`gengram` Documentation

Per Anker Jensen

Implementing ontological type coercion in an untyped unification formalism (PATR)
The case of the pre-nominal genitive in English

`Technical report`
`2010`

1[1]. `gengram`

`gengram` is a sequence of grammars each consisting of two files, a syntax-semantics file and a lexicon file. The grammars are numbered consecutively as `gengram01, gengram02`, etc., in a way such that `gengram01` consists of the files `gensyn01.pl` and `genlex01.pl`, etc.

2. `gengram01`

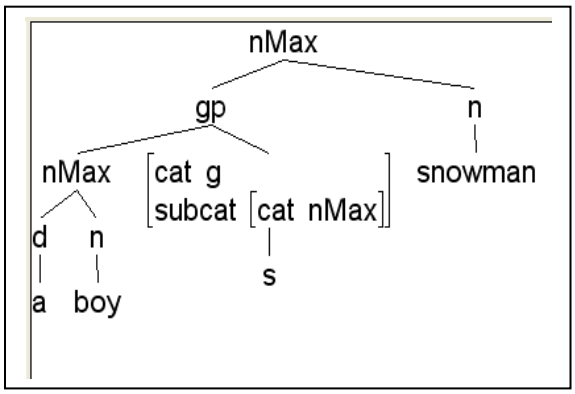
The program `gengram01` is a PATR-implementation of a formal theory of the semantics of prenominal genitive constructions proposed by Jensen &Vikner 2004. The theory is formulated within the general framework of James Pustejovsky's theory of the Generative Lexicon (GL) and addresses the problem of type coercion in relation to genitive constructions.

2.1 The syntax of genitive constructions in the implementation

The implementation of the syntax of genitive constructions models the basic features of Vikner & Jensen (2002) and Jensen &Vikner 2004. That is, for an example like (1) we have the syntactic structure in (2), where GP (covering the string *a boy's*) functions as a specifier of the topmost nMax node, ie of the full genitive construction, and the lower nMax functions as a complement subcategorised for by *s*:

(1) A boy's snowman

(2)



2.2 The semantics of nMax in the implementation

The semantics of the syntactic category nMax is modelled as a generalised quantifier using restricted quantification. Following Jensen & Vikner (1996, chapter 19), the semantic structure of an nMax in this implementation contains a quantifier with three elements in its scope: a variable, a restriction and an assertion. The overall structure of the semantic representation of an nMax is thus as shown in (3):

(3)[2] Quantifier(variable, restriction, assertion)

where *restriction* and *assertion* are well-formed formulas. This means that in an nMax consisting of two daughters d and n, d will contribute the quantifier Q, and the restriction will come from n. The

[1] I am indebted to Carl Vikner for fruitful comments and criticism of two earlier versions of this report.

[2] In the implementation we use the following abbreviations for the elements of the quantifier structure: quant (var, restr, assn).

assertion is typically provided by the semantics of the vMax with which nMax combines syntactically to form a sentence, or, in the case of genitive constructions, the assertion will hold the genitive relation as will be illustrated below.

The lexical entries for articles will take the form shown in (4):

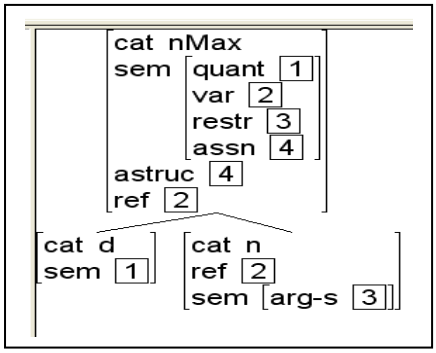
(4)
```
lex(a, L) :-
    L>>cat === d,
    L>>sem === exists.
```

Sortal nouns like *boy* are described as shown in (5):

(5)
```
lex(boy,L) :-
    L>>cat === n,
    L>>sem..'arg-s'..pred === 'boy''',
    L>>sem..'arg-s'..arg1 === L>>ref.
```

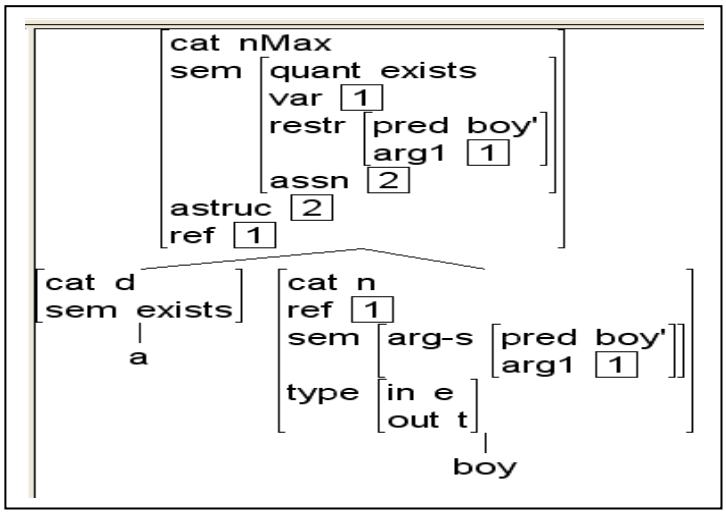
And an nMax rule like (6):

(6)
```
M ---> [D1,D2] :-       % nMax ->D N
    M>>cat === nMax,
    D1>>cat === d,
    D2>>cat === n,

    M>>sem..quant === D1>>sem,
    M>>sem..var === D2>>ref,
    M>>sem..restr === D2>>sem..'arg-s',
    M>>sem..assn === M>>astruc,
    M>>ref === M>>sem..var.
```



will combine the article and the sortal noun giving the feature structure tree shown in (7) for the nMax *a boy*:

(7)



2.3 The semantics of genitive constructions in the implementation

2.3.1 The case of relational head nouns

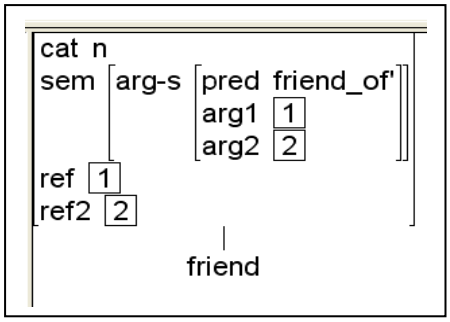
This implementation focuses on the semantic analysis of genitive constructions in which the genitive relation is assumed to originate from the head noun of the genitive construction, henceforth referred to as N_2.

In the simplest cases, N_2 is realised by a relational noun like *member*, *friend*, *daughter*, or *address*. In such cases we assume that the relational meaning of the noun is directly available from the argument structure in the lexical entry of the noun in question, and the meaning of the full genitive construction is composed such that $nMax_1$ (traditionally referred to as the "possessor"[3]) delivers an argument to the relation made available by N_2. In an example like (8):

(8) A boy's friend

the denotation of *a boy* thus acts as a semantic argument of the relation made available by *friend*, ie *A boy's friend* is interpreted as " the friend of a boy". In the implementation, the representation of the argument structure of the relational noun *friend* appears as shown in (9):

(9)



The semantic part of this feature structure corresponds to the λ-expression in (10):

(10) λy[λx[friend_of'(y)(x)]][4]

such that the variable x corresponds to `ref`, and y corresponds to `ref2`, yielding for the sentence in (11):

(11) Ann is Bo's friend

the formula in (12):

(12) friend_of'(Bo')(Ann')

where Bo' instantiates `ref2`, and Ann' instantiates `ref`.

[3] This term is strongly misleading and we do not endorse it.

[4] Actually, it is not possible to determine the sequence of the λx and λy-operators by just looking at (9). Roughly, the feature names *ref* and *ref2* correspond to the lambdas, but in the feature structure representation in (9) they are not ordered.

The semantic representation of the genitive clitic s

Vikner and Jensen (2002: 203) propose that constructional genitive interpretations (as well as pragmatic ones) are made possible by assigning the following lexical semantics to *s*:

(13) $\lambda\mathbf{P}\ [\lambda R\ [\lambda P\ [\mathbf{P}(\lambda u\ [\exists x\ [\forall y\ [R(u)(y) \leftrightarrow y = x]\ \&\ P(x)]])]]]$

Here the variable **P** represents the semantics contributed by $nMax_1$, ie the syntactic complement of *s*. The variable R represents a relation, ie an expression of type $e \rightarrow (e \rightarrow t)$. This means that at this particular variable the genitive formula accepts only a relation as an argument. This argument is to be contributed by N_2, the head of the genitive construction. Finally, the variable P is the contribution of the vMax of the sentence.

For a sentence like (14):

(14) A boy's friend laughed

we want a semantic representation like (15):

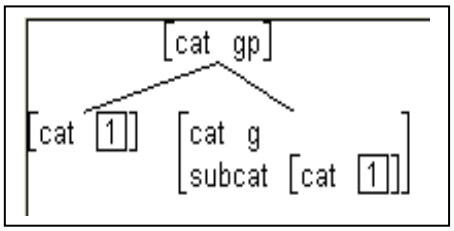
(15) the(x, exists(y, boy'(y), friend_of'(y)(x)), laugh'(x))

This means that the contribution of the full genitive construction in (14) corresponds to (16), where the vMax contribution has been abstracted:

(16) $\lambda P[\text{the}(x, \text{exists}(y, \text{boy}'(y), \text{friend\_of}'(y)(x)), P(x))]_{<<e,t>,t>}$

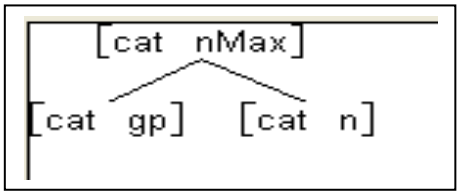
In order to construct the representation for genitive *s*, we should now extract the relevant parts from (16), since, in the syntactic structure we employ here, we have to split the semantic composition of the full genitive construction into two parts, one part corresponding to the local tree in (17):

(17)



and the other part corresponding to the local tree in (18):

(18)



Assuming that there is a uniqueness presupposition associated with *s* (as seen in the formula in (13)), we start by representing this condition by a 'quantifier' `the` representing the joint contributions of ∃ and ∀ in (13). This makes it natural to build a full quantifier structure inside the lexical entry for *s* in accordance with the formula in (13). One effect of doing this is that can assign wide scope to `the` over a quantifier contributed by $nMax_1$[5] if there is one (as is the case with *exists* in the representation in (16)). Following this line of reasoning we propose the lexical entry for *s* in (19):

(19)
```
lex(s, L) :-
    L>>cat === g,
    L>>subcat..cat === nMax,
    L>>sem..quant === the,
    L>>sem..var === L>>ref,
    L>>sem..restr === L>>restruc,
    L>>sem..assn === L>>astruc,
    L>>genRel === L>>relation.
```

What the semantic constraints in
(19) say corresponds to a "formula schema" like (20):

(20) the(x, $nMax_1$-sem, Assn) & genRel

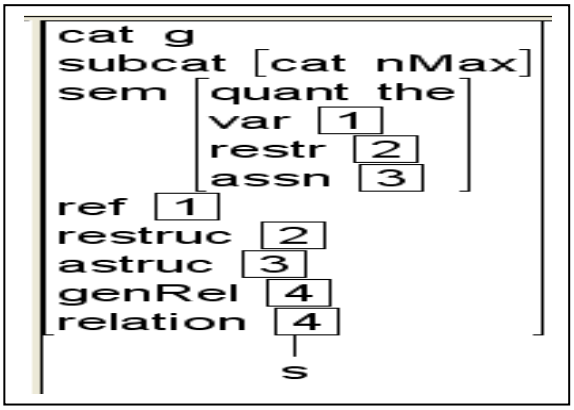
where *$nMax_1$-sem*, *Assn* and *genRel* are well-formed formulas.

The quantifier structure in the lexical entry for *s* in
(19) first introduces the uniqueness marker *the* in the constraint `L>>sem..quant === the`. The constraint L>>`sem..var === L>>ref` introduces the variable bound by *the*, ie the variable representing the referent of the full genitive construction. In the constraint `L>>sem..restr === L>>restruc`, the feature `restruc` is used to fetch the semantics provided by $nMax_1$ into the restriction of the quantifier structure of *s*. The assertion of this quantifier structure is to be instantiated by the semantics of a vMax combining with the full genitive construction, eg the semantics of *laughed* in a sentence like *Ann's friend laughed*. Finally and crucially, we have the constraint corresponding to the R variable of the formula in (13), `L>>genRel === L>>relation`. This is where the genitive relation is represented in the lexical structure of *s*. Strictly speaking, this could be done at the level of the GP without involving *s* at all. But this would be at variance with the formal representation in (13), which is why we have implemented it this way. In the GP-rule, reference will be made to this constraint ensuring that the genitive relation is inserted in the assertion of $nMax_1$; cf. the account of the GP rule below.

The lexical description in (19) corresponds to the feature structure in (21):

[5] In accordance with the scope hierarchy mentioned in Jensen & Vikner (1996:292 and 296).

(21)



The rule `gp → nMax g`

The semantic composition taking place in the GP-rule corresponding to the local syntactic tree in (17) is still not straightforward. In particular, the question of the binding of variables is tricky. We would like to maintain that the nMax *a boy* and the GP *a boy's* do not have the same meaning.[6] The differences are that, on the one hand, *s* introduces uniqueness and, on the other hand, *s* requires a relation, neither of which is the case with *a boy*. As a consequence, we assume that the GP *a boy's* denotes (22):

(22) A function from a set of relations R to a set of individuals that is stand in the relation R to some boy

This has the very important implication that the GP cannot denote the same entity as the nMax it contains! Rather, it presupposes a referent not yet linguistically expressed, namely the referent corresponding to N_2, the head noun of the full genitive construction. However, the referent of $nMax_1$ has to be made available at the GP node since it has to serve as one of the arguments of the genitive relation.

The semantic constraints of the rule `gp → nMax g` can now be formalised as follows:

(23)
```
M ---> [D1,D2] :-   % GP -> XP G[XP]
    M>>cat === gp,
    D2>>cat === g,
    D1>>cat === D2>>subcat..cat,

    M>>sem === D2>>sem,
    M>>genRel === D2>>genRel, % fetch genitive relation from
                              % n (N2)in the rule nMax -> gp n
```

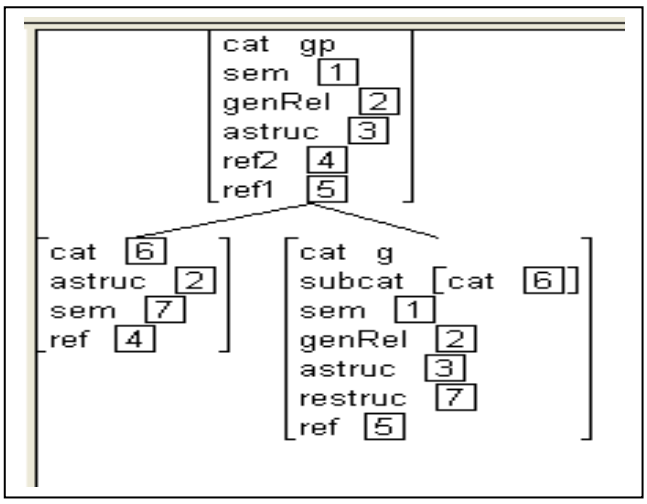
[6] This seems to be the claim proposed by Asher & Denis (2004:174), who combine $nMax_1$ and *s* using an identity function. See their example *a girl's teacher*, line (20.d).

[7] The notation `GP → XP G[XP]` follows the abbreviation conventions for feature structure notations introduced in Gerald Gazdar's paper "Generative Grammar" , in Lyons, John, Richard Coates, Margaret Deuchar & Gerald Gazdar (1987): *New Horizons in Linguistics 2*, Penguin Books, pp. 122-151. The relevant part of Gazdar's convention is that 'X' = {<CAT, X>}, e.g. GP = {<CAT, GP>}, '[XP]' ={<SUBCAT, XP>}, e.g. [NP] ={<SUBCAT, NP>}. Thus, the notation 'G[XP]' abbreviates the full feature structure {<CAT, G>, <SUBCAT, XP>}.

```
D1>>astruc === D2>>genRel,% make the genitive relation the
                          % assertion of nMax1

M>>astruc === D2>>astruc, % fetch vMax semantics via GP (and
                          % nMax2)and insert in assertion
                          % slot of genitive s

D2>>restruc === D1>>sem,  % insert nMax1 semantics in the
                          % restriction slot of genitive s

M>>ref2 === D1>>ref,      % nMax1 delivers the complement-
                          % argument (ref2) to relational
                          % nouns

M>>ref1 === D2>>ref.      % the variable bound by 'the' in
                          % s, ie the referent of n2, the head
                          % of the fullgenitive construction
```

The rule in (23) corresponds to the local tree in (24):

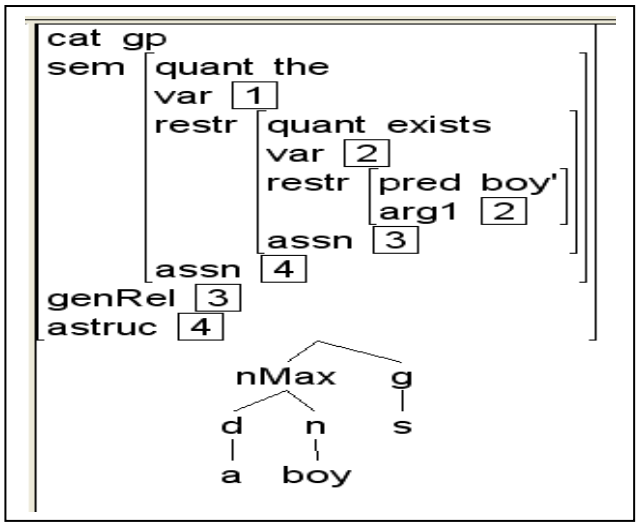
(24)



This gives the structure shown in (25) for the GP *a boy's*, which corresponds nicely to the formula in (16) with the genitive relation (genRel) and the assertion (assn) still not instantiated to the respective formulas, cf. (16'):

(16') the(x, exists(y, boy'(y), genRel), assn)

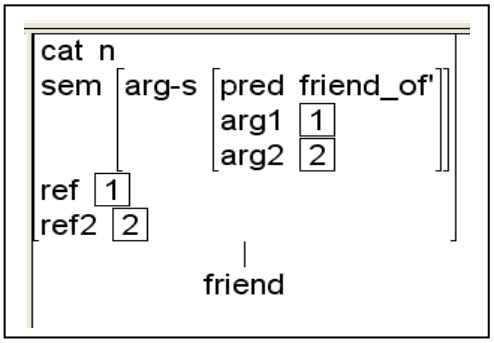
(25)



The rule `nMax → GP N`

This rule is to combine the semantic contributions of GP and N_2. The crucial semantic contribution of N_2 is the genitive relation. That is, in an example like *a boy's friend*, the genitive relation is made available by the relational noun *friend*, or, more precisely, by the information encoded in the argument structure in the lexical entry of *friend*, cf. the value of the `sem..arg-s` path in (9), which we repeat here:

(9)



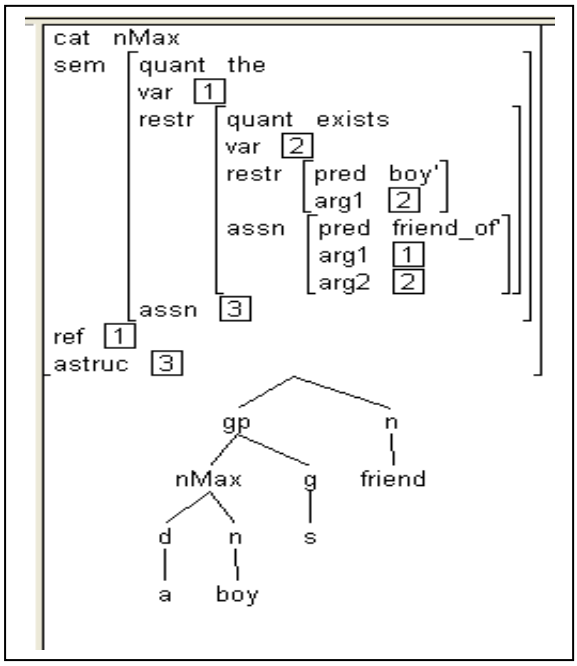
This means that we can formulate the `nMax ->GP N` rule like this:

```
(26) M ---> [D1,D2] :-   % nMax ->GP N
     M>>cat === nMax,
     D1>>cat === gp,
     D2>>cat === n,
     M>>sem === D1>>sem,              % the semantic representation
                                      % of the full genitive
                                      % construction is sent to
                                      % nMax from GP

     D1>>genRel === D2>>sem..'arg-s', % the genitive relation is
                                      % fetched from n and inserted
                                      % into the genRel slot in GP
     D1>>ref === D2>>ref,
     D1>>ref2 === D2>>ref2,
```

```
    M>>ref === D2>>ref,
    M>>astruc === D1>>astruc.
```

At this stage we generate the following representation for (8), *a boy's friend*:

(27)



At no point so far have we made explicit reference to the semantic types of the expressions involved. This means that, in its current form, the grammar will produce a correct syntactic structure for a genitive construction like (1), but the structure will be associated with a meaningless semantic structure. In the following section we address the problem of implementing type checking by unification in a formalism without types.

2.3.2 gengram01.1 Introducing Montagovian types in gengram01

In this section we address the problem of implementing Montagovian types in lexical entries and rules. The ultimate aim is to enforce type requirements by unification and to emulate type coercion by excluding certain parsing possibilities and allowing others based on the type constraints specified in lexical entries and rules.

The semantic types of sortal and relational nouns

Consider the lexical entries for *snowman* and *friend*, a sortal and a relational noun, respectively. Their respective Montagovian types are $e \rightarrow t$ and $e \rightarrow (e \rightarrow t)$. We propose to implement these functions by means of the features `in` and `out` for input type and output type, respectively. Since both input and output types may be complex types, ie functions, several occurrences of the features `in` and `out` may be embedded in a single path. For the nouns *snowman* and *friend*, this format yields the following entries:

(28)
```
    lex(snowman,L) :-
    L>>cat === n,
    L>>sem..'arg-s'..pred === 'snowman'',
    L>>sem..'arg-s'..arg1 === L>>ref,
```

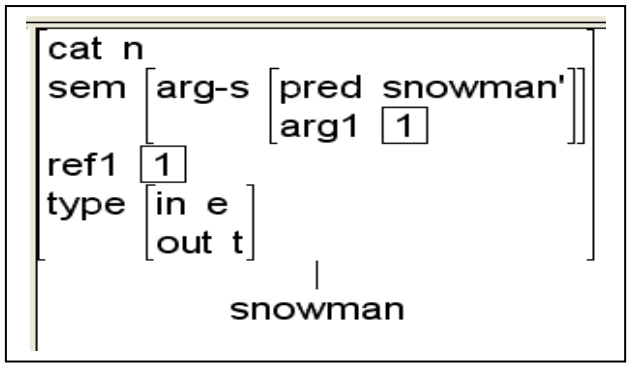
```
    L>>type..in === e,                % type e → t
    L>>type..out === t.
```

This shows an example where both input and output are atomic types. However, when we look at relational nouns, the output type is itself a function, yielding the following set of constraints for the noun *friend*:

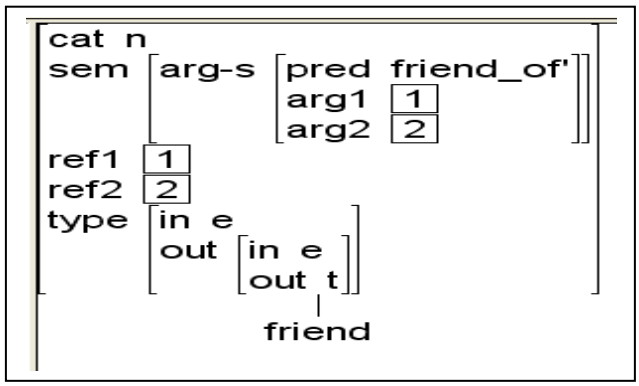
(29)
```
lex(friend,L) :-
    L>>cat === n,
    L>>sem..'arg-s'..pred === 'friend_of''',
    L>>sem..'arg-s'..arg1 === L>>ref,
    L>>sem..'arg-s'..arg2 === L>>ref2,

    L>>type..in === e,                % type e → (e → t)
    L>>type..out..in === e,
    L>>type..out..out === t.
```

The feature structures corresponding to (28) and (29) are (28') and (29'), respectively:

(28')



(29')



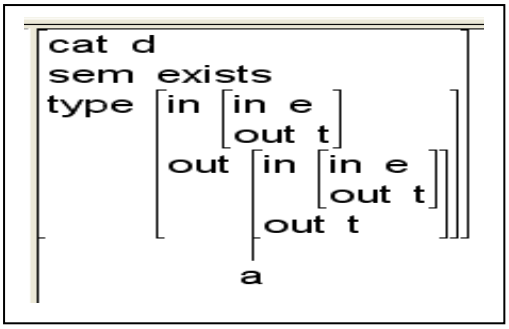
The semantic type of determiners

In classical formal semantics determiners play a crucial role due to the fact that they are the linguistic elements that introduce the quantifiers which are at the heart of logical semantics. Thus the indefinite article would be assigned a function of the type in (30):

(30) $\lambda Q[\lambda P[\exists x(Q(x) \wedge P(x)]]_{<<e,t>,<<e,t>,t>>}$

The rather complex type <<e,t>,<<e,t>,t>> looks as follows in the lexical entry for the indefinite article *a*:

(31)
```
lex(a, L) :-
    L>>cat === d,
    L>>sem === exists,

    L>>type..in..in === e,
    L>>type..in..out === t,
    L>>type..out..in..in === e,
    L>>type..out..in..out === t,
    L>>type..out..out === t.
```



In the semantic composition, the determiners have to combine with a nominal. Thus we have to check the type of that nominal, eg *boy*, which comes with the type in (32), as already mentioned above in connection with the discussion of the type of sortal nouns:

(32)
```
lex(boy,L) :-
    L>>cat === n,
    ...
    L>>type..in === e,              % type e → t
    L>>type..out === t.
```

The rule `nMax → D N` *revisited*
Based on the lexical specifications introduced above, we can now formulate the following type constraints in the `nMax → D N` rule:

(33)
```
M ---> [D1,D2] :-           % nMax ->D N
    M>>cat === nMax,
    D1>>cat === d,
    D2>>cat === n,
    ...
    D2>>type === D1>>type..in,
    M>>type === D1>>type..out.
```

The semantic type of the genitive clitic s
Turning next to the semantic type of the genitive clitic *s*, we have already presdented the function representing the constructional reading of the prenominal genitive. We repeat the function here:

(13) $\lambda\mathbf{P}\ [\lambda R\ [\lambda P\ [\mathbf{P}(\lambda u\ [\exists x\ [\forall y\ [R(u)(y) \leftrightarrow y = x]\ \&\ P(x)]])]]]$

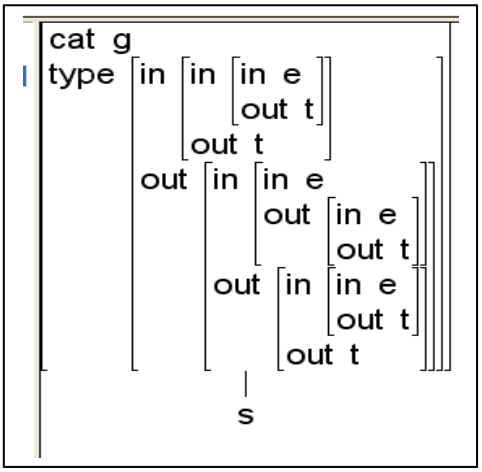
The type of this function is given in (34):

(34) <<<e,t>, t>,<<e,<e,t>>,<<e,t>, t>>>

This type is a function from a generalised quantifier to a function from a relation to a generalised quantifier. The input generalised quantifier comes from the syntactic complement of *s*, the relation comes from N_2, and the output generalised quantifier is the one corresponding to the full genitive construction.

This leads to a formalisation of the semantic type of the genitive clitic like the one shown in (35) with its corresponding feature structure:

(35)
```
lex(s, L) :-
    L>>cat === g,
    ...
    L>>type..in..in..in === e,
    L>>type..in..in..out === t,
    L>>type..in..out === t,
    L>>type..out..in..in === e,
    L>>type..out..in..out..in === e,
    L>>type..out..in..out..out === t,
    L>>type..out..out..in..in === e,
    L>>type..out..out..in..out === t,
    L>>type..out..out..out === t.
```

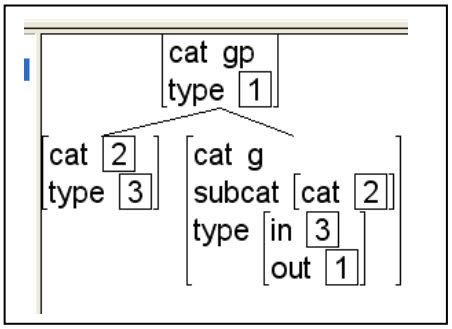


We now have the necessary ingredients to propose a semantic composition for *s* and its syntactic complement involving types.

The rule `GP -> XP G[XP]`

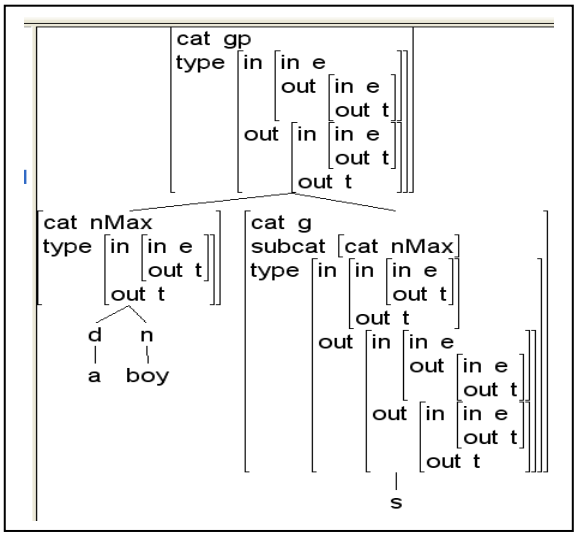
This composition can be formulated along the lines already illustrated in the rule combining determiners and their sister nominals, the only difference being that *s* acts as the functor and its syntactic complement as its semantic argument:

(36)
```
M ---> [D1,D2] :-   % GP -> XP G[XP]
    M>>cat === gp,
    D2>>cat === g,
    D1>>cat === D2>>subcat..cat,
    ...
    D1>>type === D2>>type..in,
    M>>type === D2>>type..out.
```



For a GP like *a boy's* we get the result shown in (37), where the type requirement posed by *s*, viz. <<e,t>, t>, is met by nMax, and the output type on *s*, <<e,<e,t>>,<<e,t>, t>> is passed on to the GP mother node:

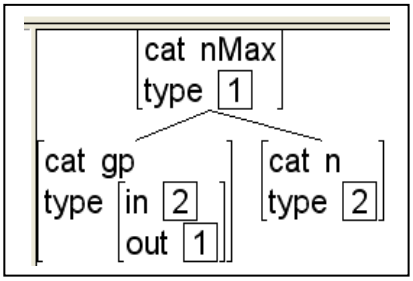
(37)



The final step in constructing the type semantics of the prenominal genitive construction is stated in the rule nMax → GP N.

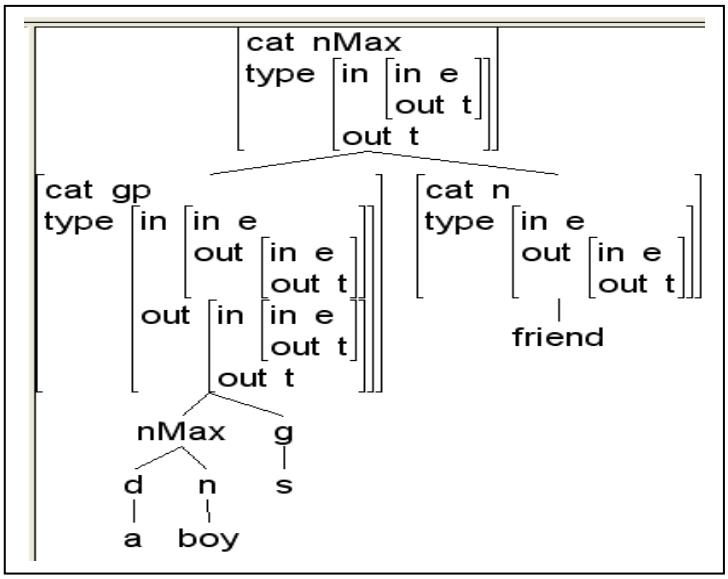
The rule nMax → GP N

(38)
```
M ---> [D1,D2] :-   % nMax ->GP N
    M>>cat === nMax,
    D1>>cat === gp,
    D2>>cat === n,
    ...
    D2>>type === D1>>type..in,
    M>>type === D1>>type..out.
```



This means that a full genitive construction like *a boy's friend* exhibits the following type composition:

(39)



This, in turn, explains why *a boy's snowman* now fails to parse since the type of *snowman*, which is <e,t> does not meet the type requirement of GP, which is <e, <e,t>>.

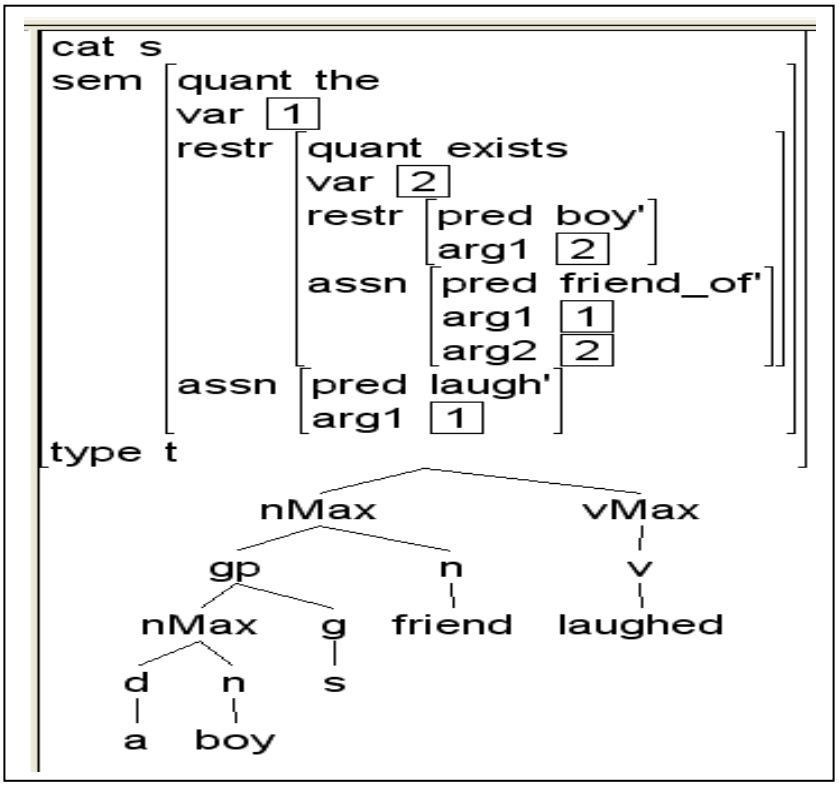
In order to produce a semantic analysis of a full sentence with a prenominal genitive construction as the subject and an intransitive verb, all that remains to do is to formulate type constraints in the vMax → V-rule and the S → nMax vMax-rule. These constraints appear as shown in (40.a) and (40.b), respectively:

(40) a.
```
M ---> [D1] :-      % vMax -> v
M>>cat === vMax,
D1>>cat === v,
D1>>subcat === null,
...
M>>type === D1>>type.
```

(40) b.
```
M ---> [D1,D2] :-   % S -> nMax vMax
M>>cat === s,
D1>>cat === nMax,
D2>>cat === vMax,
...
D2>>type === D1>>type..in,
M>>type === D1>>type..out.
```

We are now in a position to present a proposal for a semantic analysis of a full sentence like *a boy's friend laughed*, in which the subject is a pre-nominal genitive construction with a relational head noun, cf. (41):

(41)



The feature structure tree in (41) corresponds to the formula in (42):

(42) the(x, exists(y, boy'(y), friend_of'(y)(x)), laugh'(x))

In the following section, we turn to the problem presented by sortal nouns.

2.3.3 The case of non-relational head nouns

2.3.3.1 gengram01.2 Introducing Qualia structure in gengram01.1

It is now time to face the problem posed by non-relational, or sortal, head nouns of pre-nominal genitive constructions. The problem arises when N_2 is realised by nouns like *snowman*, *house*, *nose* or *cake*. The problem is that, in such cases, the relation required by the genitive is not directly available from the argument structure of the noun. This problem appears to be solvable within the framework of generative lexicon theory (Pustejovsky, 1995) by assuming that all nouns in the lexicon come equipped with a qualia structure defining a finite set of relations in which the denotata of the noun typically appear, cf. Vikner & Jensen, (2002) and Jensen &Vikner (2004).

When interpreting genitive constructions with non-relational head nouns, the agentive and the constitutive[8] qualia roles, in particular, appear to contribute relational information. Which of those two roles comes into play depends crucially on the ontological types of the N_1 and the N_2 expressions appearing in the construction. Thus, when interpreting an example like (1), *a boy's snowman*, the agentive qualia role provides a producer relation because *snowman* denotes an artefact, and in the lexicon artefact-denoting nouns are characterised by having an agentive qualia role providing the information that they denote entities which someone (a human or an animal) has produced. Thus, the fact that the qualia roles pertain to the non-linguistic entities which the nouns denote has the consequence that the semantic description of the nouns and of the constructions of which the nouns form constituents can be related to the ontological categories of the denoted entities.

We shall not consider the implementation of possible ontological subtypes of type e here. Rather, we shall first look at the enrichment of the lexical entries of nouns with a qualia structure.

2.3.3.2 Representing qualia in the lexical descripton of nouns

Consider the lexical entry proposed for *snowman* in gengram01.1:

(43)
```
lex(snowman,L) :-
    L>>cat === n,
    L>>sem..'arg-s'..pred === 'snowman''',
    L>>sem..'arg-s'..arg1 === L>>ref,

    L>>type..in === e,
    L>>type..out === t.
```

What (43) says is that *snowman* is a sortal noun of type <e,t> and is represented by a 1-place predicate and its argument, corresponding to the λ-term in (44):

[8] In more recent GL literature, what we refer to here as the constitutive qualia role, appears as the 'inverse constitutive role', cf. Pustejovsky et al. 2008: 10-11: "The Inverse Constitutive quale is used when the noun (*sic!*) is part of another entity of the same type". See also the account of this relation given in Vikner and Jensen 2002: 205-209.

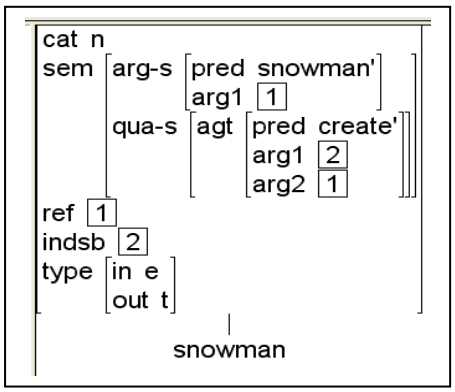
(44) λx[snowman'(x)]

Assuming that the information pertaining to the agentive qualia role is that a snowman is an entity created by someone, we might propose the following enhancement of the lexical entry in (43):

(45)
```
lex(snowman,L) :-
    L>>cat === n,
    L>>sem..'arg-s'..pred === 'snowman''',
    L>>sem..'arg-s'..arg1 === L>>ref,
    …
    L>>sem..'qua-s'..agt..pred === 'create''',
    L>>sem..'qua-s'..agt..arg1 === L>>indsb,
    L>>sem..'qua-s'..agt..arg2 === L>>ref,
```

These latter three constraints say that a snowman may enter a create-relation such that some entity created it. The `indsb` feature is used to represent the agent argument of the create'-predicate, and, crucially, the `ref` feature is the same feature as the one used to represent the referent of the snowman'-predicate, which amounts to saying that the snowman is what is created by the agent. The feature structure corresponding to (45) is shown in (46):

(46)



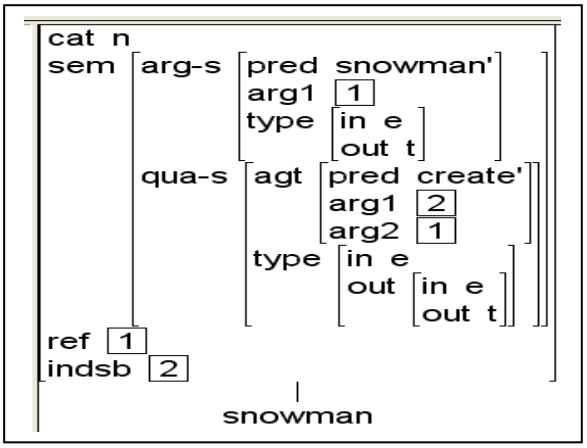
However, from the representation in (46), it is plainly seen that the type specification fits only the semantic representation in the argument structure and not the one in the qualia structure. It seems, then that we must incorporate the type specification into the appropriate part of the lexical entry. In effect this means that the type specification <e,t> should be incorporated with the argument structure, and the type specification <e,<e,t>> should should be incorporated with the qualia structure as shown in (47):

(47)
```
lex(snowman,L) :-
    L>>cat === n,
    L>>sem..'arg-s'..pred === 'snowman''',
    L>>sem..'arg-s'..arg1 === L>>ref,
    L>>sem..'arg-s'..type..in === e,    %type e -> t
    L>>sem..'arg-s'..type..out === t,

    L>>sem..'qua-s'..agt..pred === 'create''',
```

```
    L>>sem..'qua-s'..agt..arg1 === L>>indsb,
    L>>sem..'qua-s'..agt..arg2 === L>>ref,

    L>>sem..'qua-s'..type..in === e,       % type e -> (e -> t)
    L>>sem..'qua-s'..type..out..in === e,
    L>>sem..'qua-s'..type..out..out === t.
```

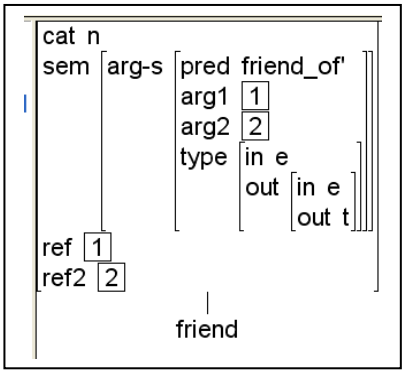
The entry in (47) now licenses the following feature structure:

(48)



Adopting this strategy entails corresponding changes in the type descriptions of all nouns in the lexicon. In particular, the relational noun *friend* will now be characterised as shown in (49), where the type constraint has been included as part of the argument structure:

(49)
```
    lex(friend,L) :-
    L>>cat === n,

    L>>sem..'arg-s'..pred === 'friend_of''',
    L>>sem..'arg-s'..arg1 === L>>ref,
    L>>sem..'arg-s'..arg2 === L>>ref2,

    L>>sem..'arg-s'..type..in === e,
    L>>sem..'arg-s'..type..out..in === e,
    L>>sem..'arg-s'..type..out..out === t.
```



Revising the rule nMax → GP N

As a consequence of the qualia-enriched description of non-relational nouns and the changed position of the type description of relational nouns, we now have to re-formulate the rule nMax → GP N. Up until now the type restriction has been formulated as shown in (50):

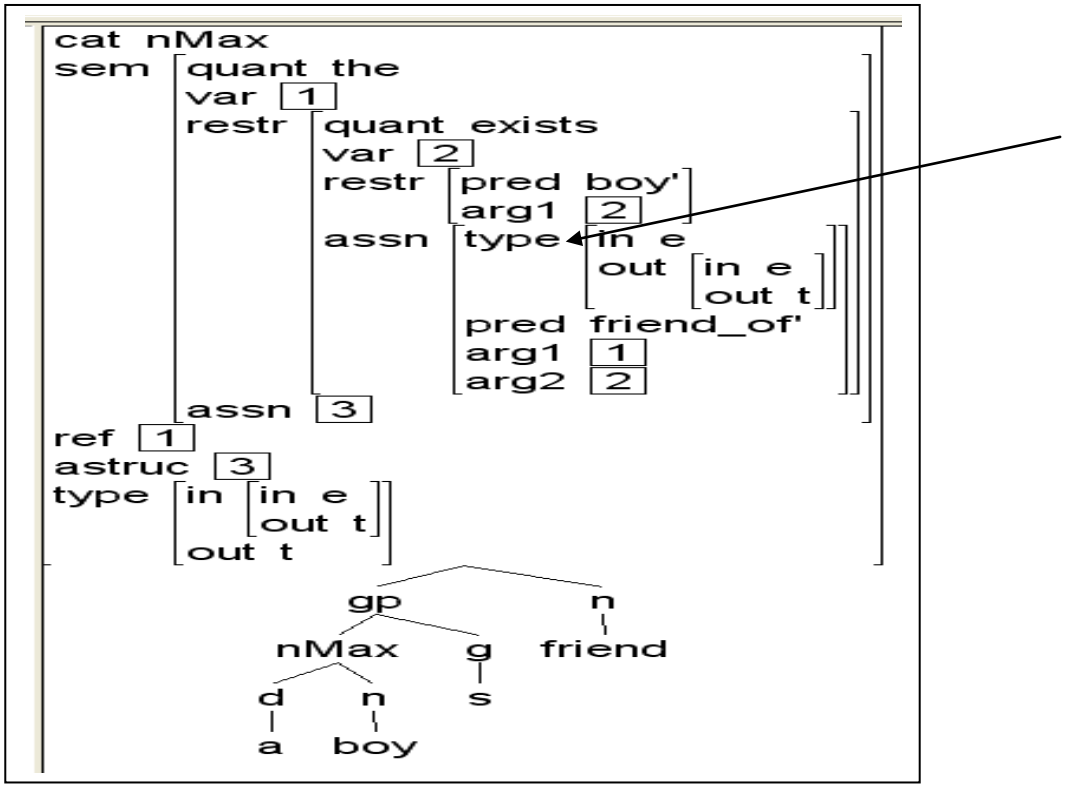
(50)
```
    M ---> [D1,D2] :-   % nMax ->GP N
    ...
    D2>>type === D1>>type..in,
    M>>type === D1>>type..out.
```

We now have to change the first of the two type constraints, thus:

(51)
```
M ---> [D1,D2] :-   % nMax ->GP N
    ...
    D2>>sem..'arg-s'..type === D1>>type..in,
    M>>type === D1>>type..out.
```

This strategy turns out to be untenable, however. The reason is seen in the following feature structure tree for the nMax *a boy's friend*:

(52)



The problem is that the type specification appears as part of the assertion of the quantifier structure of the restriction (cf. the arrow) resulting in a formula which is not well-formed. Clearly, the type information is not strictly a part of the semantic description of the phrase. Rather, it is meta-information about the semantics of friend_of'(y)(x). Therefore, we propose to keep the type information separate from the semantic representation itself, observing, however that we have to distinguish between the type information which pertains to the argument structure of a noun and the type information pertaining to each of the qualia roles of the noun in question.

For the nouns *snowman* and *friend*, this could be done as shown in (53) and (54), respectively, where the type information has been lifted out of the semantic representations themselves, as it were:

(53)
```
lex(snowman,L) :-
    L>>cat === n,
    L>>sem..'arg-s'..pred === 'snowman''',
```

```
    L>>sem..'arg-s'..arg1 === L>>ref,

    L>>'arg-s'..type..in === e,
    L>>'arg-s'..type..out === t,

    L>>sem..'qua-s'..agt..pred === 'create''',
    L>>sem..'qua-s'..agt..arg1 === L>>indsb,
    L>>sem..'qua-s'..agt..arg2 === L>>ref,

    L>>'qua-s'..type..in === e,
    L>>'qua-s'..type..out..in === e,
    L>>'qua-s'..type..out..out === t.
```

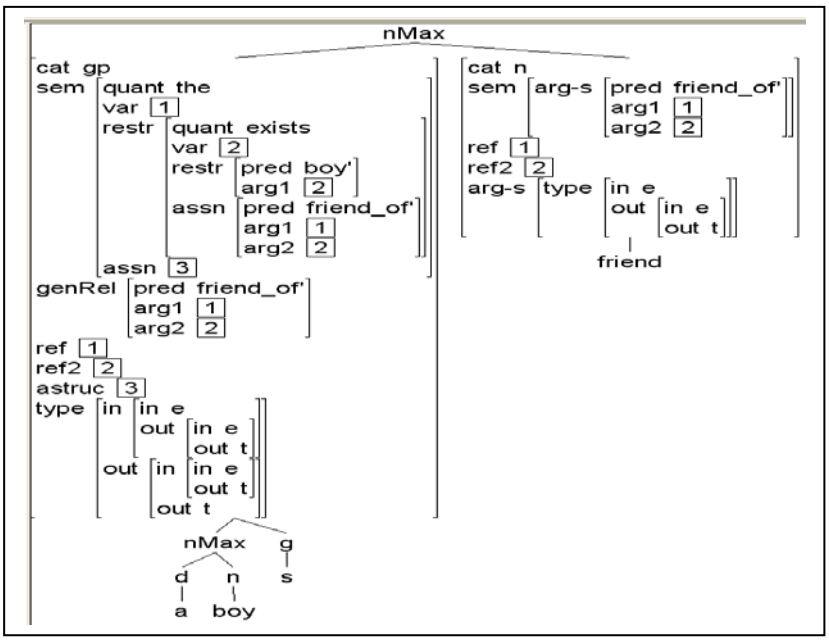
(54)
```
    lex(friend,L) :-
    L>>cat === n,

    L>>sem..'arg-s'..pred === 'friend_of''',
    L>>sem..'arg-s'..arg1 === L>>ref,
    L>>sem..'arg-s'..arg2 === L>>ref2,

    L>>'arg-s'..type..in === e,
    L>>'arg-s'..type..out..in === e,
    L>>'arg-s'..type..out..out === t.
```

Returning to the nMax ->GP N-rule, we can now formulate the relevant constraint as follows:

(55)
```
    M ---> [D1,D2] :-   % nMax ->GP N
    ...
    D2>>'arg-s'..type === D1>>type..in,
    ...
```

We now get the following feature structure tree for the nMax *a boy's friend*, where the value of the path `type..in` on the GP node has been unified with the value of the path `arg-s..type` on the N node as desired:

(56)



To sum up, we have now proposed a way of dealing with relational head nouns like *friend*, where the type of the relational noun is checked against the input type of GP as seen in (56). However, still outstanding is the problem of how to deal with sortal nouns like *snowman* and how to access the information in the qualia roles when the type of the argument structure fails to comply with the type requirement of GP. These challenges are addressed in the following section.

Gengram01.3: *Accessing qualia information*
We propose to deal with the type problem posed by sortal nouns and of how to access the information in the qualia roles by introducing dedicated *nMax → GP N* rules, which access the information in the agentive and the constitutive qualia roles, which are the roles relevant for the interpretation of pre-nominal genitive constructions, cf. Vikner & Jensen (2002).

Consider again the example (1), *a boy's snowman*. One of the senses ascribed to this nMax is the producer interpretation "the snowman that a boy has created". Presumably, the semantic representation of this sense is something like (57):

(57) the(x, and(snowman'(x), exists(y, boy'(y), **create'(x)(y)**)), ASSN)

where the restriction of the outer quantifier 'the' is a conjunction of two formulas, *snowman'(x)*, and *exists(y, boy'(y), create'(x)(y))*. Clearly, this representation is far more complex than the one we needed to represent pronominal genitives with relational head nouns, and the composition process turns out to be equally complex.

Type checking
Before turning to the problem of semantic composition, however, we shall take a look at the **type checking constraints** of the agentive nMax → GP N-rule. This is fairly straightforward in that we just have to unify the input type of GP with the type of the predicate occupying the agentive role of the sortal head noun, thus:

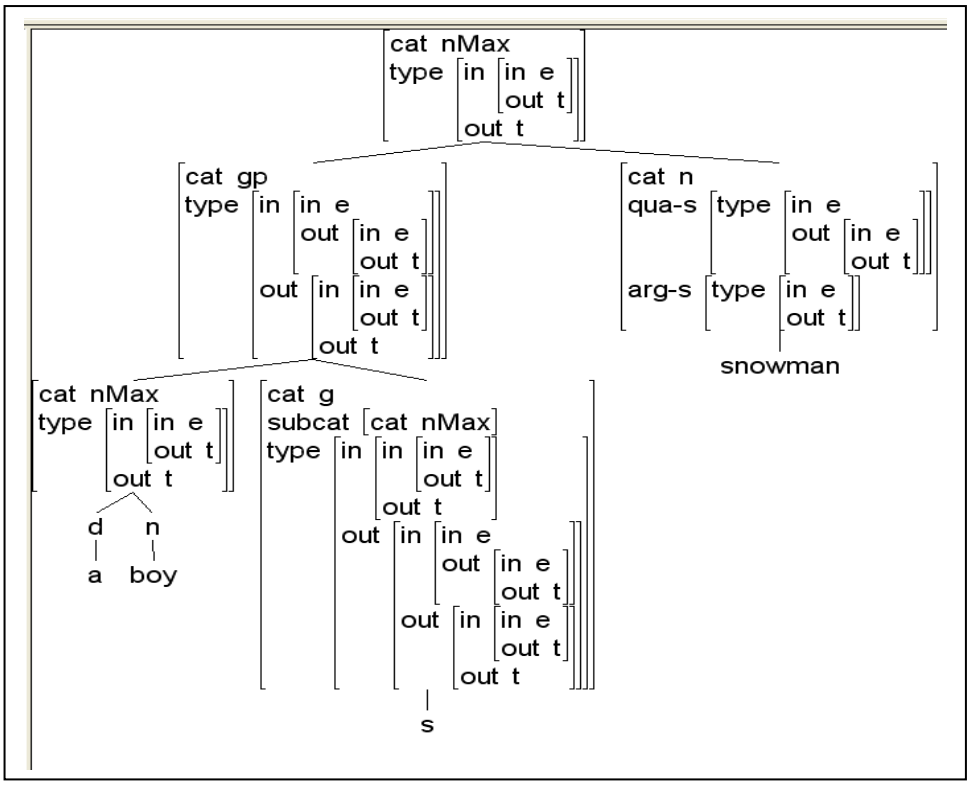
(58)
```
M ---> [D1,D2] :-  % nMax ->GP N
    M>>cat === nMax, % a boy's snowman(AGT ROLE OF SORTAL HEAD N)
    D1>>cat === gp,
    D2>>cat === n,
    ...
    D2>>'qua-s'..type === D1>>type..in,
    M>>type === D1>>type..out.
```

For the nMax in (1), *a boy's snowman*, this gives the result shown in

(59) below, where we can follow the type unifications all the way from the type of *s* to the type of the full genitive construction.

Semantic composition involving qualia information
Next, we address the problem of constructing a feature structure tree equivalent to the formula in (57). This formula shows the semantic representation of the genitive construction *a boy's snowman*, that is, it corresponds to the semantic representation of the topmost nMax node ($nMax_2$) in (59):

(59)



If we take a closer look at the structure of the formula in (57), we get the following analysis:

(57') the(x, and(snowman'(x), exists(y, boy'(y), create'(x)(y))), ASSN)

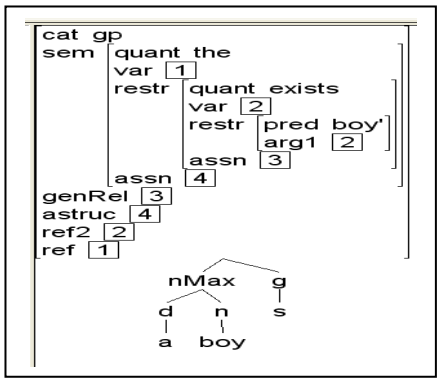
Q v r a

Q V R A

This analysis gives us a clue as to where we must look to pick up each of the several elements entering this complex semantic composition:

- The outer quantifier structure (henceforth QS-1) originates from the genitive clitic. With the semantic representation of $nMax_1$ as restriction QS-1 must be lifted to GP. This seems natural since there should be no difference between the semantic representation of a GP like *a boy's* whether it be a specifier of a genitive construction with a relational head noun or a specifier of a genitive construction with a sortal head noun. (60) shows the semantic representation of *a boy's*:

(60)

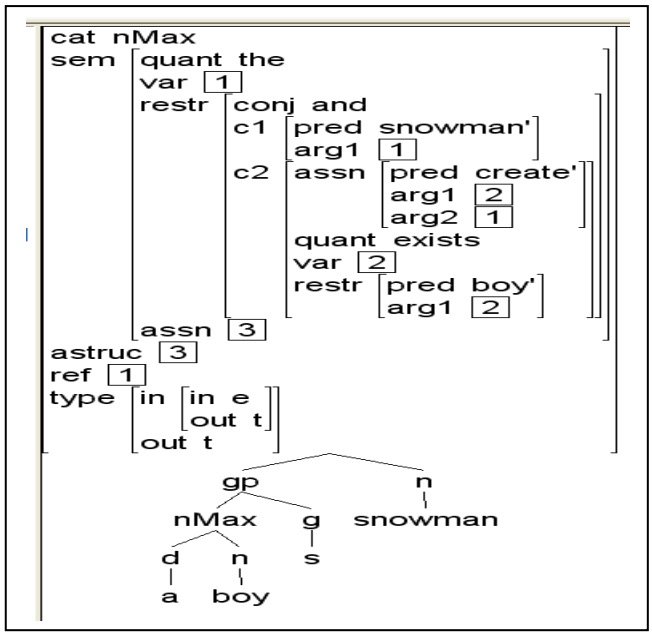


- However, as seen in (57'), when qualia information is to be involved, the restriction of QS-1 must be a conjunction, with the first conjunct, in this case 'snowman'(x)', coming from N_2 and the second conjunct coming partly from $nMax_1$ and partly from N_2. This indicates that the appropriate place for introducing the conjunction is $nMax_2$, since this node has access to information from $nMax_1$ via GP as well as information from N_2. Following this proposal, we get the formalisation of the semantic composition of the contributions from GP and N_2 shown in (61):

(61)
```
M ---> [D1,D2] :-   % nMax ->GP N
M>>cat === nMax,    % a boy's snowman
                    %(AGT ROLE OF SORTAL HEAD NOUN)
D1>>cat === gp,
D2>>cat === n,

M>>sem..quant === D1>>sem..quant,
M>>sem..var === D1>>ref,
M>>sem..restr..conj === and,
M>>sem..restr..c1 === D2>>sem..'arg-s',
M>>sem..restr..c2 === D1>>sem..restr,
M>>sem..restr..c2..assn === D1>>genRel,
M>>sem..assn === D1>>astruc,
M>>astruc === D1>>astruc,
D1>>genRel === D2>>sem..'qua-s'..agt,

D1>>ref === D2>>ref,
D1>>ref2 === D2>>indsb,
M>>ref === D2>>ref,
...
```

The semantic representation of (1), *a boy's snowman*, thus ends up as shown in (62):

(62)



A note on the constitutive role

According to Vikner & Jensen (2002), the genitive accesses either the agentive or the constitutive qualia role. Thus, we have to provide an account of the constitutive role and the way it is accessed by the genitive in semantic composition, too. However, we shall not pursue the matter further here.[9]

3. `gengram02`

The grammar `gengram02` is an implementation of the Jensen & Vikner's genitive theory seen in the light of some problems with scope in Pustejovsky's theory of control verbs pointed out by Markus Egg (2003). Our particular interest in these problem arises because we follow Pustejovsky's ideas fairly closely in our account of the pre-nominal genitive construction. So, if Pustejovsky's theory runs into problems, our account very likely does, too. Control verb syntax and semantics involve a number of problems similar to the ones met in connection with genitive constructions. In particular, semantic type coercion using information from the qualia stucture of

---

[9] It seems that there are other ways that the constitutive role might come into play. Thus, the constitutive role would probably be involved somehow in the interpretation of an example like (i):

(i) A boy's snowman melted

A snowman melts by virtue of the fact that it is constituted of snow. So, a possible rendering of the meaning of (i) would probably be (ii):

(ii) 'the x such that x is a snowman such that there exists some y such that y is snow and such that x is made of y and there exists a z such that z is a boy such that z created x melted'

So, it seems that, in order to get this interpretation, both the agentive and the constitutive role have to be involved! This means that the semantic representation of (i) is something like (iii):

(iii)
```
the (x,
  Q  V
      and (
          and (
              snowman'(x),
              exists(y, snow'(y), made_of'(y)(x))),
          exists(z, boy'(z), create'(x)(z))),
                    R
        melt'(x))
        A
```

Again, the restriction is a fairly complicated conjoined structure in which one conjunct is itself conjoined. If we look first at the inner conjunct

(iv) and (snowman'(x), exists(y, snow'(y), made_of'(y)(x)))

it represents a conjunction of the argument structure and the contents of the constitutive role of the lexical entry of the noun *snowman*. This is actually completely parallel to the situation described above in (57'). The reason why the structure in (iii) looks slightly more complicated is that the information in the constitutive role seems to be somewhat more elaborate than what is found in the agentive role. The complication arises because we have to introduce an extra predicate, snow', designating the material of which snowmen are made.

These observations are not directly related to the problems of genitive interpretation, however, they do seem to have some bearing on what information is available in the constitutive role of a noun like *snowman*, and hence also other nouns.

nouns. In relation to the formalisation of qualia proposed in Jensen & Vikner (in preparation) we are interested in finding arguments for the exact formulation of the semantic predicates in the qualia-roles, and we have to assume that a semantic predicate in a qualia role has the same form independent of whether it is to be used by a control verb or the genitive construction. Egg (2003: 165) raises three questions of prime importance to our own work: Given a formal environment for type coercion like (63):

(63) F(Op(A))

where *F* is a functor, *Op* a type-shifting operator and *A* an argument, "what does semantic construction for cases of type coercion look like, where does the operator *Op* come from, and how is it integrated with the functor argument structure?"[10]

Pustejovsky's implementation of control verbs like *begin* and *finish* holds that such verbs have an invariant semantics despite the fact that they allow two very different subcategorisation frames, one being an gerund VP and the other a transitive construction, cf. (64) and (65), respectively:

(64) Ann finished building the snowman
(65) Ann finished the snowman

Assuming an invariant semantics of these verbs then necessitates a re-interpretation in the case of the transitive construction in order to get an interpretation similar to the one we get when the complement of the control verb is an gerund VP. Pustejovsky's claim is that the re-interpretation is made possible by accessing information in the qualia roles. Thus, in (65) the agentive role may come into play in order to get the interpretation parallel with the one of (64).

Egg's claim is that under Pustejovsky's 1991-analysis the scope of quantifiers necessarily becomes too narrow with verbs like *begin* and *finish*. Pustejovsky's 1991-analysis posits a function QT which operates on the object-NP of the control verb in a way such that access to the qualia roles is made possible to a semantic predicate of the required semantic type.

References

Asher, Nicholas, Pascal Denis. 2004. Dynamic Typing for Lexical Semantics – A Case Study: The Genitive Construction. In: Varzi, Achille C. & Laure Vieu (eds.). 2004: *Formal Ontology in Information Systems*. Proceedings of the Third International Conference (FOIS-2004). IOS Press. Amsterdam. Berlin. Oxford. Tokyo. Washington, DC, pp. 165 – 176.

Egg, Markus (2003): Beginning Novels and finishing Hamburgers: Remarks on the semantics of *to begin*. *Journal of Semantics*, 20: 163-191.

Jensen, Per Anker & Carl Vikner (1996): *Natursprogsbehandling og unifikationsgrammatik. Vol. 2*. Handelshøjskolens Forlag.

Jensen, Per Anker, Carl Vikner. (2004). The English Prenominal Genitive and Lexical Semantics. In: Ji-yung Kim, Yury A. Lander, Barbara H. Partee (eds.): *Possessives and Beyond:*

[10] Note also that Egg does not endorse the term *type coercion*. He says (2003:165): "Type coercion processes characteristically involve a *sortal* shift. … [sortal shifts] change not only the combinatory potential of an expression but also the way in which it relates to the common-sense ontology that underlies the semantic sorts."

Semantics and Syntax. University of Massachusetts Occasional Papers in Linguistics 29, 2-27. Amherst, Massachusetts.

Jensen &Vikner (in preparation): Ontological Type Coercion in Lexical Semantics. Unpublished ms.

Pustejovsky, J. 1991. The Generative Lexicon. *Computational Linguistics*, 17, 409-441.

Vikner, Carl, Per Anker Jensen. (2002). A Semantic Analysis of The English Genitive. Interaction of Lexical and Formal Semantics. *Studia Linguistica,* vol. 56, 2, 191-226.

APPENDIX
The implemented grammars
****

```
% gensyn01.pl
% bruges sammen med genlex01_1.pl
% Programmet er dokumenteret i filen 'gengram Documentation.doc'


%%%%%%%%%%%
% S-RULES  %
%%%%%%%%%%%

M ---> [D1,D2] :-   % S -> nMax vMax
M>>cat === s,
D1>>cat === nMax,
D2>>cat === vMax,
M>>sem === D1>>sem,
D1>>astruc === D2>>sem,
D1>>ref === D2>>indsb.


%%%%%%%%%%%
% GP-RULES %
%%%%%%%%%%%

M ---> [D1,D2] :-   % GP -> XP G[XP]
M>>cat === gp,
D2>>cat === g,
D1>>cat === D2>>subcat..cat,
M>>sem === D2>>sem,
M>>genRel === D2>>genRel,% for at hente genitiv-relation fra n2 i reglen nMax ->
gp n
D1>>astruc === D2>>genRel,% gør genitiv-relationen til assertion i nMax1
M>>astruc === D2>>astruc, % hent vMax semantik ind i G via GP (og nMax2)
D2>>restruc === D1>>sem, % nMax1 sættes ind i restriktionen på s

M>>ref2 === D1>>ref,    % nMax leverer komplement(y)-argumentet til relationelle
substantiver dvs ref2
M>>ref === D2>>ref.     % ref er den variabel der bindes af 'the' i s, dvs
referenten for n2, kernen i genitivkonstruktionen


%%%%%%%%%%%%%
% nMax-RULES %
%%%%%%%%%%%%%


M ---> [D1] :-      % nMax -> N
M>>cat === nMax,
D1>>cat === n,

M>>sem === D1>>sem..'arg-s',
M>>ref === D1>>ref.

M ---> [D1,D2] :-   % nMax ->D N
M>>cat === nMax,
D1>>cat === d,
```

```
D2>>cat === n,

M>>sem..quant === D1>>sem,
M>>sem..var === D2>>ref,
M>>sem..restr === D2>>sem..'arg-s',
M>>sem..assn === M>>astruc,
M>>ref === M>>sem..var.

M ---> [D1,D2] :-   % nMax ->GP N
M>>cat === nMax,
D1>>cat === gp,
D2>>cat === n,

M>>sem === D1>>sem,
D1>>genRel === D2>>sem..'arg-s',
D1>>ref === D2>>ref,
D1>>ref2 === D2>>ref2,
M>>ref === D2>>ref,
M>>astruc === D1>>astruc.

%%%%%%%%%%%%%
% vMax-RULES %
%%%%%%%%%%%%%

M ---> [D1] :-      % vMax -> v
M>>cat === vMax,
D1>>cat === v,
D1>>subcat === null,
M>>sem === D1>>sem,
M>>indsb === D1>>indsb.


%genlex01.pl
% To be run with gensyn01_1.pl


%%%%%%%%%%%%%%%%%%
% DETERMINATIVES %
%%%%%%%%%%%%%%%%%%

lex(a, L) :-
L>>cat === d,
L>>sem === exists.

lex(every, L) :-
L>>cat === d,
L>>sem === all.


%%%%%%%%%%%%%%%%%%%
% GENITIVE CLITIC %
%%%%%%%%%%%%%%%%%%%

lex(s, L) :-
L>>cat === g,
L>>subcat..cat === nMax,
L>>sem..quant === the,
L>>sem..var === L>>ref,
L>>sem..restr === L>>restruc,
```

```
L>>sem..assn === L>>astruc,
L>>genRel === L>>relation.

%%%%%%%%
% NOUNS %
%%%%%%%%

lex(boy,L) :-
L>>cat === n,
L>>sem..'arg-s'..pred === 'boy''',
L>>sem..'arg-s'..arg1 === L>>ref.

lex(friend,L) :-                          % RELATIONELT
L>>cat === n,
L>>sem..'arg-s'..pred === 'friend_of''', % ref er vennen (friend) og ref2 er den
ref er ven med (eg Bo i 'Bo's friend').
                                          % Dvs: i 'Ann is a friend of Bo' er Ann
ref og Bo er ref2.
                                          % Dette skal også gælde ved genitiven
'Ann is Bo's friend', altså at Bo er ref2 her!
L>>sem..'arg-s'..arg1 === L>>ref,
L>>sem..'arg-s'..arg2 === L>>ref2.

lex(snowman,L) :-
L>>cat === n,
L>>sem..'arg-s'..pred === 'snowman''',
L>>sem..'arg-s'..arg1 === L>>ref.

%%%%%%%%
% VERBS %
%%%%%%%%

lex(laughed,L) :-
L>>cat === v,
L>>sem..pred === 'laugh''',
L>>sem..arg1 === L>>indsb.


%genlex01_1.pl
% bruges sammen med gensyn01_1.pl


%%%%%%%%%%%%%%%%%
% DETERMINATIVES %
%%%%%%%%%%%%%%%%%

lex(a, L) :-
L>>cat === d,
L>>sem === exists,

L>>type..in..in === e,
L>>type..in..out === t,
L>>type..out..in..in === e,
L>>type..out..in..out === t,
L>>type..out..out === t.

lex(every, L) :-
L>>cat === d,
L>>sem === all,
```

```
L>>type..in..in === e,
L>>type..in..out === t,
L>>type..out..in..in === e,
L>>type..out..in..out === t,
L>>type..out..out..out === t.

%%%%%%%%%%%%%%%%%%
% GENITIVE CLITIC %
%%%%%%%%%%%%%%%%%%

lex(s, L) :-
L>>cat === g,
L>>subcat..cat === nMax,

L>>sem..quant === the,
L>>sem..var === L>>ref,
L>>sem..restr === L>>restruc,
L>>sem..assn === L>>astruc,
L>>genRel === L>>relation,

L>>type..in..in..in === e,
L>>type..in..in..out === t,
L>>type..in..out === t,
L>>type..out..in..in === e,
L>>type..out..in..out..in === e,
L>>type..out..in..out..out === t,
L>>type..out..out..in..in === e,
L>>type..out..out..in..out === t,
L>>type..out..out..out === t.

%%%%%%%%%
% NOUNS %
%%%%%%%%%

lex(boy,L) :-
L>>cat === n,
L>>sem..'arg-s'..pred === 'boy''',
L>>sem..'arg-s'..arg1 === L>>ref,

L>>type..in === e,       % type e -> t
L>>type..out === t.


lex(friend,L) :-                          % RELATIONELT
L>>cat === n,

L>>sem..'arg-s'..pred === 'friend_of''', % ref er vennen (friend) og ref2 er den
ref er ven med (eg Bo i 'Bo's friend').
                                          % Dvs: i 'Ann is a friend of Bo' er Ann
ref og Bo er ref2.
                                          % Dette skal også gælde ved genitiven
'Ann is Bo's friend', altså at Bo er ref2 her!
L>>sem..'arg-s'..arg1 === L>>ref,
L>>sem..'arg-s'..arg2 === L>>ref2,

L>>type..in === e,       % type e -> (e -> t)
L>>type..out..in === e,
L>>type..out..out === t.
```

```
lex(snowman,L) :-
L>>cat === n,
L>>sem..'arg-s'..pred === 'snowman'',
L>>sem..'arg-s'..arg1 === L>>ref,

L>>type..in === e,
L>>type..out === t.

%%%%%%%%%
% VERBS %
%%%%%%%%%

lex(laughed,L) :-
L>>cat === v,
L>>sem..pred === 'laugh'',
L>>sem..arg1 === L>>indsb,

L>>type..in === e,       % type e -> t
L>>type..out === t.

% gensyn01_2.pl
% bruges sammen med genlex01_2.pl
% Programmet er dokumenteret i filen 'gengram Documentation.doc'

%%%%%%%%%%%
% S-RULES  %
%%%%%%%%%%%

M ---> [D1,D2] :-   % S -> nMax vMax
M>>cat === s,       % a boy's friend laughed
D1>>cat === nMax,
D2>>cat === vMax,
M>>sem === D1>>sem,
D1>>astruc === D2>>sem,
D1>>ref === D2>>indsb,

D2>>type === D1>>type..in,
M>>type === D1>>type..out.

%%%%%%%%%%%
% GP-RULES %
%%%%%%%%%%%

M ---> [D1,D2] :-   % GP -> XP G[XP]
M>>cat === gp,      % a boy's
D2>>cat === g,
D1>>cat === D2>>subcat..cat,

M>>sem === D2>>sem,
M>>genRel === D2>>genRel,   % for at hente genitiv-relation fra n i nMax -> gp n
D1>>astruc === D2>>genRel,  % gør genitiv-relation til assertion i nMax1
M>>astruc === D2>>astruc,   % hent vMax semantik
D2>>restruc === D1>>sem,    % nMax sættes ind i restriktionen på s

M>>ref2 === D1>>ref,        % nMax leverer komplement-argumentet til
relationelle substantiver dvs ref2
```

```
M>>ref === D2>>ref,         % den variabel der bindes af 'the' i s, dvs
referenten for n2, altså hele genitivkonstruktionen

D1>>type === D2>>type..in,
M>>type === D2>>type..out.

%%%%%%%%%%%%%
% nMax-RULES %
%%%%%%%%%%%%%
/*
M ---> [D1] :-      % nMax -> N
M>>cat === nMax,
D1>>cat === n,

M>>sem === D1>>sem..'arg-s',
M>>ref === D1>>ref,

M>>type === D1>>type.
*/

M ---> [D1,D2] :-   % nMax ->D N
M>>cat === nMax,    % a boy
D1>>cat === d,
D2>>cat === n,

M>>sem..quant === D1>>sem,
M>>sem..var === D2>>ref,
M>>sem..restr === D2>>sem..'arg-s',
M>>sem..assn === M>>astruc,
M>>ref === M>>sem..var,

D2>>'arg-s'..type === D1>>type..in,
M>>type === D1>>type..out.


M ---> [D1,D2] :-   % nMax ->GP N
M>>cat === nMax,    % a boy's friend
D1>>cat === gp,
D2>>cat === n,

M>>sem === D1>>sem,
D1>>genRel === D2>>sem..'arg-s',
D1>>ref === D2>>ref,
D1>>ref2 === D2>>ref2,
M>>ref === D2>>ref,
M>>astruc === D1>>astruc,

D2>>'arg-s'..type === D1>>type..in,
M>>type === D1>>type..out.


%%%%%%%%%%%%%
% vMax-RULES %
%%%%%%%%%%%%%

M ---> [D1] :-      % vMax -> v
M>>cat === vMax,    % laughed
D1>>cat === v,
D1>>subcat === null,
```

```
M>>sem === D1>>sem,
M>>indsb === D1>>indsb,

M>>type === D1>>type.



%genlex01_2.pl
% bruges sammen med gensyn01_2.pl


%%%%%%%%%%%%%%%%%%
% DETERMINATIVES %
%%%%%%%%%%%%%%%%%%

lex(a, L) :-
L>>cat === d,
L>>sem === exists,

L>>type..in..in === e,
L>>type..in..out === t,
L>>type..out..in..in === e,
L>>type..out..in..out === t,
L>>type..out..out === t.

lex(every, L) :-
L>>cat === d,
L>>sem === all,

L>>type..in..in === e,
L>>type..in..out === t,
L>>type..out..in..in === e,
L>>type..out..in..out === t,
L>>type..out..out..out === t.

%%%%%%%%%%%%%%%%%%%
% GENITIVE CLITIC %
%%%%%%%%%%%%%%%%%%%

lex(s, L) :-
L>>cat === g,
L>>subcat..cat === nMax,

L>>sem..quant === the,
L>>sem..var === L>>ref,
L>>sem..restr === L>>restruc,
L>>sem..assn === L>>astruc,
L>>genRel === L>>relation,

L>>type..in..in..in === e,
L>>type..in..in..out === t,
L>>type..in..out === t,
L>>type..out..in..in === e,
L>>type..out..in..out..in === e,
L>>type..out..in..out..out === t,
L>>type..out..out..in..in === e,
L>>type..out..out..in..out === t,
L>>type..out..out..out === t.
```

```
%%%%%%%%%
% NOUNS %
%%%%%%%%%

lex(boy,L) :-
L>>cat === n,
L>>sem..'arg-s'..pred === 'boy''',
L>>sem..'arg-s'..arg1 === L>>ref,

L>>'arg-s'..type..in === e,       % type e -> t
L>>'arg-s'..type..out === t.


lex(friend,L) :-                          % RELATIONELT
L>>cat === n,

L>>sem..'arg-s'..pred === 'friend_of''', % ref er vennen (friend) og ref2 er den
ref er ven med (eg Bo i 'Bo's friend').
                                          % Dvs: i 'Ann is a friend of Bo' er Ann
ref og Bo er ref2.
                                          % Dette skal også gælde ved genitiven
'Ann is Bo's friend', altså at Bo er ref2 her!
L>>sem..'arg-s'..arg1 === L>>ref,
L>>sem..'arg-s'..arg2 === L>>ref2,

L>>'arg-s'..type..in === e,         % type e -> (e -> t)
L>>'arg-s'..type..out..in === e,
L>>'arg-s'..type..out..out === t.

lex(snowman,L) :-
L>>cat === n,
L>>sem..'arg-s'..pred === 'snowman''',
L>>sem..'arg-s'..arg1 === L>>ref,

L>>'arg-s'..type..in === e,  %type e -> t
L>>'arg-s'..type..out === t,

L>>sem..'qua-s'..agt..pred === 'create''',
L>>sem..'qua-s'..agt..arg1 === L>>indsb,
L>>sem..'qua-s'..agt..arg2 === L>>ref,

L>>'qua-s'..type..in === e,       % type e -> (e -> t)
L>>'qua-s'..type..out..in === e,
L>>'qua-s'..type..out..out === t.


%%%%%%%%%
% VERBS %
%%%%%%%%%

lex(laughed,L) :-
L>>cat === v,
L>>sem..pred === 'laugh''',
L>>sem..arg1 === L>>indsb,

L>>type..in === e,       % type e -> t
L>>type..out === t.
```

```
% gensyn01_3.pl
% bruges sammen med genlex01_3.pl
% Programmet er dokumenteret i filen 'gengram Documentation.doc'

%%%%%%%%%%%
% S-RULES  %
%%%%%%%%%%%

M ---> [D1,D2] :-   % S -> nMax vMax
M>>cat === s,       % a boy's friend laughed
D1>>cat === nMax,
D2>>cat === vMax,
M>>sem === D1>>sem,
D1>>astruc === D2>>sem,
D1>>ref === D2>>indsb,

D2>>type === D1>>type..in,
M>>type === D1>>type..out.

%%%%%%%%%%%
% GP-RULES %
%%%%%%%%%%%

M ---> [D1,D2] :-   % GP -> XP G[XP]
M>>cat === gp,      % a boy's
D2>>cat === g,
D1>>cat === D2>>subcat..cat,

M>>sem === D2>>sem,

M>>genRel === D2>>genRel,   % for at hente genitiv-relation fra n i nMax -> gp n
D1>>astruc === D2>>genRel,  % gør genitiv-relation til assertion i nMax1
M>>astruc === D2>>astruc,   % hent vMax semantik
D2>>restruc === D1>>sem,    % nMax sættes ind i restriktionen på s

M>>ref2 === D1>>ref,        % nMax leverer komplement-argumentet til
relationelle substantiver dvs ref2
M>>ref === D2>>ref,         % den variabel der bindes af 'the' i s, dvs
referenten for n2, altså hele genitivkonstruktionen

D1>>type === D2>>type..in,
M>>type === D2>>type..out.

%%%%%%%%%%%%%
% nMax-RULES %
%%%%%%%%%%%%%

M ---> [D1,D2] :-   % nMax ->D N
M>>cat === nMax,    % a boy
D1>>cat === d,
D2>>cat === n,

M>>sem..quant === D1>>sem,
M>>sem..var === D2>>ref,
M>>sem..restr === D2>>sem..'arg-s',
M>>sem..assn === M>>astruc,
M>>ref === M>>sem..var,

D2>>'arg-s'..type === D1>>type..in,
```

```
M>>type === D1>>type..out.


M ---> [D1,D2] :-   % nMax ->GP N
M>>cat === nMax,    % a boy's friend (RELATIONAL HEAD NOUNS)
D1>>cat === gp,
D2>>cat === n,

M>>sem === D1>>sem,
D1>>genRel === D2>>sem..'arg-s',
D1>>ref === D2>>ref,
D1>>ref2 === D2>>ref2,
M>>ref === D2>>ref,
M>>astruc === D1>>astruc,

D2>>'arg-s'..type === D1>>type..in,
M>>type === D1>>type..out.

M ---> [D1,D2] :-   % nMax ->GP N
M>>cat === nMax,    % a boy's snowman
                    %(AGT ROLE OF SORTAL HEAD NOUN)
D1>>cat === gp,
D2>>cat === n,

M>>sem..quant === D1>>sem..quant,
M>>sem..var === D1>>ref,
M>>sem..restr..conj === and,
M>>sem..restr..c1 === D2>>sem..'arg-s',
M>>sem..restr..c2 === D1>>sem..restr,
M>>sem..restr..c2..assn === D1>>genRel,
M>>sem..assn === D1>>astruc,
M>>astruc === D1>>astruc,
D1>>genRel === D2>>sem..'qua-s'..agt,

D1>>ref === D2>>ref,
D1>>ref2 === D2>>indsb,
M>>ref === D2>>ref,

D2>>'qua-s'..type === D1>>type..in,
M>>type === D1>>type..out.


%%%%%%%%%%%%%
% vMax-RULES %
%%%%%%%%%%%%%

M ---> [D1] :-      % vMax -> v
M>>cat === vMax,    % laughed
D1>>cat === v,
D1>>subcat === null,
M>>sem === D1>>sem,
M>>indsb === D1>>indsb,

M>>type === D1>>type.
```

```
%genlex01_3.pl
% bruges sammen med gensyn01_3.pl


%%%%%%%%%%%%%%%%%%
% DETERMINATIVES %
%%%%%%%%%%%%%%%%%%

lex(a, L) :-
L>>cat === d,
L>>sem === exists,

L>>type..in..in === e,
L>>type..in..out === t,
L>>type..out..in..in === e,
L>>type..out..in..out === t,
L>>type..out..out === t.

lex(every, L) :-
L>>cat === d,
L>>sem === all,

L>>type..in..in === e,
L>>type..in..out === t,
L>>type..out..in..in === e,
L>>type..out..in..out === t,
L>>type..out..out === t.

%%%%%%%%%%%%%%%%%%%%
% GENITIVE CLITIC %
%%%%%%%%%%%%%%%%%%%%

lex(s, L) :-
L>>cat === g,
L>>subcat..cat === nMax,

L>>sem..quant === the,
L>>sem..var === L>>ref,
L>>sem..restr === L>>restruc,
L>>sem..assn === L>>astruc,
L>>genRel === L>>relation,

L>>type..in..in..in === e,
L>>type..in..in..out === t,
L>>type..in..out === t,
L>>type..out..in..in === e,
L>>type..out..in..out..in === e,
L>>type..out..in..out..out === t,
L>>type..out..out..in..in === e,
L>>type..out..out..in..out === t,
L>>type..out..out..out === t.

%%%%%%%%%
% NOUNS %
%%%%%%%%%

lex(boy,L) :-
L>>cat === n,
```

```
L>>sem..'arg-s'..pred === 'boy''',
L>>sem..'arg-s'..arg1 === L>>ref,

L>>'arg-s'..type..in === e,       % type e -> t
L>>'arg-s'..type..out === t.


lex(friend,L) :-                          % RELATIONELT
L>>cat === n,

L>>sem..'arg-s'..pred === 'friend_of''', % ref er vennen (friend) og ref2 er den
ref er ven med (eg Bo i 'Bo's friend').
                                          % Dvs: i 'Ann is a friend of Bo' er Ann
ref og Bo er ref2.
                                          % Dette skal også gælde ved genitiven
'Ann is Bo's friend', altså at Bo er ref2 her!
L>>sem..'arg-s'..arg1 === L>>ref,
L>>sem..'arg-s'..arg2 === L>>ref2,

L>>'arg-s'..type..in === e,         % type e -> (e -> t)
L>>'arg-s'..type..out..in === e,
L>>'arg-s'..type..out..out === t.

lex(snowman,L) :-
L>>cat === n,

L>>sem..'arg-s'..pred === 'snowman''',
L>>sem..'arg-s'..arg1 === L>>ref,

L>>'arg-s'..type..in === e, %type e -> t
L>>'arg-s'..type..out === t,

L>>sem..'qua-s'..agt..pred === 'create''',
L>>sem..'qua-s'..agt..arg1 === L>>indsb,
L>>sem..'qua-s'..agt..arg2 === L>>ref,

L>>'qua-s'..type..in === e,       % type e -> (e -> t)
L>>'qua-s'..type..out..in === e,
L>>'qua-s'..type..out..out === t.

%%%%%%%%%
% VERBS %
%%%%%%%%%

lex(laughed,L) :-
L>>cat === v,
L>>sem..pred === 'laugh''',
L>>sem..arg1 === L>>indsb,

L>>type..in === e,       % type e -> t
L>>type..out === t.

% gensyn01_4.pl
% bruges sammen med genlex01_4.pl
% Programmet er dokumenteret i filen 'gengram Documentation.doc'

%%%%%%%%%%%
% S-RULES  %
%%%%%%%%%%%
```

```
M ---> [D1,D2] :-   % S -> nMax vMax
M>>cat === s,       % a boy's friend laughed
D1>>cat === nMax,
D2>>cat === vMax.
/*
M>>sem === D1>>sem,
D1>>astruc === D2>>sem,
D1>>ref === D2>>indsb,

D2>>type === D1>>type..in,
M>>type === D1>>type..out.
*/
%%%%%%%%%%%
% GP-RULES %
%%%%%%%%%%%

M ---> [D1,D2] :-   % GP -> XP G[XP]
M>>cat === gp,      % a boy's
D2>>cat === g,
D1>>cat === D2>>subcat..cat.
/*
M>>sem === D2>>sem,

M>>genRel === D2>>genRel,   % for at hente genitiv-relation fra n i nMax -> gp n
D1>>astruc === D2>>genRel,  % gør genitiv-relation til assertion i nMax1
M>>astruc === D2>>astruc,   % hent vMax semantik
D2>>restruc === D1>>sem,    % nMax sættes ind i restriktionen på s

M>>ref2 === D1>>ref,        % nMax leverer komplement-argumentet til
relationelle substantiver dvs ref2
M>>ref === D2>>ref,         % den variabel der bindes af 'the' i s, dvs
referenten for n2, altså hele genitivkonstruktionen

D1>>type === D2>>type..in,
M>>type === D2>>type..out.
*/
%%%%%%%%%%%%%
% nMax-RULES %
%%%%%%%%%%%%%

M ---> [D1,D2] :-   % nMax ->D N
M>>cat === nMax,    % a boy
D1>>cat === d,
D2>>cat === n.
/*
M>>sem..quant === D1>>sem,
M>>sem..var === D2>>ref,
M>>sem..restr === D2>>sem..'arg-s',
M>>sem..assn === M>>astruc,
M>>ref === M>>sem..var,

D2>>'arg-s'..type === D1>>type..in,
M>>type === D1>>type..out.
*/

%NY
M ---> [D1,D2] :-   % nMax ->GP Nbar
M>>cat === nMax,    % a boy's friend
```

```
D1>>cat === gp,
D2>>cat === nbar.


M ---> [D1,D2] :-   % nMax ->GP N
M>>cat === nMax,    % a boy's friend (RELATIONAL HEAD NOUNS)
D1>>cat === gp,
D2>>cat === n.

/*
M>>sem === D1>>sem,
D1>>genRel === D2>>sem..'arg-s',
D1>>ref === D2>>ref,
D1>>ref2 === D2>>ref2,
M>>ref === D2>>ref,
M>>astruc === D1>>astruc,

D2>>'arg-s'..type === D1>>type..in,
M>>type === D1>>type..out.
*/

M ---> [D1,D2] :-   % nMax ->GP N
M>>cat === nMax,    % a boy's snowman
                    %(AGT ROLE OF SORTAL HEAD NOUN)
D1>>cat === gp,
D2>>cat === n.
/*
M>>sem..quant === D1>>sem..quant,
M>>sem..var === D1>>ref,
M>>sem..restr..conj === and,
M>>sem..restr..c1 === D2>>sem..'arg-s',
M>>sem..restr..c2 === D1>>sem..restr,
M>>sem..restr..c2..assn === D1>>genRel,
M>>sem..assn === D1>>astruc,
M>>astruc === D1>>astruc,
D1>>genRel === D2>>sem..'qua-s'..agt,

D1>>ref === D2>>ref,
D1>>ref2 === D2>>indsb,
M>>ref === D2>>ref,

D2>>'qua-s'..type === D1>>type..in,
M>>type === D1>>type..out.
*/

%%%%%%%%%%%%%
% nBar-RULES %
%%%%%%%%%%%%%

M ---> [D1,D2] :-   % nBar ->GP nBar
M>>cat === nbar,    % a boy's friend
D1>>cat === q,
D2>>cat === n.

%%%%%%%%%%%%%
% vMax-RULES %
%%%%%%%%%%%%%

M ---> [D1] :-      % vMax -> v
```

```
M>>cat === vMax,    % laughed
D1>>cat === v,
D1>>subcat === null.
/*
M>>sem === D1>>sem,
M>>indsb === D1>>indsb,

M>>type === D1>>type.
*/


%genlex01_4.pl
% bruges sammen med gensyn01_4.pl


%%%%%%%%%%%%%%%%%%
% DETERMINATIVES %
%%%%%%%%%%%%%%%%%%

lex(a, L) :-
L>>cat === d,
L>>sem === exists,

L>>type..in..in === e,
L>>type..in..out === t,
L>>type..out..in..in === e,
L>>type..out..in..out === t,
L>>type..out..out === t.

lex(every, L) :-
L>>cat === d,
L>>sem === all,

L>>type..in..in === e,
L>>type..in..out === t,
L>>type..out..in..in === e,
L>>type..out..in..out === t,
L>>type..out..out === t.

%%%%%%%%%%%%%%%%%%
% GENITIVE CLITIC %
%%%%%%%%%%%%%%%%%%

lex(s, L) :-
L>>cat === g,
L>>subcat..cat === nMax,

L>>sem..quant === the,
L>>sem..var === L>>ref,
L>>sem..restr === L>>restruc,
L>>sem..assn === L>>astruc,
L>>genRel === L>>relation,

L>>type..in..in..in === e,
L>>type..in..in..out === t,
L>>type..in..out === t,
L>>type..out..in..in === e,
L>>type..out..in..out..in === e,
L>>type..out..in..out..out === t,
```

```
L>>type..out..out..in..in === e,
L>>type..out..out..in..out === t,
L>>type..out..out..out === t.

%%%%%%%%%
% NOUNS %
%%%%%%%%%

lex(boy,L) :-
L>>cat === n,
L>>sem..'arg-s'..pred === 'boy''',
L>>sem..'arg-s'..arg1 === L>>ref,

L>>'arg-s'..type..in === e,       % type e -> t
L>>'arg-s'..type..out === t.


lex(friend,L) :-                          % RELATIONELT
L>>cat === n,

L>>sem..'arg-s'..pred === 'friend_of''', % ref er vennen (friend) og ref2 er den
ref er ven med (eg Bo i 'Bo's friend').
                                          % Dvs: i 'Ann is a friend of Bo' er Ann
ref og Bo er ref2.
                                          % Dette skal også gælde ved genitiven
'Ann is Bo's friend', altså at Bo er ref2 her!
L>>sem..'arg-s'..arg1 === L>>ref,
L>>sem..'arg-s'..arg2 === L>>ref2,

L>>'arg-s'..type..in === e,         % type e -> (e -> t)
L>>'arg-s'..type..out..in === e,
L>>'arg-s'..type..out..out === t.

lex(snowman,L) :-
L>>cat === n,

L>>sem..'arg-s'..pred === 'snowman''',
L>>sem..'arg-s'..arg1 === L>>ref,

L>>'arg-s'..type..in === e, %type e -> t
L>>'arg-s'..type..out === t,

L>>sem..'qua-s'..agt..pred === 'create''',
L>>sem..'qua-s'..agt..arg1 === L>>indsb,
L>>sem..'qua-s'..agt..arg2 === L>>ref,

L>>'qua-s'..type..in === e,       % type e -> (e -> t)
L>>'qua-s'..type..out..in === e,
L>>'qua-s'..type..out..out === t.

lex(snowmen,L) :-
L>>cat === n,

L>>sem..'arg-s'..pred === 'snowman''',
L>>sem..'arg-s'..arg1 === L>>ref,

L>>'arg-s'..type..in === e, %type e -> t
L>>'arg-s'..type..out === t,
```

```
L>>sem..'qua-s'..agt..pred === 'create''',
L>>sem..'qua-s'..agt..arg1 === L>>indsb,
L>>sem..'qua-s'..agt..arg2 === L>>ref,

L>>'qua-s'..type..in === e,       % type e -> (e -> t)
L>>'qua-s'..type..out..in === e,
L>>'qua-s'..type..out..out === t.

%%%%%%%%%%%%%%%%%
% QUANTIFIERS    %
%%%%%%%%%%%%%%%%%

lex(two, L) :-
L>>cat === q,
L>>sem === 2.
/*
L>>type..in..in === e,
L>>type..in..out === t,
L>>type..out..in..in === e,
L>>type..out..in..out === t,
L>>type..out..out === t.
*/

%%%%%%%%
% VERBS %
%%%%%%%%

lex(laughed,L) :-
L>>cat === v,
L>>sem..pred === 'laugh''',
L>>sem..arg1 === L>>indsb,

L>>type..in === e,       % type e -> t
L>>type..out === t.
```