Abstract

This paper shows how Walenty, a valency dictionary of Polish, was automatically converted in order to be used with an XLE/LFG grammar of Polish, discussing issues such as the grammatical function assignment under unlike category coordination and imposing constraints for lexicalised dependents.

1 Introduction

This paper discusses how Walenty, an innovative valency dictionary of Polish, was used in an LFG grammar of Polish implemented in XLE (Patejuk & Przepiórkowski, 2012b). It begins with introducing the distinctive features of Walenty that make it attractive from the perspective of use in an LFG grammar (§2), then it proceeds to presenting the procedure for interpreting and converting valency specifications from Walenty to the LFG formalism (§3). It describes the procedure of assigning the grammatical function to dependents, taking unlike category coordination into account, and it shows how relevant constraints are formalised using available LFG mechanisms, covering issues such as structural case assignment in Polish, the handling of passive voice and the treatment of optional arguments. Finally, it offers a detailed formalisation of imposing constraints on lexicalised dependents which is capable of accounting for embedding of such lexicalised specifications.

2 Walenty: an innovative valency dictionary of Polish

Walenty (Przepiórkowski et al., 2014b) is currently the largest and most precise valency dictionary of Polish – in October 2016 it contained 85,210 schemata for 16,195 lemmata. Unlike many other dictionaries, it contains not only schemata for verbs, but also for nouns, adjectives and adverbs. For reasons of space, this paper focuses on verbal schemata exclusively: there are over 65,400 schemata for 12,028 lemmata, which gives 5.4 schemata per lemma on average.

Walenty is available on an open source licence (CC BY-SA 4.0) in a variety of formats: plain text, XML and PDF. While XML is used as the input for conversion, schemata are presented below in plain text format in order to save space.

2.1 Grammatical function labels

Walenty explicitly identifies the subject position (`subj`) – understood as the argument that drives verbal agreement, regardless of its category (so it takes into account non-canonical subjects) – and the object (`obj`) – defined as the argument which can become the subject under passive voice, also regardless of its category (and case marking, if the passivisable argument is nominal).

[†]This research is partially supported by the Polish Ministry of Science and Higher Education within the CLARIN ERIC programme 2016–2018 (http://clarin.eu/). The author is grateful to anonymous reviewers for their valuable comments which made it possible to improve this paper.

The verb MANIPULOWAĆ ‘manipulate’ is an example of a predicate taking a nominal object marked for the instrumental case (`inst`), as shown in (2), while (3) demonstrates that this argument can become the subject under passive voice. The schema for MANIPULOWAĆ is given in (1) – it contains a subject marked for the structural case (`str`, see §2.3) and an instrumental object:

(1) `subj{np(str)} + obj{np(inst)}`

(2) Manipulowała mną i swoim późniejszym mężczyzną.
manipulated I.INST and SELF.INST later.INST man.INST
‘She manipulated me and her later man.’ (http://nkjp.pl)

(3) Młodzi ludzie byli manipulowani przez starsze osoby.
young.NOM people.NOM were manipulated.NOM by elder persons
‘Young people were manipulated by elder people.’ (http://nkjp.pl)

Since Walenty does not distinguish grammatical functions other than subject and object as defined above,[1] other positions are not labelled with any grammatical function and it must[2] be assigned during conversion – this is discussed in §3.1.

2.2 Syntactic positions as sets

In Walenty a syntactic schema is a list of positions (separated by +) modelled as sets of categories that can realise a given position. The set contents, enclosed in curly brackets (`{...}`), with particular elements separated by semi-colons (`;`), are specified according to the coordination test of Szupryczyńska 1996: if two or more categories can be coordinated within one position, then it is a multi-element set. For instance, in (4) the subject of BAWIĆ ‘amuse’ consists of a nominative noun phrase (*ta gra*) coordinated with a clause (*że tylu ludzi [...] dało się nabrać*), so the corresponding `subj` position in the schema in (5) is a two-element set containing the `np(str)` and `cp(że)` categories. When such coordination is not possible, a singleton set is used – the `subj` in (7), the schema for AKLIMATYZOWAĆ SIĘ ‘acclimatise’, contains only `np(str)`: a nominal marked for structural case (see §2.3). Note that the second argument of (5), `np(str)`, is a singleton set; since this position is not marked as an `obj`, it is assumed that it cannot passivise.

(4) Trochę bawiła mnie ta gra i że tylu ludzi [...]
a little amused me.ACC this.NOM game.NOM and that so many people
dało się nabrać
let REFL take in
‘This sham and (the fact) that so many people let themselves be cheated amused me a little.’ (http://nkjp.pl)

(5) `subj{np(str);cp(że)} + {np(str)}`

(6) `subj{np(str)} + {np(str)}`
`subj{cp(że)} + {np(str)}`

[1] See Patejuk & Przepiórkowski 2016 for arguments supporting such a solution.

[2] Unless an alternative approach to grammatical functions is adopted: see Patejuk & Przepiórkowski 2016 and Przepiórkowski 2016.

(7) `subj{np(str)} + {xp(locat)}`

Modelling syntactic positions as sets is innovative in that it explicitly accounts for the coordination of unlike categories. Other valency dictionaries would use separate valency schemata for different categories, as in (6), one with a nominal subject (`np(str)`) and another one with a clausal subject (`cp(że)`), which can be interpreted in two ways. If it is an `XOR` (exclusive `OR`) specification of the subject (either a nominal phrase or a clause), the possibility of having a coordinated unlike category subject such as in (4) is ruled out. If is is interpreted as an `OR` specification, it allows for such coordination at the cost of overgenerating (allowing such coordination when it is not possible). Such problems can be avoided by adopting the solution proposed in Walenty, where syntactic positions are modelled as sets which correspond to an `OR` specification, as in (5), which means that the given position can be filled by any single set element (only nominal or only clausal) or by any coordination of these elements, which accounts for unlike category coordination. If the position can be filled in more than one way but the relevant elements cannot be coordinated, the `XOR` specification is obtained by creating separate schemata with singleton sets corresponding to the relevant argument, as in (6).

Note that Walenty uses container categories such as `xp(locat)` in (7) – though it is a singleton set in the schema, it is equivalent to a multi-element set containing all the categories listed in its corresponding list of realisations (see §2.5).

2.3 Arguments marked for structural case

Walenty provides an explicit account of structural case in Polish. Unlike lexical case, which is stable in the sense that it is independent of the syntactic context, structural case is understood here as a case which may take different values depending on the syntactic environment – such arguments have the `str` value of case. The information supplied by the valency dictionary is to be processed by the grammar so as to assign an appropriate case in the given context.

When a subject is marked for structural case, its case marking may be realised in three ways.[3] The first possibility is the nominative case, the most prototypical value – it is appropriate for subjects of finite verb forms which are not non-agreeing numerals. The second value is the accusative case – it is possible when the subject of a finite verb form is a non-agreeing numeral. Finally, the third possible value is the genitive case – this is the case with the subject of gerunds.

When an object (passivisable or not) bears structural case, there are two possible values: accusative or genitive, depending on the availability of sentential negation and part of speech of the head assigning case. Gerunds require the genitive case from their structural objects regardless of negation. With other verbal forms, genitive is required when the verb assigning structural case is in the scope of sentential negation; this phenomenon is known as genitive of negation (GoN). If negation is local, GoN is obligatory, while with non-local negation (present higher in the verb

[3]Note that this does not apply to the subject of adjectival participles (determined by agreement) and infinitives (determined by control).

chain), GoN is optional (Patejuk & Przepiórkowski, 2014b). As a result, accusative is required as structural case when negation is not present at all and it is possible when it is non-local to the predicate assigning structural case.[4]

2.4 Control, raising, predicative elements

Walenty explicitly accounts for raising and control by using `controller` and `controllee` labels to establish relations between respective arguments. In (8), the schema for the verb BAĆ SIĘ 'fear, be afraid', the subject position (`subj`) containing a nominal marked for structural case (`np(str)`) is labelled as the `controller`, while the set containing the infinitival complement (`infp(_)`) is labelled as `controllee` – such notation expresses subject control, whereby the subject of BAĆ SIĘ controls the subject of the infinitive.[5]

(8) `subj,controller{np(str)} + controllee{infp(_)}`

These labels are also used to mark control relations with predicative arguments – the argument that controls the predicative element is marked as `controller`, while the predicative element is marked as `controllee`. Apart from representing what the predicative element refers to, such information may be used for ensuring proper agreement where it is applicable – as in (9), the schema for the verb UCHODZIĆ 'pass (as)', where the predicative adjective inside the prepositional phrase (`prepadjp(za,acc)`) agrees in number and gender with the subject:

(9)
```
subj,controller{np(str)}
+ controllee{prepadjp(za,acc)}
```

2.5 Semantically defined arguments

Walenty introduces a class of `xp` arguments – defined by their semantics rather than category: these include ablative, adlative, locative (see (7)), etc., arguments. For each type of `xp`, there is a defined list of its realisations (see (10),[6] with translations on the right), which results in economic, readable and coherent schemata.

(10)
```
xp(locat)-->                    xp(locat)-->
    advp(locat)                     advp(locat)
    [...]                           [...]
    cp(int[gdzie])                  cp(int[where])
    [...]                           [...]
    prepnp(koło,gen)                prepnp(near,gen)
    prepnp(między,inst)             prepnp(between,inst)
    prepnp(nad,inst)                prepnp(above,inst)
    prepnp(na,loc)                  prepnp(on,loc)
    [...]                           [...]
```

While using a plain `xp` means that all its realisations are possible with a given schema (so the given `xp` corresponds to a set containing all its realisations), sometimes it is the case that a given predicate does not allow all the realisations, though

[4] Some predicates in Polish allow partitive objects – these are covered in Walenty and treated as a variant of structural case, where genitive is additionally allowed in partitive use.

[5] It is assumed that in verb control it is always the subject of the `controlee` that is controlled.

[6] The `int` parameter in `cp(int[gdzie])` stands for interrogative – it is an interrogative clause where the interrogative item is GDZIE 'where'.

the realisations it subcategorises for have the required semantics. If the selected realisations were listed as elements of the set corresponding to the relevant argument position without stating that it is an `xp` phrase with specific semantics, the semantic information would be lost, which would be an unwelcome result (as a realisation of `xp`, `prepnp` is treated as semantic, while outside of `xp` it is non-semantic). On the other hand, classifying such an argument as a "plain" `xp` would allow all its realisations, which is also undesired as it leads to overgeneration. Walenty was designed so as to provide maximal precision of the description of valency requirements, so it offers a subtype mechanism which makes it possible to restrict the range of realisations of a given phrase (here, an `xp` phrase) to those that are applicable in the given schema using a list. For example, the schema for KIEŁKOWAĆ 'sprout' in (11) contains an `xp(adl)` phrase whose realisation list is restricted to two prepositional phrases, headed by SPOD 'from underneath' and Z 'from':

(11)
```
subj{np(str)}
+ {xp(abl[prepnp(spod,gen);prepnp(z,gen)])}
```

Elements of the subtype list, enclosed in square brackets (`[...]`), are separated by semicolons (`;`) since they may be coordinated (as in `xp` without subtypes).

The subtype mechanism is also used with clauses (`cp`): while `cp(int)` is an interrogative phrase with any interrogative element defined on the relevant list, this element may be specified using the subtype list, as in (10) (cf. fn. 6).

2.6 Lexicalised dependents

Last but not least, Walenty is one of the few valency dictionaries that include a rich phraseological component (Przepiórkowski et al., 2014a) – it explicitly specifies lexicalised arguments and constraints imposed on them, with the possibility of embedding such constraints arbitrarily deep, as in (13), the schema for the verb WITAĆ 'welcome' used in (12):

(12) Oni witali ją z (szeroko) *(otwartymi) ramionami.
they.NOM welcomed she.ACC with widely open.INST.PL arm.INST.PL
'They welcomed her with (widely) open arms.' (== very warmly)

(13)
```
subj{np(str)} + obj{np(str)} + {xp(mod);
lex(prepnp(z,inst),pl,XOR('ramię','ręka'),ratr1(
  {lex(adjp(agr),agr,agr,pos,'otwarty',atr1(
    {lex(advp(mod),pos,'szeroko',natr)}))}))}
```

There are three arguments in (13), out of which the first two, subject and object, are not lexicalised. The last one is a two-element set containing `xp(mod)` and a lexicalised (`lex`) prepositional nominal phrase (`prepnp`) with the preposition Z 'with' which requires instrumental case (`inst`) from the nominal which must in turn be specified for plural number (`pl`) and must be a form of either RAMIĘ 'arm' or RĘKA 'hand' (`XOR` specification). This lexicalised nominal must be modified by exactly one dependent (`ratr1`), an embedded `lex` specification follows: an agreeing adjectival phrase (`adjp`) headed by OTWARTY 'open', which may in turn be optionally (`atr1`) modified by a lexicalised adverbial phrase (`advp`) headed by SZEROKO 'widely', which must not be modified (`natr`).

3 Interpreting Walenty and converting it to LFG

Since Walenty uses its own formalism, it is not tied by the constraints of any particular grammar formalism and it can be used with any grammar or grammar engineering platform, provided that the grammar writer is able to interpret the specifications provided in Walenty and convert them to the relevant formalism. This section shows how this can be done for LFG on the basis of selected phenomena.

The conversion is done automatically using a Python script which ensures consistency and coherence. While schemata from Walenty are presented in plain text format, the script relies on the XML format of Walenty.

The general idea of converting a valency schema to LFG constraints is very simple: each argument must be assigned a grammatical function and then appropriate constraints relevant to this argument must be imposed.

3.1 Selecting the grammatical function

Since the grammatical function must be chosen to apply relevant constraints, let us start with the procedure of selecting the grammatical function. As mentioned in §2.1, only two grammatical functions are specified in Walenty: the subject (SUBJ) and the passivisable object (OBJ). The remaining grammatical functions are not specified in Walenty, so they are assigned using the following mapping:

- OBJ$_\theta$: thematic/secondary object – nominal, it does not passivise,
- OBL (oblique): non-semantic prepositional phrase,[7]
- OBL$_\theta$ (thematic oblique): semantic prepositional phrase,
- COMP: closed clausal complement,
- XCOMP: open infinitival complement,
- XCOMP-PRED (predicative complement): open predicative nominal or adjective (possibly embedded in a prepositional phrase).[8]

The specification outlined above works perfectly as long as the relevant argument position contains only one realisation (it is a singleton set in Walenty). If this is not the case and unlike category coordination is possible, as in (14), where a clause is coordinated with a prepositional phrase (see the last position in (15), the schema for the verb PYTAĆ ‘ask’), the choice of the grammatical function becomes problematic because different categorial realisations of the relevant argument position seem to correspond to different grammatical functions.

(14) Pytali, [jakie będą pieniądze] oraz [o to, czy zmienią się polskie
asked what will be money and about this PART change REFL Polish
szkoły].
schools
‘They asked what money will be there and whether Polish schools will change.’ (http://nkjp.pl)

[7] If there is more than one such phrase, a numeric index is appended, yielding OBL2, OBL3, etc.
[8] As an alternative, the closed PREDLINK grammatical function could be used.

(15) `subj{np(str)} + obj{np(gen)}`
`+ {prepnp(o,acc); cp(int); prepncp(o,acc,int)}`

Typically, a coordinated phrase corresponds to one grammatical function in f-structure, so a common grammatical function should be chosen. Since, according to the mapping provided above, a prepositional phrase (`prepnp`, `prepncp`) should be assigned the OBL grammatical function and an interrogative clausal complement (`cp(int)`) – the COMP grammatical function, which of these two grammatical functions should be assigned to their coordination in (14)? An analogous problem has been discussed in LFG literature in the context of OBJ as a candidate grammatical function under coordination: Dalrymple & Lødrup 2000 suggest COMP should be treated as an elsewhere grammatical function, used when only the clausal complement is possible and it cannot be coordinated with a different category. The current conversion of Walenty is inspired by this solution – it uses the ranking of grammatical functions defined in (16) to choose the common grammatical function from the set of candidates: the conversion script assigns each realisation of the relevant argument position (each element of the set) the corresponding ranking and then the highest ranked grammatical function candidate is chosen.

(16)

#	GF
4	OBL-<SEM: ABL, ADL, DUR, INSTR, LOCAT, MOD, PERL, TEMP. . . >
3	OBL
2	OBJ-<CASE: DAT, GEN, INST, STR>
1	COMP, XCOMP

According to the ranking in (16), if an argument position can be realised as a non-semantic prepositional phrase (OBL) or as a clause (COMP), as in (15), it should be assigned the OBL grammatical function. The XCOMP and COMP are the lowest ranked grammatical functions: they are only selected when the clause or infinitive are the only realisations in the set corresponding to the relevant argument position.

3.2 Imposing constraints

Once the grammatical function corresponding to a given syntactic position (the entire set) has been chosen, appropriate constraints are imposed for each realisation of the relevant syntactic position defined in the schema (each element of the set).

The method of formalising constraints corresponding to a given argument position depends on one crucial factor – whether the given position involves unlike category coordination or not. When such coordination is not involved, it is sufficient to use plain constraints such as in (17) and (18). However, when unlike category coordination is allowed, which requires that the argument GF must either have the attribute ATTR1 with V1 as its value, or the attribute ATTR2 whose value is V2, it cannot be formalised using the disjunction of two plain constraints such as in (19), because, instead of yielding the logical `OR` specification, the result will be the undesired `XOR` specification.

(17) $(\uparrow \text{GF ATTR1}) =_c \text{V1}$

(18) $(\uparrow \text{GF ATTR2}) =_c \text{V2}$

(19) $(\uparrow \text{GF ATTR1}) =_c \text{V1} \ \vee \ (\uparrow \text{GF ATTR2}) =_c \text{V2}$

As explained in Przepiórkowski & Patejuk 2012, when a plain disjunctive constraint is used, it is evaluated once (one disjunct is chosen) and applied to all conjuncts, as formalised in (20a). By constrast, the interpretation which is needed to handle unlike category coordination is the one formalised in (20b) – it evaluates the relevant statement for each conjunct separately, so it is possible that different conjuncts satisfy different constraints.

(20) a. $\forall x \in (\uparrow \text{GF}) A(x) \ \vee \ \forall x \in (\uparrow \text{GF}) B(x)$ (actual)
b. $\forall x \in (\uparrow \text{GF}) [A(x) \ \vee \ B(x)]$ (intended)

The solution to this problem described in Przepiórkowski & Patejuk 2012 relies on the use of off-path constraints in order to obtain the effect shown in (20b). In short: constraints to be imposed on a given argument are converted to their off-path equivalent and they are attached to the PRED attribute of the relevant argument – this attribute is distributive by definition, which ensures that the disjunctive off-path constraint will be distributed to each conjunct and evaluated separately.

There is a crucial difference in the formalisation of off-path constraints between LFG theory and XLE implementation: unlike in recent theoretical LFG works (including recent versions of Dalrymple 2001 and Bresnan 2001), off-path constraints are non-constructive in XLE,[9] which means that they can only be constraining or existential, but they cannot be defining – they cannot introduce new attribute-value pairs to the f-structure. As a result, constraints placed on certain attributes must be formalised as constraining equations – rather than defining ones – regardless of whether the constraint is off-path or plain, for the sake of consistency.

3.2.1 Structural case assignment

As mentioned in §2.3, Walenty provides information about the requirement of structural case and the grammatical function of the relevant argument, which is processed by the grammar, taking the syntactic context into account, in order to set the appropriate values of case. As discussed in Patejuk & Przepiórkowski 2014b for verbal heads in Polish, the structural object is marked for accusative case in the absence of negation and genitive case if negation is present – the proposed formalisation (see (21)) uses plain constraints, so it is not compatible with unlike category coordination. A formalisation of structural case assignment that does take this into consideration and uses off-path constraints is provided in Patejuk & Przepiórkowski 2014a (compare (22)):[10]

(21) $\text{STRCASE(GF)} \equiv \quad [[\neg((\text{XCOMP}^* \uparrow) \text{ NEG}) \ \wedge \ (\uparrow \text{GF CASE}) =_c \text{ ACC}] \ \vee$
$[((\text{XCOMP}^* \uparrow) \text{ NEG}) =_c + \ \wedge$
$[[(\uparrow \text{NEG}) =_c + \ \wedge \ (\uparrow \text{GF CASE}) =_c \text{ GEN}] \ \vee$
$[\neg(\uparrow \text{NEG}) \ \wedge \ (\uparrow \text{GF CASE}) \in_c \{\text{ACC, GEN}\}]]]]$

[9] See the relevant part of the XLE documentation: http://www2.parc.com/isl/groups/nltt/xle/doc/notations.html#N4.1.5b

[10] See the corresponding papers for a detailed discussion of relevant constraints.

(22) $(\uparrow \text{GF} \quad \text{PRED} \quad)$
$[[\neg((\text{XCOMP}^* \text{ GF} \leftarrow) \text{ NEG}) \wedge (\leftarrow \text{CASE}) =_c \text{ACC}] \vee$
$[((\text{XCOMP}^* \text{ GF} \leftarrow) \text{ NEG}) =_c + \wedge$
$[[((\text{GF} \leftarrow) \text{ NEG}) =_c + \wedge (\leftarrow \text{CASE}) =_c \text{GEN}] \vee$
$[\neg((\text{GF} \leftarrow) \text{ NEG}) \wedge (\leftarrow \text{CASE}) \in_c \{\text{ACC, GEN}\}]]]]$

Such constraints are placed in the lexical entry of the relevant verb – though it is less economic than placing their equivalents in c-structure rules, it allows for an appropriate treatment of implicit arguments,[11] unlike category coordination (whereby only some conjuncts are marked for structural case) and dependent sharing (whereby the shared dependent is assigned structural case by only some of the coordinated verbs). While the constraints are complex, they may be assigned to a template and then short template calls may be used in particular lexical entries, which is considerably more economic.

3.2.2 Complex constraints for clausal phrases

Walenty has two complementiser types (`żeby2` and `gdy`) which are realised as different complementisers (ŻE or ŻEBY for `żeby2`; GDY or GDYBY for `gdy`) depending on the syntactic context – this phenomenon may be thought of as similar to structural case assignment. If this analogy is accepted, the remaining complementiser types (such as `żeby`, `że`, `jeśli` and many more) may be considered lexical since they have the same form (ŻEBY, ŻE and JEŚLI, respectively) regardless of the syntactic environment, which includes factors such as the availability of negation (discussed below for `żeby2`) and mood (`gdy` is sensitive to conditional mood).

The clausal phrase `cp(żeby2)` is different from `cp(żeby)` since the former can be realised in two ways: always as ŻE and as ŻEBY only in scope of sentential negation.[12] Consider the following examples, which illustrate the schema for the verb WYOBRAZIĆ 'imagine' provided in (25):

(23) Ja *(**nie**) mogę sobie wyobrazić, **żeby** ktoś mógł zrobić coś takiego.
I NEG can SELF.DAT imagine that sb could do.INF sth such
'I cannot imagine that somebody could have done something like this.'
(http://nkjp.pl)

(24) Ja (**nie**) mogę sobie wyobrazić, **że** ktoś mógł zrobić coś takiego.
I NEG can SELF.DAT imagine that sb could do.INF sth such
'I can (not) imagine that somebody could have done something like this.'

(25)
```
subj{np(str)} + {np(str);cp(int);cp(żeby2);
ncp(str,int);ncp(str,żeby2)}
+ {lex(np(dat),_,'siebie',natr)}
```

[11] Assigning case to implicit arguments using c-structure rules would require placing such constraints on the verb directly – which has the same effect as placing them in the lexical entry.

[12] ŻEBY may also be used as the realisation of `cp(żeby2)` in generally negative contexts such as in (i), where the verb WYOBRAZIĆ 'imagine' takes `cp(żeby2)` as one of its arguments:

(i) **Z trudem** mogę sobie wyobrazić, **żeby**...
with difficulty can SELF.DAT imagine that
'It is only with difficulty that I can imagine that...'

The plain constraint corresponding to `cp(żeby2)`, one of the realisations of the second argument of (25), is provided in (26) – it states that the complementiser ŻEBY is only possible when negation is present, as in (23), while ŻE is possible at all times (there are no constraints on the value of NEG), as in (24):

(26) [(↑ NEG)=$_c$ + ∧ (↑ GF COMP-FORM)=$_c$ ŻEBY] ∨
(↑ GF COMP-FORM)=$_c$ ŻE

3.2.3 Passive voice

Another issue that is worth discussing is the method of handling passive voice. LFG grammars typically use a lexical rule, but an alternative method is used when converting Walenty – passive versions of schemata are created using the script.

In XLE the passive lexical rule manipulates the assignment of grammatical functions using string substitution: OBJ → SUBJ – the active object becomes the passive subject; SUBJ → OBL-AG/NULL – the active subject becomes the passive oblique agent or it is dropped. Such a rule is capable of changing control relations: (↑ OBJ)=(↑ XCOMP SUBJ) is the control equation used the active verb TEACH, whereby the object of TEACH is at the same time the subject of the infinitival complement of TEACH, while in the passive it is (↑ SUBJ)=(↑ XCOMP SUBJ) – because the active OBJ becomes the passive SUBJ.

Unfortunately, when applied to constraints which depend on the grammatical function, such a lexical rule has undesired effects. When the active verb takes an object marked for structural case (accusative or genitive, see §3.2.1), the case constraint will be imposed on the passive subject (simplifying, typically nominative), which results in ungrammaticality. It is, however, easy to introduce such changes in the process of conversion: when the passive version of the relevant schema is created, the script first changes the assignment of grammatical functions and then imposes the constraints, which results in changing all the appropriate constraints.

Furthermore, this method makes it possible to introduce more complex additional constraints where it is appropriate: for instance, when the active subject may only be a clause, it could not be the complement of the OBL-AG *by*-phrase because the preposition requires a certain value of case from its complement, which is normally a nominal. In this situation, a correlative pronoun might be added in the passive, resulting in a well-formed *by*-phrase.

3.2.4 Argument reduction

The next issue that must be considered when converting Walenty is the issue of argument reduction: by design, Walenty only provides maximal schemata (listing all possible arguments), but at the same time it assumes that all arguments are optional – in Polish most arguments may be dropped in the sense that they are not expressed. This is illustrated below: the schema for the verb DOWODZIĆ ‘command’ provided in (27) contains two arguments – a subject and a passivisable object. (28) shows that both arguments can be realised lexically, but they may also be omitted.

(27) `subj{np(str)} + obj{np(inst)}`

(28) (Mój ojciec) dowodził (siłami republikańskimi).
my.NOM father.NOM commanded forces.INST republican.INST
'My father commanded republican forces.' (http://nkjp.pl)

When performing the conversion, one must decide how to interpret this phenomenon in an implemented grammar. One way is to assume that the argument is present but it is not realised lexically – in this way the argument is represented syntactically, the relevant grammatical function attribute is present, but its value is 'PRO' – see the f-structure in (29), which corresponds to (28) with the object dropped. The alternative approach is to assume that the relevant argument is removed, that it is not present in the f-structure of the verb – this solution involves the creation of reduced frames, which have fewer arguments than the maximal frame, as in (30), where the object is removed from the list in PRED attribute.

(29)
$$\begin{bmatrix} \text{PRED} & \text{'COMMAND}\langle \boxed{1}, \boxed{2} \rangle\text{'} \\ \text{SUBJ} & \boxed{1} \begin{bmatrix} \text{PRED} & \text{'FATHER'} \\ \text{CASE} & \text{NOM} \\ \text{NUM} & \text{SG} \\ \text{PERS} & 3 \\ \text{ADJ} & \left\{ \begin{bmatrix} \text{PRED} & \text{'MY'} \end{bmatrix} \right\} \end{bmatrix} \\ \text{OBJ} & \boxed{2} \begin{bmatrix} \text{PRED} & \text{'PRO'} \\ \text{CASE} & \text{INST} \end{bmatrix} \end{bmatrix}$$

(30)
$$\begin{bmatrix} \text{PRED} & \text{'COMMAND}\langle \boxed{1} \rangle\text{'} \\ \text{SUBJ} & \boxed{1} \begin{bmatrix} \text{PRED} & \text{'FATHER'} \\ \text{CASE} & \text{NOM} \\ \text{NUM} & \text{SG} \\ \text{PERS} & 3 \\ \text{ADJ} & \left\{ \begin{bmatrix} \text{PRED} & \text{'MY'} \end{bmatrix} \right\} \end{bmatrix} \end{bmatrix}$$

The proposed method of interpreting Walenty uses a hybrid solution – it divides arguments into two classes: obligatory (must be present in syntactic representation) and optional (can be removed from syntactic representation).

First, if the absence of an argument changes the meaning of the predicate – as in the case of lexicalised arguments and the SIĘ marker, which can be reflexive, reciprocal or inherent (in the last case it carries no semantic information, but it is required syntactically as in BAĆ SIĘ in (8), which means 'to fear', not 'to fear oneself') – then the argument is assumed to be obligatory and it must be lexical (overtly expressed).

The second diagnostic is whether there is syntactic evidence that the relevant argument is syntactically active even though it has no surface realisation. There is evidence which supports implicit subjects and implicit controllers. As shown below using subscript indices, in Polish it is the subject which binds[13] the SIEBIE anaphor (see (31)) and controls participles (see (32)). If the subject were removed from the schema, sentences with no lexical subject could not be parsed (because the subject position would have no value, resulting in incompleteness) and would be expected to be ungrammatical, counter to fact:

[13] With the exception of reciprocal predicates – in (i) *sobie* is bound by the object, *sąsiadów*:

(i) Przedstawił sobie$_s$ (nawzajem) sąsiadów$_s$.
introduced SELF reciprocally neighbours
'He introduced the neighbours to one another.'

(31) (Antek$_a$) opowiedział Erykowi$_e$ o sobie$_{a/*e}$.
Antek.NOM told Eryk.DAT about SELF
'(Antek) told Eryk about himself.'

(32) Wychodząc$_{a/*e}$, (Antek$_a$) pocieszał Eryka$_e$.
leaving Antek.NOM comforted Eryk.ACC
'Leaving, (Antek) was comforting Eryk.'

The second group of arguments which may be implicit are controllers of infinitives and predicative complements – the reason for having implicit arguments is the same as for controlling participles: the subject of the controlled element is structure-shared with the controller, so the controller must be present in the f-structure. In this case, however, the controller may be different than the subject, see the examples below:

(33) Dowódca kazał (nam wszystkim) uciekać.
commander.NOM ordered us.DAT all.DAT escape.INF
'The commanding officer ordered us all to run away'. (http://nkjp.pl)

(34) Antek zawsze uczyni (Eryka) szczęśliwym.
Antek always make Eryk happy
'Antek will always make (Eryk) happy.'

According to the schema in (35), the controller of the infinitival complement of the verb KAZAĆ 'order' in (33) is the dative nominal. By contrast, the schema in (36) specifies the passivisable object marked for structural case as the controller of the predicative complement of the verb UCZYNIĆ 'make' in (34).

(35) `subj{np(str)} + controller{np(dat)}`
`+ controllee{cp(żeby); infp(_)}`

(36) `subj{np(str)} + obj,controller{np(str)}`
`+ controllee{adjp(inst)}`

In (33)–(34) controllers may have no lexical realisation, they are nevertheless required by syntax (controlled phrases must have controllers), so they are represented in the f-structure representation as implicit arguments ('PRO' is the value of their PRED attribute) – the f-structure in (37) corresponds to (33), while (38)[14] provides a representation of (34) without the lexical object:

(37)
$$\begin{bmatrix} \text{PRED} & \text{'ORDER}\langle\boxed{1},\boxed{2},\boxed{3}\rangle\text{'} \\ \text{SUBJ} & \boxed{1}\begin{bmatrix}\text{PRED} & \text{'COMMANDER'}\\ \text{CASE} & \text{NOM}\end{bmatrix} \\ \text{OBJ}_\theta & \boxed{2}\begin{bmatrix}\text{PRED} & \text{'PRO'}\\ \text{CASE} & \text{DAT}\end{bmatrix} \\ \text{XCOMP} & \boxed{3}\begin{bmatrix}\text{PRED} & \text{'RUN\_AWAY}\langle\boxed{2}\rangle\text{'}\\ \text{SUBJ} & \boxed{2}\end{bmatrix} \end{bmatrix}$$

(38)
$$\begin{bmatrix} \text{PRED} & \text{'MAKE}\langle\boxed{1},\boxed{3}\rangle\boxed{2}\text{'} \\ \text{SUBJ} & \boxed{1}\begin{bmatrix}\text{PRED} & \text{'ANTEK'}\\ \text{CASE} & \text{NOM}\end{bmatrix} \\ \text{OBJ} & \boxed{2}\begin{bmatrix}\text{PRED} & \text{'PRO'}\\ \text{CASE} & \text{ACC}\end{bmatrix} \\ \text{XC-PRED} & \boxed{3}\begin{bmatrix}\text{PRED} & \text{'HAPPY}\langle\boxed{2}\rangle\text{'}\\ \text{SUBJ} & \boxed{2}\end{bmatrix} \end{bmatrix}$$

Such implicit arguments are introduced optionally (in brackets), so as not to block lexical realisations of the relevant argument – see (39):

(39) $((\uparrow \text{GF PRED}) = \text{'PRO'} \wedge (\uparrow \text{GF CASE}) = \text{CASE\_VALUE})$

[14] For typesetting reasons, XC-PRED is used in (38) instead of XCOMP-PRED.

(40) ((↑ GF PRED)= 'PRO' ∧ [(↑ GF CASE)= ACC ∨ (↑ GF CASE)= GEN])

The first conjunct in (39) introduces an implicit argument (PRO) as the value of GF, while the second one assigns case to this argument – in accordance with respective constraints from Walenty. When the implicit argument is marked for structural case, the constraint in (40) is used instead.[15] There is no need to introduce the implicit subject – this is done by the grammar rules (at the level of c-structure).

When none of the criteria presented above is satisfied, the relevant argument is assumed to be optional and it may be reduced – this is done by removing it from the PRED attribute and removing the respective constraints that apply to it. Removing arguments in such a way requires care because controllers must not be removed unless the corresponding controllee is removed. However, once the controllee is removed, the `controller` label is removed from the controller and then it can also be reduced (unless is it a subject – as explained above, it is assumed that subjects do not undergo reduction).

An alternative approach to argument reduction would be to introduce implicit PRO arguments for all arguments, but this would result in implicit clauses and prepositional phrases, which would introduce a lot of additional ambiguity – many predicates take both and a parse would be created for each such argument. Besides, there seems to be no syntactic evidence for introducing such implicit arguments.

3.2.5 Lexicalised dependents: formalisation of modification patterns[16]

Only one aspect of the formalisation of lexicalised dependents is discussed in this section, namely the constraints corresponding to the modification pattern defined in Walenty – these include:

- `natr`: no further modification,
- `atr[(...)]`: modification allowed (optional), it may be iterated,
- `atr1[(...)]`: at most one modifier allowed,
- `ratr[(...)]`: modification required (obligatory), it may be iterated,
- `ratr1[(...)]`: exactly one modifier required.

Apart from `natr` which precludes any modification,[17] the modification pattern symbol can be followed by a list of dependents (its optionality is expressed using square brackets: `[...]`) whose elements are separated by +, as in "top level" Walenty schemata. They may be non-lexicalised or lexicalised – in the latter case an embedded `lex` specification is used, it can be arbitrarily deep.

The **`natr`** modification pattern, forbidding any dependents, is converted as the negative constraint in (41) (plain) or in (42) (off-path),[18] where `PATH` is the f-

[15] The values of CASE introduced by (40) are constrained by equations discussed in §3.2.1.

[16] Since §3.2.5 and §3.2.6 are implementational, they use XLE notation. See http://www2.parc.com/isl/groups/nltt/xle/doc/notations.html#N0A for the complete notation mapping used by XLE.

[17] Though the word "modification" is used, the "modifier" is understood as any dependent: it may either be an argument or an adjunct – this is not specified in Walenty as it is assumed that this is restricted by the lexical entry of the lexicalised dependent.

[18] Off-path counterparts of subsequent plain statements may be provided without comments.

structure path leading to the lexicalised dependent, while GFALL is defined as in (43)[19] – as the disjunction of all grammatical functions used in the grammar.[20]

(41) `~(PATH GFALL)`
(42) `(PATH PRED: ~(<- GFALL))`
(43) `GFALL = {SUBJ|OBJ|OBL(-?*)|(X)COMP|(X)ADJUNCT|POSS}`

The **ratr** specification requires a dependent which may be constrained (an embedded argument list is provided then) or not. In the latter case it is assumed that it may be any dependent allowed by the particular head – the constraint in (44) uses the GFDEP variable, which is resolved to the disjunction of grammatical functions allowed by the given head.

(44) `(PATH GFDEP)` (45) `(PATH PRED: (<- GFDEP))`

On the other hand, when the dependent is given explicitly (as an element of the embedded list), the schematic constraint in (46) is used, where GFDEP is the grammatical function of the particular dependent, chosen according to its specification, while ATTR stands for the relevant attribute and v for its required value.

(46) `(PATH GFDEP ATTR)=c v`
(47) `(PATH PRED: (<- GFDEP ATTR)=c v)`

When there is more than one element on the list of possible modifiers, the following constraints are used: the first line in (48) is the disjunctive constraint where particular disjuncts contain existential equations requiring the grammatical functions which correspond to particular dependents on the list inside ratr. Its purpose is to satisfy this modification requirement by ensuring that at least one of the listed required dependents is present. The following lines contain disjunctive constraints for each of the dependents on the list (GFDEP1 for the first one, etc.) which ensure that either the dependent corresponding to the given grammatical function is present and it satisfies the relevant requirements (the positive first disjunct – it corresponds to (46)) or that it is not present (the negative second disjunct).

(48)
```
{(PATH GFDEP1) | (PATH GFDEP2) | ...}
{(PATH GFDEP1 ATTR)=c v | ~(PATH GFDEP1)}
{(PATH GFDEP2 ATTR)=c v | ~(PATH GFDEP2)}
...
```
(49)
```
(PATH PRED:
      {(<- GFDEP1) | (<- GFDEP2) | ...}
      {(<- GFDEP1 ATTR)=c v | ~(<- GFDEP1)}
      {(<- GFDEP2 ATTR)=c v | ~(<- GFDEP2)}
      ...)
```

Finally, it is necessary to block all dependents other than those specified in ratr – in (50) the GFDEP variable corresponds to a disjunction of all grammatical functions allowed in ratr, while GFALL corresponds to all grammatical functions possible with the given head.

(50) `~(PATH GFALL - GFDEP)`

[19] The actual version accepted by XLE does not use regular expressions.

[20] The expansion of GFALL could be narrowed down to grammatical functions possible with the given head, reducing the number of disjuncts.

(51) `(PATH PRED: ~(<- GFALL - GFDEP))`

The **`ratr1`** specification is a modified version of `ratr` – the difference is that the former limits the number of required dependents to exactly one. As with `ratr`, the `ratr1` can be constrained (using an embedded argument list) or not.

When `ratr1` is not constrained using a list, the constraints in (52) and (53) are used – note that these are a conjunction of `ratr` constraints: the first conjunct is the positive constraint shown in (44) or (45), while the second conjunct is the negative constraint in (50) or (51).

(52)
```
(PATH GFDEP)
~(PATH GFALL - GFDEP)
```

(53)
```
(PATH PRED:
(<- GFDEP)
~(<- GFALL - GFDEP))
```

If the given head can take more than one dependent, more complex constraints must be used: (54) and (55) are disjunctive constraints, where each disjunct corresponds to one category allowed by the given head – the first disjunct requires the `GFDEP1` grammatical function and blocks all grammatical functions other than `GFDEP1`, the second disjunct is analogous.

(54)
```
{
(PATH GFDEP1)
~(PATH GFALL - GFDEP1)
|
(PATH GFDEP2)
~(PATH GFALL - GFDEP2)
|
...
}
```

(55)
```
(PATH PRED:
{
(<- GFDEP1)
~(<- GFALL - GFDEP1)
|
(<- GFDEP2)
~(<- GFALL - GFDEP2)
|
...
})
```

However, when the dependent list is given in `ratr1`, the constraints on particular dependents are imposed as described for `ratr` – when there is only one element on the list, the constraints in (46) or (47) are used for imposing positive requirements for the given phrase and (50) or (51) are used for blocking all other dependents. The following constraints result:

(56)
```
(PATH GFDEP ATTR)=c v
~(PATH GFALL - GFDEP)
```

(57)
```
(PATH PRED:
(<- GFDEP ATTR)=c v
~(<- GFALL - GFDEP))
```

When the list of dependents contains more than one element, the following constraints are used, where, as in (54) and (55), each disjunct corresponds to one element on the list of dependents – the difference is that each disjunct constrains the relevant dependent appropriately:

(58)
```
{
(PATH GFDEP1 ATTR)=c v
~(PATH GFALL - GFDEP1)
|
(PATH GFDEP2 ATTR)=c v
~(PATH GFALL - GFDEP2)
|
...
}
```

(59)
```
(PATH PRED:
{
(<- GFDEP1 ATTR)=c v
~(<- GFALL - GFDEP1)
|
(<- GFDEP2 ATTR)=c v
~(<- GFALL - GFDEP2)
|
...
})
```

Since the **atr** specification expresses the optionality of a certain requirement, it either requires a certain dependent using the appropriate ratr specification or it blocks any dependents using the natr specification. For this reason, instead of rewriting all the constraints, only general schemata are provided here: (60) is appropriate for plain atr constraints, while (61) is its off-path counterpart – <ratr_constraint> is the placeholder for the relevant ratr constraint (atr, like ratr, may be followed by an embedded list specifying dependents or unspecified), which may be disjunctive or not, and <natr_constraint> is the placeholder for the natr constraint.

(60)
```
{
<ratr_constraint>
|
<natr_constraint>
}
```

(61)
```
(PATH PRED:
{
<ratr_constraint>
|
<natr_constraint>
})
```

To give an example, (62) is the plain version of constraints for atr with a specified list of dependents, where the list contains more than one dependent – the last disjunct is the negative constraint corresponding to the natr specification in (41), while the remaining disjuncts are taken from the corresponding ratr specification in (48). (63) is the off-path counterpart of (62) – its last disjunct corresponds to (42), while the remaining ones correspond to the off-path version of ratr specification given in (49).

(62)
```
{
{(PATH GFDEP1) | (PATH GFDEP2) | ...}
{(PATH GFDEP1 ATTR)=c v | ~(PATH GFDEP1)}
{(PATH GFDEP2 ATTR)=c v | ~(PATH GFDEP2)}
...
|
~(PATH GFALL)
}
```

(63)
```
(PATH PRED:
      {
      {(<- GFDEP1) | (<- GFDEP2) | ...}
      {(<- GFDEP1 ATTR)=c v | ~(<- GFDEP1)}
      {(<- GFDEP2 ATTR)=c v | ~(<- GFDEP2)}
      ...
      |
      ~(<- GFALL)
      )
```

Finally, the treatment of **atr1** is fully analogous to atr discussed above – since atr1 expresses that a certain optional dependent can occur only once, it is formalised as a disjunction of the appropriate ratr1 constraint and the natr constraint, as presented in the general schemata provided below:

(64)
```
{
<ratr1_constraint>
|
<natr_constraint>
}
```

(65)
```
(PATH PRED:
{
<ratr1_constraint>
|
<natr_constraint>
})
```

3.2.6 Lexicalised dependents: an example

Let us now consider an example on the basis of the schema for WITAĆ 'welcome' in (13) discussed in §2.6, repeated as (66) for the sake of convenience:

(66)
```
subj{np(str)} + obj{np(str)} + {xp(mod);
lex(prepnp(z,inst),pl,XOR('ramię','ręka'),ratr1(
  {lex(adjp(agr),agr,agr,pos,'otwarty',atr1(
    {lex(advp(mod),pos,'szeroko',natr)}))})}
```

It consists of 3 positions, of which the last one contains a non-lexicalised `xp(mod)` phrase which can be coordinated with a lexicalised (`lex`) `prepnp` phrase – according to the ranking in (16), the entire position is assigned the OBL-MOD grammatical function (see §3.1). Since the position involves coordination (the set contains two phrases, one of which, `xp(mod)`, is additionally a container category, see §2.5), off-path constraints must be used.

Note that constraints for lexicalised (`lex`) phrases consist of two parts: the non-lexicalised constraints appropriate for the given base category (the first parameter of `lex`) and lexicalised constraints – these two constraint types are marked using comments below (enclosed in quotes: `"..."`).

Let us discuss the constraints for the last position of (66) in detail, stepwise: first, only fragments of relevant constraints are presented (all partial constraints use the same off-path anchor: the `PRED` attribute of the `OBL-MOD` grammatical function) and placeholders such as `<constraints_for_...>` are used for the rest of the relevant constraint (discussed later as the next fragment). Finally, the entire constraint, consisting of fragments discussed earlier, is presented.

For the prepositional nominal phrase `prepnp(z,inst)`, base category constraints include the preposition form (assigned in `PRED` since it is a semantic[21] preposition) and case required from the nominal element (since the preposition is semantic, the nominal is analysed as its `OBJ`). Furthermore, the specification of lexicalised constraints such as number, lemma and modification pattern of the prepositional nominal phrase (`prepnp`) applies to its nominal component – it must be plural, its lemma can either be `ramię` or `ręka` (it is constrained using a two element list with the `XOR` operator)[22] and it requires exactly one modifier (`ratr1`) which is specified further as another lexicalised phrase, an adjectival phrase: `adjp`.

(67)
```
(^ OBL-MOD PRED:
      {
      <constraints_for_xp(mod)>
      |
      "base category constraints: prepnp(z,inst)"
      (-> FN)=c z  (<- OBJ CASE)=c inst
      "lexicalised constraints"
      (<- OBJ NUM)=c pl
      (<- OBJ PRED FN)=c ramię
         <constraints_for_ratr1>
      })
```

[21] It is semantic because the grammatical function assigned to the entire position is `OBL-MOD`.

[22] The constraint in (67) handles `ramię` as the lemma, an analogous constraint is used for `ręka`.

The adjp dependent of the nominal (np – this is because constraints apply to the nominal inside the prepositional phrase) is assigned[23] the ADJUNCT grammatical function (GFDEP is resolved to ADJUNCT in ratr1 specification). The only base category constraint for adjp restricts the value of its _CAT attribute to one of the following categories: adj is an adjective, ppas and pact are adjectival participles, passive and active. When it comes to lexicalised constraints, the values of case, number and gender are specified as agreeing (agr), so no constraints are introduced – such agreement is handled by the general grammar rules. The adjective is specified for positive degree (pos), so a DEGREE constraint is used. The lemma of adjp must be otwarty – a simple PRED specification is used here. Finally, the modification pattern of adjp is specified as atr1 – it may optionally take exactly one dependent, which is another lexicalised phrase, an advp(misc).

(68)
```
(^ OBL-MOD PRED:
   "base category constraints: adjp(agr)"
   (<- OBJ ADJUNCT CHECK _CAT)$c {adj ppas pact}
   "lexicalised constraints"
   (<- OBJ ADJUNCT DEGREE)=c positive
   (<- OBJ ADJUNCT PRED FN)=c otwarty
   ~(<- OBJ GFALL - ADJUNCT)
      <constraints_for_atr1>)
```

The constraints for the advp(misc) in (69) include the base category constraint restricting its _CAT to adv, followed by lexicalised constraints on degree (pos) and lemma (szeroko) and natr as its modification pattern (see (70)):

(69)
```
(^ OBL-MOD PRED:
   {
   "base category constraints: advp(misc)"
   (<- OBJ ADJUNCT ADJUNCT CHECK _CAT)=c adv
   "lexicalised constraints"
   (<- OBJ ADJUNCT ADJUNCT DEGREE)=c positive
   (<- OBJ ADJUNCT ADJUNCT PRED FN)=c szeroko
   ~(<- OBJ ADJUNCT GFALL - ADJUNCT)
      <constraints_for_natr>)
   |
   ~(<- OBJ ADJUNCT GFALL)
   })
```

(70)
```
(^ OBL-MOD PRED: ~(<- OBJ ADJUNCT ADJUNCT GFALL))
```

The entire (except for the placeholder for xp(mod) constraints, used for reasons of space) constraint for the last argument of (13) is given in (71):[24]

(71)
```
(^ OBL-MOD PRED:
   {
   <constraints_for_xp(mod)>
   |
   "base category constraints: prepnp(z,inst)"
   (-> FN)=c z (<- OBJ CASE)=c inst
   "lexicalised constraints"
   (<- OBJ NUM)=c pl
```

[23] For reasons of space, the entire mapping for dependents of non-verbal predicates cannot be presented here. As is standard in LFG and ParGram, the adjp dependent of np is an ADJUNCT, the advp dependent of adjp is also an ADJUNCT.

[24] All instances of GFALL variable used in (71) could replaced with unique, indexed variables such as GFALL_ADJP, GFALL_ADVP so that they have an expansion which is appropriate for a given head–dependent pair of categories, as explained in fn. 20.

```
(<- OBJ PRED FN)=c ramię
   "<constraints_for_ratr1>"
   "base category constraints: adjp(agr)"
   (<- OBJ ADJUNCT CHECK _CAT)$c {adj ppas pact}
   "lexicalised constraints"
   (<- OBJ ADJUNCT DEGREE)=c positive
   (<- OBJ ADJUNCT PRED FN)=c otwarty
   ~(<- OBJ GFALL - ADJUNCT)
      "<constraints_for_atr1>"
      {
      "base category constraints: advp(misc)"
      (<- OBJ ADJUNCT ADJUNCT CHECK _CAT)=c adv
      "lexicalised constraints"
      (<- OBJ ADJUNCT ADJUNCT DEGREE)=c positive
      (<- OBJ ADJUNCT ADJUNCT PRED FN)=c szeroko
      ~(<- OBJ ADJUNCT GFALL - ADJUNCT)
         "<constraints_for_natr>)"
         ~(<- OBJ ADJUNCT ADJUNCT GFALL)
      |
      ~(<- OBJ ADJUNCT GFALL)
      }
})
```

4 Conclusion

This paper presented how valency information from Walenty, currently the largest and most precise valency dictionary of Polish, can be used in an LFG grammar of Polish, presenting selected issues in more detail, together with a full formalisation.

The quality of the performed conversion of Walenty is evaluated and improved by building an LFG structure bank of Polish: human annotators manually disambiguate structures produced by the grammar which uses the lexicon with converted valency information from Walenty – see Patejuk & Przepiórkowski 2014c.

It is perhaps worth noting that the work presented here from the implementational side also supported theoretical work on the definition of grammatical functions in LFG (Patejuk & Przepiórkowski, 2016, 2014a) and formal issues such as imposing constraints in LFG under unlike category coordination (Przepiórkowski & Patejuk, 2012; Patejuk & Przepiórkowski, 2012a).

References

Arnold, Doug, Miriam Butt, Berthold Crysmann & Tracy Holloway King (eds.). 2016. *The proceedings of the HeadLex16 conference*. Stanford, CA: CSLI Publications.

Bresnan, Joan. 2001. *Lexical-functional syntax* Blackwell Textbooks in Linguistics. Blackwell.

Butt, Miriam & Tracy Holloway King (eds.). 2014. *The proceedings of the LFG'14 conference*. Stanford, CA: CSLI Publications.

Dalrymple, Mary. 2001. *Lexical-Functional Grammar*. Academic Press.

Dalrymple, Mary & Helge Lødrup. 2000. The grammatical functions of complement clauses. In Miriam Butt & Tracy Holloway King (eds.), *The proceedings of the LFG'00 conference*, University of California, Berkeley: CSLI Publications.

Patejuk, Agnieszka & Adam Przepiórkowski. 2012a. A comprehensive analysis of constituent coordination for grammar engineering. In *Proceedings of the 24rd international conference on computational linguistics (COLING 2012)*, Mumbai, India.

Patejuk, Agnieszka & Adam Przepiórkowski. 2012b. Towards an LFG parser for Polish: An exercise in parasitic grammar development. In *Proceedings of the eighth international Conference on Language Resources and Evaluation, LREC 2012*, 3849–3852. Istanbul, Turkey: ELRA.

Patejuk, Agnieszka & Adam Przepiórkowski. 2014a. Control into selected conjuncts. In Butt & King (2014) 448–460.

Patejuk, Agnieszka & Adam Przepiórkowski. 2014b. Structural case assignment to objects in Polish. In Butt & King (2014) 429–447.

Patejuk, Agnieszka & Adam Przepiórkowski. 2014c. Synergistic development of grammatical resources: a valence dictionary, an LFG grammar, and an LFG structure bank for Polish. In *Proceedings of the thirteenth workshop on Treebanks and Linguistic Theories (TLT13)*, Tübingen, Germany.

Patejuk, Agnieszka & Adam Przepiórkowski. 2016. Reducing grammatical functions in LFG. In Arnold et al. (2016).

Przepiórkowski, Adam. 2016. How *not* to distinguish arguments from adjuncts in LFG. In Arnold et al. (2016).

Przepiórkowski, Adam, Elżbieta Hajnicz, Agnieszka Patejuk & Marcin Woliński. 2014a. Extended phraseological information in a valence dictionary for NLP applications. In *Proceedings of the workshop on lexical and grammatical resources for language processing (LG-LP 2014)*, 83–91. Dublin, Ireland: Association for Computational Linguistics and Dublin City University.

Przepiórkowski, Adam, Elżbieta Hajnicz, Agnieszka Patejuk, Marcin Woliński, Filip Skwarski & Marek Świdziński. 2014b. Walenty: Towards a comprehensive valence dictionary of Polish. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk & Stelios Piperidis (eds.), *Proceedings of the ninth international Conference on Language Resources and Evaluation, LREC 2014*, 2785–2792. Reykjavík, Iceland: ELRA.

Przepiórkowski, Adam & Agnieszka Patejuk. 2012. On case assignment and the coordination of unlikes: The limits of distributive features. In Miriam Butt & Tracy Holloway King (eds.), *The proceedings of the LFG'12 conference*, 479–489. Stanford, CA: CSLI Publications.

Szupryczyńska, Maria. 1996. Problem pozycji składniowej. In Krystyna Kallas (ed.), *Polonistyka toruńska uniwersytetowi w 50. rocznicę utworzenia UMK. Językoznawstwo*, 135–144. Toruń: Wydawnictwo Uniwersytetu Mikołaja Kopernika.