Discussion: Defaults/defeasible constraints in the LKB

DELPH-IN 2025

Spencer Brooks

July 8, 2025

Outline

1 Problem Overview

2 Questions

3 Discussion

Goals for Today

Today, I'm hoping to:

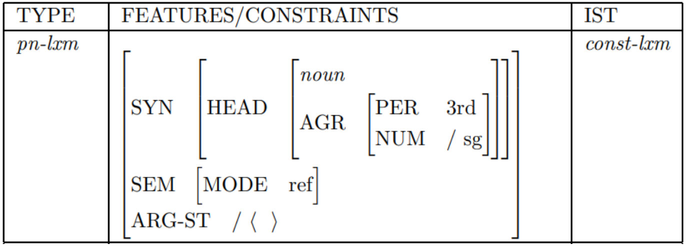
- develop my understanding of the problem and its known challenges (in theory and application)
- hear thoughts on some specific questions
- facilitate general thoughts and discussion

In addition, I'm eager to hear:

- about readings, grammars, and other resources you would suggest I examine
- other guidance/thoughts

Problem Overview

- Sag, Wasow, and Bender (2003) [eng] use **defeasible** constraints, aka **defaults**, in lexical rule and lexeme type definitions
- Example from a lexeme type definition:

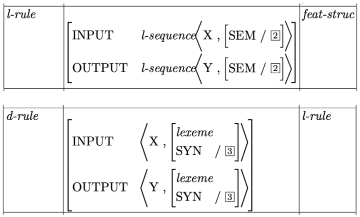


TYPE	FEATURES/CONSTRAINTS	IST
pn-lxm	$\begin{bmatrix} \text{SYN} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \textit{noun} & \\ \text{AGR} & \begin{bmatrix} \text{PER} & \text{3rd} \\ \text{NUM} & /\ \text{sg} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{SEM} & \begin{bmatrix} \text{MODE} & \text{ref} \end{bmatrix} \\ \text{ARG-ST} & /\ \langle\ \rangle \end{bmatrix}$	*const-lxm*

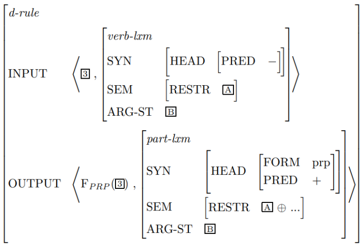
- Overridden by specific lexical types: The Seattle Reign

Problem Overview

- Where it gets interesting: **defeasible identity**
- (Motivating) example from lexical rule type definition:


l-rule
INPUT l-sequence⟨X , [SEM / 2]⟩
OUTPUT l-sequence⟨Y , [SEM / 2]⟩
feat-struc
d-rule
INPUT ⟨X , [lexeme SYN / 3]⟩
OUTPUT ⟨Y , [lexeme SYN / 3]⟩
l-rule

Present Participle Lexical Rule


d-rule
INPUT ⟨3 , [verb-lxm SYN [HEAD [PRED −]] SEM [RESTR A] ARG-ST B]⟩
OUTPUT ⟨F_PRP(3) , [part-lxm SYN [HEAD [FORM prp PRED +]] SEM [RESTR A ⊕ ...] ARG-ST B]⟩

- Emerson, 2021: ‘Would be nice: The ability to say “identify everything other than that which is specified as different between mother and daughter”’

Problem Overview

Present Participle Lexical Rule

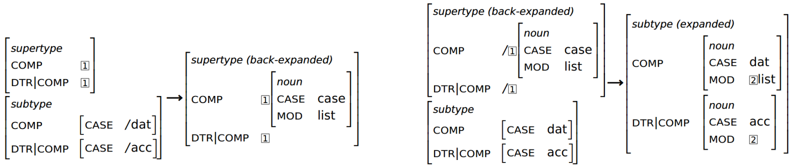
$$\begin{bmatrix} \textit{d-rule} & \\ \text{INPUT} & \left\langle \boxed{3}\,,\ \begin{bmatrix} \textit{verb-lxm} & \\ \text{SYN} & \begin{bmatrix}\text{HEAD} & \begin{bmatrix}\text{PRED} & -\end{bmatrix}\end{bmatrix} \\ \text{SEM} & \begin{bmatrix}\text{RESTR} & \boxed{A}\end{bmatrix} \\ \text{ARG-ST} & \boxed{B} \end{bmatrix} \right\rangle \\ \text{OUTPUT} & \left\langle \text{F}_{PRP}(\boxed{3})\,,\ \begin{bmatrix} \textit{part-lxm} & \\ \text{SYN} & \begin{bmatrix}\text{HEAD} & \begin{bmatrix}\text{FORM} & \text{prp} \\ \text{PRED} & + \end{bmatrix}\end{bmatrix} \\ \text{SEM} & \begin{bmatrix}\text{RESTR} & \boxed{A} \oplus \ldots\end{bmatrix} \\ \text{ARG-ST} & \boxed{B} \end{bmatrix} \right\rangle \end{bmatrix}$$

- Example: copy up the value of FORM and all other head features and constraint on the head type; copy up predications
- Because of the inherited defeasible identity of SYN, we wouldn't have to explicitly copy: FORM, INF, AUX, POL, INV, AGR, ...

Problem Overview

- Current LKB reality: when you override a defeasible constraint, info is thrown away instead of pushed down
- (Is this accurate?)
- Instead, Emerson 2021: Do YADU (a default unification operation from Copestake and Lascarides 1996) forwards and backwards, which (somehow) both expands feature paths and assigns the correct identities:

From Emerson 2021:



Questions

- How does Guy's proposal expand types?
- Why (intuitively) should Guy's proposal get us the desired unification result?
- Why are there ???'s for a COMPS...HEAD value in Guy's 2021 example?
- Can we do anything about the potential for YADU meltdown?
- Can we sidestep problems by providing detailed error messages?
- Are incompatibilities with append lists expected?
- Are incompatibilities with min-types expected?
- Are persistent defaults in-scope?

How does Guy's proposal expand types?

- My understanding is a key reason to do backward YADU is to get access to (here) CASE and MOD for the forward step. How do you get these?
- Emerson, 2021: The "pushed down" identity combines something more specific and something less specific:
  1. Existence of new feature path: more specific
  2. Identity constraint on new path: less specific
- Is this a key question or in the weeds?

How does Guy's proposal expand types?

From Emerson 2021:

$$\begin{array}{l}\begin{bmatrix}\textit{supertype} & \\ \text{COMP} & \boxed{1} \\ \text{DTR}|\text{COMP} & \boxed{1}\end{bmatrix} \\ \begin{bmatrix}\textit{subtype} & \\ \text{COMP} & \begin{bmatrix}\text{CASE} & /\text{dat}\end{bmatrix} \\ \text{DTR}|\text{COMP} & \begin{bmatrix}\text{CASE} & /\text{acc}\end{bmatrix}\end{bmatrix}\end{array} \longrightarrow \begin{bmatrix}\textit{supertype (back-expanded)} & \\ \text{COMP} & \boxed{1}\begin{bmatrix}\textit{noun} & \\ \text{CASE} & \text{case} \\ \text{MOD} & \text{list}\end{bmatrix} \\ \text{DTR}|\text{COMP} & \boxed{1}\end{bmatrix}$$

$$\begin{array}{l}\begin{bmatrix}\textit{supertype (back-expanded)} & \\ \text{COMP} & /\boxed{1}\begin{bmatrix}\textit{noun} & \\ \text{CASE} & \text{case} \\ \text{MOD} & \text{list}\end{bmatrix} \\ \text{DTR}|\text{COMP} & /\boxed{1}\end{bmatrix} \\ \begin{bmatrix}\textit{subtype} & \\ \text{COMP} & \begin{bmatrix}\text{CASE} & \text{dat}\end{bmatrix} \\ \text{DTR}|\text{COMP} & \begin{bmatrix}\text{CASE} & \text{acc}\end{bmatrix}\end{bmatrix}\end{array} \longrightarrow \begin{bmatrix}\textit{subtype (expanded)} & \\ \text{COMP} & \begin{bmatrix}\textit{noun} & \\ \text{CASE} & \text{dat} \\ \text{MOD} & \boxed{2}\text{list}\end{bmatrix} \\ \text{DTR}|\text{COMP} & \begin{bmatrix}\textit{noun} & \\ \text{CASE} & \text{acc} \\ \text{MOD} & \boxed{2}\end{bmatrix}\end{bmatrix}$$

Why (intuitively) should Guy's proposal get us the desired unification result?

- In particular, what's the intuition behind switching between default and nondefault constraints?

From Emerson 2021:

$$
\begin{array}{l}
\begin{bmatrix} \textit{supertype} & \\ \text{COMP} & \boxed{1} \\ \text{DTR}|\text{COMP} & \boxed{1} \end{bmatrix} \\
\begin{bmatrix} \textit{subtype} & \\ \text{COMP} & \begin{bmatrix} \text{CASE} & /\text{dat} \end{bmatrix} \\ \text{DTR}|\text{COMP} & \begin{bmatrix} \text{CASE} & /\text{acc} \end{bmatrix} \end{bmatrix}
\end{array}
\longrightarrow
\begin{bmatrix} \textit{supertype (back-expanded)} & \\ \text{COMP} & \boxed{1}\begin{bmatrix} \textit{noun} & \\ \text{CASE} & \text{case} \\ \text{MOD} & \text{list} \end{bmatrix} \\ \text{DTR}|\text{COMP} & \boxed{1} \end{bmatrix}
$$

$$
\begin{array}{l}
\begin{bmatrix} \textit{supertype (back-expanded)} & \\ \text{COMP} & /\boxed{1}\begin{bmatrix} \textit{noun} & \\ \text{CASE} & \text{case} \\ \text{MOD} & \text{list} \end{bmatrix} \\ \text{DTR}|\text{COMP} & /\boxed{1} \end{bmatrix} \\
\begin{bmatrix} \textit{subtype} & \\ \text{COMP} & \begin{bmatrix} \text{CASE} & \text{dat} \end{bmatrix} \\ \text{DTR}|\text{COMP} & \begin{bmatrix} \text{CASE} & \text{acc} \end{bmatrix} \end{bmatrix}
\end{array}
\longrightarrow
\begin{bmatrix} \textit{subtype (expanded)} & \\ \text{COMP} & \begin{bmatrix} \textit{noun} & \\ \text{CASE} & \text{dat} \\ \text{MOD} & \boxed{2}\text{list} \end{bmatrix} \\ \text{DTR}|\text{COMP} & \begin{bmatrix} \textit{noun} & \\ \text{CASE} & \text{acc} \\ \text{MOD} & \boxed{2} \end{bmatrix} \end{bmatrix}
$$

Why are there ???'s for a COMPS...HEAD value in Guy's 2021 example?

- Emerson, 2021: The HEAD value gets lost in this example, because CASE is part of head; can't identify HEAD while changing CASE
- Would we want to identify the HEAD type here? I'm not sure what the bottom line here is.

```
defeasible-identity-lex-rule := lex-rule &
  [ SYNSEM.LOCAL.CAT /#cat,
    ARG-ST /#arg-st,
    C-CONT.HOOK /#hook,
    DTR [ LOCAL [ CAT /#cat,
                  CONT.HOOK /#hook ],
          ARG-ST /#arg-st ]].

acc-to-dat-obj-lex-rule := defeasible-identity-lex-rule &
 [ SYNSEM.LOCAL.CAT.COMPS.FIRST.LOCAL.CAT.HEAD.CASE dat,
   DTR.SYNSEM.LOCAL.CAT.COMPS.FIRST.LOCAL.CAT.HEAD.CASE acc ].
```

Why are there ???'s for a COMPS...HEAD value in Guy's 2021 example?

```
acc-to-dat-obj-lex-rule := defeasible-identity-lex-rule &
  [ SYNSEM.LOCAL.CAT [ HEAD /#head,
                       VAL [ SPR #/spr,
                             SUBJ #/subj,
                             SPEC #/spec,
                             COMPS [ REST /#rest,
                                     FIRST [ LOCAL [ CAT [ VAL /#val,
                                                           HEAD ??? &
                                                           [ CASE dat
                                                             MOD /#mod ]],
                                                     CONT /#cont ]]]],
    DTR [ LOCAL [ CAT [ HEAD /#head,
                        VAL [ SPR #/spr,
                              SUBJ #/subj,
                              SPEC #/spec,
                              COMPS [ REST /#rest,
                                     FIRST [ LOCAL [ CAT [ VAL /#val,
                                                           HEAD [ CASE acc,
                                                                  MOD /#mod ]],
                                                     CONT /#cont ]]]]]]].
```

Can we do anything about the potential for YADU meltdown?

- Ann: There's a question of where a re-entrancy is overriden at the top level... There was an idea that you could set up lexical rules where F is coindexed with G, even if there is some overriding of that coindexation (e.g. F.H   G.H, but keep F.W = G.W). But there are cases where this can't work. When you start relying on that stuff, the YADU algorithm goes into complete meltdown. So the danger of using defaults in the grammar is that it can grind to a complete halt. One of those cases is complicated re-entrancies.
- Francis Bond: if we're only using defaults before compiling the grammar, can we check if it's okay?

Can we do anything about the potential for YADU meltdown?

- Ann: It won't melt-down at run-time. But YADU will just go away and not come back. Can't give you a static checker. Just to quickly say what's going on in YADU, in the case of a Nixon diamond, you generate the possibilities - it's okay if there's 2, but not if there's thousands. Mostly it's okay, but it can sometimes blow up.
- Dan: If the feedback to the developer was relatively quick (even just a blue screen of death), it could help increase the clarity of our grammars, and that could be incremental. So fixing the easy things would be a major boost. For the more complicated recursive parts, there's less benefit.
- ...
- Ann: I don't think discovering this could be speeded up. Put something in and check it, then go onto the next thing.

Can we sidestep problems by providing detailed error messages?

- In the context of discussion about min-types and persistent defaults:
- MWG: This is all convenience to the grammar writer and not increasing the power of the grammar. Seems like we're getting caught up on edge cases. What if we were able to detect edge cases and warn the grammar engineer about things that aren't supported. The difficult things are also difficult for the grammar engineer to reason about and that defeats the purpose.

Can we sidestep problems by providing detailed error messages?

- Guy: I guess you could compile the grammar and then check back through the feature structure to where the default identity constraint was introduced and then see whether any of the types along the way have further subtypes – and then say these are things that might potentially go wrong.
- EMB: I think the edge cases are ones where the grammarian hasn't provided enough info to get what they want. So hard to check for at the level of sorry we won't compile that grammar, but maybe helpful as Guy says to flag where there might be more to think about.

Are incompatibilities with min-types expected?

- Woodley: The min types might be an unfortunate interaction with the default unification idea. Default constraint says make the whole structure identified, and if you don't do things right, what you'll end up with is expansion only to the min type that lets you write the change, but then lose the upper-level identity and when the min type gets expanded to the full type, lose the other features. In Guy's example, suppose that noun is a type with more features than CASE and MOD, but you've only got noun-min.
- ...
- Dan: ...a consequence of the forwards-backwards approach is that you need a fully realized feature structure by the time you're done. All features need to be expanded is a scary conception to me, since it means I [can't] use the min features to reduce the size of the things being manipulated at parse time... If you did this to apply defeasibility, could you go back up to min-types before parse time...?

Are interactions with append lists expected?

- Emily: So this might not be simpler in the end: less typing, more debugging. If we are telling the grammarian “it’s all taken care of” we have to be very clear about what isn’t.
- Guy: True, but there is also a lot of reasoning about the thicket of types.
- Olga (in chat): Similar considerations for the append lists.

Are persistent defaults in-scope?

- (I don't think so, but if so, what are persistent defaults exactly?)
- Guy: From what I understand, YADU has two settings for each defeasible constraint, to say whether it's persistent or not. All are kept as defaults within the type hierarchy, but at the instances, you say whether it remains default or becomes strict.
- ...
- EMB: Also as a grammarian I can't reason about persistent defaults.

Thank you!