Rule verbalization notes for the `de.tuebingen.sfs.psl.talk` package

Verena Blaschke

August 14, 2022

This document describes the algorithm for generic PSL rule verbalizations as currently implemented in our PSL infrastructure. It also serves as a supplement to the forthcoming paper "Navigable atom-rule interactions in PSL models enhanced by rule verbalizations, with an application to etymological inference"[1]

When creating custom PSL problems, overriding the default rule summaries and atom verbalizations by implementing subclasses of `TalkingRuleOrConstraint` and `TalkingPredicate` is recommended (see p. ii).

Developers can decide to override some or all of the rule verbalization types (if-cases in the *verbalize* function on pp. iii–iv). We find that overriding these defaults while still modelling the structure of the custom verbalizations after the default cases yields the best results. The implementation of the `LivesToKnowsRule`[2] from the example in our paper showcases what this can look like. The verbalization of the `KnowsSymmetryConstraint`,[3] by constrast, is structured in a much more idiosyncratic way as this fits the symmetry of the rule (`Knows(P1,P2) = Knows(P2,P1).`) best. More complex examples of rule verbalizations can be found in the talking rule classes of the etinen-etymology package.[4]

[1] Verena Blaschke, Thora Daneyko, Jekaterina Kaparina, Zhuge Gao, & Johannes Dellert. Navigable atom-rule interactions in PSL models enhanced by rule verbalizations, with an application to etymological inference. To appear in the *Proceedings of the 31st International Conference on Inductive Logic Programming (ILP 2022)*.

[2] `https://github.com/jdellert/psl-infrastructure/blob/master/src/main/java/de/tuebingen/sfs/psl/examples/livesknows/LivesToKnowsRule.java`

[3] `https://github.com/jdellert/psl-infrastructure/blob/master/src/main/java/de/tuebingen/sfs/psl/examples/livesknows/KnowsSymmetryConstraint.java`

[4] `https://github.com/verenablaschke/etinen-etymology/tree/master/src/main/java/de/tuebingen/sfs/eie/components/etymology/talk/rule`

Given: A rule $r = (A, P)$ containing a number of atoms, the first b of which are in the body of the rule and the last h of which are in the head of the rule: $A = (a_0, a_1, ..., a_b, a_{b+1}, ..., a_{b+h})$, as well as the corresponding polarities $P = (p_0, ..., p_{b+h}) \in \{+1, -1\}^{b+h}$ indicating whether each atom is negated (-1) or not (+1). If the rule is an arithmetic rule, $h = 0$.
Given: A context atom $c \in A$
Given: A boolean *whyNotLower* indicating whether we want to explain why the atom's value isn't lower (vs. why it isn't higher)
Given: A map of atoms to their inferred values: $score : A \mapsto [0; 1]$

def *summary(rule)*:
  return (default) "*rule*"
  return (custom, e.g.) "If there is evidence that two people live at the same address, this makes it more likely that they know each other."

def *sentence(atom, score(atom))*:
  return (default) "*atom* is score(*atom*)"
  return (custom, e.g.) "Alice and Bob probably know each other"

def *nounPhrase(atom)*:
  return (default) "*atom*"
  return (custom, e.g.) "the acquaintance between Alice and Bob"

def *nounPhraseScore(atom, score(atom))*:
  return (default) "*atom* (score(*atom*))"
  return (custom, e.g.) "the likely acquaintance between Alice and Bob"

```
def verbalize(r, c, whyNotLower, score):
    v ← summary(r)
    if (score(c) = 0 ∧ whyNotLower) ∨ (score(c) = 1 ∧ ¬whyNotLower)
      ∨ h = 0 then
        /* We're either dealing with an arithmetic rule or a
           case where a complicated explanation isn't needed
           since it is trivially true that the context atom's
           value cannot be any lower than 0 / higher than 1.
           */
        if score(c) = 0 ∧ whyNotLower then
            v+ = "This atom has taken the lowest value it can take."
        else if score(c) = 1 ∧ ¬whyNotLower then
            v+ = "This atom has taken the highest value it can take."
        end
        for n ∈ (0, ..., b + h) \ {c} do
            v+ = sentence(a_n) + "and"
        end
        delete last "and"
        return v
    end
    if c > b then
        /* If the context atom is in the rule's head, the
           explanation is straightforward:                    */
        v+ = "Since"
        for n ∈ (0, ..., b) do
            v+ = sentence(a_n) + "and"
        end
        delete last "and"
        if p_c = −1 then
            v+ = "the value of nounPhrase(a_c) has an upper limit."
        else
            v+ = "the value of nounPhrase(a_c) has a lower limit."
        end
        /* In the future, this might be extended to provide
           more nuance when it comes to rules with multi-atom
           heads.  In the current state of the Java
           implementation, we assume that each rule head only
           contains a single atom.                            */
        return v
    end
```

```
    for n ∈ (0, ..., b) do
        v+ = nounPhraseScore(a_n) + "and"
    end
    delete last "and"
    v+ = "determine"
    posHeadIndices ← { n ∈ (b + 1, ..., b + h) | p_n = +1 }
    negHeadIndices ← { n ∈ (b + 1, ..., b + h) | p_n = −1 }
    if |posHeadIndices| > 0 then
        v+ = "minimum values for"
        for n ∈ posHeadIndices do
            v+ = nounPhrase(a_n) + "and"
        end
    end
    if |negHeadIndices| > 0 then
        v+ = "maximum values for"
        for n ∈ negHeadIndices do
            v+ = nounPhrase(a_n) + "and"
        end
    end
    delete last "and"
    if the Lukasiewicz t-norm of the head is 1 then
        /* The rule is trivially satisfied:                    */
        v+ = "However, since they already have values of 100%/0%,
         changing the value of any of the other atoms would not result in a
         rule violation."
        return v;
    end
    v+ = "Since"
    for n ∈ posHeadIndices do
        v+ = nounPhrase(a_n) + "is only" + score(a_n) + "and"
    end
    for n ∈ negHeadIndices do
        v+ = nounPhrase(a_n) + "is already" + score(a_n) + "and"
    end
    delete last "and"
    v+ = "the value nounPhrase(a_n) can take is limited."
    return v
end
```