A constraint-programming approach to parsing with resource-sensitive categorial grammar

Katrin Erk & Geert-Jan M. Kruijff

Computational Linguistics
Saarland University
Saarbrücken, Germany
{erk,gj}@coli.uni-sb.de

Abstract. Parsing with resource-sensitive categorial grammars (up to the Lambek-Van Benthem calculus LP) is an NP-complete problem. The traditional approach to parsing with such grammars is based on *generate & test* and cannot avoid this high worst-case complexity. This paper proposes an alternative approach, based on *constraint programming*: Given a grammar, constraints formulated on an abstract interpretation of the grammar's logical structure are used to prune the search space during parsing. The approach is provably sound and complete, and reduces the search space in steps that are mostly linear or low-polynomial.

1 Introduction

The problem we are concerned with here is the efficient parsing of *resource-sensitive categorial grammars*. Resource-sensitive categorial grammar [Mor94,Hep95,Moo97] (ResCG) overcomes the linguistic inadequacy of classical categorial grammar (CG). However, parsing with ResCGs up to the Lambek-Van Benthem calculus LP is NP-complete, and undecidable in the case of unrestricted ResCG [Car95]. The proof search algorithm used for parsing with ResCGs (up to LP) [Moo02] does not avoid this NP-complete worst-case complexity. We use constraint programming to formulate a parsing algorithm for ResCG that shows a better average case complexity, pruning the search space using linear or low-polynomial computations while maintaining soundness. Like Moot we consider ResCG up to LP.

In CG, we only have inference over types that indicate how expressions can be used to form larger expressions. The languages we can thus generate are at most context-free. Consequently, we cannot e.g. handle discontinuity or represent generalizations about word order flexibility. The basic idea behind ResCG is to enable additional inference over the trees that expressions form as a reflection of type inference. This additional inference, or *structural reasoning*, takes the form of *tree rewriting*. Besides a lexicon and a logic describing type inference, a ResCG contains a set of *structural rules* that describe how to rewrite one tree into another tree. In this paper we concentrate on structural rules that can regroup trees (associativity, or 'flexible constituency') or reorder them (commutativity). ResCGs with suchlike rules are commonly known as the Lambek calculi [Moo97], with which detailed accounts of phenomena beyond context-freeness can be given, e.g. [Kru01].

Classical CG can be modelled using the Lambek calculus L [Lam58], which is weakly equivalent to context-freeness [Pen97]. For L we have various efficient parsing algorithms. [Kön90] and [Hep92] present early versions of chart-based approaches, which e.g. [Mor96] and [dG99] improve upon. For ResCG, it is more difficult to find efficient parsing algorithms.

Commonly, the proof search algorithm described in [Moo02] is used to parse with ResCGs. However, because that algorithm uses *generate & test*, it cannot avoid the high worst-case complexity. We propose an alternative approach based on *constraint programming*, cf. [JL87,MS98]. Constraint programming is typically applied to solving NP-hard combinatoric problems that would traditionally be tackled with a *generate & test* strategy. The idea of constraint programming is to use a *propagate & distribute* strategy instead: We put off *distribution* steps that constitute search as long as possible, and first apply *propagators*, deterministic inferences that prune the search space. Good propagation can reduce the size of the search space considerably. Although the worst-case complexity remains the same, we usually obtain a better average-case complexity. [1]

For parsing ResCGs, we propose two types of propagators, which are constructed on an analysis of the grammar. These propagators operate on (compact) tree descriptions that are abstract representations of the possible results of rewriting. *Stationary terms* indicate structures that, once composed, the grammar leaves invariant under rewriting. *Transit terms* state how structural rules license subtrees to move. We use these two types of information to prune the search space. The idea is to allow only movements that are licensed on the grammar.[2]

This approach can be seen as an *abstract interpretation* of computations in a ResCG. Abstract interpretation is a theory for approximating the semantics of discrete dynamic systems, e.g. computations of programming languages. With stationary terms and transit terms we create an abstraction from the actual grammar itself, and show that one can compute on this abstraction more efficiently than with the actual grammar while maintaining soundness and completeness.

Overview. In §2 we introduce ResCG and a running example we use throughout the paper, and briefly sketch ResCG parsing algorithms. §3 and §4 form the core of the paper, introducing the basic concepts of our approach, and discussing the constraint-based algorithm in detail. In §5 we discuss soundness and completeness results for our approach. §6 deals with complexity. We end with conclusions.

Assumptions. We focus in this paper on eliminative and structural reasoning in ResCG. We do not consider hypothetical reasoning here: This can be modeled using the *dominance* constraints we discuss in §4, or avoided altogether [Hep98]. For the purposes of this paper we also assume that there is no categorial ambiguity, in order to present the key intuitions of our approach as clearly as possible.

Acknowledgements. We would like to thank Denys Duchier, Alexander Koller and the reviewers of NLULP for many helpful comments. Geert-Jan Kruijff's work is supported by the DFG Sonderforschungsbereich 378 *Resource-Sensitive Cognitive Processes*, Project NEGRA EM6.

[1] Note that even in the case of LP we can reduce the search space in steps that are linear or low-polynomial, not exponential.

[2] Note that we employ movement as a metaphor, not as a theoretical construct in CG.

2 Reasoning with trees

The purpose of the current section is to introduce resource-sensitive categorial grammars and a running example we use throughout the paper (§2.1); and, to discuss the algorithm of [Moo02] and present the intuitions behind the constraint-based approach we propose (§2.2).

2.1 Resource-sensitive categorial grammar (ResCG)

Like a classical categorial grammar (CG), a resource-sensitive categorial grammar (ResCG) assigns types to expressions, and has a calculus defining inference over these types. What sets ResCG apart from CG is that we can have additional inference over the trees that expressions form as a reflection of the type inference.

Lexicon:

$\langle$dass, $relc/_{rel}scon\rangle$ $\langle$einen, $np/_{d}np\rangle$
$\langle$maria, $np\rangle$ $\langle$roman, $np\rangle$
$\langle$zu-schreiben, $np\backslash_{dc}(np\backslash_{sc}zuinf)\rangle$
$\langle$verspricht, $(np\backslash_{sc}scon)/_{con}(np\backslash_{sc}zuinf)\rangle$

Structural rules:

[1] $A \circ_{con} (B \circ_{dc} C) \rightarrow B \circ_{dc} (A \circ_{con} C)$
[2] $A \circ_{sc} (B \circ_{con} C) \rightarrow (A \circ_{sc} B) \circ_{con} C$
[3] $A \circ_{con} (B \circ_{dc} C) \rightarrow (B \circ_{dc} C) \circ_{con} A$

Fig. 1. A small fragment for scrambling phenomena in German

$$
\begin{array}{ll}
[\mathsf{E}\backslash] & \langle \mathcal{E}_1, A\rangle \;\&\; \langle \mathcal{E}_2, A\backslash_\mu B\rangle \rightarrow \langle (\mathcal{E}_1 \circ_\mu \mathcal{E}_2), B\rangle \\
[\mathsf{E}/] & \langle \mathcal{E}_2, B/_\mu A\rangle \;\&\; \langle \mathcal{E}_1, A\rangle \rightarrow \langle (\mathcal{E}_2 \circ_\mu \mathcal{E}_1), B\rangle \\
[\mathsf{I}\backslash] & [\langle \mathcal{E}_1, A\rangle] \;\&\; \langle (\mathcal{E}_1 \circ_\mu \mathcal{E}_2), B\rangle \rightarrow \langle \mathcal{E}_2, A\backslash_\mu B\rangle \\
[\mathsf{I}/] & \langle (\mathcal{E}_2 \circ_\mu \mathcal{E}_1), B\rangle \;\&\; [\langle \mathcal{E}_1, A\rangle] \rightarrow \langle \mathcal{E}_2, B/_\mu A\rangle
\end{array}
$$

Fig. 2. Base logic for resource-sensitive categorial grammar

Consider the lexicon in Figure 1. Lexical entries take the form $\langle word, type\rangle$. The basic behavior of the type-forming operators $\{\backslash_\mu, /_\mu\}$ is defined by the base logic, given in Figure 2. Rather than reasoning just over types, the base logic reasons with tuples $\langle Tree, Type\rangle$. For example, $\langle$einen, $np/_{d}np\rangle$ can combine with $\langle$roman, $np\rangle$ using $[\mathsf{E}/]$ to form the tree $(einen \;\circ_d\; roman)$ of type np, with $\circ_d$ reflecting the *mode* d of the slash $/_d$ in the type of $einen$.

In addition to the base logic we can define *structural rules* that operate on trees. A structural rule takes the form $\mathbb{I} \rightarrow \mathbb{O}$, rewriting an input tree of the form $\mathbb{I}$ into the output tree $\mathbb{O}$. Important is that the applicability of a structural rule is conditioned by the shape of the tree $\mathbb{I}$ it operates on – that is, by the products $\circ_\mu$ used in building that tree. As we can introduce different modes μ [Hep95], we have a fine-grained means to control the applicability of structural rules. That way we can avoid unrestricted applicability of commutativity and associativity, like in the Lambek-Van Benthem calculus LP.

Figure 1 gives a few structural rules dealing with German scrambling, a phenomenon that is best handled by a grammar stronger than context-freeness, e.g. [DD01]. Variables A, B, C stand for arbitrary substructures. The fragment is for illustrative purposes only – see e.g. [DD01] for a more comprehensive account. In (1) through (3) we provide examples that can be analyzed; between brackets we list the structural rule(s) needed in their derivation.

(1) *dass Maria verspricht einen Roman zu-schreiben* [base logic]
that Maria promises a novel to-write
"that Maria promises to write a novel."
(2) dass Maria einen Roman verspricht zu-schreiben [1]
(3) dass einen Roman zu-schreiben Maria verspricht [2,3]

It is easy to verify that we get the following tree for (1), using the [E\], [E/] rules of the base logic, and the lexical entries for the words:

– (dass $\circ_{rel}$ (maria $\circ_{sc}$ (verspricht $\circ_{con}$ ((einen $\circ_{d}$ roman) $\circ_{dc}$ zu-schreiben))))

Structural rule [1] can rewrite the subtree for "verspricht einen roman zu-schreiben", to yield sentence (2):

– (verspricht $\circ_{con}$ ((einen $\circ_{d}$ roman) $\circ_{dc}$ zu-schreiben))
$\stackrel{[1]}{\longrightarrow}$ ((einen $\circ_{d}$ roman) $\circ_{dc}$ (verspricht $\circ_{con}$ zu-schreiben))
– (dass $\circ_{rel}$ (maria $\circ_{sc}$ ((einen $\circ_{d}$ roman) $\circ_{dc}$ (verspricht $\circ_{con}$ zu-schreiben))))

Structural rules are linear rewriting rules – they rearrange nodes in a tree. The notion of *id-mapping function* helps us to talk about (and keep track of) what is being manipulated, explicitly. Given two trees τ_1, τ_2, an id-mapping is a bijection that maps each node u of τ_1 to a node v of τ_2 such that u, v bear the same node label. Each structural rule $R = \mathbb{I} \rightarrow \mathbb{O}$ comes with an id-mapping function $\mathbf{idmap}_R$. We use $\mathbf{idmap}_R(u) = v$ to express that the node u of $\mathbb{I}$ has been relocated to be node v of $\mathbb{O}$. When we apply a structural rule R to a tree τ to yield τ', we can extend the function $\mathbf{idmap}_R$ to an id-mapping from τ to τ', since the structural rule only rearranges the nodes in the substructure of τ matching $\mathbb{I}$. The id-mapping shows where R moves each node of τ in τ'.

2.2 Parsing with ResCGs

Structural rules like [1] and [3] (Figure 1) reorder nodes in a tree, taking the fragment beyond context-freeness. [Moo02] describes an approach to parsing with ResCG that covers the full range of Lambek calculi.

The *parsing problem* for an ResCG can be defined as follows: Given a grammar $\mathcal{G}$ with lexicon $\mathcal{L}$ and set of structural rules $\mathcal{R}$, an input sentence $\mathcal{E} = w_1 ... w_n$ and a goal type t_G prove that $\mathcal{E}$ can be assigned a tree of type t_G in $\mathcal{G}$. We call $\mathcal{E}, t_G$ an *input* to $\mathcal{G}$ if $\mathcal{G}$'s lexicon $\mathcal{L}$ contains entries $\langle w_i, \varphi_i \rangle$ for all the words in $\mathcal{E}$.

The *generate & test* approach to ResCG, basically as shown in Figure 3 (a), then proceeds as follows. Given the types for the words of $\mathcal{E}$, it constructs all trees with

Given a grammar $\mathcal{G}$, an input sentence $\mathcal{E} = w_1 ... w_n$ and a goal type t_G prove that $\mathcal{E}$ can be assigned a tree of type t_G in $\mathcal{G}$.

(a)

1. Construct a set of starting trees, combining the lexical types that $\mathcal{G}$ assigns to the words w_i in $\mathcal{E}$ in all possible ways (ignoring the word order in $\mathcal{E}$) to form t_G;
2. Take a starting tree, try to rewrite it into a tree that has $\mathcal{E}$ as its yield, using any applicable structural rules of $\mathcal{G}$.
3. If rewriting is successful, we have an analysis for $\mathcal{E}$ in $\mathcal{G}$. Otherwise, repeat 2 for another starting tree.

(b)

1. Compute the set of starting trees for $\mathcal{E}$ that are licensed by stationary terms.
2. For each such starting tree τ_S: Compute a description of its set of destination trees that are licensed by stationary and transit terms.
3. If τ_S has licensed destination trees, validate whether some such tree constitutes a proof in $\mathcal{G}$.

Fig. 3. Parsing algorithms for ResCG: (a) *generate & test* approach, (b) constraint-based approach

root type t_G that match functions with arguments. For this, the algorithm uses a more powerful logic than the base logic in Figure 2: It uses LP to combine the function- and argument-types of words in every possible way, irrespective of the linear order in which these words occur in $\mathcal{E}$. In this paper, we call these graphs combining function- and argument-types the *starting trees*, as illustrated in Figure 4. We denote the set of all starting trees for $\mathcal{E}$ and t_G as $\mathsf{starting}_{t_G}(\mathcal{E})$.

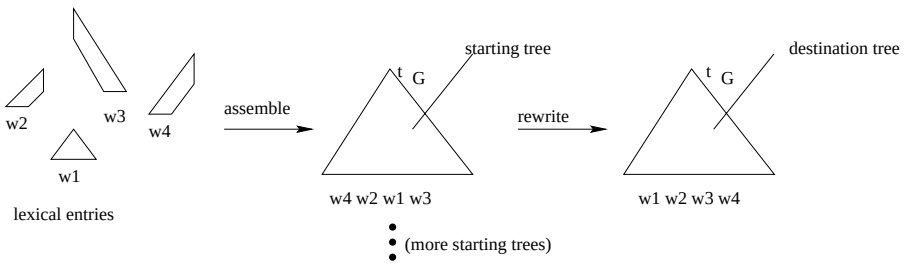


Fig. 4. Obtaining starting trees and destination trees from lexical entries

Suppose we choose $\mathcal{E}$ to be sentence (2), and t_G to be $relc$. Then, for this example there are four different starting trees, shown in Figure 5. The nonterminal node labels are the modes of the slashes in the types. (We simplify modes $\circ_\mu$ to μ in all figures.)

In the second phase of the *generate & test* computation, each starting tree is rewritten using the structural rules. If rewriting can produce a tree in which the leaves read from left to right (the *yield* of the tree) are $\mathcal{E}$, the parse is successful.

We propose to replace the *generate & test* strategy by the constraint-based approach shown in Figure 3 (b). It restricts the search space by restricting the number of starting trees: Some starting trees that could never lead to a successful parse are never generated

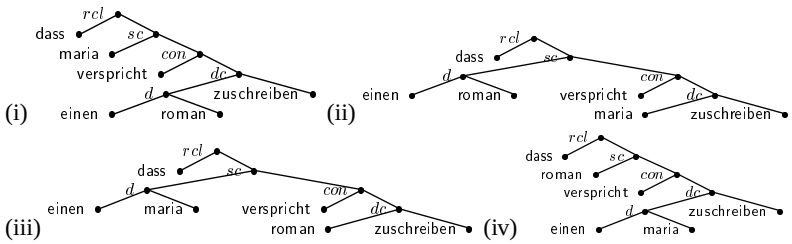


Fig. 5. All starting trees for sentence (2) (same for (1) and (3))

in the first place (step 1 of the algorithm); others are eliminated before any rewriting is applied (step 2). Additionally, the information we gather can be used to guide and restrict the validation phase in step 3.

In step 2 of the algorithm, we work with a description of the *destination trees* for a starting tree, as illustrated in Figure 4. The set of destination trees for a starting tree τ_S and an expression $\mathcal{E}$, $\mathsf{destination}_{\mathcal{E}}(\tau_S)$, is the set of all trees τ with yield $\mathcal{E}$ such that there exists an id-mapping from τ_S to τ.

In a destination tree, the nodes of the starting tree τ_S may be rearranged, and it has the sentence $\mathcal{E}$ as its yield, i.e. the words are in the right order. Intuitively, a destination tree is a possible outcome of rewriting τ_S. We use a tree description language to give a single, compact description of the set of all destination trees for τ_S, and we use *stationary terms* and *transit terms* that describe restrictions on possible outcomes of rewriting to constrain this description. If a starting tree is left with an inconsistent description (i.e. there is no destination tree given $\mathcal{G}$), then we can eliminate it.

In step 3, if we have not discarded τ_S, we reuse information we have gathered during the computation of transit terms to verify (by rewriting) whether any of τ_S's destination trees forms a proof in $\mathcal{G}$.

3 Stationary and transit terms

In this section we discuss the two main concepts of our approach in more detail: stationary and transit terms. In §4 we show how to use these concepts to prune the search space.

Stationary terms. Consider the mode dc in Figure 1: In all rules in which it occurs (rules [1] and [3]), its left child B stays constant from the input to the output side of the rule. If a tree contains a node labelled $\circ_{dc}$, that node is *left stationary*: Rewriting can work inside its left subtree, but it will never move a node out of the left subtree, or into it from outside. A node labelled $\circ_d$ will be both left and *right stationary*, since no structural rule in our fragment involves d. [3]

A stationary term for $\circ_\mu$ describes whether some other node remains in the same position relative to $\circ_\mu$ ($\Downarrow$), moves above $\circ_\mu$ ($\Uparrow$), or changes position under $\circ_\mu$ ($\Leftrightarrow$) in R.

[3] Note that we can derive this information offline from the grammar's lexicon and structural rules.

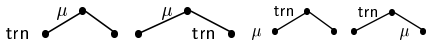
Given a node u of a tree τ, we write $\mathbb{L}_\tau(u)$ and $\mathbb{R}_\tau(u)$ for the subtrees starting at u's left and right child, and $\mathbb{S}ub_\tau(u)$ for the subtree with root u.

Definition 1 (Stationary terms, stationary nodes). *Given a set $\mathcal{R}$ of structural rules and a rule $R = \mathbb{I} \to \mathbb{O}$ from $\mathcal{R}$. Let v_i be a node of $\mathbb{I}$ labelled by some algebraic product $\circ_\mu$, and let $v_o = \mathbf{idmap}_R(v_i)$. Then $\circ_\mu(c_\mathbb{L}, c_\mathbb{R})$ is a* stationary term *for R if the following condition holds:*

- *$c_\mathbb{L} = \Downarrow$ ($c_\mathbb{R} = \Downarrow$) iff exactly the variables from $\{A, B, C\}$ that occur in $\mathbb{L}_\mathbb{I}(v_i)$ also occur in $\mathbb{L}_\mathbb{O}(v_o)$ (iff exactly the variables that occur in $\mathbb{R}_\mathbb{I}(v_i)$ also occur in $\mathbb{R}_\mathbb{O}(v_o)$).*
- *Otherwise, $c_\mathbb{L} = \Leftrightarrow$ ($c_\mathbb{R} = \Leftrightarrow$) iff exactly those variables from $\{A, B, C\}$ that occur in $\mathbb{S}ub_\mathbb{I}(v_i)$ still occur in $\mathbb{S}ub_\mathbb{O}(v_o)$.*
- *Otherwise $c_\mathbb{L} = \Uparrow$. (Otherwise $c_\mathbb{R} = \Uparrow$.)*

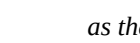
Let τ be a tree with a nonterminal node u labelled by an algebraic product $\circ_\mu$. Let $\mathsf{statterms}(\circ_\mu)$ *be the set of all stationary terms for $\mathcal{R}$ with root node $\circ_\mu$. Then u is* stationary *iff for all $t \in \mathsf{statterms}(\circ_\mu)$ no child is $\Uparrow$;* left stationary *iff for all $t \in \mathsf{statterms}(\circ_\mu)$ the left child is $\Downarrow$; and* right stationary *iff for all $t \in \mathsf{statterms}(\circ_\mu)$ the right child is $\Downarrow$.* □

The transit relation, transit terms. Whereas rewriting leaves some structures intact, it changes others. The *transit relation* charts how structures may travel, and the trees such travel gives rise to. We describe these changes in terms of *patterns*. A pattern is a parent/child pair of nodes where one node travels (the "transit node" *trn*) and the other node provides the context in which *trn* is allowed to move. We have the following schemata of patterns:



We also write the first pattern as $(\mathit{trn})\mu$, and analogously for the other three. We use the letter $\mathbb{P}$ to denote patterns. A pattern $\mathbb{P}$ occurs in a tree *at* the node u if u is in the position of the transit node *trn* in $\mathbb{P}$. In that case, $\mathbb{P}$ also occurs *for* the label of the node u. For example, the pattern $\circ_{con}(\mathit{trn})$ describes a transit node that is the right child of a $\circ_{con}$-labelled node. This pattern occurs in the starting tree (i) of Figure 5 for the label $\circ_{dc}$: $\circ_{con}(\circ_{dc})$.

To construct the transit relation, we look at each structural rule in the grammar to chart the patterns that change. For example, by rule [1] the right child of a $\circ_{dc}$-labelled node can become the right child of a $\circ_{con}$-labelled node. We record this change as $\circ_{dc}(\mathit{trn}) \xrightarrow{trns} \circ_{con}(\mathit{trn})$. Similarly, focusing now on the parent of the node at which the rule is applied, the parent has a con node as child before rule application, and a dc node afterwards. We formalize this latter point as the *parent extension*.

Definition 2 (Transit relation). *For a tree τ, we define the* left parent extension of τ *as the tree*  *and the* right parent extension *as the tree*  .

Let $\mathcal{R}$ be a set of structural rules. The transit relation $\xrightarrow{trns}$ *is defined as follows: $\mathbb{P} \xrightarrow{trns}_R \mathbb{P}'$ iff $R = \mathbb{I} \to \mathbb{O}$ is a rule in $\mathcal{R}$ and $\mathbb{P}, \mathbb{P}'$ are patterns such that $\mathbb{P}$ occurs in a parent extension of $\mathbb{I}$ at a node u, $\mathbb{P}'$ occurs in the same parent extension of $\mathbb{O}$ at $\mathbf{idmap}_R(u)$, and $\mathbb{P} \neq \mathbb{P}'$.* □

The transit relation for our example fragment is charted in Figure 6. The edges are annotated with the structural rules they stem from, and with conditions that we explain below.

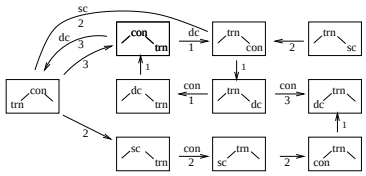


Fig. 6. The transit relation for Figure 1

Definition 3 (Transit term). *Combining a pattern $\mathbb{P}$ with a node label ℓ, we get a* transit term $\mathbb{P}(\ell)$. *We use that term to express that the pattern $\mathbb{P}$ occurs for the node label ℓ. The* inverse *of a transit term consisting of pattern $\circ_\mu(trn)$ and label $\circ_\nu$ is the term consisting of the pattern $trn(\circ_\nu)$ and the label $\circ_\mu$. Likewise, the transit term consisting of pattern $(trn)\circ_\mu$ and label $\circ_\nu$ is the inverse of the pattern $(\circ_\nu)trn$ with label $\circ_\mu$.* □

Inverse transit terms describe the same parent/child pair, but the focused transit node is different.

The logical roadmap. We combine the transit relation with a starting tree τ_S to form a *logical roadmap*.

Definition 4 (Log. roadmap). *Given an input $\mathcal{E}$, t_G, a* logical roadmap *for a grammar $\mathcal{G}$ and a starting tree $\tau_S \in \mathsf{starting}_{t_G}(\mathcal{E})$ is a tuple $\mathsf{rmap} = \langle \tau_S, \xrightarrow{trns} \rangle$ of τ_S and the transit relation $\xrightarrow{trns}$ of $\mathcal{G}$.* □

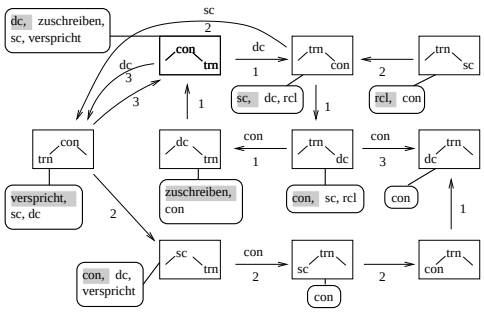


Fig. 7. Saturated logical roadmap for tree (i)

Using the logical roadmap we compute an approximation of possible transit terms that we can obtain by rewriting τ_S. This computation is done by *saturating* the logical

roadmap (Definition 5). We start by annotating the transit relation with all labels for which a pattern occurs in τ_S, and then we propagate these labels through the graph. A pattern annotated with a label is a transit term. A saturated logical roadmap for starting tree (i) is given in Figure 7. In the figure, the labels read off from τ_S are shaded. According to the following definition (Definition 5), the labels are propagated along the edges of the graph **r.transit**, and whenever we attach a label to a pattern, we also get the inverse transit term **(r.par/ch)**.[4] This simple algorithm can be restricted by further conditions to rule out pattern propagations that can never model an actual rewriting sequence. We present two possible such conditions: First, if an edge is annotated with a mode, only this mode can traverse the edge. In our example, the right child of $\circ_{con}$ can become the left child of $\circ_{con}$ by rule [3], but only if this child is labelled $\circ_{dc}$ **(c.label)**. Second, we do not propagate a label $\circ_\mu$ to a pattern $(\mathit{trn})\circ_\mu$ since in our starting tree each label occurs only once **(c.resource)**.

Definition 5 (Computing the transit set). *Given a logical roadmap* $\mathsf{rmap} = \langle \tau_S, \xrightarrow{trns} \rangle$, *a set* S *of transit terms, and a set* C *of conditions. Then the algorithm* $\mathsf{walk}(\mathsf{rmap}, C, S)$ *consists of the following rules that add pattern s to a set of transit terms* $\mathsf{transset}(\mathsf{rmap})$.

(r.transit) $\mathbb{P}(\ell) \to \mathbb{P}'(\ell)$ *if* $\mathbb{P} \xrightarrow{trns}_R \mathbb{P}'$ *and* C.
(r.par/ch) $\mathbb{P}(\nu) \to \mathbb{P}'(\mu)$ *if* $\mathbb{P}(\nu)$ *and* $\mathbb{P}'(\mu)$ *are inverse transit terms.*

The algorithm stops as soon as no rule can add a new node label anymore to $\mathsf{transset}(\mathsf{rmap})$ *(i.e.* $\mathsf{transset}(\mathsf{rmap})$ *is saturated).*

The set C *of conditions that we are going to use in this paper is* $C = \{$**(c.label)**, **(c.resource)**$\}$ *with*

(c.label) $\mathbb{P}$ *occurs in a parent extension of* $\mathbb{I}$ *either for* ℓ*, where* ℓ *is an algebraic product, or it occurs, but not for any algebraic product.*
(c.resource) *either* ℓ *does not occur in* $\mathbb{P}'$*, or* ℓ *occurs in* τ_S *at least twice.*

Let $\mathcal{G}$ *be a grammar with input* $\mathcal{E}, t_G$*, let* $\tau_S \in \mathsf{starting}_{t_G}(\mathcal{E})$ *be a starting tree, and let* rmap *be the logical roadmap for* $\mathcal{G}$ *and* τ_S*. Let* $S_0 := \{\mathbb{P}(\ell) \mid \mathbb{P}$ *is a transit pattern that occurs in* τ_S *for the label* $\ell\}$*. Then the* set of transit terms for rmap, $\mathsf{transset}(\mathsf{rmap})$, *is the result of* $\mathsf{walk}(\mathsf{rmap}, C, S_0)$. □

Saturation explores all possible rewriting sequences from τ_S *at the same time* without having to perform actual rewriting, and such that for common subsequences of two rewriting sequences the computation is done only once. We show in §6 that the time complexity of constructing a saturated logical roadmap is linear in the length of $\mathcal{E}$.

4 A constraint-based algorithm

In the previous section, we have introduced *stationary terms* and *transit terms*. In this section we show how these terms fit into the constraint-based algorithm for parsing

[4] We need to be able to change focus between parent and child. Adding the inverse means we can recognize changes to the context node of a pattern.

with resource-sensitive CG given in Figure 3 (b): We use stationary and transit terms to eliminate starting trees that can never lead to a successful parse. Again, we illustrate our approach with the fragment of §2.1 and sentence (2).

Step 1. We construct the set of starting trees that are *licensed by stationary terms*: If a node u of a starting tree τ_S is stationary, then no node can travel out from under u, and no node can travel below u by rewriting. Thus, it must be possible to order the leaves in $\mathbb{S}ub_{\tau_S}(u)$ such that they form an infix of $\mathcal{E}$, as any successful parse has to have $\mathcal{E}$ as its yield. If u is left stationary, then the same must hold for $\mathbb{L}_{\tau_S}(u)$, and similarly for right stationary products. If u is both left and right stationary, then it must be possible to order the leaves of both u's left and right subtrees such that they form adjacent infixes of $\mathcal{E}$. This condition rules out starting trees (iii) and (iv) of Figure 5 because they both have a subtree $(einen \circ_d maria)$, and the $\circ_d$-node is left as well as right stationary. This leaves us with two starting trees licensed by stationary nodes: (i) and (ii). We explain the remaining two steps of the algorithm using starting tree (i).

Step 2. In §3 we have defined a *destination tree* τ for a given starting tree τ_S as a possible result of rewriting performed on τ_S. It consists of the same nodes and node labels as τ_S, but they may be composed differently, and the yield of τ must be $\mathcal{E}$. In step 2 of the algorithm we construct a single, underspecified description of all possible destination trees for a given starting tree τ_S. This description is constrained by stationary and transit terms. If the description becomes inconsistent, then τ_S can never lead to a successful parse, and we can eliminate τ_S.

We use stationary nodes to constrain the way in which the nodes of a destination tree may be composed, and we use the logical roadmap to constrain the patterns that may occur in a destination tree: Each pattern occurring in a destination tree must either be present in τ_S, or it has been created by rewriting, in which case we must be able to read it off the logical roadmap.

To describe the set of all destination trees for a starting tree τ_S, we make statements about the nodes of τ_S and the new positions they may occupy in a destination tree. We need three constructs. *Labelling* gives a node's label and its children, *precedence* (some node is left of another in the tree) encodes the requirement that the yield of a destination tree be $\mathcal{E}$, and *dominance* (some node is above another in the tree) encodes licensing by stationary nodes: In any destination tree, a stationary node must dominate exactly the same nodes that it dominates in τ_S.

Tree description languages of this kind are studied e.g. in [DT99,KMN00]. We use the set-based language of [DT99]. The advantage is that we can exploit an existing constraint solver to draw inferences on our destination tree descriptions. This solver already enforces tree-shapedness.

Variables stand for nodes of the tree we describe. Each variable x is associated with five sets of variables: $up(x)$, $dn(x)$, $eq(x)$, $left(x)$, and $right(x)$. The set $up(x)$ contains all variables above x in the tree, and $dn(x)$ contains all variables below x. $eq(x)$ is the set of variables that describe the same node as x. $left(x)$ and $right(x)$ are the variables preceding x and preceded by x, respectively.

Figure 8 illustrates the five sets. They form a partition of the set of variables – every variable must stand in one of these five relations to x. Initially, these sets are only

(c1)	$label(x_u) = \mu(x_{uleft}, x_{uright})$	$x_u \in \mathcal{V}_{nont}$, u labelled μ in τ_S
(c2)	$label(x_u) = w$	$x_u \in \mathcal{V}_{leaf}$, u labelled w in τ_S
(c3)	$eq(x) = \{x\} \cup filler(x)$, $\lvert filler(x)\rvert = 1$,	$x \in \mathcal{V}_{hole}$
(c4)	$filler(x) \in \mathcal{V}_{\tau_S}$	$x \in \mathcal{V}_{hole}$
(c5)	$x_u \in left(x_v), x_v \in right(x_u)$	$x_u, x_v \in \mathcal{V}_{leaf}$, u labelled w_i, v labelled w_j in τ_S, $i < j$
(c6)	$eqdn(x_u) = \{x_v \in \mathcal{V}_{\tau_S} \mid v \text{ in } \mathbb{S}ub_{\tau_S}(u)\} \cup \{x_{vleft}, x_{vright} \mid x_v \in eqdn(x_u)\}$	$x_u \in \mathcal{V}_{nont}$, u stationary
(c7)	$eqdn(x_{uleft}) = \{x_v \in \mathcal{V}_{\tau_S} \mid v \text{ in } \mathbb{L}_{\tau_S}(u)\} \cup \{x_{vleft}, x_{vright} \mid x_v \in eqdn(x_{uleft})\} \cup \{x_{uleft}\}$	$x_u \in \mathcal{V}_{nont}$, u left stationary
(c8)	$eqdn(x_{uright}) = \{x_v \in \mathcal{V}_{\tau_S} \mid v \text{ in } \mathbb{R}_{\tau_S}(u)\} \cup \{x_{vleft}, x_{vright} \mid x_v \in eqdn(x_{uright})\} \cup \{x_{uright}\}$	$x_u \in \mathcal{V}_{nont}$, u right stationary
(c9)	$filler(x_{uleft}) \subseteq lefttransit(x_u)$	$x_u \in \mathcal{V}_{nont}$
(c10)	$filler(x_{uright}) \subseteq righttransit(x_u)$	$x_u \in \mathcal{V}_{nont}$

Fig. 9. Describing destination trees, given starting tree τ_S and expression $\mathcal{E} = w_1 \dots w_n$

partially determined. The solver then uses *propagation* and *distribution* to narrow the sets down until they are fully determined, and describe a destination tree.

Apart from these sets, we use $eqdn(x) = eq(x) \cup dn(x)$. Furthermore, $label(x) = \ell(x_1, \dots, x_n)$ states that x stands for a node labelled ℓ with children $x_1, \dots, x_n$ in that order.

Now to describe all destination trees for a given starting tree τ_S, we use a variable set $\mathcal{V}$ consisting of a set of tree variables $\mathcal{V}_{\tau_S} = \mathcal{V}_{nont} \uplus \mathcal{V}_{leaf}$, where $\mathcal{V}_{nont} = \{x_u \mid u \text{ nonterminal node of } \tau_S\}$ and $\mathcal{V}_{leaf} = \{x_u \mid u \text{ leaf node of } \tau_S\}$, and a set of hole variables $\mathcal{V}_{hole} = \{x_{uleft}, x_{uright} \mid x_u \in \mathcal{V}_{nont}\}$.

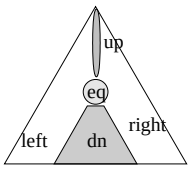


Fig. 8. Five variable sets

The constraints we use in propagation are given in Figure 9. A destination tree has exactly the same nodes and node labels as τ_S; the constraints (c1) and (c2) contribute these labels. The variables x_{uleft} and x_{uright} are *hole variables*. Each hole variable needs to be identified with *exactly one* $\mathcal{V}_{\tau_S}$-variable, its *filler*, (c3) and (c4). When we have identified each hole with a filler (and checked tree-shapedness with the constraint solver), then we have arranged the nodes of τ_S into a new tree. Furthermore, we know that the yield of a destination tree must be $\mathcal{E}$ (c5).

The constraints (c6) through (c10) are the most interesting ones: They involve stationary and transit terms. Stationary terms impose strong restrictions on the possible structure of a destination tree: A stationary node will dominate exactly the same nodes in any destination tree that it dominated in τ_S (c6). Likewise, if a node is left stationary, then its left child will dominate exactly the same nodes in any destination tree that it dominated in τ_S (c7), and if it is right stationary, the same holds for its right child (c8).

The constraints (c9) and (c10) use transit terms: We define sets $lefttransit(x_u)$, $righttransit(x_u)$ for each variable $x_u \in \mathcal{V}_{nont}$. Suppose the node u is labelled ℓ in τ_S, and the node v is labelled ℓ'. Then the set $lefttransit(x_u)$ contains x_v iff the set of transit terms for the logical roadmap (i.e. for $\mathcal{G}$ and τ_S) contains the transit term consisting of the pattern $(trn)\ell$ and the label ℓ'. (This can be the case either because the

label ℓ' is attached to the pattern $(\mathit{trn})\ell$ in the saturated logical roadmap, or the pattern $(\mathit{trn})\ell$ is not in the roadmap but it occurs in τ_S with ℓ' taking the role of the transit node *trn*.) The $righttransit$ set is defined accordingly, using the pattern $\ell(\mathit{trn})$ instead. So the constraints (c9, c10) state that whenever we fill a hole, we must do it such that we obtain a pattern licensed by the logical roadmap, i.e. a pattern of which we predict that it may arise by rewriting.

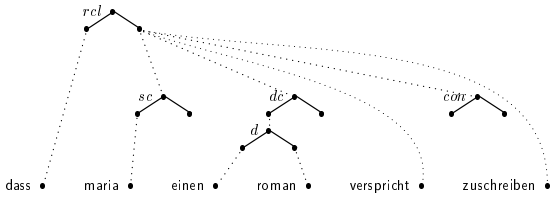


Fig. 10. Initial constraint describing all destination trees for starting tree (i) and sentence (2) (before inferences)

When we create a destination tree description for starting tree (i) in this way, we arrive at Figure 10: The edges from a labelled node to its children are drawn as solid lines. If we just know that x dominates y, this is drawn as a dotted line from x to y. The dotted lines in Figure 10 all stem from stationary nodes: The nodes labelled $\circ_{rcl}$ and $\circ_d$ are both left and right stationary, while $\circ_{sc}$ and $\circ_{dc}$ are left stationary.

Now the constraint solver can draw inferences to narrow down the destination tree description. We say x_{sc} for short if we mean the variable labelled $\circ_{sc}$, and analogously for the other labels.

We know that $x_{rclleft}$ dominates just x_{dass}, but $x_{rclleft}$ must be identified with a filler that it dominates, so we must identify $x_{rclleft}$ with x_{dass}. Likewise, the left child of x_{sc} must be x_{maria}, the children of x_d must be x_{einen} and x_{roman}, and the left child of x_{dc} must be x_d.

Now consider $x_{rclright}$: Since $x_{rclleft}$ is equal to x_{dass}, the next leaf to its right must be x_{maria}, so $x_{rclright}$ can only be identified with either x_{sc} or x_{con}, any other choice would violate this condition. By the same argument from precedence, we get the following restrictions:

$$\begin{aligned} filler(x_{rclright}) &\subseteq \{x_{sc}, x_{con}\} \\ filler(x_{scright}) &\subseteq \{x_{dc}, x_{con}\} \\ filler(x_{dcright}) &\subseteq \{x_{con}, x_{verspricht}\} \end{aligned}$$

However, by constraint (c10), $filler(x_{dcright}) \subseteq \{x_{con}, x_{zuschreiben}\}$ (cf. Figure 7) so the right child of x_{dc} must be x_{con}. Hence the right child of x_{sc} must be x_{dc} and the right child of x_{rcl} must be x_{sc}. So by using stationary and transit terms, we have narrowed down the description of destination trees for τ_S to a single tree, shown in Figure 11.

Step 3. Using stationary nodes and the logical roadmap we can efficiently prune the search space. However, the logical roadmap is only an *approximation*: the conditions

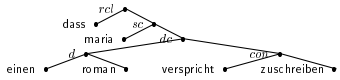


Fig. 11. The only destination tree for starting tree (i) and sentence (2)

on traversing an edge in the logical roadmap are less strict than the conditions on actually applying a structural rule – the saturated roadmap may contain patterns that cannot be obtained using just the grammar. Thus, the approximation comes with a definite computational gain (cf. also §6) but also at a formal cost: Although we can prove that the first two steps of the algorithm in Figure 3 are *sound*, they are not complete. Therefore, we need an additional step of *validation*, checking whether a destination tree can be obtained from a starting tree, using the grammar's structural rules. [5] To validate a destination tree τ against a starting tree τ_S, we create a *validation table* in a chart parsing-like fashion. However, rather than words the validation table contains *itineraries* read off the logical roadmap. An itinerary is a sequence of structural rule-applications changing τ_S into τ. Suppose we have a starting tree τ_S for which the destination tree description is satisfiable, and τ is one of the destination trees described. Then we put up an *itinerary* for each transit term $\mathbb{P}(\ell)$ that occurs in the destination tree τ: An itinerary for $\mathbb{P}(\ell)$ is a sequence $\mathbb{P}_0(\ell_0)R_0\mathbb{P}_1(\ell_1)\ldots R_{n-1}\mathbb{P}_n(\ell_n)$ that starts with some transit term $\mathbb{P}_0(\ell_0)$ occurring in τ_S and ends with the transit term $\mathbb{P}_n(\ell_n) = \mathbb{P}(\ell)$. Inbetween there must be a "path" of transit terms and structural rules read off the logical roadmap: for $0 \leq i < n$,

- either $\mathbb{P}_i(\ell_i) \rightarrow \mathbb{P}_{i+1}(\ell_{i+1})$ by **(r.transit)**, and $\mathbb{P} \stackrel{trns}{\longrightarrow}_{R_i} \mathbb{P}'$,
- or $\mathbb{P}_i(\ell_i) \rightarrow \mathbb{P}_{i+1}(\ell_{i+1})$ by **(r.par/ch)**, and $R_i = A \rightarrow A$.

All the itineraries for τ_S and τ are placed into a validation table, and are then used in a chart-like fashion. Initially, we place a $\bullet$ before the first structural rule R_0 in each itinerary. Now, to rewrite τ_S into τ, the next structural rule we choose must always be one directly to the right of some $\bullet$. Applying a structural rule R moves $\bullet$'s on to the next rule occurring in the itinerary: At least one $\bullet$ that occurs to the left of R in some itinerary must be moved; but a single rule application may change more than one pattern.

We obtain all possible itineraries already as a side product when saturating the logical roadmap. (Thus, we now simply reuse this information, rather than letting validation be a *generate & test* step in disguise.) If we can use the itineraries to rewrite τ_S into τ, then τ is valid on the grammar.

Finally, how much validation we need to do depends on the strength of the conditions on traversing an edge of the transit relation. Stronger conditions than those used here are possible, and would move computation on the logical roadmap closer to logical completeness with respect to the grammar.

[5] We can prove that, if there exists a validation for a destination tree, then we find it – thus, the overall computation *is* complete.

5 Licensing, soundness and completeness

The core of our constraint-based inferences is formed by the licensing steps. In this section we give a formal definition of the licensing steps that we have used in the previous section, and prove theorems about the soundness of licensing. We also state a theorem about the obtainability of a validation and the resulting completeness.

To make the definitions given below more readable, we use the following notation: Given a grammar $\mathcal{G}$ with input $\mathcal{E}$, t_G, we call a pair of trees $\langle \tau_S, \tau \rangle$ a start/destination pair iff $\tau_S \in \mathsf{starting}_{t_G}(\mathcal{E})$ and $\tau \in \mathsf{destination}_{\mathcal{E}}(\tau_S)$.

We have used *licensing by stationary terms* in steps 1 and 2 of the algorithm in Figure 3 (b). Licensing by stationary terms means that we can eliminate starting trees by comparing subtrees with stationary roots against the expression $\mathcal{E}$, and we can restrict the form of destination trees by inferring dominance information from stationary nodes. Formally, licensing by stationary terms can be defined as follows.

Definition 6 (Licensing by stationary terms). *Let $\mathcal{G}$ be a grammar with input $\mathcal{E}, t_G$ and start/destination pair $\langle \tau_S, \tau \rangle$. Let S be the set of stationary, left stationary, or right stationary nonterminal nodes in τ_S. For each node $u \in S$, let*

$$v_u = \begin{cases} u & \text{for } u \text{ stationary} \\ u\text{'s left child} & \text{for } u \text{ left stationary} \\ u\text{'s right child} & \text{for } u \text{ right stationary} \end{cases}$$

Then the starting tree τ_S *is* licensed by stationary terms *iff two conditions hold: First, for each node $u \in S$, some permutation of the yield of $\mathbb{S}ub_{\tau_S}(v_u)$ is an infix of $\mathcal{E}$. Second, for each node u that is both left and right stationary, some permutations of the yields of $\mathbb{L}_{\tau_S}(u)$ and $\mathbb{R}_{\tau_S}(u)$ are adjacent infixes of $\mathcal{E}$.*

The destination tree τ *is* licensed by stationary terms *iff there exists an id-mapping function* $\mathbf{idmap}$ *from τ_S to τ such that for each node $u \in S$, and all nodes w in τ_S, w lies in $\mathbb{S}ub_{\tau_S}(v_u)$ iff $\mathbf{idmap}(w)$ lies in $\mathbb{S}ub_{\tau}(\mathbf{idmap}(v_u))$.* □

We have used *licensing by transit terms* in step 2 of the algorithm: A hole variable must choose a filler that will form a valid transit term together with the hole's parent, i.e. a term that either occurs in the starting tree or in the saturated logical roadmap.

Definition 7 (Licensing by transit terms). *Let $\mathcal{G}$ be a grammar with input $\mathcal{E}, t_G$ and start/destination pair $\langle \tau_S, \tau \rangle$. Let* rmap *be the logical roadmap for $\mathcal{G}$ and τ_S. Then the destination tree τ is* licensed by transit *iff for all patterns $\mathbb{P}$ occuring in τ for some product $\circ_\mu$, the term $\mathbb{P}(\circ_\mu)$ is in* $\mathsf{transset}(\mathsf{rmap})$. □

Licensing by stationary and transit terms is sound: If a destination tree is not licensed, it cannot be reached by rewriting using the grammar itself.

For the two following theorems, let $\mathcal{G}$ be a grammar with lexicon $\mathcal{L}$, structural rules $\mathcal{R}$, a base logic as in Figure 2, and an input $\mathcal{E}$, t_G.

Theorem 1 (Licensing by stationary terms is sound). *Let $\tau_S \in \mathsf{starting}_{t_G}(\mathcal{E})$ be a starting tree that can be rewritten to $\tau \in \mathsf{destination}_{\mathcal{E}}(\tau_S)$. Then both τ_S and τ are licensed by stationary terms.*

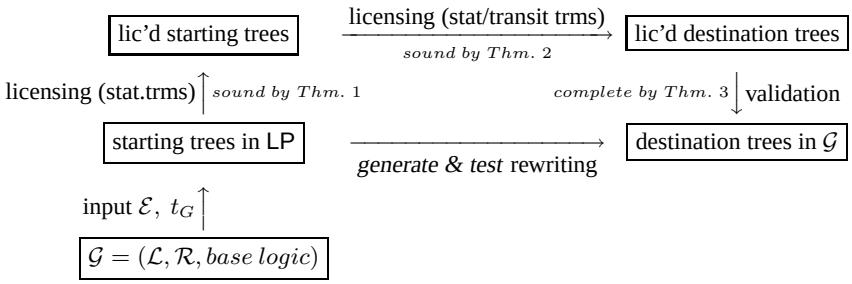


Fig. 12. Abstract interpretation, soundness & completeness

Proof. It can easily be shown that when a structural rule R is used to rewrite a tree τ_0 to τ_1, and u is a stationary node of τ_0, then a node v is in $\mathbb{S}ub_{\tau_0}(u)$ iff $\mathbf{idmap}_R(v)$ lies in $\mathbb{S}ub_{\tau_1}(\mathbf{idmap}_R(u))$, and similarly for left and right stationary nodes. So if τ_S or τ were not licensed by stationary terms, there could not be a successful rewriting sequence. □

Theorem 2 (Licensing by transit is sound). *Let $\tau_S \in \mathsf{starting}_{t_G}(\mathcal{E})$ be a starting tree that can be rewritten to $\tau \in \mathsf{destination}_{\mathcal{E}}(\tau_S)$. Then τ is licensed by transit.*

Proof. Let rmap be the logical roadmap for $\mathcal{G}$ and τ_S. Let $\mathbb{P}$ be a pattern occurring in τ for the label ℓ. We show that $\mathbb{P}(\ell) \in \mathsf{transset}(\mathsf{rmap})$, proceeding by induction on the length of the rewriting sequence starting in $\tau_0 = \tau_S$ and ending in $\tau_n = \tau$.

If $\mathbb{P}$ occurs in τ_0 for a label ℓ, then $\mathbb{P}(\ell)$ is in the set S_0 of Def. 5 and hence in $\mathsf{transset}(\mathsf{rmap})$.

Now suppose that for some $n \geq 1$, τ_{n-1} gets rewritten to τ_n by the structural rule $R = \mathbb{I} \rightarrow \mathbb{O}$. If $\mathbb{P}(\ell)$ occurs in τ_{n-1} already, we are done. So suppose otherwise, and suppose $\mathbb{I}$ occurs in τ_{n-1} at some node v that is the child of v'. Then $\mathbb{P}$ occurs in τ_n within the parent extension of $\mathbb{O}$ starting at $\mathbf{idmap}_R(v')$.

Suppose $\mathbb{P}$ occurs at $\mathbf{idmap}_R(v')$. Then the label of $\mathbf{idmap}_R(v)$ in τ_n is different from the label of v in τ_{n-1}. W.l.o.g. let v be labelled μ, $\mathbf{idmap}_R(v)$ labelled ν, and $\mathbb{P} = \mathit{trn}(\nu)$. Then by Def. 2, the transit relation contains $\mathit{trn}(\mu) \xrightarrow{trns}_R \mathit{trn}(\nu)$. By the inductive hypothesis, $\mathit{trn}(\mu)(\ell) \in \mathsf{transset}(\mathsf{rmap})$. Condition **(c.label)** is fulfilled: The pattern $\mathbb{P}$ occurs in a parent extension of $\mathbb{I}$, but not for any label that is an algebraic product. Now we check condition **(c.resource)**. If the label ℓ occurs in $\mathit{trn}(\nu)$, then $\ell = \nu$. Then τ_n contains at least two nodes labelled ν, and since structural rules only rearrange nodes, the same holds for τ_S. So in either case, whether $\ell = \nu$ or $\ell \neq \nu$, condition **(c.resource)** holds. Hence by the rule **(r.transit)**, $\mathit{trn}(\nu)(\ell) \in \mathsf{transset}(\mathsf{rmap})$, and by **(r.par/ch)**, $\ell(\mathit{trn})(\nu) \in \mathsf{transset}(\mathsf{rmap})$ as well.

For the other possible positions of $\mathbb{P}$, the argument is analogous. Note that if a pattern $\mathbb{P}'$ occurs in $\mathbb{I}$ for some algebraic product μ, then $\mathbb{P}'$ occurs in τ_{n-1} for the same label μ because $\mathbb{I}$ matches the tree τ_{n-1} at v. Hence the edge $\mathbb{P}'(\mu) \xrightarrow{trns}_R \mathbb{P}(\mu)$ can still be traversed by condition **(c.label)**. □

Construction of stationary terms for $\mathcal{G}$	–	Computed offline
Licensing starting trees by stationary terms	$\mathcal{O}(\lvert\mathcal{E}^3\rvert)$	For every node v of τ_S that is (left, right) stationary, mark each word in $\mathcal{E}$ occuring in the subtree with root v; then check if marked nodes form an infix of $\mathcal{E}$
Construction of compact description of dest. trees	$\mathcal{O}(\lvert\mathcal{E}^2\rvert)$	
Construction of a saturated logical roadmap	$\mathcal{O}(\lvert\mathcal{E}\rvert)$	The number of node labels on each node of the transit relation graph is bounded by the number of nodes in τ_S.
Checking satisfiability of dominance, precedence	$\mathcal{O}(\lvert\mathcal{E}^2\rvert)$	[KMN00]
Checking satisfiability against logical roadmap	$\mathcal{O}(\lvert\mathcal{E}\rvert)$	The destination tree description contains $\mathcal{O}(\lvert\mathcal{E}\rvert)$ transit patterns that need to be checked against the saturated roadmap for licensing by transit.

Fig. 13. Complexity results for individual steps of Figure 3

So licensing by stationary and transit terms is sound – we eliminate only starting trees that do not lead to a successful parse. Together with the validation step that we have sketched in §4, we gain a complete algorithm for ResCG parsing:

Theorem 3 (Validation is complete). *Let $\tau_S \in \mathsf{starting}_{t_G}(\mathcal{E})$ be a starting tree that can be rewritten to $\tau \in \mathsf{destination}_{\mathcal{E}}(\tau_S)$. Then there is a validation for τ from τ_S.*

The computation of stationary and transit terms form an *abstract interpretation* of a computation with the grammar: Using sound and complete operations, we compute on licensed descriptions rather than with the grammar itself. Figure 12 recapitulates the interrelation between the abstract interpretation, the licensing steps, and the soundness and completeness results with respect to a grammar $\mathcal{G}$.

6 Complexity

In this section we mention some complexity results for the algorithm we propose in Figure 3 (b). The baseline is formed by the abstract complexity results for ResCG: Without the restriction to linear structural rules, ResCG is Turing Complete; limited to at most LP) it is NP-complete, cf. [Moo02].

Figure 13 gives the (time) complexity of various computation steps our algorithm performs. One point to note is that stationary terms, which account for a lot of pruning, are computed offline since they rely just on $\mathcal{G}$. Furthermore, most construction and satisfaction steps in Figure 3 are either linear or low-polynomial. Thus, while the worst-case complexity of proof search using itineraries is the same as that of proof search in LP, we can reduce the search space in steps of low complexity – and potentially obtain significant reductions.

7 Conclusions

We presented a constraint-based approach to parsing with resource-sensitive CG that uses propagators derived from the logical formulation of a grammar $\mathcal{G}$ to prune the (proof) search space. Although the *generate & test* approach to parsing with resource-sensitive CGs and our approach have the same worst-case complexity, our approach has a (potentially) better average-case complexity since we reduce the search space using steps that are mostly linear or low-polynomial in the length of the sentence, not exponential as in *generate & test*. We plan to extend the approach to include unary modalities $\Diamond$ and $\Box^{\downarrow}$ (available in [Moo02], but not in [Hep95]) and to obtain practical results using treebank-grammars and an implementation of our algorithm. (Parsing with treebank grammars leads to exponential explosion in the *generate & test* approach, (Moot,p.c.)).

References

[Car95] Bob Carpenter. The Turing-completeness of multimodal categorial grammar. Unpublished manuscript. Carnegie Mellon University, 1995.

[DD01] Denys Duchier and Ralph Debusmann. Topological dependency trees: A constraint-based account of linear precedence. In *Proceedings ACL'01*, France, 2001.

[dG99] Philippe de Groote. A dynamic programming approach to categorial deduction. In *Proceedings of CADE'99*, 1999.

[DT99] Denys Duchier and Stefan Thater. Parsing with tree descriptions: a constraint-based approach. In *Proceedings NLULP'99*, New Mexico, 1999.

[Hep92] Mark Hepple. Chart parsing lambek grammars: Modal extensions and incrementality. In *Proceedings COLING'92*, France, 1992.

[Hep95] Mark Hepple. Mixing modes of linguistic description in categorial grammar. In *Proceedings EACL'95*, Ireland, 1995.

[Hep98] Mark Hepple. Memoisation for glue language deduction and categorial parsing. In *Proceedings COLING-ACL'98*, Canada, 1998.

[JL87] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *ACM Symposium on Principles of Programming Languages*, pages 111–119. The ACM Press, 1987.

[KMN00] Alexander Koller, Kurt Mehlhorn, and Joachim Niehren. A polynomial-time fragment of dominance constraints. In *Proceedings ACL'00*, Hong Kong, 2000.

[Kön90] Esther König. The complexity of parsing with extended categorial grammars. In *Proceedings COLING'90*, Finland, 1990.

[Kru01] Geert-Jan M. Kruijff. *A Categorial-Modal Logical Architecture of Informativity: Dependency Grammar Logic & Information Structure*. PhD thesis, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic, April 2001.

[Lam58] Joachim Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–169, 1958.

[Moo97] Michael Moortgat. Categorial type logics. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*. Elsevier Science, 1997.

[Moo02] Richard Moot. *Proofnets for linguistic analysis*. PhD thesis, University of Utrecht, The Netherlands, 2002.

[Mor94] Glyn V. Morrill. *Type Logical Grammar: Categorial Logic of Signs*. Kluwer Academic Publishers, 1994.

[Mor96] Glyn V. Morrill. Memoisation of categorial proof nets: parallelism in categorial processing. In *Proceedings of the 1996 Roma Workshop*, Italy, 1996.

[MS98] Kim Marriott and Peter J. Stuckey. *Programming with Constraints*. Kluwer Academic Publisher, 1998.

[Pen97] Mati Pentus. Product-free Lambek calculus and context-free grammars. *The Journal of Symbolic Logic*, 62(2):648–660, June 1997.