Natural Language Processing

Class 07:
Advanced topics in Fluid Construction Grammar

VRIJE UNIVERSITEIT BRUSSEL

20 April 2018

Katrien Beuls/ Paul Van Eecke

Assignment 1 feedback

Multiplying probabilities is not a good idea (numeric underflow): instead sum their logs

Use clear function and variable names, add enough comments to the code.

Every function should execute one specific “task”/”idea”. Use the docstring to explain this.

Assignment 1 feedback

Think about efficient data structures!!! (ESSENTIAL)

In this task you want to find bigrams efficiently

Dictionaries in Python have O(1) access

Best representation: dictionary with bigrams as keys and as values another dictionary with letters as keys and probabilities as values.

To save time and memory, write your model to a file and read it in (check if file exists, if not generate model)

Example JSON output
by Mourad Akondouch

(Australian English character LM)


```
{
    "aa": {
        "a": 0.026470588235294117,
        "b": 0.008823529411764706,
        "c": 0.03235294117647059,
        "d": 0.023529411764705882,
        "e": 0.0058823529411764705,
        "f": 0.020588235294117647,
        "g": 0.008823529411764706,
        "h": 0.011764705882352941,
        "i": 0.011764705882352941,
        "j": 0.0058823529411764705,
        "k": 0.008823529411764706,
        "l": 0.026470588235294117,
        "m": 0.008823529411764706,
        "n": 0.047058823529411764,
        "o": 0.008823529411764706,
        "p": 0.011764705882352941,
        "q": 0.0058823529411764705,
        "r": 0.06470588235294118,
        "s": 0.09705882352941177,
        "t": 0.026470588235294117,
        "u": 0.01764705882352941,
        "v": 0.0058823529411764705,
        "w": 0.08235294117647059,
        "x": 0.0058823529411764705,
        "y": 0.0058823529411764705,
        "z": 0.008823529411764706,
        "_": 0.09411764705882353
    },
    "ab": {
        "a": 0.025525186356449063,
```

Assignment 1 feedback

Code must be clear and self-explanatory but

README file is also important!

It allows others to understand the structure of your code immediately

About deadlines: plan ahead, don’t mail me two days before the deadline for an extension

Exam date

Problem for VUB 1MA students in exam schedule
(two exams planned on subsequent days)

We have to find a new date (**after 18/6**) and all students have to sign a paper stating that they agree, which then goes to the dean for approval

Doodle for new date (please complete before 1 May ‘18):
https://doodle.com/poll/9vra77k3tawxmu4r

Today

Recap of basic concepts of Fluid Construction Grammar

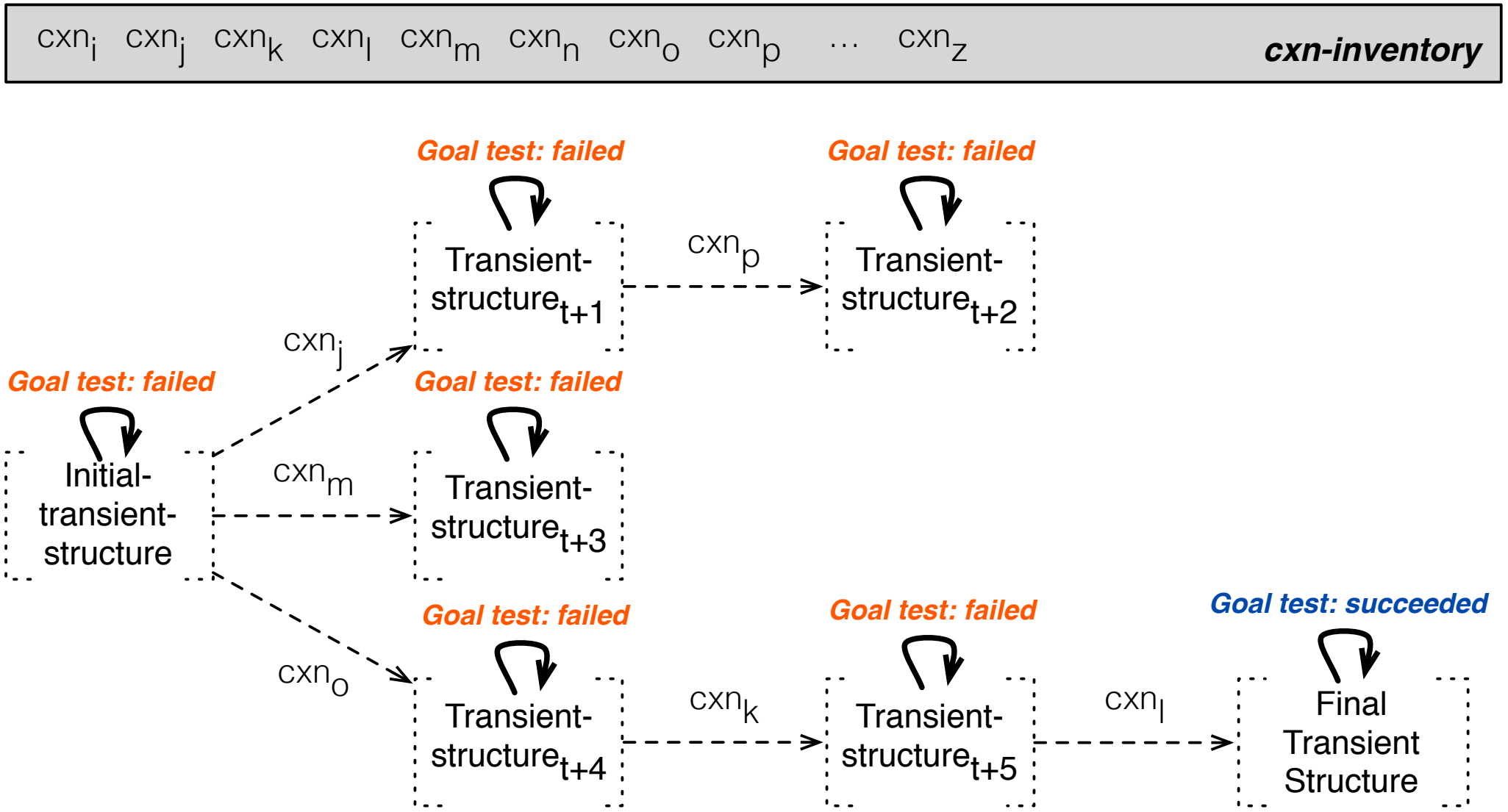
Managing complexity

Robustness and learning

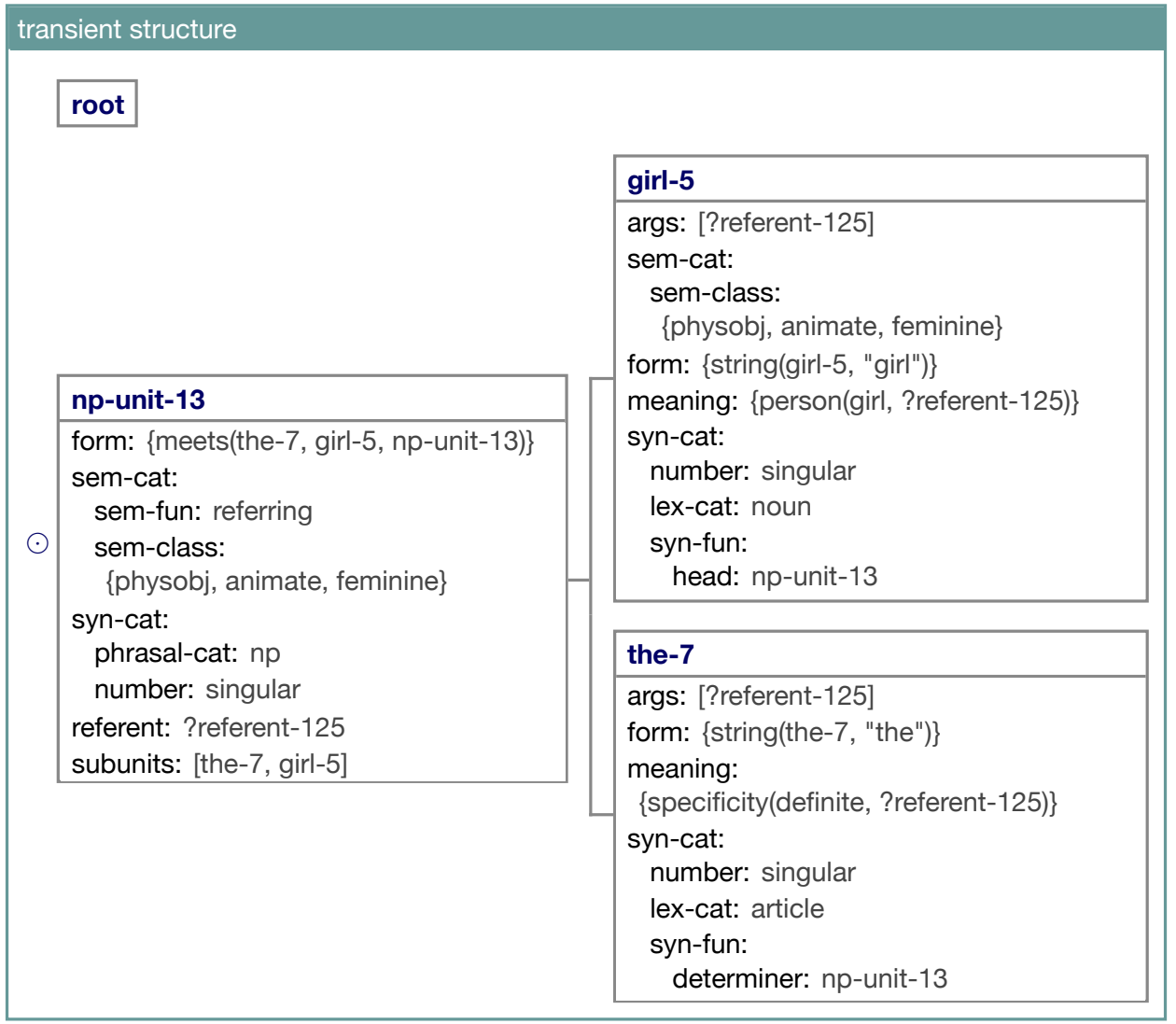
Installation

Problem solving in FCG

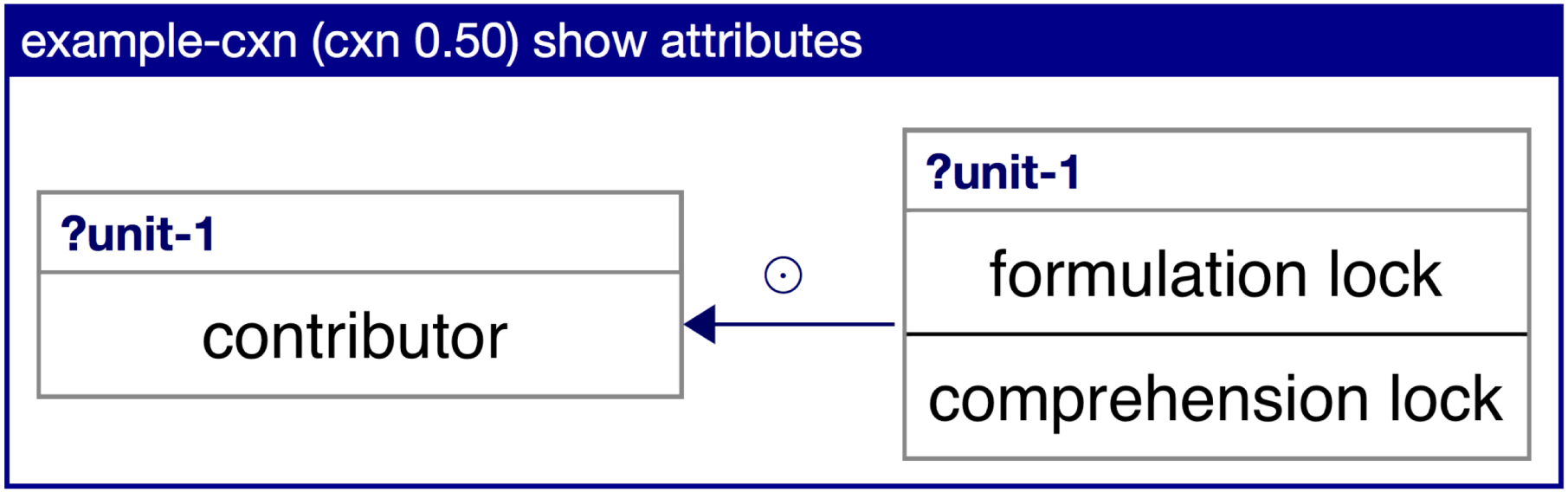
State Representation:	*Transient Structure*
Actions:	*Construction Schemas*
Initial State:	*Initial Transient Structure*
Goal State:	*Goal test returns true*
Action Selection	*E.g. Best-first with backtracking*


cxn_i cxn_j cxn_k cxn_l cxn_m cxn_n cxn_o cxn_p … cxn_z
cxn-inventory
Goal test: failed
Initial-transient-structure
cxn_j
cxn_m
cxn_o
Goal test: failed
Transient-structure_t+1
cxn_p
Goal test: failed
Transient-structure_t+2
Goal test: failed
Transient-structure_t+3
Goal test: failed
Transient-structure_t+4
cxn_k
Goal test: failed
Transient-structure_t+5
cxn_l
Goal test: succeeded
Final Transient Structure

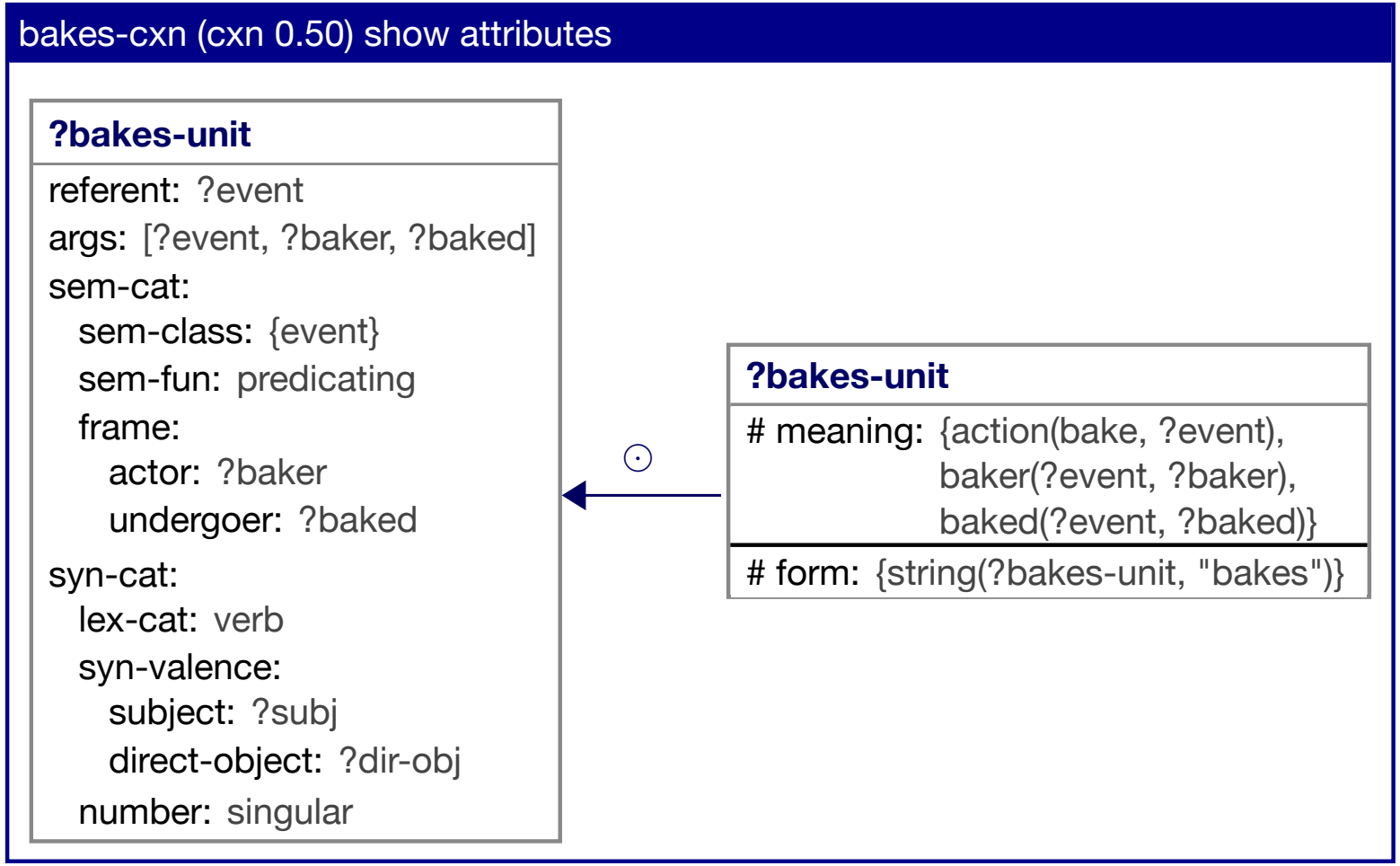
Transient structures



Construction schemas



Construction schemas



Managing complexity

VRIJE
UNIVERSITEIT
BRUSSEL

Construction application and search

Aim = find a pathway that leads from the initial transient structure to a transient structure that qualifies as a goal state

Two main mechanisms to steer cxn application process:

1. queue regulator
2. construction supplier

Construction application and search

(Most basic setting)

- Queue regulator selects transient structure that was most recently added to the queue (depth first)
- Construction supplier randomly picks a construction from the construction inventory

If cxn applies, add transient structure to queue

Else, backtrack to previous node in which not all cxns have been tried

Exploring the search tree in an uninformed way is not feasible for large search spaces!

Construction sets and construction networks

Internally structure the constructions in the construction inventory

1. Divide cxns into number of ordered sets (formulation/comprehension) >> cxn supplier only supplies constructions from current set

2. Priming networks inform the cxn supplier to first supply cxns that are primed by outgoing links (trained on test corpus or in multi-agent experiment)

Hashing constructions

Morphological and lexical constructions allow hashing for quick retrieval based on string/meaning predicate

Hash table with strings and meaning predicates as keys and a list of constructions that match these as values

Hugely improves performance with large cxn inventories

Scoring constructions and transient structures

Construction supplier can take into account cxn scores to select next construction

Queue regulator sorts transient structures based on score (how far this state is from a solution state)

- Little research in this area
- Most commonly used heuristic = nr of cxns that have applied so far

Grammar design

General rule: avoid that two constructions with conflicting features can apply to the same transient structure

Underspecify corresponding features instead of splitting the search tree into multiple hypotheses

- E.g. morphological construction for indefinite article “a”/”an” should only be applied after the noun phrase has been build and the first letter of the following word is known

Goal tests and solutions

Every node in the search tree contains a transient structure

At the moment of creation, all goal tests specified are run

- If all tests succeed, node is returned as a solution
- If one or more tests fail, search process continues

Comprehension goal tests: other cxns can still apply, ROOT unit contains strings, meaning predicates not linked into a single semantic network

Formulation goal tests?

Rendering

Final transient structure (solution state) is rendered

In comprehension:

- Render method extracts all predicates from the meaning feature in every unit of the transient structure
- Drawn as a semantic network

In formulation:

- Render method extracts all predicates from the form feature in every unit of the transient structure
- String predicates concatenated taking into account the ordering constraints in the other predicates

Robustness and learning

VRIJE
UNIVERSITEIT
BRUSSEL

Robustness is crucial

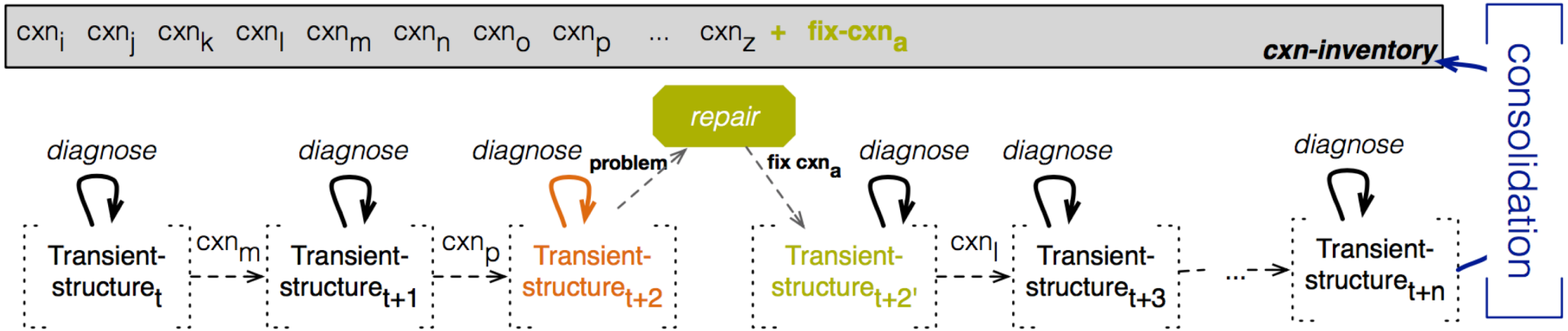
- Language use is creative, open-ended and full of innovations
- Language needs to be learned and needs to adapt to novel situations

Grammar and processing mechanisms need to be flexible enough to handle the utterances and meaning representations that deviate from the norm

Meta-layer architecture

Routine layer and meta-layer

- Diagnostics
- Repairs
- Consolidation strategies



Diagnostics

Run after each construction application

Inspect the resulting transient structure for any abnormalities or potential problems

Create a problem of a certain type, e.g. ‘unknown-string’

Diagnostics have access to the complete construction application process, including the search tree, the previously applied constructions, the transient structures and the construction inventory

Repairs

Methods implementing problem solving strategies

Specialize on one or more classes of problems that are triggered by diagnostics

Operates on same node as diagnostic, or earlier nodes

Repair tries to find a solution in the form of a fix object (open-ended)

- Typically, fix-cxn

Also specified on grammar-level and stored in cxn-inventory

Consolidation strategies

Designed to store solutions to these problems for later reuse during routine processing

- E.g. add fix-cxns to construction inventory (only successful ones!)

Not easy to strike the optimal generality-specificity balance for a fix

Library of basic diagnostics and repairs

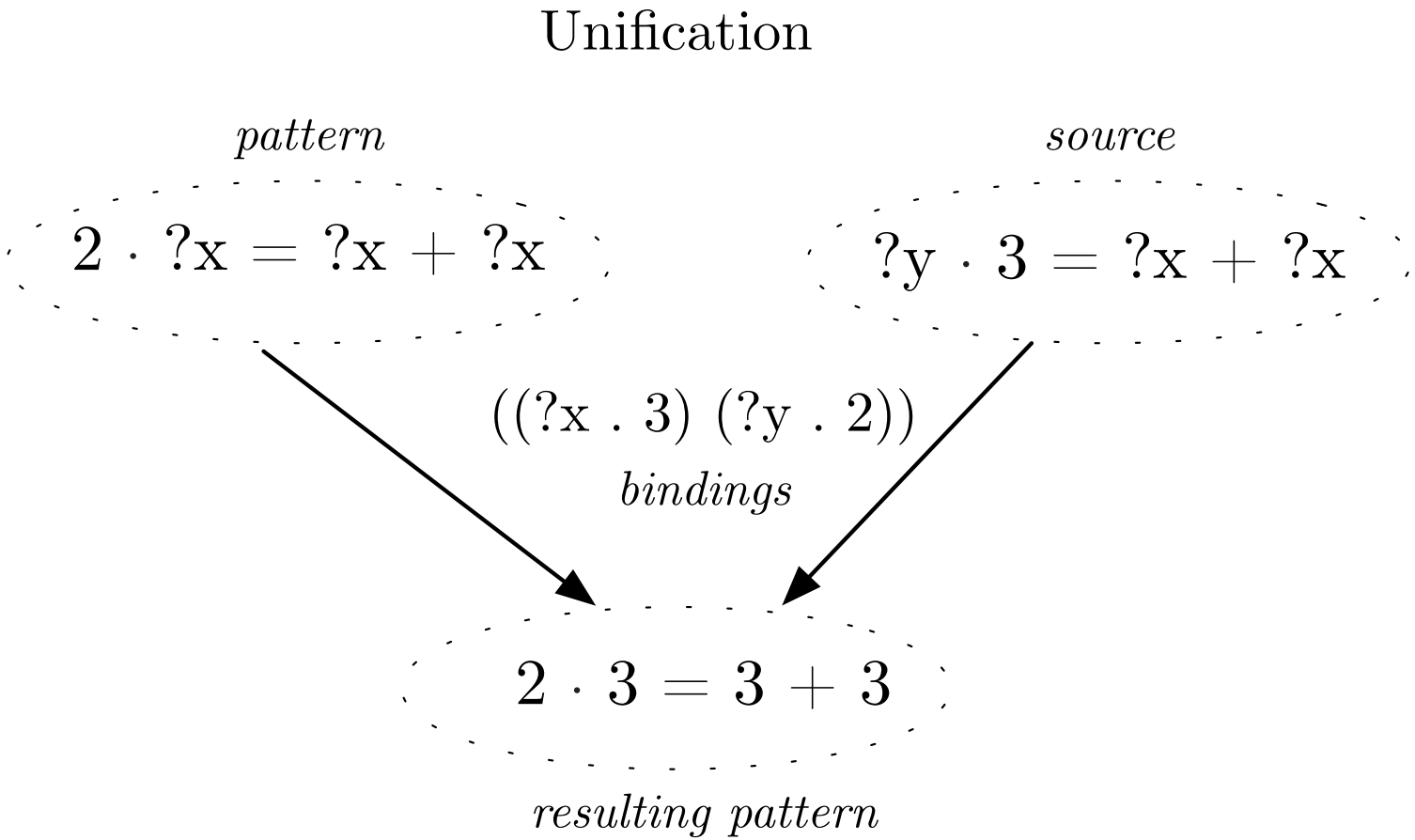
Problem	Diagnostic	Repair	Consolidation
Unknown-word (comprehension)	No more applicable cxns + strings in root	Create new lexical cxn (hypothesized meaning).	Add to cxn-inventory.
Unknown-word (formulation)	No more applicable cxns + meaning predicates in root.	Create new lexical cxn (new form).	Add to cxn-inventory.
Missing-phrasal-cxn (comprehension or formulation)	No more applicable cxns + unconnected meaning + anti-unification possible with low cost.	Anti-unification of existing construction with transient structure.	Pro-unification and add to cxn-inventory.
Matching-conflict (comprehension or formulation)	No more applicable cxns + unconnected meaning + anti-unification not possible with low cost.	Create new phrasal cxn (variable equalities and word order / markers).	Pro-unification and add to cxn-inventory.

Generalising constructions using anti-unification

= Opposite of unification:
Least general generalisation of two or more terms

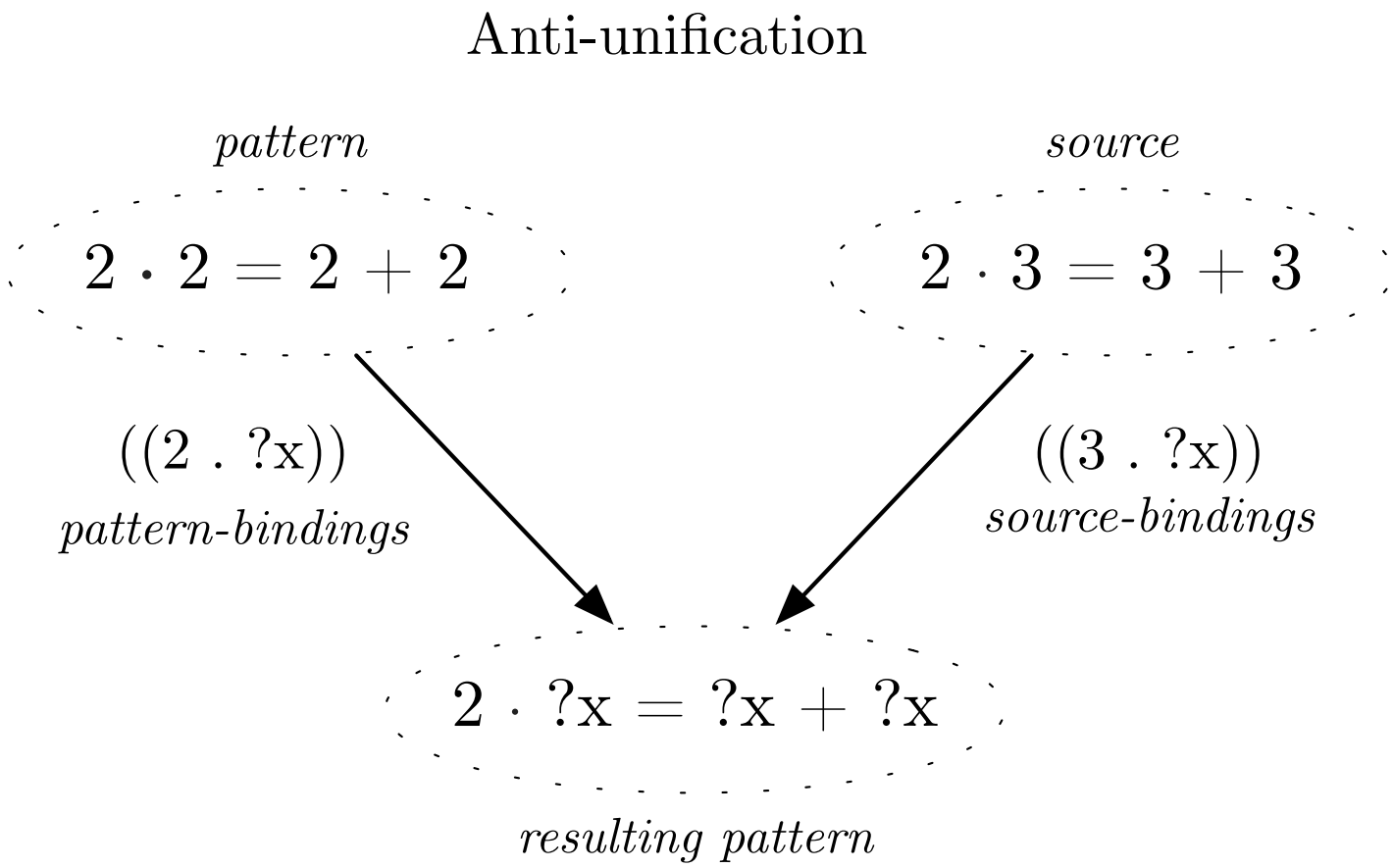
Unification

- Most general specialisation (MGS)
- Minimal set of bindings that (after substitution) makes the two terms equal
- Resulting pattern computed by substituting the set of bindings obtained through unification



Anti-unification

- Least general specialisation (LGS)
- Two sets of bindings
- Resulting pattern cannot be reliably computed through the substitution of bindings
- Always succeeds!



About pattern and source

Operations involved in construction application are not commutative

Match operation includes subset constraint

Pattern: construction
Source: transient structure
Resulting pattern: construction

Cost calculation

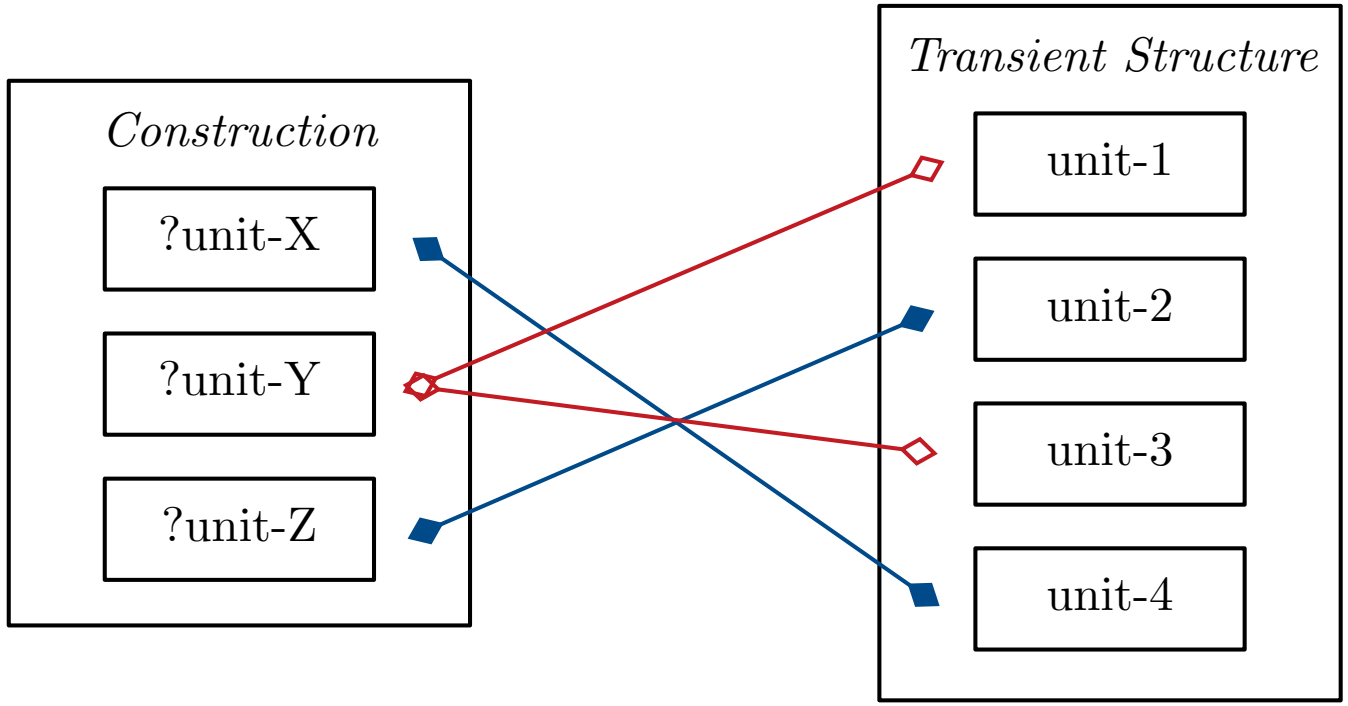
Generalisations might become meaningless...

Cost reflects in how far the construction needed to be generalised before it could apply to the transient structure

Repair prefers construction that required least amount of generalisation

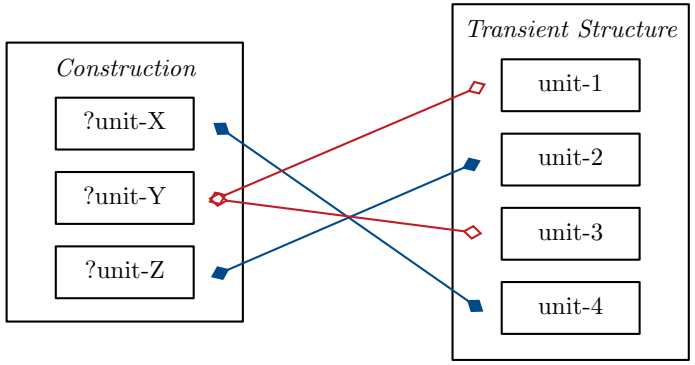
1. Pairing units

Pattern and Source

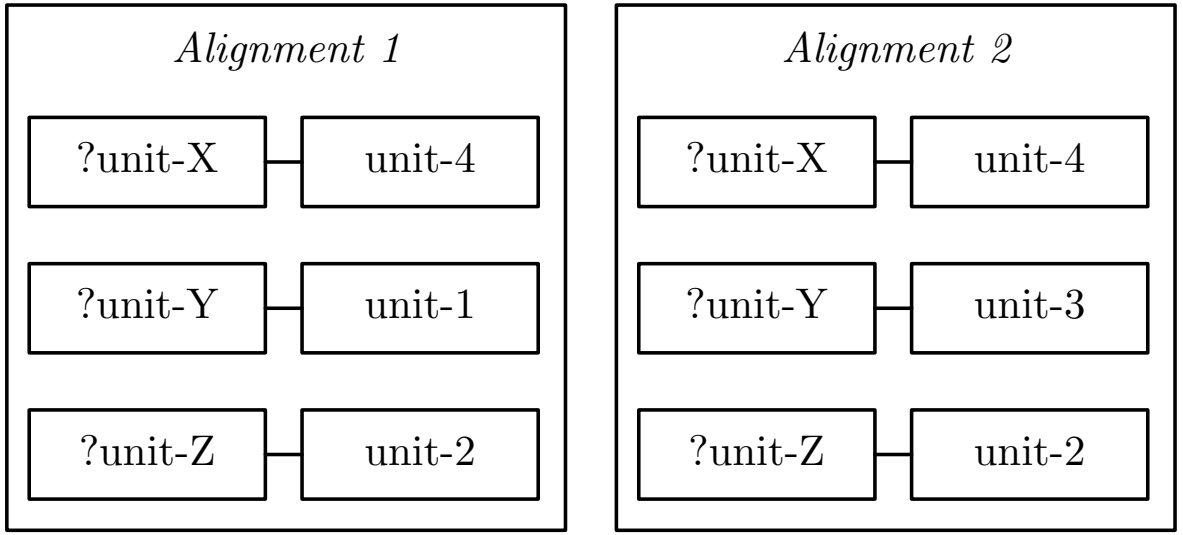


Three possible strategies

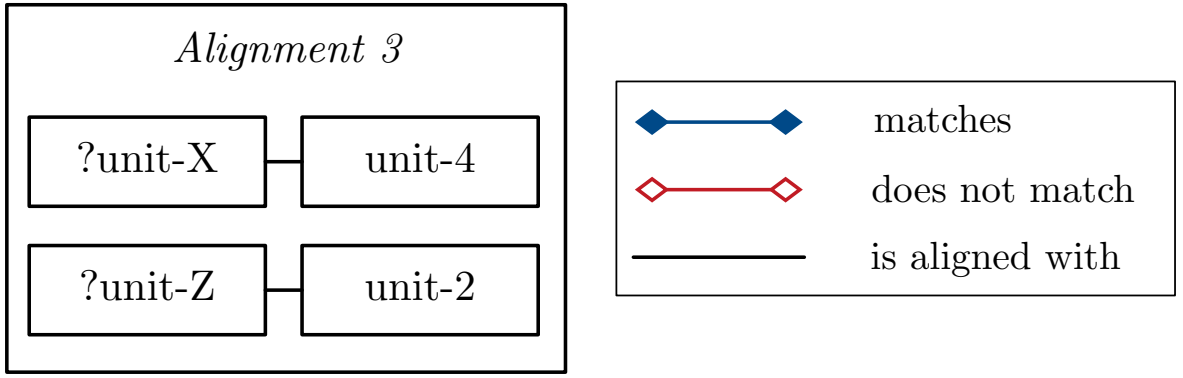
Pattern and Source



Unit Pairing (no deletion)



Unit Pairing (deletion)



2. Anti-unifying features and values

Feature type system for influencing the behaviour of the unification algorithm (subset unification, negation, overwriting)

>> Anti-unification algorithm needs to take into account the meaning of these feature types to appropriately generalise features and values

Demonstration

Simple French noun phrases

Grammar contains lexical constructions and one noun phrase construction

Four types of generalisation:

1. variable decoupling
2. value relaxation
3. feature/predicate deletion
4. unit deletion

Variable decoupling

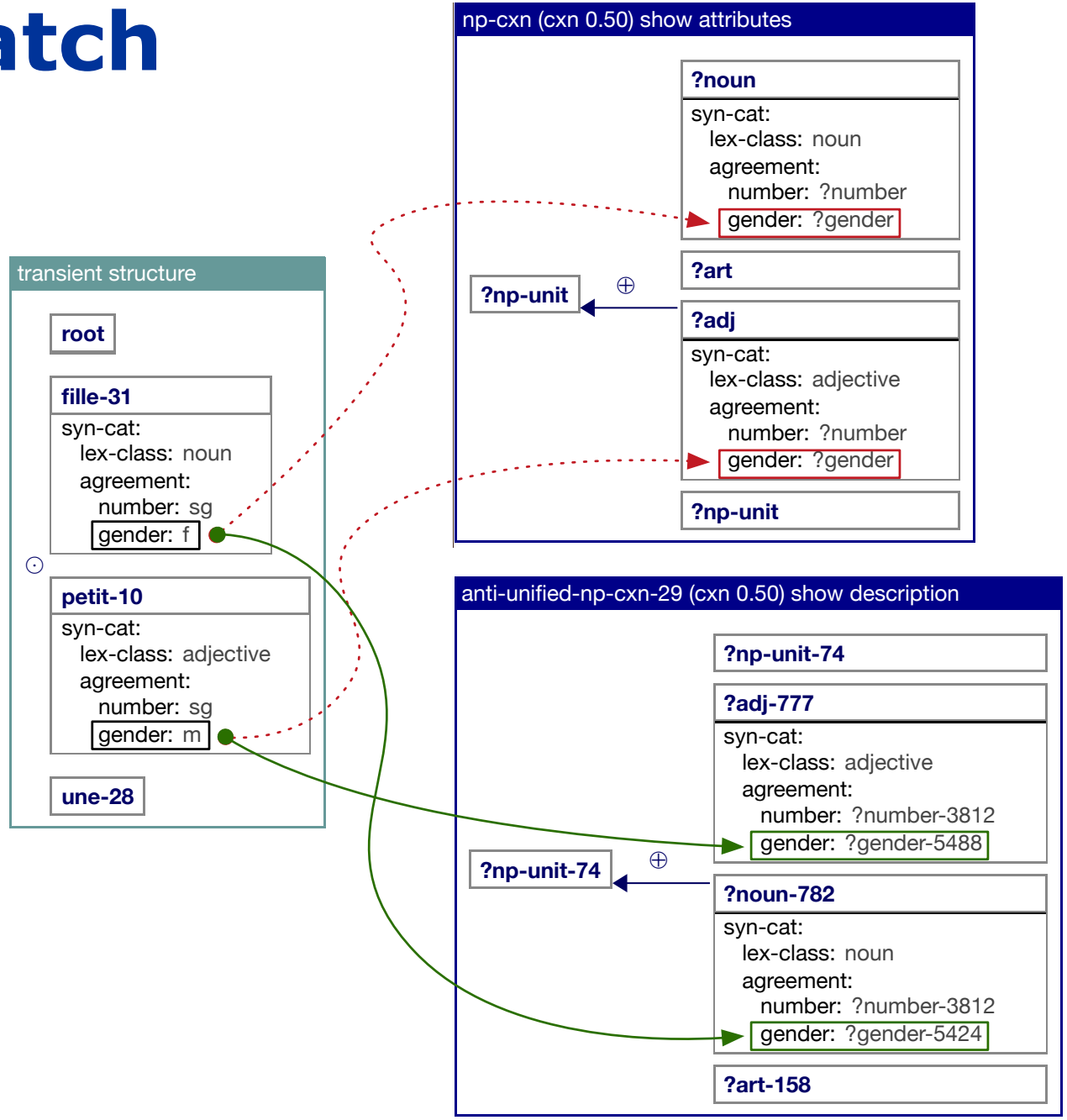
Same variable occurs at multiple places in the construction and different, non-unifying values occur at these places in the transient structure

Solution: ”decoupling the variables”

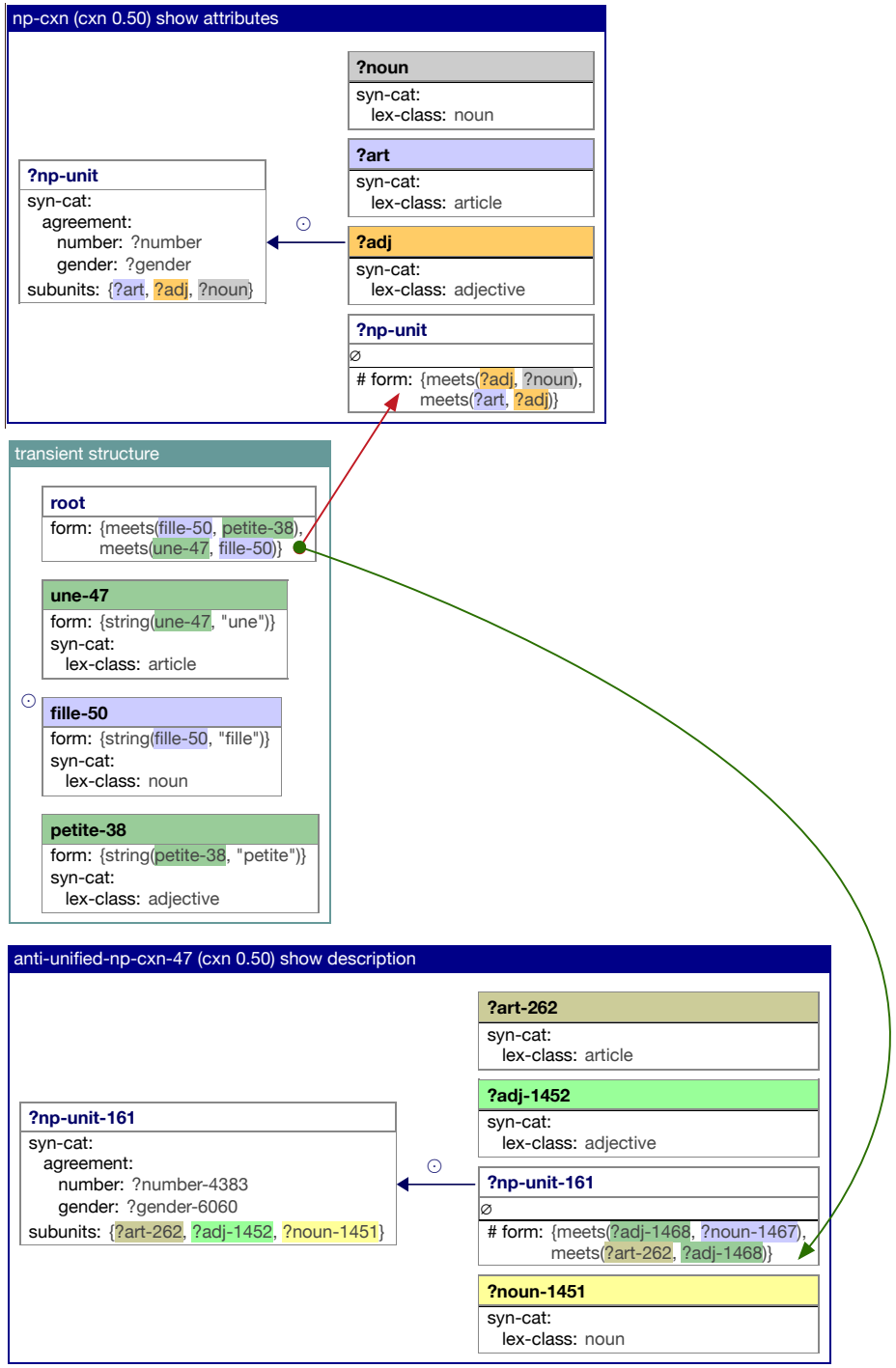
Examples:

- Gender mismatch
- Deviating word order

Gender mismatch



Deviating word order

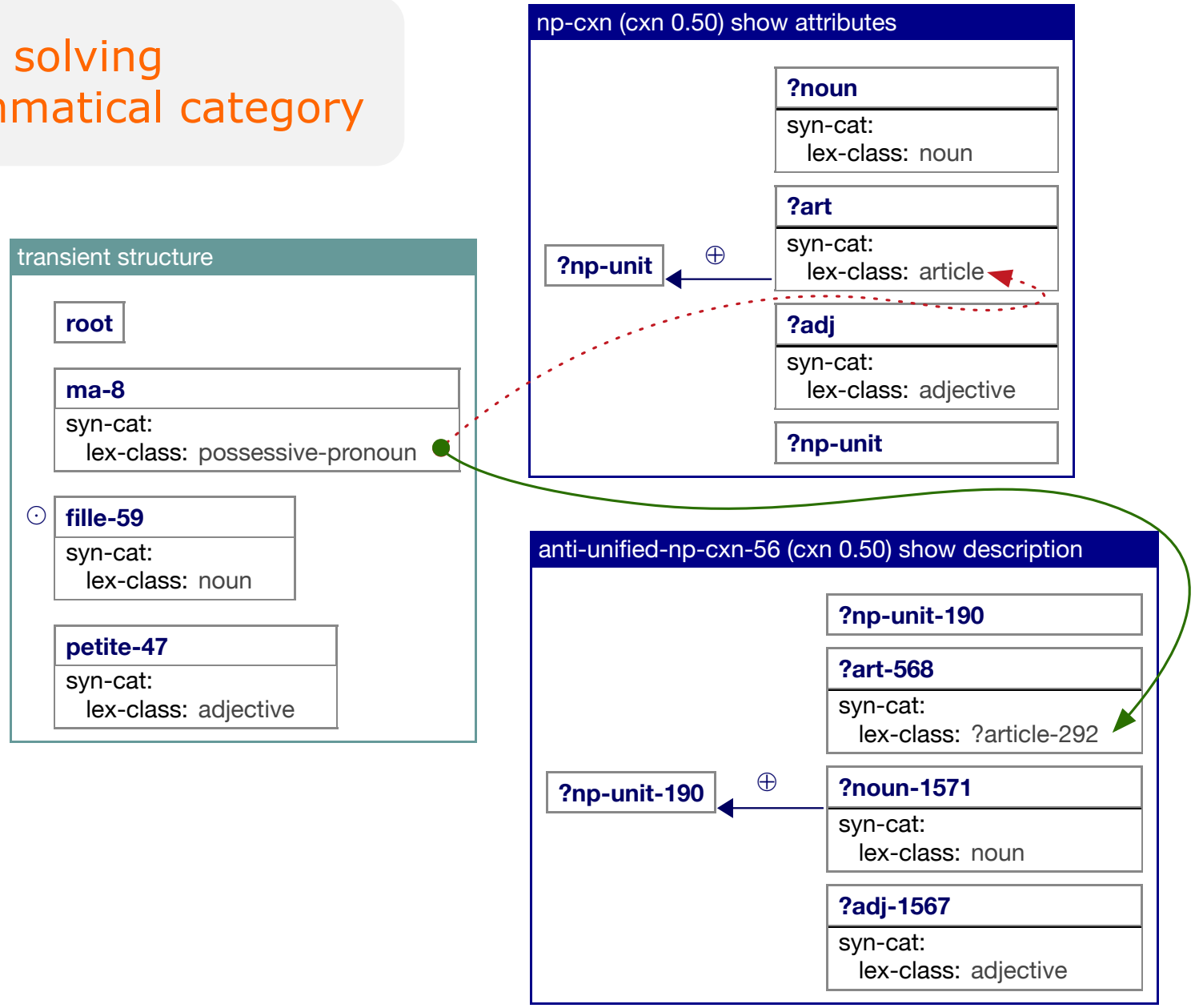


Value relaxation

Applied when a value of a particular feature in the construction is different from the value of that feature in the transient structure

Solution: feature value replaced with a free variable

Value relaxation for solving a mismatch in grammatical category

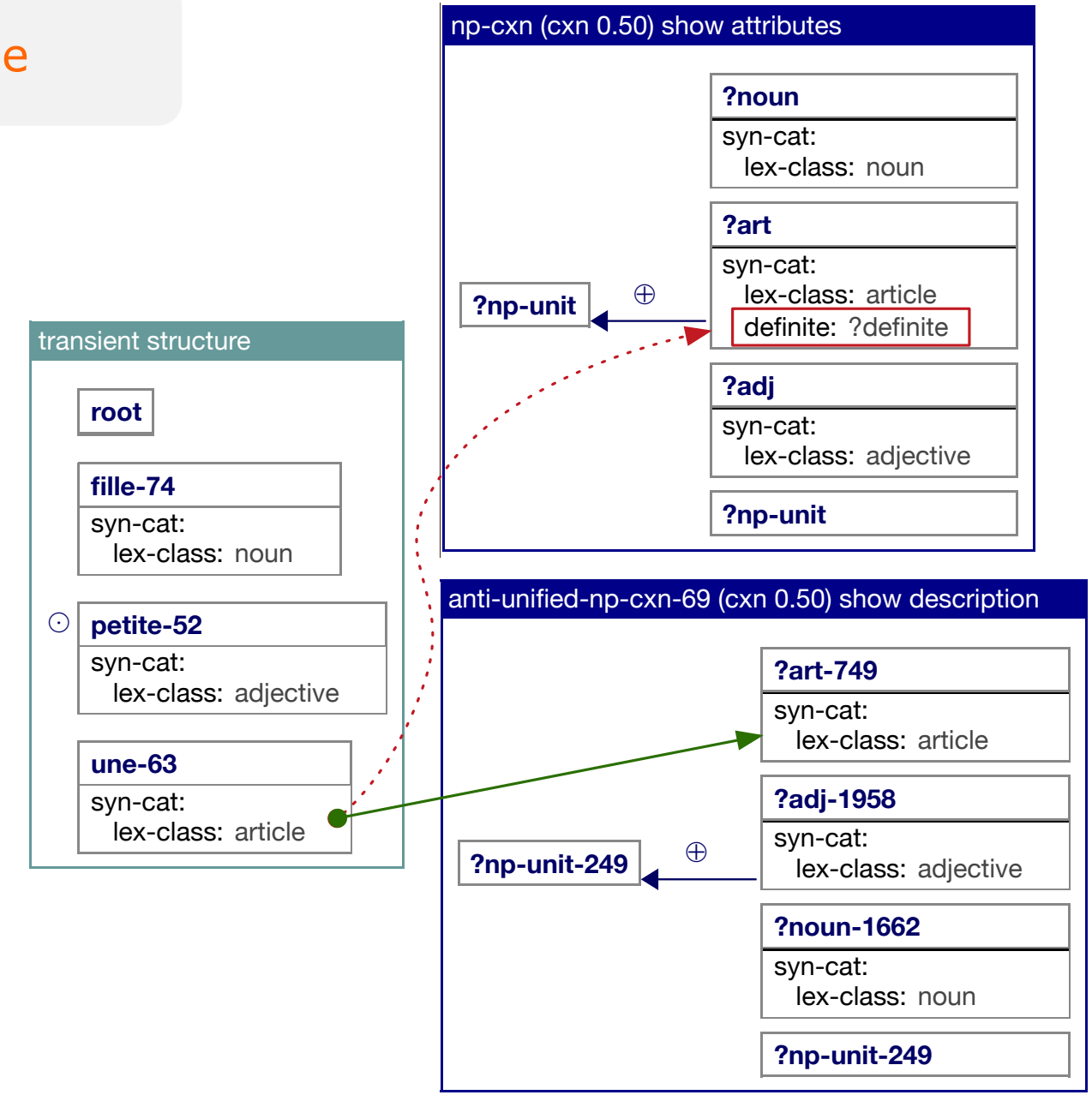


Feature/predicate deletion

Happens when a feature or predicate in the construction is not found at the corresponding place in the transient structure

Solution: deleting the feature/predicate from the construction

Feature deletion example



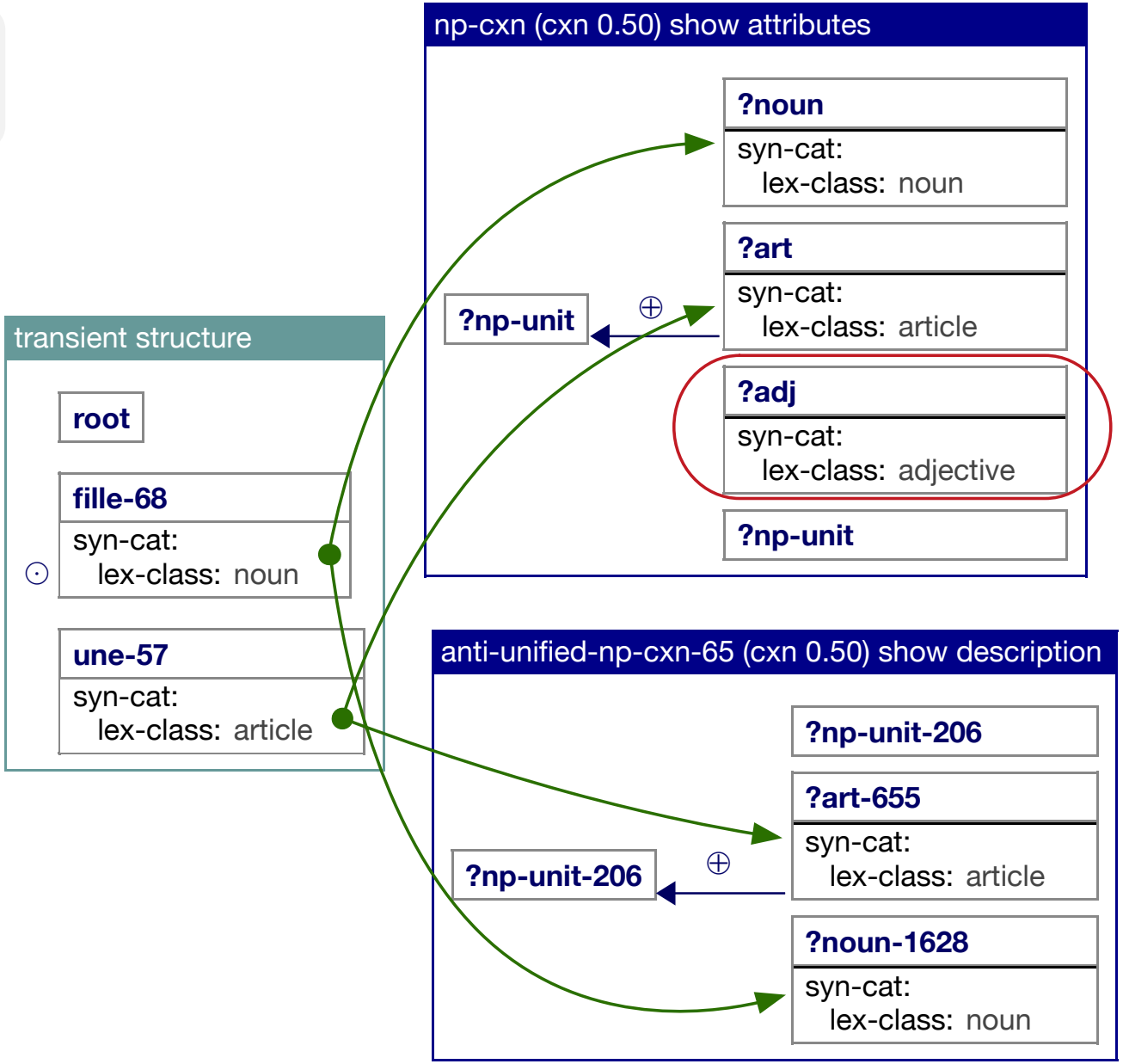
Unit deletion

Applied when a unit from the construction does not find a suitable unit in the transient structure to match with

Always considered but high cost!

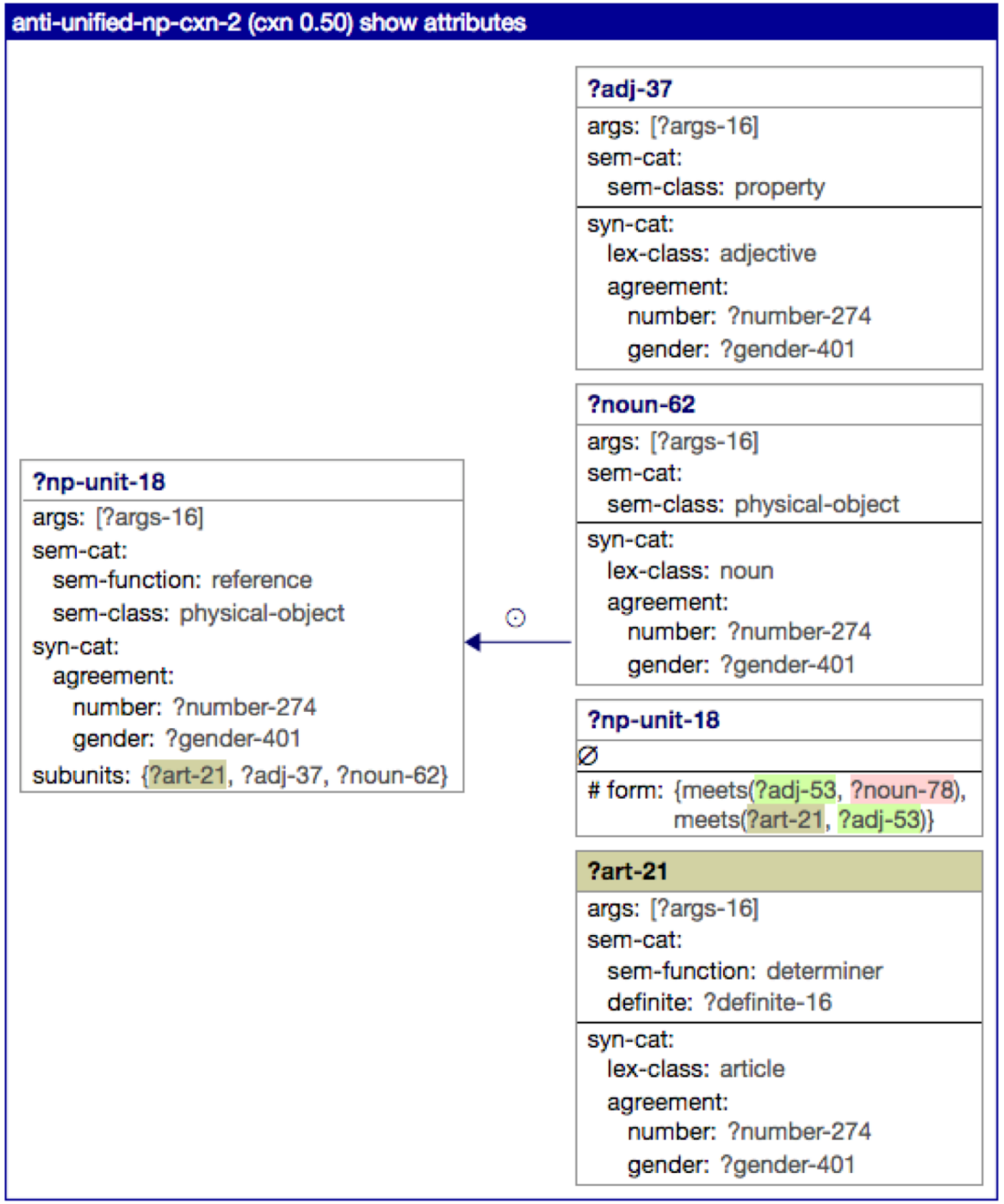
Only returned as best solution when cost of anti-unifying features of that unit with any available unit is higher than deleting the unit

Unit deletion example



Problem

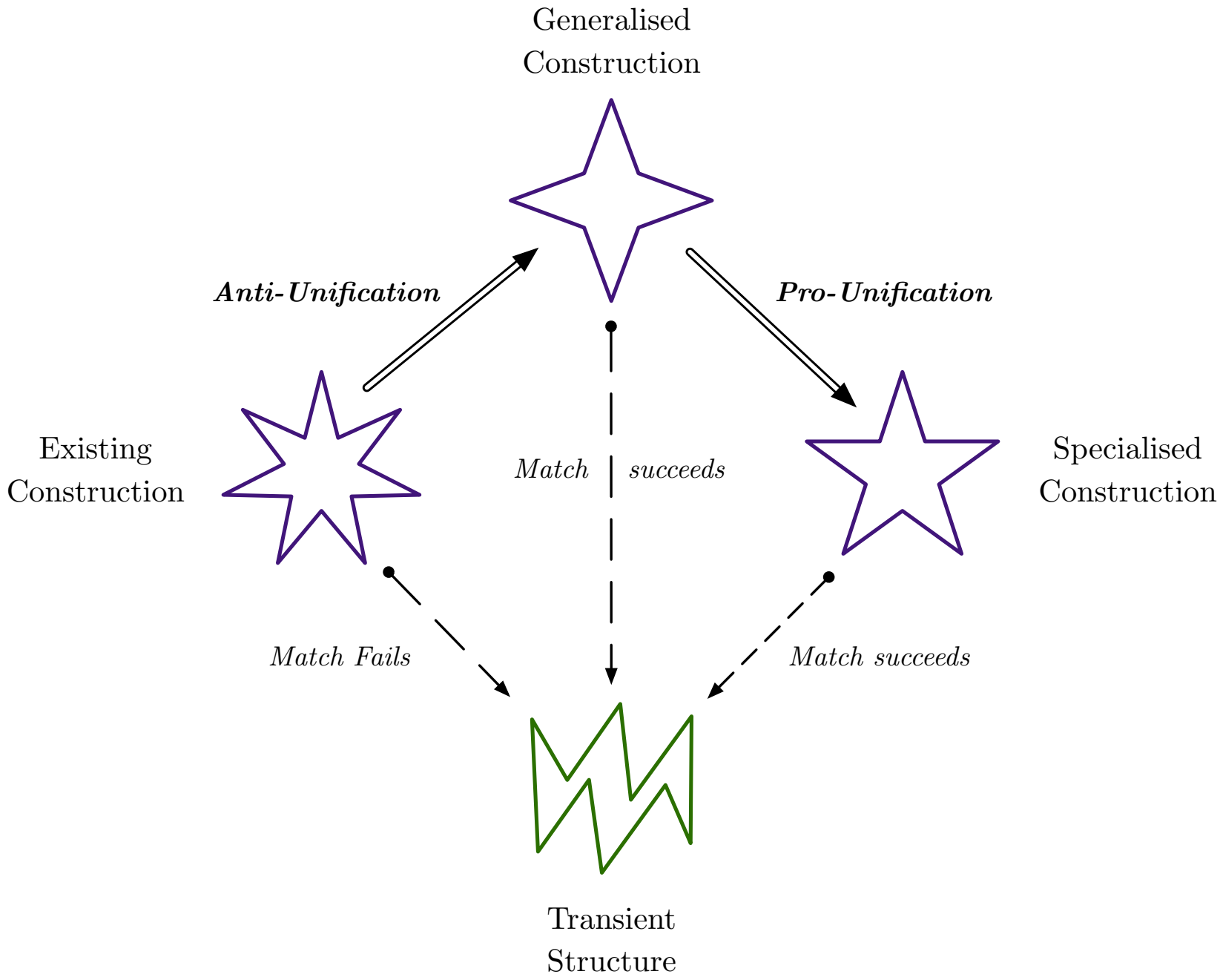
Generalised cxns
too unconstrained
to be stored for later use



Learning new constructions also requires a specialization step

Integrates specific elements from the observation into the new construction

Word order example: encode observed word order into the new construction


Generalised
Construction
Anti-Unification
Pro-Unification
Existing
Construction
Match succeeds
Specialised
Construction
Match Fails
Match succeeds
Transient
Structure

Pro-unification

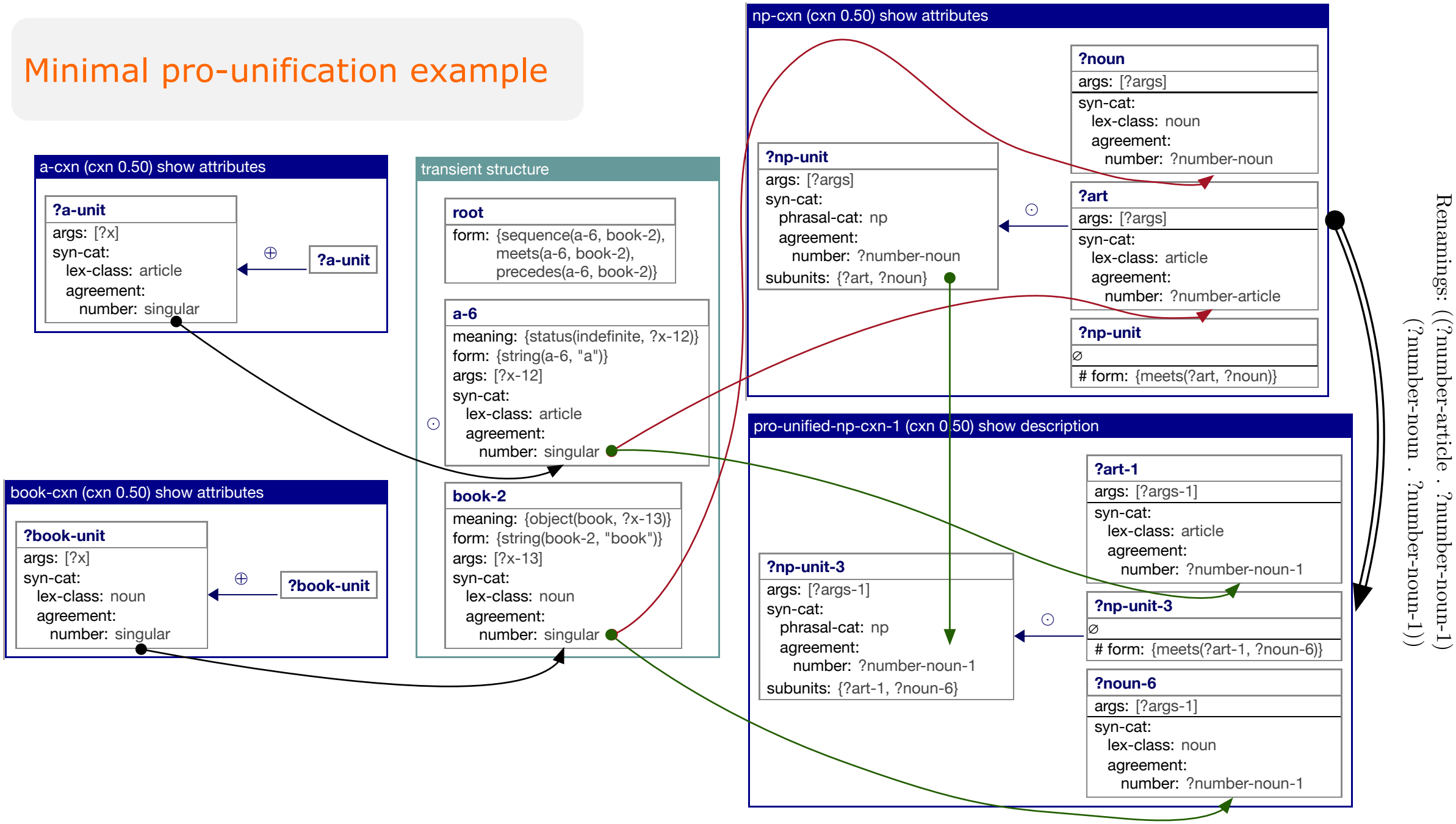
Difficult to find an adequate level of specialisation:

- Incorporate additional constraints
- But not all constraints ...

Choice of elements from transient structure to incorporate is experiment and grammar specific

Pro-unification = collection of strategies

Minimal pro-unification example



Integration into FCG's meta-layer architecture

Diagnostic: 'no-match-or-solution'

Repair: `anti-unify-pro-unify'

Consolidation strategy: `add-cxn'

Homework: Installation of FCG software

Installation scripts for OS X and Linux:

https://github.com/EvolutionaryLinguisticsAssociation/Babel2/wiki/Installation-Script

Installation instructions for Windows:

https://github.com/EvolutionaryLinguisticsAssociation/Babel2/wiki/Windows-Installation

Send me an e-mail if you are stuck

Master thesis topics 2018-2019

Available Monday 23 April 4pm on Pointcarré

FCG-related, multi-agent language evolution experiments in VR worlds, NLU for virtual assistants

Make an appointment to discuss topics (suggest your own) or visit us on the Open Day on Wednesday 25/4

Pointers

https://www.fcg-net.org/demos/insight-grammar-learning/

https://github.com/EvolutionaryLinguisticsAssociation/Babel2/wiki/Syntax-and-Semantics-of-FCG

Next class in Leuven: 26/4

Prof. em. Luc Steels

https://www.arts.kuleuven.be/english/francqui-chair

Language grounding based on cognitive semantics and pragmatics

26 April 2018, 16:00 – 18:00

MSI1 01.16, Mgr. Sencie Instituut Erasmusplein 2, 3000 Leuven

=> No class on 27 April