Navigable atom-rule interactions in PSL models enhanced by rule verbalizations, with an application to etymological inference

Verena Blaschke, Thora Daneyko, Jekaterina Kaparina,
Zhuge Gao & Johannes Dellert

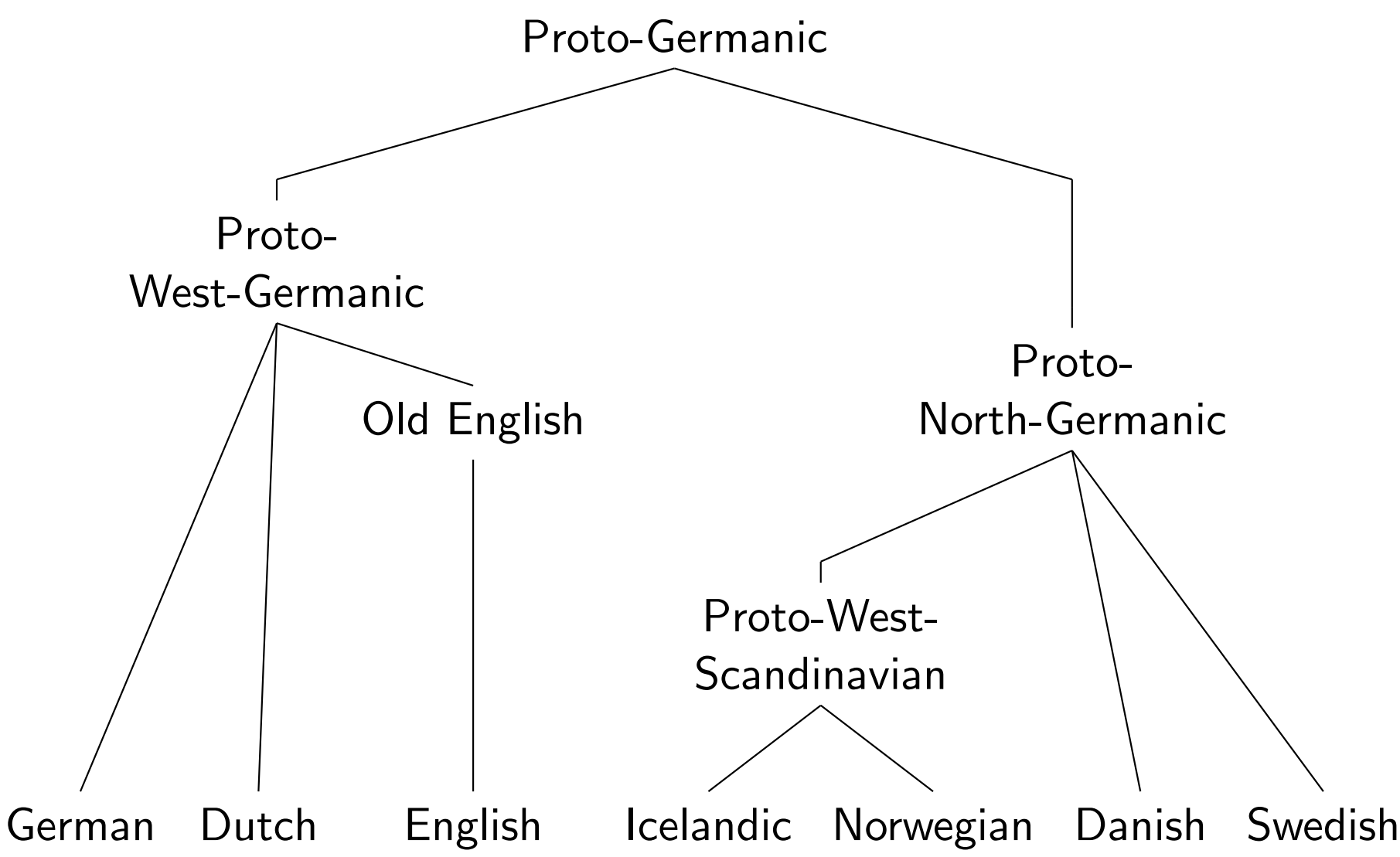
ILP 2022, Windsor (UK), September 29, 2022

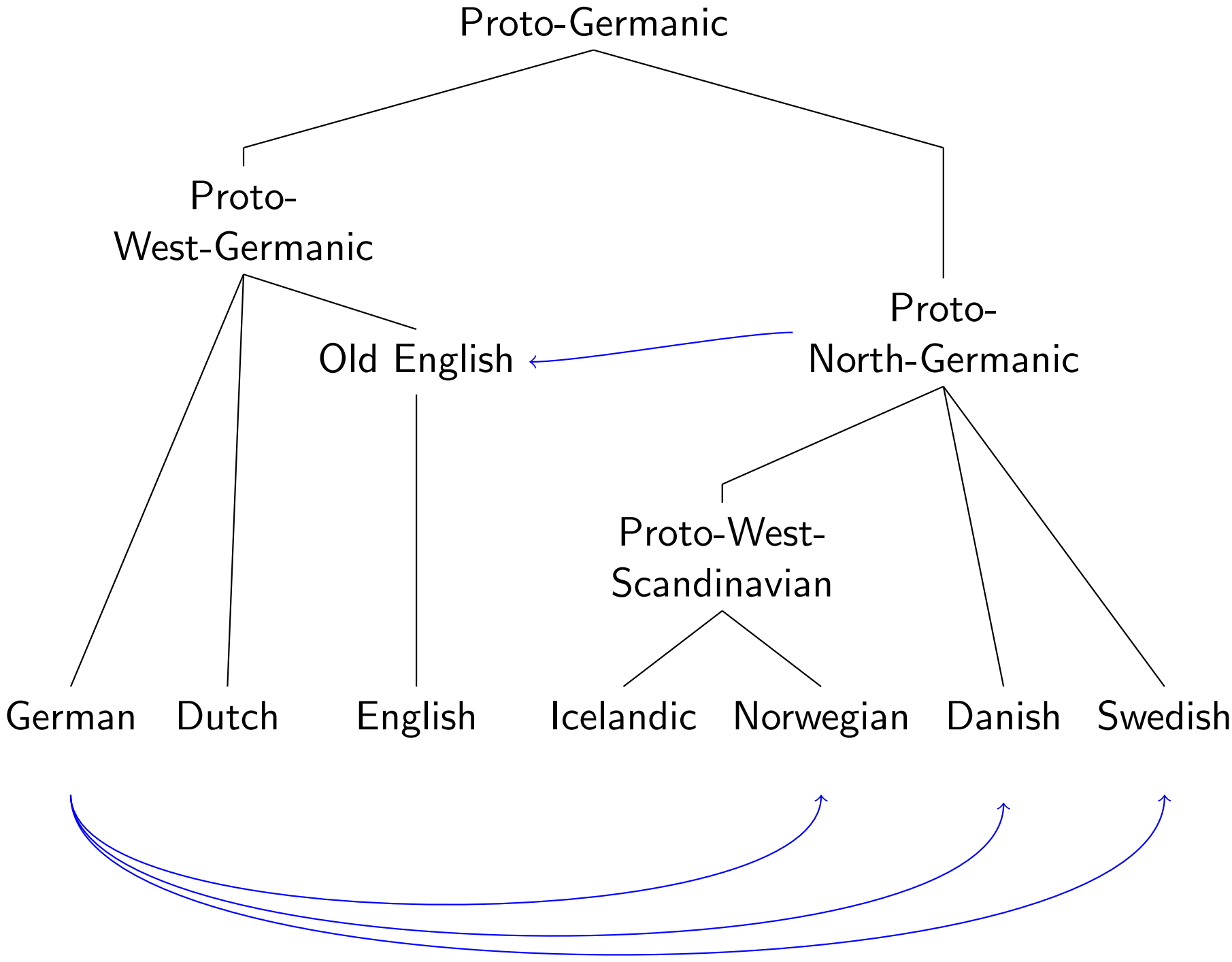
University of Tübingen

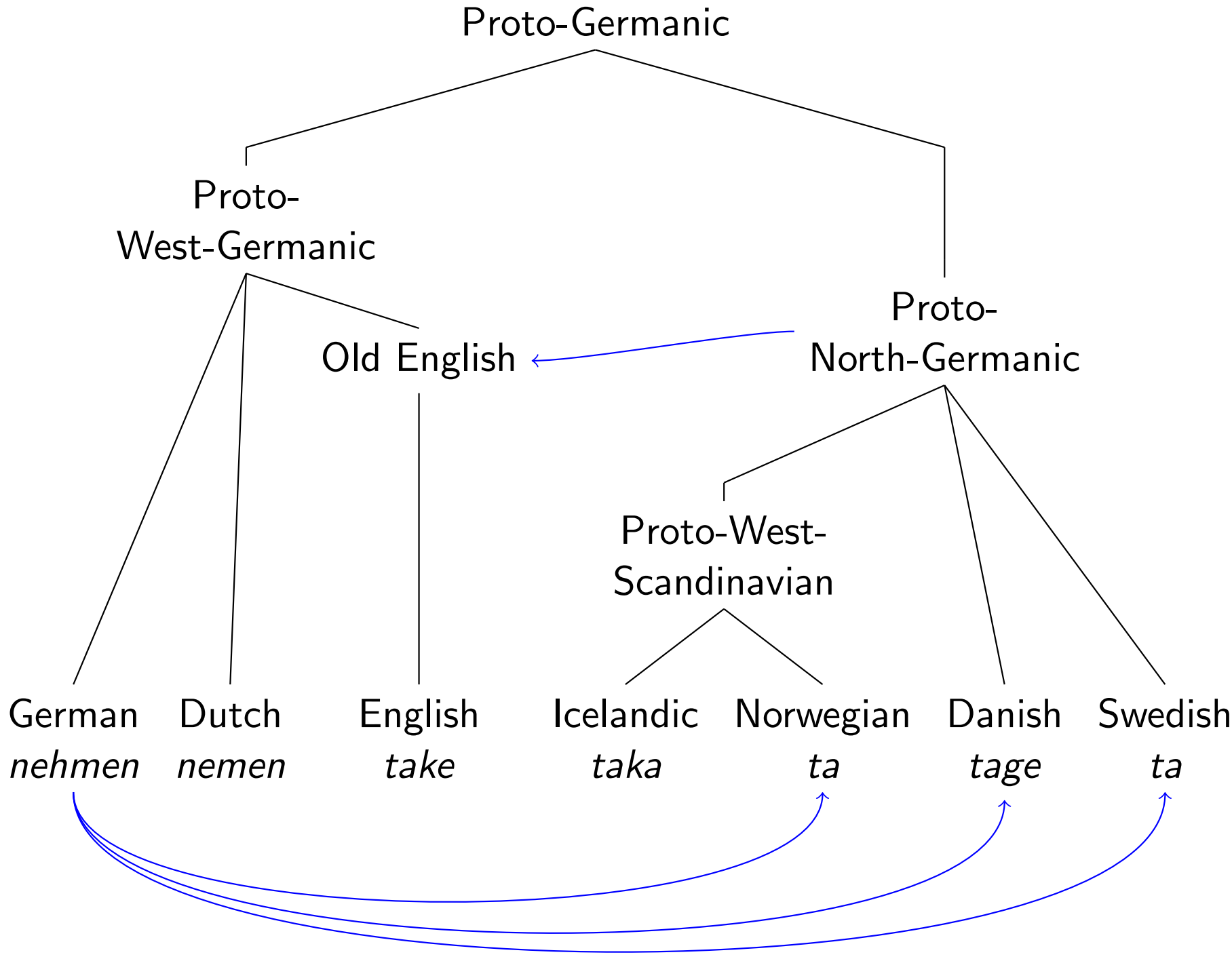
Probabilistic soft logic[1]

- Templating language for a probabilistic graphical model (hinge-loss Markov random fields)
- First-order logic + statistics
- Used for modelling relation graphs – in our case in the context of historical linguistics

[1] Bach, S. H., Broecheler, M., Huang, B., & Getoor, L. (2017). "Hinge-Loss Markov Random Fields and Probabilistic Soft Logic." *JMLR* **18**(109)


Proto-Germanic
Proto-
West-Germanic
Old English
Proto-
North-Germanic
Proto-West-
Scandinavian
German
Dutch
English
Icelandic
Norwegian
Danish
Swedish


Proto-Germanic
Proto-
West-Germanic
Old English
Proto-
North-Germanic
Proto-West-
Scandinavian
German
Dutch
English
Icelandic
Norwegian
Danish
Swedish


Proto-Germanic
Proto-
West-Germanic
Old English
Proto-
North-Germanic
Proto-West-
Scandinavian
German
nehmen
Dutch
nemen
English
take
Icelandic
taka
Norwegian
ta
Danish
tage
Swedish
ta

PSL atoms

- **Atom**
  `Inherited(X,Y)`, `Borrowed(X,Y)`, `UnknownOrigin(X)`, `Similar(X,Y)`, ...
- **Ground atom**
  `Inherited("English ‘take’", "Old English ‘…’")`

  `Borrowed("Old English ‘…’", "Proto-North-Germanic ‘…’")`

  `UnknownOrigin("Proto-West-Germanic ‘…’")`

  `Similar("English ‘take’", "Norwegian ‘ta’")` $= 0.7$

  ...

  $\rightarrow$ We want to assign scores $\in [0; 1]$ to the ones whose values are initially undefined

PSL rules

- **Logical rules**
  Disjuncts of literals, can be written as implications of the form

```
P1(A,B) & P2(A,B) -> P3(A,B) | P4(A,B)
```

```
Inherited(X,Z) & Inherited(Y,Z) & (X != Y)
  -> Similar(X,Y)
```

- **Arithmetic rules**: (in)equalities

```
Similar(X,Y) = Similar(Y,X) .
```

```
Borrowed(X,Y) + Borrowed(Y,X) <= 1 .
```

```
Inherited(X,+Y) + Borrowed(X,+Z)
  + UnknownOrigin(X) = 1 .
```

- Ground rules: where all atoms are ground atoms

```
Similar("German ‘nehmen’","Dutch ‘nemen’") =
Similar("Dutch ‘nemen’","German ‘nehmen’") .
```

Distance to satisfaction

- Arithmetic rules are **satisfied** if their (in)equalities are fulfilled
- Logical rules are **satisfied** if the consequent's score is at least as high as the antecedent's score:
  - Łukasiewicz logic, e.g. conjunction:
    $A \,\&\, B := \max\{A + B - 1, 0\}$

```
Inherited(X,Z) & Inherited(Y,Z) & (X != Y)
    -> Similar(X,Y)
```

```
  Inherited("German ‘nehmen’","P-W-Gmc ‘…’")    0.9
& Inherited("Dutch ‘nemen’","P-W-Gmc ‘…’")      0.8
& ("German ‘nehmen’" != "Dutch ‘nemen’")        1.0
-> Similar("German ‘nehmen’","Dutch ‘nemen’")       0.9
```

Distance to satisfaction

- Arithmetic rules are **satisfied** if their (in)equalities are fulfilled
- Logical rules are **satisfied** if the consequent's score is at least as high as the antecedent's score:
  - Łukasiewicz logic, e.g. conjunction:
    $A \,\&\, B := \max\{A + B - 1, 0\}$

```
Inherited(X,Z) & Inherited(Y,Z) & (X != Y)
    -> Similar(X,Y)
```

```
  Inherited("German ‘nehmen’","P-W-Gmc ‘…’")     0.9 ⎫
& Inherited("Dutch ‘nemen’","P-W-Gmc ‘…’")       0.8 ⎬ 0.7
& ("German ‘nehmen’" != "Dutch ‘nemen’")         1.0 ⎭  ≤
-> Similar("German ‘nehmen’","Dutch ‘nemen’")          0.9
```

✔

Distance to satisfaction

```
  Inherited("Icelandic ‘taka’","P-W-Sca ‘…’")      0.9 }
& Inherited("Norwegian ‘ta’","P-W-Sca ‘…’")        0.9 } 0.8
& ("Icelandic ‘taka’" != "Norwegian ‘ta’")        1.0 }  ≤
-> Similar("Icelandic ‘taka’","Norwegian ‘ta’")          0.7
                                                         ✗
```

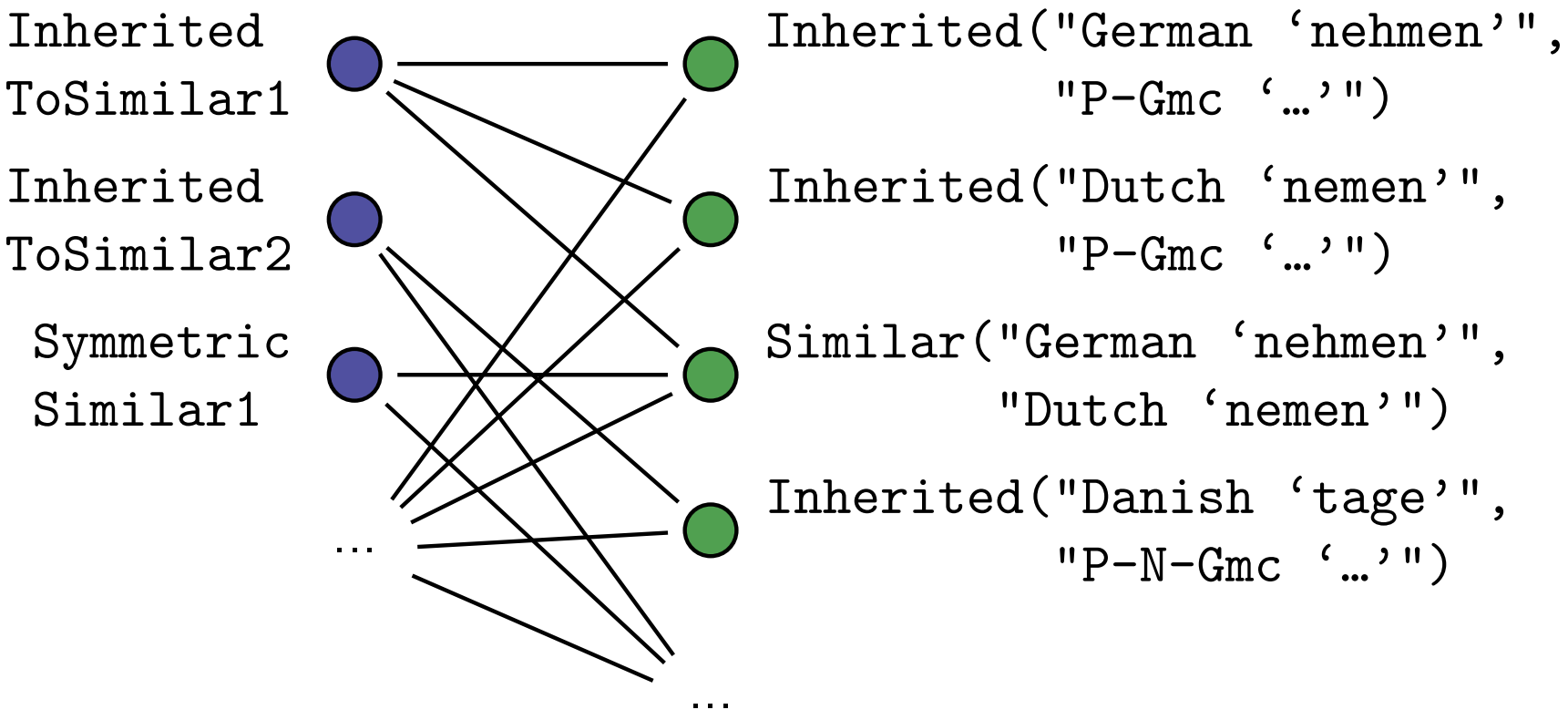
- **Distance to satisfaction:** *How* unsatisfied is a rule?
  max{antecedent score - consequent score, 0}
  (arithmetic constraints are a natural extension)

Distance to satisfaction

- **Distance to satisfaction:** *How* unsatisfied is a rule?
  max{antecedent score - consequent score, 0}
  (arithmetic constraints are a natural extension)
- Rules can be constraints (that should always be satisfied) or weighted according to how much their dissatisfaction should be penalized
  - Rule weights can be learned!
- PSL inference goal: assign atom values in such a way that the weighted sum of distances to satisfaction is minimized (MAP estimate)

RAG

Rule-atom graph



Very large $\rightarrow$ hard to interpret

Existing debugging tool: VMI-PSL[2]

Rule Overview

ID	Rule	Weighted	Count	Total Dissatisfaction	Mean Dissatisfaction
1	SGD_RATING(U, I) → RATING(U, I)	true	19799	567.63	0.03
2	USER(U) & AVG_USER_RATING(U) & ITEM(I) & RATED(U, I) → RATING(U, I)	true	15125	721.17	0.05
3	RATING(U, I) → SGD_RATING(U, I)	true	18623	692.60	0.04
4	ITEM_PEARSON_RATING(U, I) → RATING(U, I)	true	19769	369.31	0.02
5	RATING(U1, I) & RATED(U1, I) & RATED(U2, I) & SIM_PEARSON_USERS(U1, U2) → RATING(U2, I)	true	14505	1137.08	0.08
6	RATING(U1, I) & RATED(U1, I) & USERS_ARE_FRIENDS(U1, U2) & RATED(U2, I) → RATING(U2, I)	true	39662	2629.90	0.07
7	BPMF_RATING(U, I) → RATING(U, I)	true	19809	312.74	0.02
8	USER(U) & ITEM(I1) & RATED(U, I1) & ITEM(I2) & RATED(U, I2) & SIM_MF_EUCLIDEAN_ITEMS(I1, I2) & RATING(U, I1) → RATING(U, I2)	true	11623	797.98	0.07
9	USER(U) & ITEM(I1) & RATED(U, I1) & ITEM(I2) & RATED(U, I2) & SIM_MF_COSINE_ITEMS(I1, I2) & RATING(U, I1) → RATING(U, I2)	true	11111	754.27	0.07
10	RATED(U, I1) & RATED(U, I2) & SIM_ADJCOS_ITEMS(I1, I2) & RATING(U, I1) → RATING(U, I2)	true	0	0.00	0.00
11	USER(U) & RATING(U, I) & ITEM(I) & RATED(U, I) → AVG_USER_RATING(U)	true	18159	4348.77	0.24
12	USER(U1) & RATING(U1, I) & RATED(U1, I) & SIM_MF_EUCLIDEAN_USERS(U1, U2) & ITEM(I) & RATED(U2, I) & USER(U2) → RATING(U2, I)	true	5878	414.48	0.07
13	RATING(U, I) → ITEM_PEARSON_RATING(U, I)	true	19091	484.80	0.03
14	USER(U1) & RATING(U1, I) & RATED(U1, I) & ITEM(I) & RATED(U2, I) & SIM_MF_COSINE_USERS(U1, U2) & USER(U2) → RATING(U2, I)	true	5892	446.29	0.08
15	RATING(U1, I) & RATED(U1, I) & SIM_COSINE_USERS(U1, U2) & RATED(U2, I) → RATING(U2, I)	true	22874	1431.46	0.06
16	RATING(U, I) → BPMF_RATING(U, I)	true	19807	559.84	0.03
17	RATED(U, I1) & SIM_PEARSON_ITEMS(I1, I2) & RATED(U, I2) & RATING(U, I1) → RATING(U, I2)	true	20179	1394.86	0.07

- (Largely) focused on analyzing rules (aggregate (dis)satisfaction, number of groundings)
- Our approach: focus on ground atoms (examine atoms with surprising/undesired inferred values)

[2]Rodden, A., Salh, T., Augustine, E., & Getoor, L. (2020). "VMI-PSL: Visual Model Inspector for Probabilistic Soft Logic." *RecSys '20*, pp. 604–606. Graphic via supplementary material (demo video)

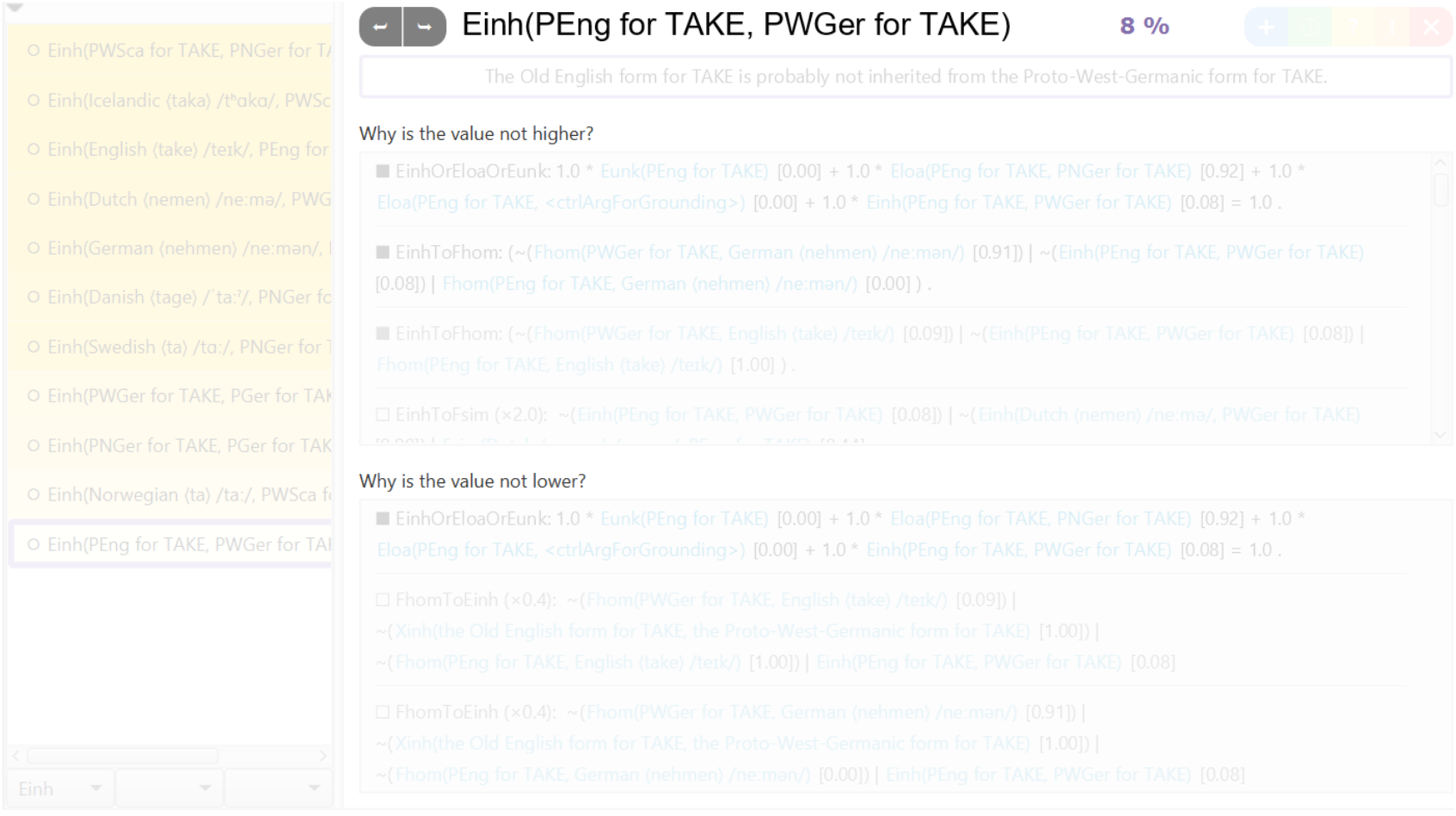
Our RAG inspection tool

`Inherited("Old English '…'", "P-West-Germanic '…'")`

Atom sorting/filtering

Rules and associated atoms

Upward/downward pressure



Upward/downward pressure

- Each rule-atom link exerts upward and/or downward pressure on the value of the atom

```
Inherited(X,Z) & Inherited(Y,Z) & (X != Y)
    -> Similar(X,Y)
```

- If unsatisfied:
  - upward pressure on the value of `Similar(X,Y)`
  - downward pressure on `Inherited(X,Z)` and `Inherited(Y,Z)`

Upward/downward pressure

```
Inherited(X,Z) & Inherited(Y,Z) & (X != Y)
    -> Similar(X,Y)
```

- If satisfied: potentially relevant for explaining
  - why `Similar(X,Y)` receives a higher value than expected
  - why `Inherited(X,Z)` or `Inherited(Y,Z)` receives a lower value than expected

Upward/downward pressure

```
Borrowed(X,Y) + Borrowed(Y,X) <= 1 .
```

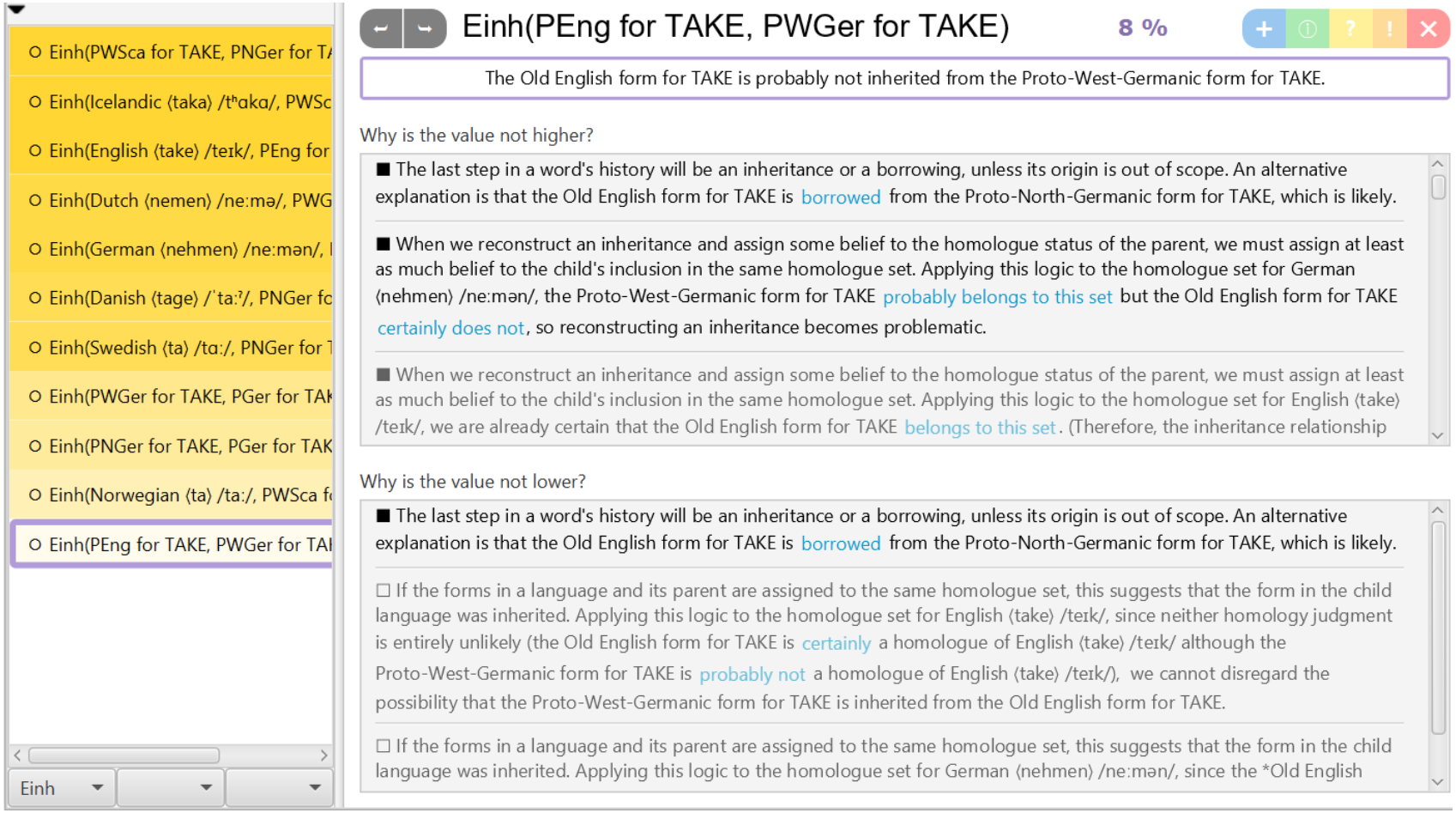
- If unsatisfied:
  - downward pressure on `Borrowed(X,Y)` and `Borrowed(Y,X)`

Rule activity

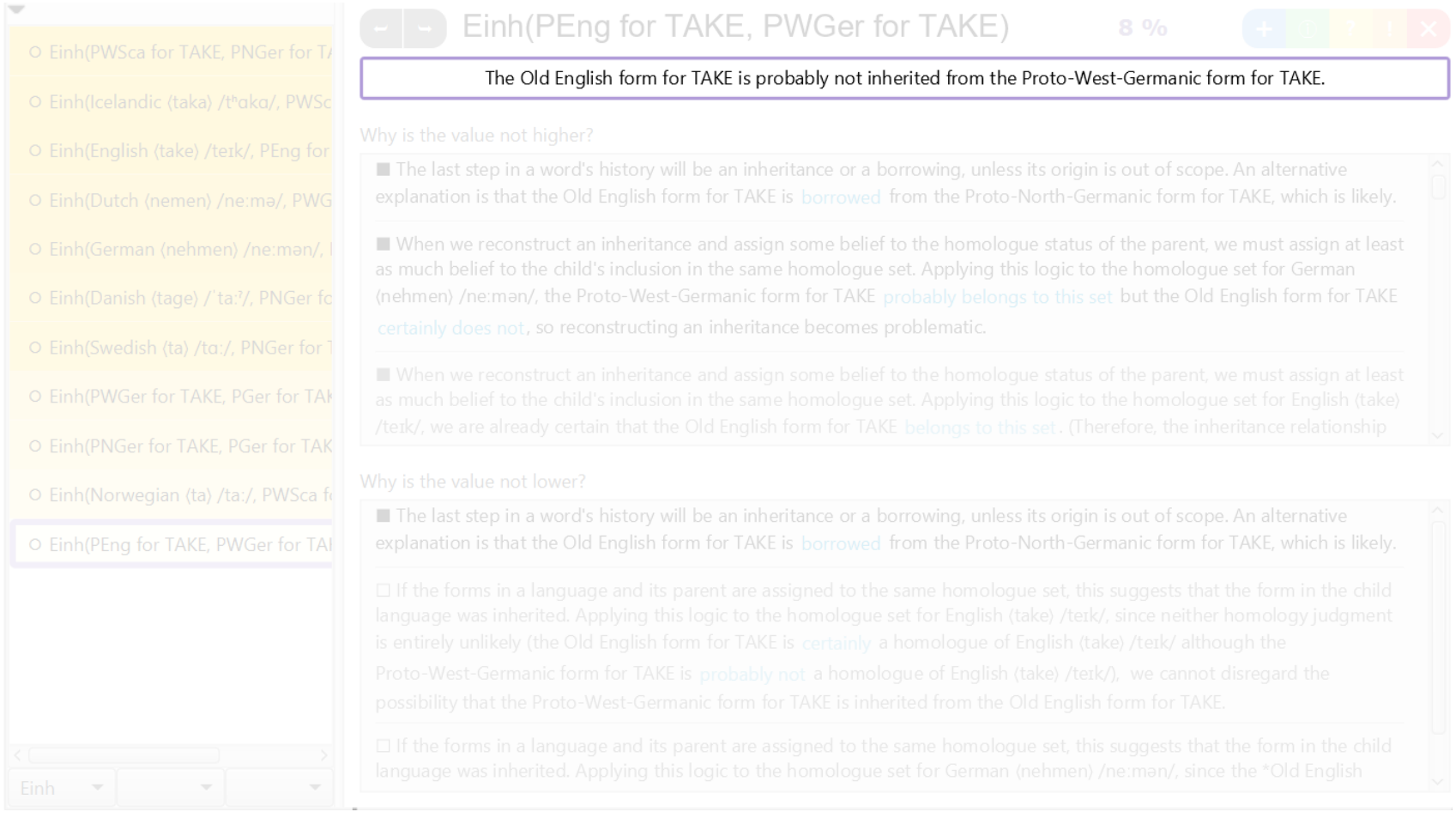
Rule activity

- A ground rule is **active** with respect to a ground atom if
  - it is dissatisfied (changing the atom's value in the direction of the rule pressure would alleviate the dissatisfaction)
  - it *would* be dissatisfied if the atom's value were changed slightly (against the rule pressure) – counterfactual test
- Typically, most ground rules will be inactive for any given atom
- Only active rules actually contribute to the MAP estimate and therefore its explanation
- If many groundings of some rule turn out to be inactive,
  - its reasoning pattern is already covered by (a combination of) other rules, which might mean the rule is not needed, or
  - the rule needs to be adjusted in order to increase its influence on the results

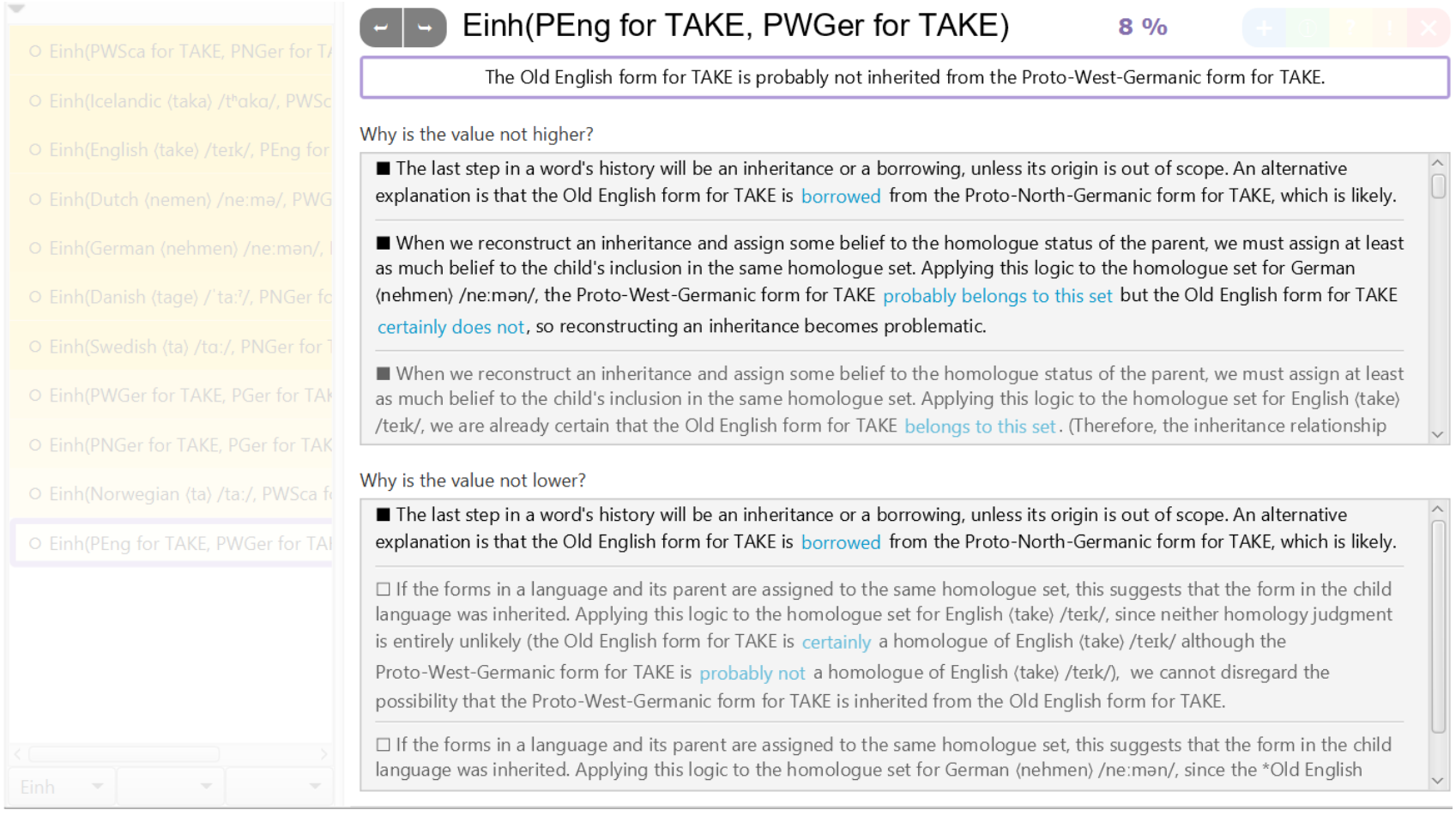
Atom & rule verbalization



Atom verbalization



Rule verbalization



Rule verbalization

- Expressing the mechanics of each ground rule in terms of domain-specific natural language
- Templates for various scenarios
  - Which position within the rule does the inspected atom have?
  - Upwards/downwards pressure

Rule verbalization

```
Inherited(X,Z) & Inherited(Y,Z) & (X != Y)
    -> Similar(X,Y)

Inherited("German ‘nehmen’", "P-West-Gmc ‘…’")   0.9 }
& Inherited("Old Eng ‘…’", "P-West-Gmc ‘…’")     0.1 } 0.0
& ("German ‘nehmen’" != "Old Eng ‘…’")           1.0 }
-> Similar("German ‘nehmen’", "Old Eng ‘…’")              0.4
```

Rule verbalization

Expressing the mechanics of each ground rule in terms of domain-specific natural language

```
Inherited(X,Z) & Inherited(Y,Z) & (X != Y)
    -> Similar(X,Y)
```

- Introductory sentence summarizing the reasoning pattern expressed by the rule
  "If two words are inherited from the same source word, they should be phonetically similar."

Rule verbalization

```
Inherited("German ‘nehmen’", "P-West-Gmc ‘…’")   0.9
& Inherited("Old Eng ‘…’", "P-West-Gmc ‘…’")     0.1
& ("German ‘nehmen’" != "Old Eng ‘…’")           1.0
-> Similar("German ‘nehmen’", "Old Eng ‘…’")     0.4
```

- Introductory sentence
  “If two words are inherited from the same source word, they should be phonetically similar.”
- Then fill in specifics of grounding & inference
  “However, since it is very likely that German ‘nehmen’ was inherited from Proto-West-Germanic, but it is very unlikely that the Old English word was as well, the similarity score is actually not constrained.”

Rule verbalization

```
Inherited("German ‘nehmen’", "P-West-Gmc ‘…’")   0.9
& Inherited("Old Eng ‘…’", "P-West-Gmc ‘…’")     0.1
& ("German ‘nehmen’" != "Old Eng ‘…’")           1.0
-> Similar("German ‘nehmen’", "Old Eng ‘…’")     0.4
```

- Introductory sentence
  “If two words are inherited from the same source word, they should be phonetically similar.”
- Then fill in specifics of grounding & inference
  “Since it is very likely that German ‘nehmen’ was inherited from Proto-West-Germanic, but the German and Old English words are moderately dissimilar, it should be at least somewhat unlikely that the Old English word comes from the same source.”

Code can be used for any PSL model:

- github.com/jdellert/psl-infrastructure
  - Java API for defining/running PSL inferences
  - Rule-atom graph analysis tools (rule pressure, rule activity, verbalizations)
  - (More PSL tools released and to come: inference parallelization, model evaluation, ...)
- github.com/verenablaschke/psl-ragviewer
  - GUI

Thank you!
Questions?

github.com/jdellert/psl-infrastructure
github.com/verenablaschke/psl-ragviewer

verena.blaschke@cis.lmu.de
johannes.dellert@uni-tuebingen.de

European Research Council
Established by the European Commission

This work is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (CrossLingference, grant agreement no. 834050).

Appendix

Etymological inference: predicates

Closed predicates:

`InheritancePoss(X,Y)`	lang of Y is parent of lang of X
`BorrowingPoss(X,Y)`	lang of Y influenced lang of X

Open predicates used for reasoning:

`Homologue(X,H)`	belief that word X belongs to homologue set H
`Similar(X,Y)`	phonetic similarity of the words X and Y (already known for pairs of modern words)

Open predicates for encoding etymologies (= the inference result):

`Inherited(X,Y)`	belief that X was inherited Y
`Borrowed(X,Y)`	belief that X was borrowed from Y
`UnknownOrigin(X)`	belief that etymology of X is outside scope

Etymological Inference: constraints and priors

Homologue set assignment for each form is a distribution:

```
Homologue(X,+H) = 1.
```

Form similarity is symmetric and (roughly) transitive:

```
Similar(X,Y) = Similar(Y,X) .
```

```
(X != Y) & (X != Z) & (Y != Z) &
   Similar(X,Y) & Similar(Y,Z) -> Similar(X,Z) .
```

Weak negative prior on borrowing (= prefer inheritance by default):

```
0.5: !Borrowed(X,Y)
```

Strong negative prior on unknown (= try hard to find etymology):

```
2.5: !UnknownOrigin(X)
```

Etymological Inference: rules

Belief assigned to etymologies for each form is a distribution:

```
Inherited(X,+Y) + Borrowed(X,+Z) + UnknownOrigin(X) = 1 .
```

Borrowing is monodirectional:

```
Borrowed(X,Y) + Borrowed(Y,X) <= 1 .
```

Etymological links are only possible between homologues:

```
Inherited(X,Y) & Homologue(Y,H) -> Homologue(X,H).
```

```
Borrowed(X,Y) & Homologue(Y,H) -> Homologue(X,H).
```

Etymological inference: rules

Strong preference for two words which are inherited from the same word to be phonetically similar:

```
2.0: Inherited(X,Z) & Inherited(Y,Z) & (X != Y)
    -> Similar(X,Y)
```

All pairs of inherited words are at least as similar as their parents (preventing forms from becoming more similar with time):

```
1.0: Similar(X,Y) & Inherited(X,W) & Inherited(Y,Z)
    & (W != Z) -> Similar(W,Z)
```

A loan should be more similar to its source than to any other word:

```
1.0: Borrowed(X,Y) & Similar(X,Z) & (Y != Z) & (X != Z)
    -> Similar(X,Y)
```

Etymological inference: rules

Evidence of homologue set presence is propagated along parent-child links, with the child-to-parent direction being dominant:

```
0.6: Homologue(X,H) & InheritancePoss(X,Z)
    -> Homologue(Z,H)

0.2: Homologue(Z,H) & InheritancePoss(X,Z)
    -> Homologue(X,H)
```

Etymological inference: rules

If parent and child are homologues, that suggests inheritance:

```
0.4: Homologue(X,H) & Homologue(Y,H)
    & InheritancePoss(X,Y) -> Inherited(X,Y)
```

By contrast, if there is any reason to doubt the presence of a homologue set in the parent, an available loanword etymology becomes much more likely:

```
1.0: Homologue(X,H) & !Homologue(Y,H) &
    InheritancePoss(X,Y) & BorrowingPoss(X,Z)
    -> Borrowed(X,Z)
```