Defeasible Identity Constraints in the DELPH-IN Joint Reference Formalism

Emily M. Bender
Guy Emerson

DELPH-IN Summit 2021

Desiderata

- Most valuable use of defeasible identity constraints in SWB 2003: Lexical rules

l-rule		*feat-struc*
	$\begin{bmatrix} \text{INPUT} & \textit{l-sequence}\left\langle \text{X} , \begin{bmatrix} \text{SEM} \ / \ \boxed{2} \end{bmatrix} \right\rangle \\ \text{OUTPUT} & \textit{l-sequence}\left\langle \text{Y} , \begin{bmatrix} \text{SEM} \ / \ \boxed{2} \end{bmatrix} \right\rangle \end{bmatrix}$	

d-rule		*l-rule*
	$\begin{bmatrix} \text{INPUT} & \left\langle \text{X} , \begin{bmatrix} \textit{lexeme} \\ \text{SYN} \ / \ \boxed{3} \end{bmatrix} \right\rangle \\ \text{OUTPUT} & \left\langle \text{Y} , \begin{bmatrix} \textit{lexeme} \\ \text{SYN} \ / \ \boxed{3} \end{bmatrix} \right\rangle \end{bmatrix}$	

- Identity gets “pushed down” if overridden in part.

Desiderata

- The Grammar Matrix provides supertypes for lexical rules that get specialized in customization, including phenomena not easily modeled with monotonic addition of constraints

- Current solution: a thicket of rules for copying up different substructures and then subtypes that inherit different constellations of those

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

Example: Non-defeasible supertype

```
lex-rule := phrase-or-lexrule & word-or-lexrule &
  [ NEEDS-AFFIX bool,
    SYNSEM.LOCAL [ CAT.WH #wh,
                   CONT [ RELS.APPEND < #r1, #r2 >,
        HCONS.APPEND < #h1, #h2 >,
        ICONS.APPEND < #i1, #i2 > ] ],
    DTR #dtr & word-or-lexrule &
  [ SYNSEM.LOCAL [ CAT.WH #wh,
                   CONT [ RELS #r1,
              HCONS #h1,
              ICONS #i1 ] ],
    ALTS #alts ],
C-CONT [ RELS #r2,
    HCONS #h2,
    ICONS #i2 ],
ALTS #alts,
ARGS < #dtr > ].
```

Example: Type with defeasible constraints & overriding type

```
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 ].
```

Example: Expanded version of subtype, with inherited defeasible identities ‘pushed down’

```
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 ]]]]]]].
```

NB: Hugely abbreviated!

Notes from the distant past (Paris, 2010)

- Probably best not to put the defeasible identity constraints directly on the rule, but in a separate file that instructs the processor to do something special at compile time.

- Interacts with type well-formedness: Can only “push down” defeasible identity constraints to features that are necessitated by types invoked in the rule definitions.
    - What about types invoked by subtypes of these rule definitions?
    - Impossible: types invoked only by structure unified in as daughter

- The HEAD value gets lost in this example, because CASE is part of *head*; can’t identify HEAD while changing CASE