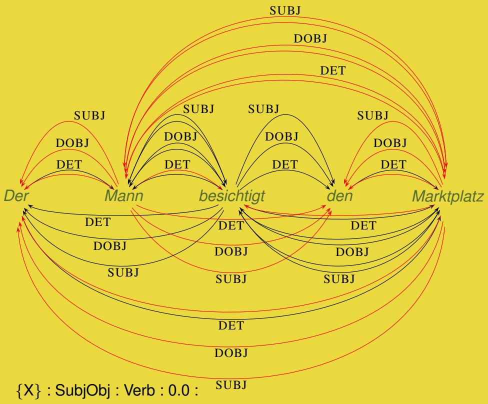


{X} : DetNom : Det : 0.0 : X↓cat=det → X↑cat=noun ∧ X.label=DET

Constraining structures

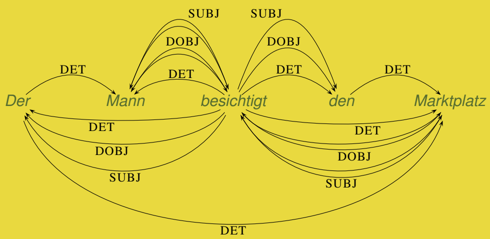


Constraining structures

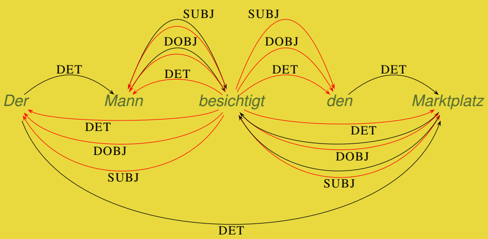


{X} : SubjObj : Verb : 0.0 :
X↓cat=noun → X↑cat=vfin ∧ X.label=SUBJ ∨ X.label=DOBJ

Constraining structures

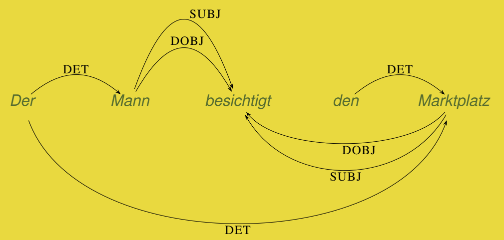


Constraining structures

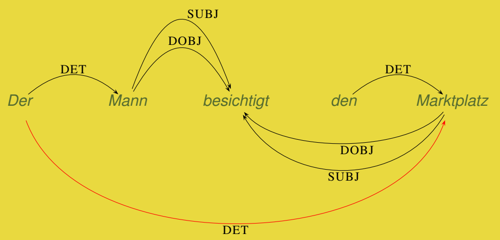


{X} : Root : Verb : 0.0 :
X↓cat=vfin → X↑cat=nil

Constraining structures

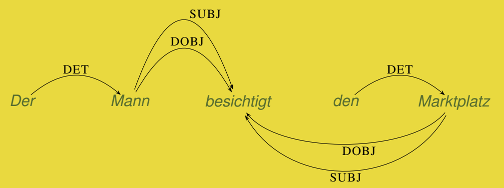


Constraining structures

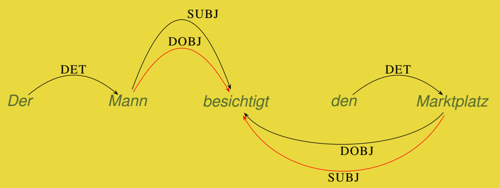


{X,Y} : Unique : General : 0.0 :
$X{\uparrow}id = Y{\uparrow}id \rightarrow X.label \neq Y.label$

Constraining structures

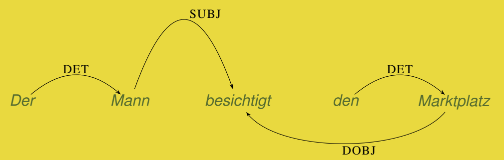


Constraining structures



{X,Y} : SubjAgr : Subj : 0.0 :
$X.label = \textsc{subj} \wedge Y.label = \textsc{det} \wedge X{\downarrow}id = Y{\uparrow}id \rightarrow Y{\uparrow}case = Y{\downarrow}case = nom$

Constraining structures



Weighted constraints

- no diagnosis, thus far
  - only error detection possible
  - diagnosis would require constraint retraction
  - constraint retraction makes the problem intractable
  - constraining information is lost
- the fundamental dilemma
  - constraints are needed to constrain the search space
  - diagnosis requires constraints to be retracted
- how can the lost information be compensated for?

Weighted constraints

- constraints are annotated with weights:
  $c_i \mapsto w_i,\ 0 \leq w_i < 1$
- depencency edges receive a scores: $e_j \mapsto s_j$
- the score of a dependency edge e_j which violates a constraint c_i is deprecated:

$$s_j = \prod w_i \mid \text{violates}(e_j, c_i)$$

Weighted constraints

- constraint *relaxation* instead of retraction
  - constraints (with a non-zero score) can be violated
  - constraints are weakened, but not neutralised
  - scores can guide the search for the best solution

Weighted constraints

- relaxation is not a repair mechanism, but the fundamental mechanism for evidence arbitration
  - handling of contradictory information, e.g. preferences
    - e.g. the subject usually preceedes the object
  - knowledge sources can be used even if they are uncertain

  additional constraining information becomes available!
- parsing becomes an optimisation problem
  - various optimisation procedures can be applied

Constraint violations

- many constraints are defeasible
- almost arbitrary input can be analysed
- constraint violations in the optimum solution can be interpreted as diagnoses

Constraint violations

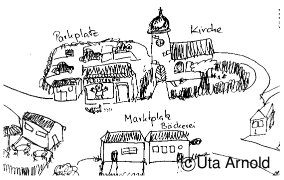
- combination with error simulation in a two phase-diagnosis (MENZEL AND SCHRÖDER 1998)
- constraint-based error simulation
  - is highly precise and supports multiple explanation perspectives
  - but requires a syntactic structure being given
- parsing as constraint optimisation
  - (so far) no precise error diagnosis in transitive correctness conditions (like agreement)
  - but provides a syntactic structure
- good synergy when used in combination

Constraint violations

- phase one:
  - constraint optimisation with WCDG
  - results in a syntactic structure
  - and possibly constraint violations
  - severe constraint violations can be directly explained
- phase two
  - build a constraint net using the syntactic relationships established in the first phase
  - perform a precise error simulation by propagating constraints

System design

- semantic preferences, world knowledge and context information can easily be integrated into the optimisation process



in_front_of(church,marketplace).
left_of(church,parking_lot).
at(market_place,bakers)
count(church,1).
...

- if the "world" contains just a single church, prefer the singular reading

System design

- preferences can also come from the interaction with the student
  - verbal: asking back
  - non-verbal: use direct manipulation gestures to illustrate the desired propositional content
- intention-based diagnosis becomes feasible

Error-sensitive parsing

- summing up
  - constraint retraction can render error anticipation superfluous
    - but: distinction between "incorrect" and "don't know" is lost
    - extremely careful grammar design becomes necessary
  - error simultation based on flexible parsing strategies makes possible to also diagnose structural faults
- grammar development is laborious and error-prone
  - requires expert knowledge

Error-sensitive parsing

- summing up (cont.)
  - weighted constraints extend the idea of constraint retraction to the parsing process in general
    - constraints are relaxed but not retracted
    - guiding information is kept
    - global optimisation facilitates the integration of all available evidence (including uncertain knowledge)
    - contextual expectations can be integrated into the parsing process

Summing up

- wide variety of techniques for grammar diagnosis available
  - anticipation-based vs. anticipation-free
  - instance-based vs. generalised
  - error simulation vs. constraint retraction
- none of them is perfect for every purpose

Summing up

- system design:
  choose the techniques matching your problem best
  - problem area
    - generalised methods are not easily applicable to irregular language phenomena
    - instance-based ones are unsuitable for communication-oriented exercises
  - human resources available
    - instance-based modelling is easy
    - generalisations are extremely hard to be made reliable

Summing up

- system design (cont.)
  - system resources available
    - generalised methods are resource-hungry
    - will your server handle this?
  - effort for scaling up
    - instance-based methods: easily predictable
      almost constant effort
    - generalised methods: hard to predict
      usually intensive debugging necessary

Summing up

- if your goal is a real application, then ”small is beautiful!”
  - better a limited but reliable solution
  - instead of an overambitious but buggy one
- if your goal is research, then ”think deep and think big!”
  - there is no ready-made solution of NLP for CALL
  - but there is a rich body of knowledge and techniques which must and can be adapted to the purpose of CALL applications

Summing up

- there is little awareness for CALL-specific requirements in the NLP community
- there is little awareness of NLP techniques in the CALL community
- building bridges is a necessity

Summing up

- How to proceed?
  - problem analysis:
    - What are the shortcomings of the current technology?
    - What prevents us from reaching our vision?
  - technology review:
    - How far is the problem solution from currently available technology?
    - Are there solutions adressing similar problems?
    - Can solutions be combined to achieve the desired behaviour?

Summing up

- How to proceed? (cont.)
  - system design and development:
    - Which basic functionality can be isolated?
    - Which basic functionality is required?
    - Can existing software be adapted/extended?
    - How can the necessary components be combined?
  - system evaluation:
    - corpus-based evaluation
    - field tests

Summing up

- generalised descriptions and anticipation-free diagnosis techniques brought us closer to the vision
  - communicate with your tutoring system in a close-to-reality setting
  - receive feedback about possibilities for improvement if desired
- there are prototype systems
  - for limited domains
  - for particular languages
  - for demonstration purposes only

Summing up

- thus, the problem isn't solved yet!
- badly needed:
  - better analysis and diagnosis techniques
  - more efficient solution methods
  - better authoring and debugging tools
  - better system integration
    - spelling correction
    - world knowledge
    - ...

Summing up

- badly needed: (cont.)
  - harmonised exercise designs
    - bringing together task, virtual world, help and feedback facilities, ...
  - better feedback modes
    - why always error explanations?
    - many possible alternatives: graphics, asking back, giving intuitive counterexamples, ...
  - innovative exercise types
    - *experiments* with limited NLP models