Finite-State Rule Deduction for Parsing Non-Constituent Coordination

Sina Zarrieß and Wolfgang Seeker

Potsdam University, Germany

Abstract. In this paper, we present a finite-state approach to constituency and therewith an analysis of coordination phenomena involving so-called non-constituents. We show that non-constituents can be seen as parts of fully-fledged constituents and therefore be coordinated in the same way. We have implemented an algorithm based on finite state automata that generates an LFG grammar assigning valid analyses to non-constituent coordination structures in the German language.

1 Introduction

In standard syntactic theories, coordination is usually seen as a structure that conjuncts syntactic entities which are both of the same phrasal category and maximal projections (see [1] for an example). However, in various languages one finds examples for coordinated structures where conjoins don't fit into any standard concept of syntactic constituent, or don't even share their phrasal properties. The main insight the following examples should give is that, in general, syntactic entities can be coordinated if the material they share completes them in a syntactically well-formed way.

(1)	Asterix	darf	und	Obelix	darf	nicht	vom	Zaubertrank	trinken.
	Asterix	may	and	Obelix	may	not	of	magic potion	drink

(2)	Asterix	gibt	Obelix	ein	Wildschwein	und	Idefix	einen	Knochen.
	Asterix	gives	Obelix	a	boar	and	Idefix	a	bone

(3)	Obelix	verschlingt	ein	kleines	und	nascht	von	einem	großen
	Obelix	devours	a	small	and	snacks	of	a	big
	Wildschwein.								
	boar								

Even on the level of constituent structure, it is quite difficult to assign a valid analysis to this kind of constructions, see figure 1.

In [2], the author gives an exhaustive overview of strategies that have been implemented for covering these phenomena. In his paper, he mainly shows that despite their nearly identical behaviour, constituent and non-constituent coordination are often treated as completely seperated structures. Sometimes even the underlying parsing algorithm is modified for being able to parse this particular

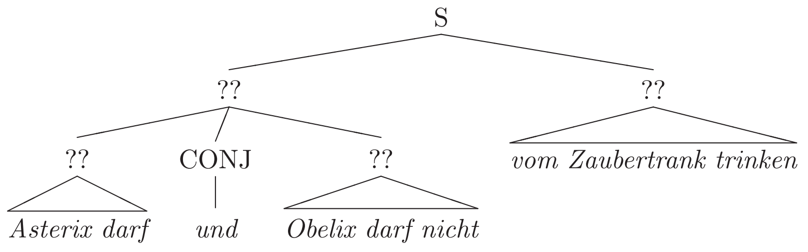

Fig. 1. Which category has a non-constituent?

structure. What seems to make it so difficult to analyze non-constituent phenomena is that they undermine the fundamental concept of syntactic category.

We have implemented an algorithm that generates a grammar allowing partial constituents in the context of coordination from a standard context-free LFG grammar in the well-known XLE format. This automatically generated grammar then covers the main types of non-constituent coordination as *Right Node Raising* and *Conjunction Reduction*. Our approach has been inspired by [3] who make use of the fact that the right rule sides of an LFG grammar are regular languages and can therefore be represented by finite state automata. But still, their strategy operates on the level of the parsing algorithm and suffers from the drawback of being barely formalized.

In the remaining of this paper, we will describe the way we apply the theory of automata to the problem of partial rule generation. Our basic goal is to assign the same category to partial constituents that expect identical completing material. When appropriate, we will go into the details of the XLE grammar implementation that realizes the assignment of a well-formed syntactic analysis to non-constituent coordination structures.

2 Preliminaries

2.1 Formal Language Devices

Initially, since we want to conceptualize constituency in a finite-state frame, we give some notations for formal devices that specify regular as well as context-free languages. For instance, regular expressions are instances of regular languages over an alphabet Σ. They are usually defined recursively as follows:

Definition 1. *The empty set $\emptyset$, the empty word ϵ and all symbols $a \in \Sigma$ denote regular expressions. If r and s are regular expressions, their disjunction $(r + s)$, concatenation (rs) and closure r^* also denote regular expressions.*

An alternative device for manipulating regular languages are the so-called finite-state automata.

Definition 2. *A finite-state automaton (or FSA) A is a 5-tuple $A = (\Sigma, Q, I, F, E)$ where:*

1. *Σ is the finite input alphabet of the automaton;*
2. *Q is the finite set of states;*
3. *$I \subseteq Q$ is the set of initial states;*
4. *$F \subseteq Q$ is the set of final states;*
5. *$E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ the set of transitions.*

Given a transition $e \in E$, we denote by $s[e]$ its input label, by $p[e]$ its origin (previous) and by $n[e]$ its destination (next) state. A path $\pi = e_1 \dots e_n$ in the automaton A is defined as an element of E^* where forall $n[e_{i-1}] = p[e_i]$. A *successful* path in A is a path $\pi = e_1 \dots e_n$ where $p[e_1] \in I$ and $n[e_n] \in F$, thus a path from an initial to a final state. The concept of an origin and destination state can be extended to paths so that $p[\pi]$ and $n[\pi]$ are meant to be the origin and destination state of the path π. Accordingly, $s[\pi]$ denotes the concatenation of the input symbols of a path π.

We define the right language of a state $q \in Q$ as follows: $\overrightarrow{L}(q) = \{w \mid \pi(q, w, q'), q' \in F\}$. The language accepted by an automaton A is the union of the right languages of all initial states $L(A) = \bigcup_{q \in I} \overrightarrow{L}(q)$.

The definition of a finite-state automaton can be easily extended to a finite-state transducer (FST) $T = (\Sigma, \Delta, Q, I, F, E)$ where Δ is the finite output alphabet and $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times Q$. The composition of two finite state transducers $T = T_1 \circ T_2$ is defined as $T = (\Sigma_1, \Delta_2, Q_1 \times Q_2, I_1 \times I_2, F_1 \times F_2, E)$ where $(< q_1, q_1{}' >, a, b, < q_2, q_2{}' >) \in E$ if $(q_1, a, c, q_2) \in E_1$ and $(q_1{}', c, b, q_2{}') \in E_2$.

Whereas the language $L(A)$ accepted by an FSA A is a set $L \subseteq \Sigma^*$, the language recognized by an FST can be seen as a relation $L(T) \subseteq (\Sigma \times \Delta)^*$. We will sometimes use the common set theoretic operations like union and product to denote operations on FSTs. Furthermore, we define an operation of *projection*, where $P_1(T) \subseteq \Sigma^*$ corresponds to the input and $P_2(T) \subseteq \Delta^*$ to the output language of T.

The LFG grammar that will provide input for our rule generator is an instance of a context-free language specified as a context-free grammar.

Definition 3. *By the classical definition, CFGs are tuples $G_c = \langle V_T, V_N, S, R \rangle$ including the requirements:*

1. *V_T is the finite alphabet of terminals;*
2. *V_N is the finite alphabet of nonterminals;*
3. *S is the start symbol;*
4. *R is the set of rules $r : \sigma \rightarrow \omega$, so that $\sigma \in V_N$ and $\omega \in (V_T \cup V_N)^*$.*

In the following section, we will mainly show how these computationally quite distinct devices can be interrelated to give rise to a formal concept of (partial) constituency.

2.2 LFG Grammars in XLE

The actual implementation of our coordination grammar, as described in the following section, has been done within the XLE development environment for

LFG grammars (see [4] or [5] for a general introduction). To enable the reader to follow the presented examples, we will give a short description of some common XLE notations.

In general, XLE grammars have to be specified as *left-canonical* (see definition 4). An example for a typical rule in XLE is shown in figure 2. Disjunction is expressed by curly brackets, optionality by round brackets. The category symbol stands to the left of a colon, its f-annotation to the right. An expression ($SUBJ$) =! in the f-annotation means that whatever is derived by that symbol is in the $SUBJ$ feature of the current f-structure. The $-symbol is the ∈-symbol and is used for sets.

```
NP -->
    { (D) A*: !$(^ADJUNCT); N:(^SUBJ)=!;
    | PRON:(^SUBJ)=!;
    }
    (PP:!$(^ADJUNCT);).
```

Fig. 2. A simple NP rule

To formulate category independent structures in a general way one can define so-called XLE *macros*. In figure 3, our macro for coordination is shown. A category symbol placed in **_cat** is copied to all occurences in the rule body and associated with the specified f-annotation. Note that the $-operator causes the coordinated elements to be unified in a set and their f-structures to be treated as a single set.

```
COORD(_cat) =
    _cat: !$^;
    ( COMMA _cat: !$^; )*
    {CONJ[konj]: (^NUM)=pl | CONJ[disj]: (^NUM)=sg }
    _cat: !$^.
```

Fig. 3. The coordination macro

Another useful XLE mechanism called *complex category symbols* allows enriching category symbols with different parameters to generalize information about different instances of the same constituent. A parameterized symbol then stands for all possible instanciations of its parameters. An example is shown in figure 4.

However, one should not think of category parameters in terms of Prolog-style variables. Parameters are only interpreted when they occur at the left side of a rule. Thus, if parametrized categories appear on the right side of a rule,

```
NP[_param $ { pl sg }] -->
    {
        D N
    |
        N: _param = pl;
    }.
```

Fig. 4. Parameterized NP rule

their mother category is parametrized by at least the sum of all its daughter parameters. In consequence, underspecified parameters aren't licensed and we are forced to enumerate all possible instanciations of a parameter at least at the root symbol.

3 A Finite-State Concept of Constituency

3.1 A First Approach to Partial Constituents

One can easily see that the right side ω of a standard context-free rule as defined in definition 3 is equivalent to a regular expression by the finite concatenation of symbols $x_i \in (V_T \cup V_N)$, where $x_1 \ldots x_n = \omega$. Very often, context-free grammars (LFG grammars for instance) especially make use of the regular syntax in that they allow operations like disjunction or closure to appear on the right rule side. This fact enables us to define the concept of a *left-canonical* grammar.

Definition 4. *A context-free grammar $G_c = \langle V_T, V_N, S, R\rangle$ is called left-canonical if and only if for every $\sigma \in V_N$ there exists at most one $r \in R$ such that $r : \sigma \to \omega$.*

For every context-free grammar G that doesn't satisfy this property, it is possible to define an equivalent *left-canonical* grammar G'. Given a grammar G_1 that contains one pair of rules r_1, r_2 such that $\sigma_1 = \sigma_2$, you simply have to unify the right sides ω_1, ω_2 such that $r_{1+2} : \sigma_{1+2} \to \omega_1 + \omega_2$.

This type of grammar is a useful construction when you want to define the regular language $L_1(\sigma)$ that can be derived from a nonterminal $\sigma \in G$ by a single rule application. If G is left-canonical, for every $\omega_i \in R$, ω_i denotes exactly the regular expression that corresponds to the language $L_1(\sigma_i)$. Thus, it is possible to represent the set of rules R of a context-free grammar by a set of automata $A_R = \{A_{\omega_i} \mid L(A_{\omega_i}) = L_1(\sigma_i)\}$.

Given these connections between regular and context-free languages, one can formalize the syntactic concept of constituency in the following way:

Definition 5. *A linguistic entity ω_i is called a constituent with respect to a context-free grammar G, if there exists a path $\pi \in A_\omega$ such that A_ω is equivalent to some right rule side $\sigma \to \omega \in G$ and $s[\pi] = \omega_i$.*

In case $p[\pi] \notin I$ or $n[\pi] \notin F$, thus if π isn't *successful*, the related constituent $s[\pi]$ is called *partial* and labelled by $\omega^{i,j}$ where $i = p[\pi]$ and $j = n[\pi]$. Otherwise, if π is a *successful* path in A_ω we call the constituent $\omega^{0,n}$ that corresponds to $s[\pi]$ *complete*. We define an isomorphism $\mu : L_1(\sigma^{i,j}) \rightarrow \Pi_{e_i \dots e_j}$ that maps the regular language representing a (possibly partial) constituent to its corresponding set of paths in the rule automaton.

Now, we are able to state the hypothesis in [3] concerning coordination of non-constituents in a formal way:

Lemma 1. *Two constituents* $\omega^{i,j}, \omega^{u,v}$ *with* $\omega^{i,j} = s[\pi_{i,j}]$ *and* $\omega^{u,v} = s[\pi_{u,v}]$, $\pi_{i,j}, \pi_{u,v} \in A_\omega$ *can be coordinated iff* $i = u$ *and* $j = v$.

3.2 A Precise Formalization of Coordination

The intuition underlying this generalized coordination rule is that (partial) constituents can be conjoined if the constituents that make them complete are identical. However, this lemma doesn't completely capture this intuition. Consider the automaton in figure 5 that represents a simplified NP-rule.

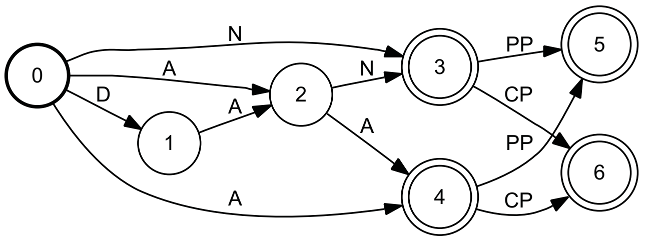

Fig. 5. An automaton representing the right side of a simple NP rule

Although the constituents labelled by the input sequences N and A can be completed by identical constituents PP or CP, their corresponding paths don't have a single destination state in common. Therefore, we have to refine our concept of a valid rule automaton A_ω.

It can be shown that for every regular set there exists a canonical minimum state automaton. The well-known Myhill-Nerode theorem (see [6]) states that a regular language is the union of equivalence classes on a right invariant equivalence relation R_L of finite index. R_L is defined by: xR_Ly if and only if for all $z \in \Sigma^*$, $xz \in L$ if and only if $yz \in L$. In terms of an automaton this means that its set of states can be partitioned into equivalence classes with respect to a right invariant relation $R_L{}'$ where $q_i R_L{}' q_j$ if $\overrightarrow{L}(q_i) = \overrightarrow{L}(q_j)$.

The concept of a state representing an equivalence class with respect to its right language is exactly what we need for the partial constituents being assigned a meaningful category. In consequence of the Myhill-Nerode theorem, one can now say that all partial constituents $\omega^{i,j}$ that can be completed by an identical constituent $\omega^{j,k}$ lead the canonical rule automaton into an unique state k, since they have the same right language $\overrightarrow{L}(k)$. This leads us to a more precise generalized coordination rule:

Lemma 2. *Two constituents $\omega^{i,j}, \omega^{u,v}$ with $\omega^{i,j} = s[\pi_{i,j}]$ and $\omega^{u,v} = s[\pi_{u,v}]$, $\pi_{i,j}, \pi_{u,v} \in A_\omega$ can be coordinated iff $i, u \in ||i||$ and $j, v \in ||j||$.*

Hence, our goal is to generate all possible partial right rule sides $\omega^{i,j} = s[\pi_{i,j}]$ from the canonical rule automaton and assign them a category which is parametrized by the pair of states (i, j) in the canonical rule automaton. Formally, this can be expressed by a mapping that relates a *left-canonical* context-free grammar G to a grammar G' that is equivalent with respect to the maximal projections but explicits its partial constituents:

Definition 6. *Given a canonical CFG $G_c = \langle V_T, V_N, S, R\rangle$ and its equivalent canonical rule automaton A_ω we define a mapping onto a grammar $G_p = \langle V_T`, V_N`, S`, R`\rangle$ that fulfils the following conditions:*

1. *$S = S`$ and $V_T = V_T`$.*
2. *$V_N` = V_N \cup \bigcup_{\sigma \in V_N, i,j \in Q_{A_\omega}} \sigma^{i,j}$.*
3. *Every rule $r : \sigma \rightarrow \omega$, $r \in R$ is mapped by a function f to a set of rules $f(r) = \{\sigma^{i,j} \rightarrow \omega^{i,j} \mid s[\pi_{i,j}] = \omega^{i,j}, \pi_{i,j} \in {E^*}_{A_\omega}, \pi_{i,j} \in \mu(L(\sigma^{i,j}))\}$.*
4. *$R` = \bigcup_{r \in R} f(r) \cup R$.*

However, in our XLE implementation we avoid blowing up the set of rules as is described above. The mechanism of *complex category symbols* allows us to treat all partial constituents $\omega^{i,j}$ of a mother category σ as derivations of the same rule that parametrize σ in a different way. Figure 6 shows an example of a parametrized NP-rule, where the origin and destination state of the constituent are realized as the parameters **_from** and **_to**. To indicate the possibly partial status of this category it is called *XPsub*. Every *XPsub* is parameterized by all parameters $\phi_1 ... \phi_n$ of its original category, by a parameter **_koord**, which marks the coordination status of the constituent, and two parameters **_from** and **_to** to mark the start and end index of a particular substring (we use the values **sa** and **se** for start and end state). Thus, *XPsub* is equal to the set of rules yielded by the mapping $f(XP)$.

Because of the constituent status of the non-constituents we can now coordinate them with the same macro we use for constituent coordination, see figure 3. The restriction, that only constituents with congruent origin and destination state can be coordinated, is inherent to the macro since only categories with identical parameters form the same category symbol.

3.3 Assembling Complete Constituents

Up to now, we have completely ignored the fact that we don't want our grammar to derive partial constituents in contexts other than coordination. Finally, we

```
NPsub[_ntype $ {std rel int},_koord,_from,_to] -->
{
AP: _ntype = std; e: _from=s19 _to=s21 _koord=no;
|
...
|
@(COORD_PART NPsub[_ntype $ {std rel int},no,_from,_to])
e: _koord=yes;
|
@(COORD NPsub[_ntype $ {std rel int},_koord,_from,_to]
e: _from=sa _to=se)
}.
```

Fig. 6. *NPsub* generated from an NP rule

will have to add rules to the grammar that reassemble partial constituents to complete ones. The formalization is straightforward:

Lemma 3. *A sequence of constituents* $\omega^{i_0,j_0}, \omega^{i_1,j_1}, \ldots, \omega^{i_n,j_n}$, $\omega^{i_i,j_i} \in A_\omega$ *forms a complete constituent, iff for every* $j_n = i_{n+1}$ *and* i_0 *is the initial state and* i_n *some final state in* A_ω.

Thus, we would have to add to our coordination grammar G_c' all possible instances of the rule $r : \sigma^{i,k} \rightarrow \omega^{i_i,j_i}, \ldots, \omega^{i_k,j_k}$. But this would lead to an extreme ambiguity in the resulting grammar where every complete constituent could, in addition to its original derivations, be derived by numerous ways of putting together its partial right rule sides $\omega^{i,j}$. To cope with this serious overgeneration, we restrict the completion to require at least one partial constituent that has been formed by a conjunction of partial constituents. Every partial constituent is further parametrized with a boolean feature that marks its coordination status to check for this condition.

Lemma 4. *A sequence* ${\omega_{coord}}^{i_0,j_0}, {\omega_{coord}}^{i_1,j_1}, \ldots, {\omega_{coord}}^{i_n,j_n}$, $\omega^{i_i,j_i} \in A_\omega$ *forms a complete constituent, iff there exists some* ${\omega_{coord}}^{i,j}$ *with coord = true and for every* $j_n = i_{n+1}$ *and* i_0 *is the initial state and* i_n *some final state in* A_ω.

In our implementation we have restricted the respective completion rules to binary branching.

We are now able to give an elegant analysis for e.g. the phenomenon of Right Node Raising as shown in figure 8.

4 Automated Deduction of Partial Constituents

We could now extract all partial constituents from the rule automaton by performing a standard breadth-first search on its transitions. However, we prefer this operation to be defined on the algebra of finite state automata.

```
NPkompl[_ntype $ {std rel int},_from,_inter,_to] -->
{
NPsub[_ntype $ {std rel int},yes,_from,_inter]
NPsub[_ntype $ {std rel int},no,_inter,_to]
|
NPsub[_ntype $ {std rel int},no,_from,_inter]
NPsub[_ntype $ {std rel int},yes,_inter,_to]
|
NPmiss[_ntype $ {std rel int},yes,_inter]
PPsub[std,no,_inter,se]: !$(^ADJUNCT); e:_from=sa _to=se;
}.
```

Fig. 7. *NPkompl* generated from an NP rule

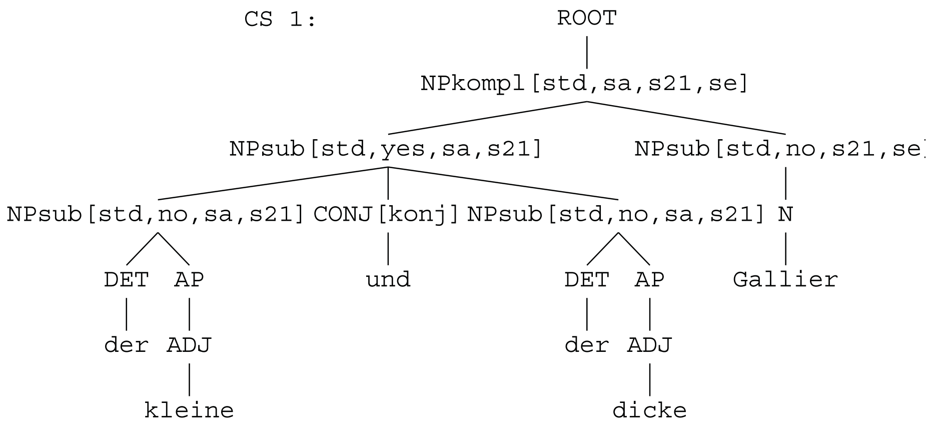

Fig. 8. Analysis for the German NP *the small and the big gaul*

In a first step, we define a transducer T_{sub} that can be notated as a regular expression $T_{sub} = (ID(\Sigma)^* \cdot (\Sigma \times \{\epsilon\})^* \cdot ID(\Sigma)^*)$. By definition of the composition of FSTs, the transducer $ID(A_\omega) \circ T_{sub}$ yields a relation that maps arbitrary long prefixes and suffixes in A_ω to ϵ, so that the second projection of this relation $P_2(ID(A_\omega) \circ T_{sub})$ yields an automaton $A_{\omega^{i,j}}$ that accepts all partial and complete constituents of A_ω. The problem with this first approach is that the states in $A_{\omega^{i,j}}$ of course don't correspond to those in A_ω anymore. However, we have shown in the previous section that the state information inherent to the canonical rule automaton A_ω is exactly what we need to parametrize partial categories.

Thus, before generating partial constituents, we have to explicitely encode the states in the FSA A_ω. Broadly speaking, we do this by indexing the regular expression that constitutes the right rules sides ω to label every distinct position of the expression. Then, the automaton ${A_\omega}^{ind}$ equivalent to the indexed right rule side contains position labels such that for every given state $n[e] \in A$ with the right language $\overrightarrow{L} \setminus I$ there exists a unique input label $i[e] \in I_\omega$. By a series of compositions with deletion transducers we obtain an automaton ${A_\omega}^{i,j}$ whose input sequences $s[\pi]$, where π is *successful*, are strings of the form $s = i_i \cdot \omega^{i,j} \cdot i_j$.

First, we have to define the operation of indexing positions in regular expressions.

Definition 7. *Let r, s and t denote regular expressions and $I = \{1, 2, 3 \ldots n\}$ an index alphabet such that n is the number of subexpressions of r, t or s respectively. If $r = a, a \in \Sigma$, $r^{ind} = i_0 \cdot a \cdot i_n$. If $r = (s + t)$, $r^{ind} = (s^{ind} + t^{ind}) \cdot i_n$. If $r = st$, $r^{ind} = (s^{ind} t^{ind}) \cdot i_n$. If $r = s^*$, $r^{ind} = (s^{ind})^* \cdot i_n$.*

The automaton ${A_\omega}^{ind} = (\Sigma \cup I_\omega, Q, I, F, E)$ corresponding to an indexed right rule side ω^{ind} accepts input sequences $s[\pi_{0,n}] = i_0 \cdot {\omega^{0,n}}_{ind} \cdot i_n$. To remove redundant position labels in ${A_\omega}^{ind}$, it is composed with the transducer T_{delpos}:

$$T_{delpos} = (ID(i_0) \cdot (I \times \{\epsilon\})^* \cdot ID(\Sigma)) \cdot ((I \times \{\epsilon\})^* \cdot (ID(I) \cdot ID(\Sigma))^*) \quad (1)$$

The second projection ${A_\omega}^{delpos} = P_2({A_\omega}^{ind} \circ T_{delpos})$ yields input sequences of the form $s[\pi_{0,n}] = i_0 x_0 i_1 x_1 \ldots x_n i_n$ where $i_i \in I, x_i \in \Sigma$ so that $x_0 \ldots x_n = \omega$.

Lemma 5. *If there are two constituents ${\omega^{i,j}}_{ind} = s[\pi_{i,j}], {\omega^{u,v}}_{ind} = s[\pi_{u,v}]$, with $\pi_{i,j}, \pi_{u,v} \in {A_\omega}^{delpos}$, there will be a pair of sequences $s[\pi_{i-1,j+1}] = i_i \cdot x_i \ldots x_j \cdot i_j, s[\pi_{u-1,v+1}] = i_i \cdot x_u \ldots x_v \cdot i_j$ if and only if $i = u, j = v$ and $i - 1 = u - 1, j + 1 = v + 1$.*

This lemma shows that 1) if there are some indexed constituents (some paths whose input sequences start and end with a symbol $a \in \Sigma$) that have an identical origin and destination state, there are some paths in the same automaton that accept these constituents labeled with an identical start and end position and 2) if there are some input sequences that have an identical start and end label, their corresponding constituents have an identical origin and destination state.

Now, the deletion transducer has to be changed to preserve the right labels for each partial constituent. Given the following transductions:

$$\begin{aligned} T_{sub} &= (ID(\Sigma) \cup (\Sigma \times \{\epsilon\}) \cup ID(I))^* \\ T_{normpos} &= ((I \times \{\epsilon\})^* \cdot (ID(I) \cdot ID(\Sigma))^+ \cdot ID(I) \cdot (I \times \{\epsilon\})^*) \\ T_{uni} &= (ID(I) \cdot (ID(\Sigma) \cup (I \times \{\epsilon\}))^+ \cdot ID(I)) \end{aligned}$$

and the composition:

$$T_{\omega^{i,j}} = A_\omega \circ T_{delpos} \circ T_{sub} \circ T_{normpos} \circ T_{uni}$$

The second projection $P_2(T_{\omega^{i,j}})$ finally produces an automaton $A_\omega{}^{i,j}$ where each input sequence $s[\pi_{i-1,j+1}] = i_i \cdot \omega^{i,j} \cdot i_j$.

To conclude, we have shown that the mapping defined in 6 can be performed on the algebra of FSTs.

5 Extensions

Consider the following coordination:

(4)	Asterix	spielt	mit	und	Obelix	schimpft	auf	Idefix.
	Asterix	plays	with	and	Obelix	rails	against	Idefix

Currently, our grammar doesn't cover this type of constructions because there aren't some partial constituents being coordinated, but two complete phrases that contain a partial constituent of the same type. To be able to cover these phenomena, we define a third constituent type called *missing*.

Definition 8. *A constituent $\omega^{i,j}$ has the missing-property if it fulfils the following conditions:*

1. *i and j correspond to some initial and final state in A_ω.*
2. *For the rightmost symbol $\sigma_r = \omega^{j-1,j}$ there is a rule in G such that $r : \sigma_r \rightarrow \omega$.*
3. *The rightmost symbol σ_r is a partial constituent $\sigma_r{}^{i,j}$ where i corresponds to the inital state in $A_{\omega r}$ and $j \notin F_{A_{\omega r}}$*

In figure 9, we give an example for a *NPmiss*-category. The *XPmiss* symbol also contains all parameters $\phi_1 ... \phi_n$ of its original category and a parameter **_koord** to mark its coordination status. In figure 10 we then present an analysis for a category with the missing property.

Of course, there are still some other specific constructions that aren't covered by the presented model. For instance, one could imagine the coordination of two partial constituents that don't have the same *right language*, but whose intersection of right languages isn't empty, and the material that completes them is contained in this intersection. Or else, coordinated constituents may have an identical right language, but not in the same rule automaton. In this case, their origin and destination state won't help identifying them. Thus, an alternative parametrization for partial constituents would be their right and left language in contrast to their origin and destination state.

```
NPmiss[_ntype $ {std rel int},_koord,_missing] -->
{
DET: ^=!; AP: _ntype = std; N: _ntype = std;
PPsub[std,no,sa,_missing]: ! $ (^ADJUNCT); e: _koord=no;
|
...
|
@(COORD_PART NPmiss[_ntype $ {std rel int},no,_missing])
e: _koord=yes;
}.
```

Fig. 9. *NPmiss* generated from an NP rule

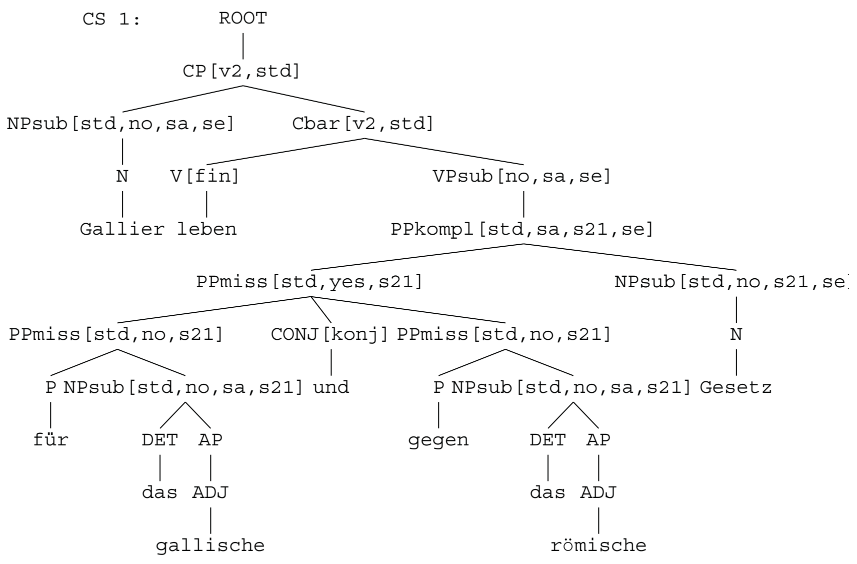

Fig. 10. Analysis for the German sentence *Gauls live for the gaulic and against the roman law*

6 Conclusion

These considerations outlined some formal properties as well as the limitations of the coordination approach proposed by [3]. We have shown that the problem of non-constituent coordination can be solved by expliciting partial constituents on the surface of the grammar. An adequate way of describing these partial categories is to see them as paths in a canonical right rule side automaton. This formalization gives rise to an efficient mechanism of subrule generation. We have also extended the approach of [3] to coordination of complete constituents with embedded partial constituents at the right periphery.

References

1. Kaplan, R.M., Maxwell, J.T.: Constituent coordination in lexical-functional grammar. In: Proc. of the 12th COLING, Budapest, Hungary (1988) 303–305
2. Milward, D.: Non-constituent coordination: Theory and practice. In: Proceedings of the 15th International Conference on Computational Linguistics (COLING94), Kyoto. (1994)
3. Maxwell, J., Manning, C.: A theory of non-constituent coordination based on finite-state rules (1996)
4. Butt, M., Dyvik, H., King, T.H., Masuichi, H., Rohrer, C.: The parallel grammar project. In: Proc. of COLING-2002 Workshop on Grammar Engineering and Evaluation. (2002) 1–7
5. Kaplan, R.M., Maxwell, J.T.: Lfg grammar writer's cookbook. Technical report, Xerox PARC (1996)
6. Hopcroft, J., Ullmann, J.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)