Polynomially-parsable Unification Grammars

Hadas Peled

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE MASTER DEGREE

University of Haifa
Faculty of Social Sciences
Department of Computer Science

November, 2011

Polynomially-parsable Unification Grammars

By: Hadas Peled
Supervised By: Prof. Shuly Wintner

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE MASTER DEGREE

University of Haifa
Faculty of Social Sciences
Department of Computer Science

November, 2011

Approved by: ________________________________ Date:________
(supervisor)

Approved by: ________________________________ Date:________
(Chairman of M.Sc. Committee)

Acknowledgments

I would like to thank Prof. Shuly Wintner for his devoted guidance, and for sharing his wide knowledge and enthusiasm for the world of Unification Grammars.

Contents

Abstract V
List of Figures VII

1 Introduction **1**
1.1 Typed Unification Grammars 1
1.2 Constrained Unification Grammars 3
1.3 Range Concatenation Grammars 4
1.4 Structure of the thesis 12

2 Restricted Typed Unification Grammars **13**
2.1 Representing Lists of Terminals with TFSs 13
2.2 Restricted type signatures 15
2.3 Restricted TFS 17
2.4 Restricted lexicon 17
2.5 Restricted rules 18
2.6 Restricted Unification Grammars 19
2.7 Examples of Restricted TUG 19

3 Simulation of RTUG by RCG **25**
3.1 Mapping of type signature to RCG clauses 26
3.2 Mapping of UG rules to RCG clauses 26
3.3 Mapping the lexicon to RCG clauses 28
3.4 Examples 30

4 Simulation of RCG by RTUG **33**
4.1 Examples 36

5 A sketch of the proof **49**
5.1 Instantiated grammar 49
5.2 Direction 1: $\mathcal{L}_{RCG} \subseteq \mathcal{L}_{RTUG}$ 50
5.3 Direction 2: $\mathcal{L}_{RTUG} \subseteq \mathcal{L}_{RCG}$ 51

6 Conclusions **55**

Bibliography **57**

A Representing bidirectional lists with TFSs **59**
A.1 Restrictions over node TFSs and bi_list TFSs 60
A.2 Explicit bi_lists 62

A.3 Bi_list operations 64

B Proofs of the main results **69**
B.1 Instantiated grammar 69
B.2 Direction 1: $\mathcal{L}_{RCG} \subseteq \mathcal{L}_{RTUG}$ 73
B.2.1 $L(G) \subseteq L(G_{tug})$ 73
B.2.2 $L(G_{tug}) \subseteq L(G)$ 82
B.3 Direction 2: $\mathcal{L}_{RTUG} \subseteq \mathcal{L}_{RCG}$ 84
B.3.1 Predicate subsumption 85
B.3.2 $L(G) \subseteq L(G_{rcg})$ 86
B.3.3 $L(G_{rcg}) \subseteq L(G)$ 92

Polynomially-parsable Unification Grammars

Hadas Peled

Abstract

Unification grammars (UG) underlie several contemporary linguistic theories, including Lexical-functional Grammar (LFG) and Head-driven Phrase-structure Grammar (HPSG). UG is an attractive grammatical formalism, *inter alia*, because of its expressivity: it facilitates the expression of complex linguistic structures and relations. Formally, UG is Turing-complete, generating the entire class of recursively enumerable languages (Francez and Wintner, 2012, Chapter 6). This expressivity, however, comes at a price: the universal recognition problem is undecidable for arbitrary unification grammars (Johnson, 1988).

Several constraints on UGs were suggested in order to reduce the expressiveness of the formalism and thereby guarantee more efficient processing. A series of works (see Jaeger et al. (2005) and references therein) define various *off-line parsability* constraints, which guarantee the decidability of the universal recognition problem, but not its tractability. The recognition problem for off-line parsable grammars is NP-hard (Barton et al., 1987). Other works define highly restricted versions of UG, such that efficiency of parsing is ensured: Feinstein and Wintner (2008) define *non-reentrant* UGs, which generate exactly the class of context-free languages; and *one-reentrant* UGs, which generate the class of tree-adjoining languages (TALs). Keller and Weir (1995) define *PLPATR*, an extension of Linear Indexed Grammars that manipulates feature structures rather than stacks, which has a polynomial-time parsing algorithm. PLPATR languages are included in the set of languages generated by Linear Context-Free Rewriting Systems. The expressivity and flexibility of these constrained formalisms, however, are severely limited, and seriously handicap the grammar designer.

In this work we define a constrained version of UG that is equivalent to Range Concatenation Grammar (RCG). RCG is a formalism that generates exactly the class of languages recognizable in deterministic polynomial time (Boullier, 1998b); specifically, it strictly contains the class of TALs (Boullier, 1998a). Boullier (1999) shows that RCG can express natural language phenomena such as Chinese numbers and German word scrambling, that lie beyond the expressive power of TALs. RCG is closed under union, concatenation, Kleene iteration, intersection and complementation (Boullier, 1998b). Since RCG has a polynomial parsing algorithm (Boullier, 1998b; Kallmeyer et al., 2009), a restricted version of UG that is equivalent to RCG (along with an efficient conversion procedure) guarantees the efficiency of parsing.

The main contribution of this work is thus a constrained version of UG that is on one hand expressive enough so as to allow the expression of complex linguistic structures in terms of typed feature structures that linguists favor, and on the other hand guarantees efficient processing for all grammars that can be expressed in the formalism.

List of Figures

1.1 The clauses of G_{SCR} . . . 11

2.1 An example TFS representing a bidirectional list of terminals . . . 15
2.2 An RTUG, G_{abc}, generating the language $a^n b^n c^n$ . . . 20
2.3 A derivation tree for the string *"aabbcc"* . . . 21
2.4 An RTUG, $G_{longdist}$ . . . 22
2.5 An RTUG, $G_{longdist}$ (continued) . . . 23
2.6 A derivation tree of the string *"Laban wondered whom Jacob loves"* . . . 24

4.1 The type signature of G_{prime} . . . 37
4.2 The root of the derivation tree . . . 39
4.3 Derivation subtree showing how the input word is collected . . . 40
4.4 Subtree showing actual simulation of G_{prime} (part 1) . . . 41
4.5 Subtree showing actual simulation of G_{prime} (part 2) . . . 42
4.6 The type signature of G_{SCR} . . . 43
4.7 The rules of G_{SCR} (part 1) . . . 44
4.8 The rules of G_{SCR} (part 2) . . . 45
4.9 Derivation tree of the string $n_2 n_3 n_1 v_1 v_2$ (part 1) . . . 46
4.10 Derivation tree of the string $n_2 n_3 n_1 v_1 v_2$ (part 2) . . . 47
4.11 Derivation tree of the string $n_2 n_3 n_1 v_1 v_2$ (part 3) . . . 48

Chapter 1

Introduction

Unification grammars (UG) underlie several contemporary linguistic theories, including Lexical-functional Grammar (LFG) and Head-driven Phrase-structure Grammar (HPSG). UG is an attractive grammatical formalism because of its expressivity: it facilitates the expression of complex linguistic structures and relations. Formally, UG is Turing-complete, generating the entire class of recursively enumerable languages (Francez and Wintner, 2012, Chapter 6). This expressivity, however, comes at a price: the universal recognition problem is undecidable for arbitrary unification grammars (Johnson, 1988).

In this work we define a constrained version of UG that guarantees efficient processing, while allowing the expression of complex linguistic structures. We do so by showing that the constrained UG is equivalent to Range Concatenation Grammar (RCG), a formalism that generates exactly the class of languages recognizable in deterministic polynomial time (Boullier, 1998b).

In this introduction we set up notation for the two formalisms this work deals with, Typed Unification Grammars (Section 1.1) and Range Concatenation Grammars (Section 1.3). In addition, in Section 1.2 we list other constraints on UG that were suggested in the past, in order to guarantee more efficient processing, at a price of reduced expressiveness.

1.1 Typed Unification Grammars

We assume familiarity with typed unification grammars, as formulated, e.g., by Carpenter (1992). For a partial function F, '$F(x)\downarrow$' (and similarly, '$F(x)\uparrow$') means that F is defined (undefined) for the value x. The following definitions recapitulate basic notions.

Definition 1 (Type hierarchy)**.** *A partial order $\sqsubseteq$ over a finite, non-empty set* TYPES *of types is a* **type hierarchy** *if it is bounded complete, i.e., if every up-bounded subset T of* TYPES *has a (unique) least upper bound, $\sqcup T$. If $t_1 \sqsubseteq t_2$ we say that t_1* **subsumes***, or is* **more general than***, t_2; t_2 is a* **subtype** *of (more* **specific** *than) t_1. We say that t_1 is an* ***immediate subtype*** *of t_2, denoted $t_2 \overset{\circ}{\sqsubset} t_1$ if $t_2 \sqsubseteq t_1$, $t_1 \neq t_2$, and for every $t' \in$* TYPES*, if $t' \sqsubseteq t_1$, then $t' \sqsubseteq t_2$. If t is such that for no $t' \neq t$, $t \sqsubseteq t'$, then t is a* ***maximal*** *type, or a* **species***. Let $\sqcap T$ be the greatest lower bound of the set T, if it exists. $\bot = \sqcup\emptyset$ is the most general type.*

Definition 2 (Appropriateness)**.** *Given a set of types* $\textsc{Types}$ *and a set of features* $\textsc{Feats}$*, an* ***appropriateness specification*** *is a partial function* $Approp : \textsc{Types} \times \textsc{Feats} \to \textsc{Types}$*, such that:*

- *for every* $f \in \textsc{Feats}$*, let* $T_f = \{t \in \textsc{Types} \mid Approp(t, f)\downarrow\}$*; then* $T_f \neq \emptyset$ *and* $Intro(f) = \sqcap T_f \in T_f$*.*

- *if* $Approp(t_1, f)\downarrow$ *and* $t_1 \sqsubseteq t_2$ *then* $Approp(t_2, f)\downarrow$ *and* $Approp(t_1, f) \sqsubseteq Approp(t_2, f)$*.*

A type t *is* ***featureless*** *if for every* $f \in \textsc{Feats}$*,* $Approp\,(t, f)\uparrow$*.*

Definition 3 (Type signatures)**.** *A* ***type signature*** *is a quadruple* $\langle \textsc{Types}, \sqsubseteq, \textsc{Feats}, Approp \rangle$*, where* $\langle \textsc{Types}, \sqsubseteq \rangle$ *is a type hierarchy and* $Approp : \textsc{Types} \times \textsc{Feats} \to \textsc{Types}$ *is an appropriateness specification.*

In this work we use the LKB notation for defining a type signature, where a subtype is listed below its super type, with increasing indentation. The features and the appropriate types of each type are listed in the same line as the type. For example, the following specification:

$\textsc{Types} = \{t_1, t_2, t_3, t_4\}$, $\textsc{Feats} = \{f_1, f_2\}$
t_1
  t_2 f_1: t_3 f_2: t_4
t_3
  t_4

represents a typed signature where $\bot \sqsubseteq t_1$, $t_1 \sqsubseteq t_2$, $\bot \sqsubseteq t_3$, $t_3 \sqsubseteq t_4$, and $Approp(t_2, f_1) = t_3$, $Approp(t_2, f_2) = t_4$.

Definition 4 (Typed feature graphs)**.** *A* ***typed feature graph*** $\langle Q, \bar{q}, \delta, \theta \rangle$ *is a directed, connected, labeled graph consisting of a finite, nonempty set of nodes* Q*, a root* $\bar{q} \in Q$*, a partial function* $\delta : Q \times \textsc{Feats} \to Q$ *specifying the arcs such that every node* $q \in Q$ *is accessible from* $\bar{q}$ *and a total function* $\theta : Q \to \textsc{Types}$ *marking the nodes with types.*

Let $\hat{\delta}$ be the reflexive-transitive closure of δ. In the sequel we abuse notation and refer to $\hat{\delta}$ as δ. Let $\textsc{Paths} = \textsc{Feats}^*$.

Definition 5 (Paths)**.** *The* ***paths*** *of a feature graph* A *are* $\Pi(A) = \{\pi \in \textsc{Paths} \mid \delta_A(\bar{q}_A, \pi)\downarrow\}$*.*

Definition 6 (Path value)**.** *for a feature graph* $A = \langle Q_A, \bar{q}_A, \delta_A, \theta_A \rangle$ *and a path* $\pi \in \Pi(A)$ *the* ***value*** $val_A(\pi)$ *of* π *in* A *is a feature graph* $B = \langle Q_B, \bar{q}_B, \delta_B, \theta_B \rangle$*, over the same signature as* A*, where:*

- $\bar{q}_B = \delta_A(\bar{q}_A, \pi)$

- $Q_B = \{q' \in Q_A \mid$ *for some* $\pi', \delta_A(\bar{q}_B, \pi') = q'\}$ *(*Q_B *is the set of nodes reachable from* $\bar{q}_B$*)*

- *for every feature* f *and for every* $q' \in Q_B$*,* $\delta_B(q', f) = \delta_A(q', f)$ *(*δ_B *is the restriction of* δ_A *to* Q_B*)*

- *for every $q' \in Q_B$, $\theta_B(q') = \theta_A(q')$ (θ_B is the restriction of θ_A to Q_B)*

Definition 7 (Reentrancy)**.** *Let $A = \langle Q, \bar{q}, \delta, \theta \rangle$ be a feature graph. Two paths $\pi_1, \pi_2 \in \Pi(A)$ are **reentrant** in A, iff $\delta(\bar{q}, \pi_1) = \delta(\bar{q}, \pi_2)$ implaying $val(\pi_1) = val(\pi_2)$.*

Definition 8 (Subsumption)**.** *Let $A_1 = \langle Q_1, \bar{q}_1, \delta_1, \theta_1 \rangle$ and $A_2 = \langle Q_2, \bar{q}_2, \delta_2, \theta_2 \rangle$ be two typed feature graphs over the same signature. A_1 **subsumes** A_2 (denoted by $A_1 \sqsubseteq A_2$) iff there exists a total function $h : Q_1 \to Q_2$, called a **subsumption morphism**, such that $h(\bar{q}_1) = \bar{q}_2$; for every $q \in Q_1$ and for every f such that $\delta_1(q, f)\downarrow$, $h(\delta_1(q, f)) = \delta_2(h(q), f)$; and for every $q \in Q_1$, $\theta_1(q) \sqsubseteq \theta_2(h(q))$.*

A **typed feature structure** (TFS) is an equivalence class of isomorphic feature graphs (ignoring the identities of the nodes). A **multi-rooted structure** (MRS) is a sequence of TFSs, with possible **reentrancies** (shared nodes) across the members of the sequence. Following the linguistic convention, we depict TFSs and MRSs as attribute-value matrices (AVMs) in the sequel. Example 6 (page 14) depicts a TFS represented as an AVM.

Definition 9 (Maximally specific TFS)**.** *A TFS F_1 is **maximally specific** if no TFS F_2 exists such that $F_1 \sqsubseteq F_2$.*

Definition 10 (Rules)**.** *A **rule** is an MRS of $n > 0$ TFSs, with a distinguished first element. The first elemnt is its **head** and the rest of the elements are the rule's **body**. We adopt a convention of depicting rules with an arrow ($\to$) separating the head from the body.*

Since a rule is simply an MRS, there can be reentrancies among its elements: both between the head and (some element of) the body and among elements in its body.

Definition 11 (Typed unification grammar)**.** *A **typed unification grammar** over a finite set* WORDS *of words and a type signature $\langle \text{TYPES}, \sqsubseteq, \text{FEATS}, Approp \rangle$ is a tuple $G = \langle \mathcal{R}, A_s, \mathcal{L} \rangle$, where $\mathcal{R}$ is a finite set of **rules**, each of which is an MRS, A_s is the **start symbol** (a TFS), and $\mathcal{L}$ is the **lexicon** which associates with each word $w \in$ WORDS a set of TFSs $\mathcal{L}(w)$.*

The **language** generated by a UG is defined in terms of a **derivation** relation over MRSs. See Carpenter (1992); Francez and Wintner (2012) for the details. Figure 2.2 (page 20) depicts a unification grammar and specifies the language it generates.

1.2 Constrained Unification Grammars

UG, as defined above, is Turing-equivalent (Francez and Wintner, 2012, Chapter 6). In other words, it generates the entire class of recursively enumerable languages. Consequently, the universal recognition problem for UG is undecidable (Johnson, 1988). Several constraints on UGs were suggested in order to reduce the expressiveness of the formalism and thereby guarantee more efficient processing.

Off-line parsability (OLP) constraints These constraints guarantee that the recognition problem for grammars that obey them is decidable (Jaeger et al., 2005). The idea behind all the OLP definitions

is to rule out grammars which license trees in which an unbounded amount of material is generated without expanding the frontier word. This can happen due to two kinds of rules: ϵ-rules (whose bodies are empty) and unit rules (whose bodies consist of a single element). However, even for unification grammars with no such rules the recognition problem is NP-hard (Barton et al., 1987).

Other works define highly restricted versions of UG, which guarantee th efficiency of parsing:

Non-reentrant unification grammars A unification grammar is non-reentrant if it includes no reentrancies. Non-reentrant unification grammars generate exactly the class of context free grammars (Feinstein and Wintner, 2008).

One-reentrant unification grammars A unification grammar is one-reentrant if every rule includes at most one reentrancy, between the head of the rule and some element of the body. One-reentrant unification grammars generate exactly the class of Tree-Adjoining languages (Feinstein and Wintner, 2008).

Partially Linear PATR (PLPATR) A unification grammar is PLPATR if it obeys the following constraints:

- the start symbol contains no reentrancies;
- every rule includes at most one reentrancy, between the head of the rule and some element of the body;
- reentrancies are allowed between elements in the rule's body, as long as they are not also in the rule's head.

PLPATR is more powerful then Tree-Adjoining grammar (TAG) since it can generate the k-copy language for any fixed k: $\left\{w^k \mid w \in L\right\}$ for any $k \geq 1$ and context-free language L. PLPATR languages are included in the set of languages generated by Linear Context-Free Rewriting System (LCFRS) (Keller and Weir, 1995).

These versions ensure efficiency, by limiting the expressivity and flexibility of the formalism, thereby handicapping the grammar designer. Our goal in this work is to define a constrained version of UG that on one hand is expressive enough so as to allow the expression of complex linguistic structures, and on the other hand guarantees efficient (polynomial time) processing. This is achieved by mapping constrained UGs to RCGs, a formalism that guarantees polynomial-time processing (in the size of the input string), but is maximally expressive. (Note that recognition time with RCGs can still be exponential in the size of the grammar; we are only concerned with complexity as a function of the length of the input string below.)

1.3 Range Concatenation Grammars

Range Concatenation Grammars (RCG) is a syntactic formalism that was introduced by Boullier (1998b). The basis of RCG is the notion of *ranges*, pairs of integers which denote occurrences of substrings in

a source text. RCG generates exactly the class of languages recognizable in polynomial time (Boullier, 1998b), and it is closed under union, concatenation, Kleene iteration, intersection and complementation (Boullier, 1998b).

Boullier (2000) introduces both Positive and Negative RCG, where the formalism is the union of the two. Since the negative variant has no additional generative power over the positive one, however, we only use Positive RCG, referring to it as RCG, for simplicity. The following definitions are taken from Boullier (2000).

Definition 12 (Range Concatenation Grammar (RCG))**.** ***A range concatenation grammar** (RCG) $G = \langle N, T, V, P, S\rangle$ is a 5-tuple, where:*

- *N is a finite set of **nonterminal symbols** (also called **predicate names**); each non-terminal $A \in N$ is associated with an **arity**, $ar(A)$.*
- *T is a finite set of **terminal symbols**,*
- *V is a finite set of **variable symbols**, such that $T \cap V = \emptyset$.*
- *$S \in N$ is the **start predicate**, or the* axiom*; $ar(S) = 1$.*
- *P is a finite set of **clauses** of the form*

  $$\psi_0 \to \psi_1 ... \psi_j ... \psi_m$$

  *where $m \geq 0$ and each ψ_i, $0 \leq i \leq m$, is a **predicate** of the form*

  $$A(\alpha_1, ..., \alpha_i, ..., \alpha_{ar(A)}).$$

  *where $A \in N$, and each $\alpha_i \in \{T \cup V\}^*, 1 \leq i \leq ar(A)$, is an **argument**.*

Example 1. *Following is an RCG grammar G. While we only define the notion presently, the language of this grammar is $\{a^n b^n c^n\}$. $G = \langle N, T, V, P, S\rangle$ where $N = \{S, A\}$, $T = \{a, b, c\}$, $V = \{X, Y, Z\}$, S is the start symbol, $ar(S) = 1$, $ar(A) = 3$, and P is given by:*

$$\begin{array}{ll} (1) & S(XYZ) \to A(X, Y, Z) \\ (2) & A(aX, bY, cZ) \to A(X, Y, Z) \\ (3) & A(\epsilon, \epsilon, \epsilon) \to \epsilon \end{array}$$

The language defined by an RCG is based on the notion of **range**.

Definition 13 (Range)**.** *For a given input string $w = a_1 \ldots a_n$, a **range** is a pair $(i, j), 0 < i < j \leq n$, of integers, which denotes the occurrence of some substring $a_{i+1} \ldots, a_j$ in w. The number i is its **lower bound**, j is its **upper bound** and $j - i$ is its **length**. If $i = j$, the range is **empty**. For $w \in T^*$ such that $|w| = n$, its **set of ranges** is $R_w = \{\rho \mid \rho = (i, j), 0 \leq i \leq j \leq n\}$. R_w^k is the **set of vectors of ranges in** R_w **with** k **elements**: $R_w^k = \{\langle \rho_1, \ldots \rho_h\rangle \mid \rho_i \in R_w, 0 \leq i \leq k\}$.*

Let $w = a_1 \dots a_n$ be an input string. Let $w_1 = a_1 \dots a_i$, $w_2 = a_{i+1} \dots a_j$ and $w_3 = a_{j+1} \dots a_n$ be three substrings of w. w_1 is denoted by $w^{\langle 0..i \rangle}$, w_2 is denoted by $w^{\langle i..j \rangle}$ and w_3 is denoted by $w^{\langle j..n \rangle}$. Therefore, $w^{\langle j..j \rangle} = \epsilon$, $w^{\langle j-1..j \rangle} = a_j$ and $w^{\langle 0..n \rangle} = w$. If $\vec{\rho} = \rho_1, \dots, \rho_i, \dots, \rho_p$ is a vector of ranges, by definition $w^{\vec{\rho}}$ denotes the tuple of strings $w^{\rho_1}, \dots, w^{\rho_i}, \dots, w^{\rho_p}$.

Definition 14 (Concatenation of ranges)**.** ***Range concatenation*** *is defined by* $w^{\langle i_1..j_1 \rangle} . w^{\langle i_2..j_2 \rangle} = w^{\langle i_1..j_2 \rangle}$ *if and only if* $j_1 = i_2$.

In any RCG, terminals, variables and arguments in a clause are bound to ranges by a substitution mechanism. For the follows discussion, fix an RCG $G = \langle N, T, V, P, S \rangle$

Definition 15 (Instantiation)**.** *A pair* (X, ρ)*, denoted by* X/ρ*, where* $X \in V$ *and* ρ *is a range, is called a* ***variable binding****.* ρ *is the* ***range instantiation*** *of* X *and* w^ρ *is its* ***string instantiation****. A set* $\sigma = \{X_1/\rho_1, ..., X_p/\rho_p\}$ *of variable bindings is a* ***variable substitution*** *if and only if* $X_i/\rho_i \neq X_j/\rho_j$ *implies* $X_i \neq X_j$*. A pair* (a, ρ) *is a* ***terminal binding****, denoted by* a/ρ *if and only if* $\rho = \langle j-1..j \rangle$ *and* $a = a_j$*.*

Example 2. $A(w^{\langle g..h \rangle}, w^{\langle i..j \rangle}, w^{\langle k..l \rangle}) \to B(w^{\langle g+1..h \rangle}, w^{\langle i+1..j-1 \rangle}, w^{\langle k..l-1 \rangle})$ *is an instantiation of the clause* $A(aX, bYc, Zd) \to B(X, Y, Z)$ *if the input word* $w = a_1...a_n$ *is such that* $a_{g+1} = a, a_{i+1} = b, a_j = c$ *and* $a_l = d$*. In this case, the variables* X, Y *and* Z *are bound to* $w^{\langle g+l..h \rangle}, w^{\langle i+1..j-1 \rangle}$ *and* $w^{\langle k..l-1 \rangle}$*, respectively.*

For brevity, in the following discussion we often use the term **instantiation** to indicate **string instantiation**, rather then **range instantiation**. In any case, every variable substitution (by range or by substring) is subject to the constraints of Definition 15 above.

Definition 16 (Argument instantiation)**.** *Let* $p = \psi_0 \to \psi_1 \dots \psi_m, m \geq 0$ *be a clause in* P*. Let* $\alpha \in \{T \cup V\}^*$ *be an argument of some predicaate* $\psi_i, 0 \leq i \leq m$*. Given a string* w*, and a substring of* w*,* w^ρ*,* w^ρ *is an* ***instantiation*** *of* α *if and only if:*

- $\alpha = w^\rho = \epsilon$*, hence* $\rho = \langle i, i \rangle$*, or,*
- $\alpha = X \in V$*, or,*
- $\alpha = w^\rho = a \in T$*, hence* a/ρ *is a terminal binding, or,*
- $\alpha = \beta \cdot \gamma$*, such that* $\beta, \gamma \in \{T \cup V\}^*$*, and* $\rho = \mu \cdot \sigma$*, such that* w^μ *is an instantiation of* β*, and* w^σ *is an instantiation of* γ*.*

We now define the sets of instantiated predicates and instantiated clauses for a given RCG G, and a given word w. The set of instantiated clauses is the set of all the clauses that can be generated by instantiating the clauses in P by substrings of w.

Definition 17 (The set of instantiated predicates)**.** *For an RCG* $G = (N, T, V, P, S)$ *and a string* $w \in T^*$ *we define the set of* ***instantiated predicates*** *as*

$$IP_{G,w} = \{A(\vec{\rho}) \mid A \in N, \vec{\rho} \in R_w^h, h = ar(A)\}.$$

Definition 18 (The set of instantiated clauses)**.** *For an RCG $G = (N, T, V, P, S)$ and a string $w \in T^*$, $p' = A_0\left(\vec{\beta_0}\right) \to A_1\left(\vec{\beta_1}\right) \dots A_m\left(\vec{\beta_m}\right)$ is an* ***instantiated clause of*** *G if and only if:*

- *For every i, $0 \le i \le m$, $A_i\left(\vec{\beta_i}\right) \in IP_{G,w}$, and,*
- *there is a clause $p = A_0(\vec{\alpha_0}) \to A_1(\vec{\alpha_1}) \dots A_m(\vec{\alpha_m}) \in P$, such that:*
  - *for every i, $0 \le i \le m$, $\vec{\beta_i}$ is the set of instantiated arguments of $\vec{\alpha_i}$, and,*
  - *if $\{X_1, \dots, X_l\}$ is the set of variables in p, and $\{\rho_1, \dots, \rho_l\}$ is the variable binding of $\{X_1, \dots, X_l\}$ in p', then $\{X_1/\rho_1, \dots, X_l/\rho_l\}$ is a variable substitution.*

The set of instantiated clauses *of G and w is denoted $IC_{G,w}$.*

As is customary in rewriting systems, we now define the language of an RCG grammar by first defining an immediate derivation relation, and then taking the set of strings derived by its reflexive-transitive closure as the language. However, RCG differs from standard rewriting systems by the fact that derivations begin with the full input words and end with the empty word ϵ.

Definition 19 (Derivation relation)**.** *For an RCG $G = (N, T, V, P, S)$ and a string $w \in T^*$, a* ***derivation relation****, denoted by $\Rightarrow_{G,w}$, is defined on strings of instantiated predicates. If Γ_1 and Γ_2 are strings of instantiated predicates in $(IP_{G,w})^*$:*

$$\Gamma_1 A_0(\vec{\rho_0}) \Gamma_2 \Rightarrow_{G,w} \Gamma_1 A_1(\vec{\rho_1}) \dots A_m(\vec{\rho_m}) \Gamma_2 \in IC_{G,w}$$

if and only if

$$A_0(\vec{\rho_0}) \to A_1(\vec{\rho_1}) \dots A_m(\vec{\rho_m}) \in IC_{G,w}.$$

The reflexive-transitive closure of $\Rightarrow_{G,w}$ is denoted by $\overset{*}{\Rightarrow}_{G,w}$.

Definition 20 (The language of an RCG)**.** ***The language of an RCG*** *$G = (N, T, V, P, S)$ is the set*

$$L(G) = \left\{ w \mid S(w) \overset{*}{\Rightarrow}_{G,w} \epsilon \right\}$$

An input string $w = a_1 ... a_n$ is a ***sentence*** *if and only if the empty string (of instantiated predicates) can be derived from $S(w^{\langle 0..n \rangle})$, the instantiation of the start predicate on w.*

More generally, the ***string language*** *of a nonterminal A is defined as*

$$L(A) = \bigcup_{w \in T^*} L(A, w)$$

where

$$L(A, w) = \left\{ w^{\vec{\rho}} \mid \vec{\rho} \in R_w^h, h = ar(A), A(\vec{\rho}) \overset{+}{\Rightarrow}_{G,w} \epsilon \right\}$$

Observe that $L(G) = L(S)$, as expected.

Example 3. *The following grammar defines the language* $\{www \mid w \in \{a,b\}^*\}$*:*

$$\begin{array}{ll} (1) & S(XYZ) \rightarrow A(X,Y,Z) \\ (2) & A(aX,aY,aZ) \rightarrow A(X,Y,Z) \\ (3) & A(bX,bY,bZ) \rightarrow A(X,Y,Z) \\ (4) & A(\epsilon,\epsilon,\epsilon) \rightarrow \epsilon \end{array}$$

Below we demonstrate that the input string $w = ababab$ *is a sentence:*

$$S(ababab) \Rightarrow_{G,w} A(ab,ab,ab)$$

using clause (1) and variable substitution $\{X/ab, Y/ab, Z/ab\}$

$$A(ab,ab,ab) \Rightarrow_{G,w} A(b,b,b)$$

using clause (2) and variable substitution $\{X/b, Y/b, Z/b\}$

$$A(b,b,b) \Rightarrow_{G,w} A(\epsilon,\epsilon,\epsilon)$$

using clause (3) and variable substitution $\{X/\langle 2..2\rangle, Y/\langle 4..4\rangle, Z/\langle 6..6\rangle\}$

$$A(\epsilon,\epsilon,\epsilon) \Rightarrow_{G,w} \epsilon$$

using clause (4)

Definition 21 (RCLs)**.** *If G is an RCG, then $L(G)$ is a* ***range concatenation language (RCL)****. Let $\mathcal{L}_{RCG}$ be the class of RCLs.*

We demonstrate the formalism by presenting two RCG grammars: The grammar G_{prime} whose language is $a^{prime} = \{a^p \mid p \text{ is a prime}\}$; and a grammar, G_{SCR}, accounting for the phenomenon of word scrambling which occurs in several natural languages such as German.

Example 4 (G_{prime})**.** *The idea behind the grammar is as follows: Given a string a^n, if $n = 2$ or $n = 3$, accept. Otherwise, try to divide n by any number in the range between 2 and $(n-1)/2$. If n is not divisible by any of these numbers, accept.*

Division of n by k is done by RCG, using strings, as follows:

1. *Let $x = a^k$*
2. *Let $y = a^n$*
3. *Repeat while x and y are not empty:*
   (a) *Remove one a from x*
   (b) *Remove one a from y*
4. *If x is empty and y is not:*

(a) $x = a^k$

(b) Go to line 3

5. If y is empty and x is not, x is not a factor of y. Stop.

6. If both x and y are empty, x is a factor of y. Stop.

The clauses of G_{prime} are:

1.	$S(aa) \to \epsilon$	*accept aa*
2.	$S(aaa) \to \epsilon$	*accept aaa*
3.	$S(XaY) \to A(X, XaY, X, XaY)\, eq(X, Y)$	*given a^n, try to divide n by $k = (n-1)/2$*
4.	$A(aX, aY, Z, W) \to A(X, Y, Z, W)$	*start the loop in the algorithm above, line 3*
5.	$A(\epsilon, Y, Z, W) \to A(Z, Y, Z, W)$	*X is empty, restart the loop, line 4*
6.	$A(X, \epsilon, aZ, W) \to A(Z, W, Z, W)$	*Y is empty, X is not,*
	$NonEmpty(X)\, MinLen2(Z)$	*k is greater than 2, try $k = k - 1$*
7.	$A(X, \epsilon, Z, W) \to NonEmpty(X)\, Len2(Z)$	*Y is empty, X is not, $k = 2$, n is a prime.*
8.	$eq(aX, aY) \to eq(X, Y)$	
9.	$eq(\epsilon, \epsilon) \to \epsilon$	
10.	$NonEmpty(aX) \to \epsilon$	
11.	$Len2(aa) \to \epsilon$	
12.	$MinLen2(aX) \to NonEmpty(X)$	

To demonstrate the operation of the grammar, we describe below a derivation of the string $aaaaa$

with G_{prime}

$$
\begin{array}{llll}
(clause3) & S(aaaaa) & \rightarrow & A(aa, aaaaa, aa, aaaaa)\, eq(aa, aa) \\
(clause8) & eq(aa, aa) & \rightarrow & eq(a, a) \\
(clause8) & eq(a, a) & \rightarrow & eq(\epsilon, \epsilon) \\
(clause8) & eq(\epsilon, \epsilon) & \rightarrow & \epsilon \\
(clause4) & A(aa, aaaaa, aa, aaaaa) & \rightarrow & A(a, aaaa, aa, aaaaa) \\
(clause4) & A(a, aaaa, aa, aaaaa) & \rightarrow & A(\epsilon, aaa, aa, aaaaa) \\
(clause5) & A(\epsilon, aaa, aa, aaaaa) & \rightarrow & A(aa, aaa, aa, aaaaa) \\
(clause4) & A(aa, aaa, aa, aaaaa) & \rightarrow & A(a, aa, aa, aaaaa) \\
(clause4) & A(a, aa, aa, aaaaa) & \rightarrow & A(\epsilon, a, aa, aaaaa) \\
(clause5) & A(\epsilon, a, aa, aaaaa) & \rightarrow & A(aa, a, aa, aaaaa) \\
(clause4) & A(aa, a, aa, aaaaa) & \rightarrow & A(a, \epsilon, aa, aaaaa) \\
(clause7) & A(a, \epsilon, aa, aaaaa) & \rightarrow & NonEmpty(a)\, Len2(aa) \\
(clause10) & NonEmpty(a) & \rightarrow & \epsilon \\
(clause11) & Len2(aa) & \rightarrow & \epsilon
\end{array}
$$

Example 5 (Word scrambling)**.** *Scrambling is a word-order phenomenon which occurs in several languages such as German, Japanese and Hindi, and is known to be beyond the formal power of TAGs (Becker et al., 1991). Scrambling can be seen as a leftward movement of verb arguments (nominal, prepositional or clausal), and can be abstracted by the formal language $SCR = \{\pi(n_1, \ldots, n_p)\, v_1, \ldots, v_q\}$, such that π is a permutation, $n_1, \ldots, n_p$ are terminals of the set of nouns $\mathcal{N}$, and $v_1, \ldots, v_q$ are terminals of the set of verbs $\mathcal{V}$. In addition, there is a mapping function $h : \mathcal{N} \rightarrow \mathcal{V}$, such that, for every $1 \leq i \leq q$, there is a j, $1 \leq j \leq p$, such that, $h(n_j) = v_i$, which indicates that the noun n_j is an argument of the verb v_i. Several nominal arguments can be attached to a single verb (hence, $p \geq q$). The grammar below is a simplified version of the scrambling RCG that is presented in Boullier (1999)[1].*

G_{SCR} includes the terminals $T = \{n_1, \ldots, n_l, v_1, \ldots, v_m\}$, and the following non-terminals:

- *$\mathcal{N}$ defines the set of nouns,*
- *$\mathcal{V}$ defines the set of verbs,*
- *h defines the mapping between $\mathcal{N}$ and $\mathcal{V}$,*
- *$\mathcal{N}^+\mathcal{V}^+$ separates the input string into two parts: a prefix of nouns and a suffix of verbs,*
- *$\mathcal{N}_s$ verifies that every noun n in the nouns part has a corresponding verb v in the verbs part, such that $h(n) = v$,*
- *$\mathcal{N}in\mathcal{V}^+$ verifies that a single noun n has a corresponding verb v in the verbs part, such that $h(n) = v$,*

[1] The original grammar is a negative RCG that also checks the uniqueness of the nouns and verbs in the sentence.

- *$\mathcal{V}_s$ verifies that every verb v in the verbs part has a corresponding noun n in the verbs part, such that $h(n) = v$,*

- *$\mathcal{V}in\mathcal{N}^+$ verifies that a single verb v has a corresponding noun n in the nouns part, such that $h(n) = v$,*

The clauses of G_{SCR} are listed in Figure 1.1.

$$
\begin{array}{lllll}
(1) & S(W) & \rightarrow & \mathcal{N}^+\mathcal{V}^+(W,W) \\
(2) & \mathcal{N}^+\mathcal{V}^+(W,TY) & \rightarrow & \mathcal{N}(T)\,\mathcal{N}^+\mathcal{V}^+(W,Y) \\
(3) & \mathcal{N}^+\mathcal{V}^+(XTY,TY) & \rightarrow & \mathcal{V}(T)\,\mathcal{N}_s(X,TY)\,\mathcal{V}_s(X,TY) \\
\\
(4) & \mathcal{N}_s(TX,Y) & \rightarrow & \mathcal{N}in\mathcal{V}^+(T,Y)\,\mathcal{N}_s(X,Y) \\
(5) & \mathcal{N}_s(\epsilon,Y) & \rightarrow & \epsilon \\
(6) & \mathcal{N}in\mathcal{V}^+(T,T'Y) & \rightarrow & h(T,T') \\
(7) & \mathcal{N}in\mathcal{V}^+(T,T'Y) & \rightarrow & \mathcal{N}in\mathcal{V}^+(T,Y) \\
\\
(8) & \mathcal{V}_s(X,TY) & \rightarrow & \mathcal{V}in\mathcal{N}^+(T,X)\,\mathcal{V}_s(X,Y) \\
(9) & \mathcal{V}_s(X,\epsilon) & \rightarrow & \epsilon \\
(10) & \mathcal{V}in\mathcal{N}^+(T,T'Y) & \rightarrow & h(T',T) \\
(11) & \mathcal{V}in\mathcal{N}^+(T,T'Y) & \rightarrow & \mathcal{V}in\mathcal{N}^+(T,Y) \\
\\
(12) & \mathcal{N}(n_1) & \rightarrow & \epsilon \\
 & \ldots & & \\
 & \mathcal{N}(n_l) & \rightarrow & \epsilon \\
\\
(13) & \mathcal{V}(v_1) & \rightarrow & \epsilon \\
 & \ldots & & \\
 & \mathcal{V}(v_m) & \rightarrow & \epsilon \\
\\
(14) & h(n_1,v_1) & \rightarrow & \epsilon \\
 & \ldots & & \\
 & h(n_l,v_m) & \rightarrow & \epsilon
\end{array}
$$

Figure 1.1: The clauses of G_{SCR}

To demonstrate the operation of the grammar, let the set of nouns be $\mathcal{N} = \langle n_1, n_2, n_3 \rangle$, the set of verbs be $\mathcal{V} = \langle v_1, v_2 \rangle$, and the mapping between $\mathcal{N}$ and $\mathcal{V}$ be:

$$
\begin{array}{llll}
(h1) & h(n_1,v_1) & \rightarrow & \epsilon \\
(h2) & h(n_2,v_1) & \rightarrow & \epsilon \\
(h3) & h(n_3,v_2) & \rightarrow & \epsilon
\end{array}
$$

We describe below a derivation of the string $n_2 n_3 n_1 v_1 v_2$ with G_{SCR}. First, seprate the string to two

substrings, one of nouns and one of verbs:

$$\begin{array}{llll}
(clause1) & S\,(n_2n_3n_1v_1v_2) & \rightarrow & \mathcal{N}^+\mathcal{V}^+\,(n_2n_3n_1v_1v_2, n_2n_3n_1v_1v_2) \\
(clause2) & \mathcal{N}^+\mathcal{V}^+\,(n_2n_3n_1v_1v_2, n_2n_3n_1v_1v_2) & \rightarrow & \mathcal{N}\,(n_2)\,\mathcal{N}^+\mathcal{V}^+\,(n_2n_3n_1v_1v_2, n_3n_1v_1v_2) \\
(clause2) & \mathcal{N}^+\mathcal{V}^+\,(n_2n_3n_1v_1v_2, n_3n_1v_1v_2) & \rightarrow & \mathcal{N}\,(n_3)\,\mathcal{N}^+\mathcal{V}^+\,(n_2n_3n_1v_1v_2, n_1v_1v_2) \\
(clause2) & \mathcal{N}^+\mathcal{V}^+\,(n_2n_3n_1v_1v_2, n_1v_1v_2) & \rightarrow & \mathcal{N}\,(n_1)\,\mathcal{N}^+\mathcal{V}^+\,(n_2n_3n_1v_1v_2, v_1v_2) \\
(clause3) & \mathcal{N}^+\mathcal{V}^+\,(n_2n_3n_1v_1v_2, v_1v_2) & \rightarrow & \mathcal{V}\,(v_1)\,\mathcal{N}_s\,(n_2n_3n_1, v_1v_2)\,\mathcal{V}_s\,(n_2n_3n_1, v_1v_2)
\end{array}$$

Now, check that every noun n, has a verb v, such that $h\,(n,v)$:

$$\begin{array}{llll}
(clause4) & \mathcal{N}_s\,(n_2n_3n_1, v_1v_2) & \rightarrow & \mathcal{N}in\mathcal{V}^+\,(n_2, v_1v_2)\,\mathcal{N}_s\,(n_3n_1, v_1v_2) \\
(clause6) & \mathcal{N}in\mathcal{V}^+\,(n_2, v_1v_2) & \rightarrow & h\,(n_2, v_1) \\
(h2) & h\,(n_2, v_1) & \rightarrow & \epsilon \\
(clause4) & \mathcal{N}_s\,(n_3n_1, v_1v_2) & \rightarrow & \mathcal{N}in\mathcal{V}^+\,(n_3, v_1v_2)\,\mathcal{N}_s\,(n_1, v_1v_2) \\
(clause7) & \mathcal{N}in\mathcal{V}^+\,(n_3, v_1v_2) & \rightarrow & \mathcal{N}in\mathcal{V}^+\,(n_3, v_2) \\
(clause6) & \mathcal{N}in\mathcal{V}^+\,(n_3, v_2) & \rightarrow & h\,(n_3, v_2) \\
(h3) & h\,(n_3, v_2) & \rightarrow & \epsilon \\
(clause4) & \mathcal{N}_s\,(n_1, v_1v_2) & \rightarrow & \mathcal{N}in\mathcal{V}^+\,(n_1, v_1v_2)\,\mathcal{N}_s\,(\epsilon, v_1v_2) \\
(clause6) & \mathcal{N}in\mathcal{V}^+\,(n_1, v_2v_2) & \rightarrow & h\,(n_1, v_1) \\
(h1) & h\,(n_1, v_1) & \rightarrow & \epsilon \\
(clause5) & \mathcal{N}_s\,(\epsilon, v_1v_2) & \rightarrow & \epsilon
\end{array}$$

Finally, check that every verb v, has a noun n, such that $h\,(n,v)$:

$$\begin{array}{llll}
(clause8) & \mathcal{V}_s\,(n_2n_3n_1, v_1v_2) & \rightarrow & \mathcal{V}in\mathcal{N}^+\,(v_1, n_2n_3n_1)\,\mathcal{V}_s\,(n_3n_1, v_2) \\
(clause10) & \mathcal{V}in\mathcal{N}^+\,(v_1, n_2n_3n_1) & \rightarrow & h\,(n_2, v_1) \\
(h2) & h\,(n_2, v_1) & \rightarrow & \epsilon \\
(clause8) & \mathcal{V}_s\,(n_2n_3n_1, v_2) & \rightarrow & \mathcal{V}in\mathcal{N}^+\,(v_2, n_2n_3n_1)\,\mathcal{V}_s\,(n_3n_1, \epsilon) \\
(clause11) & \mathcal{V}in\mathcal{N}^+\,(v_2, n_2n_3n_1) & \rightarrow & \mathcal{V}in\mathcal{N}^+\,(v_2, n_3n_1) \\
(clause10) & \mathcal{V}in\mathcal{N}^+\,(v_2, n_3n_1) & \rightarrow & h\,(n_3, v_2) \\
(h3) & h\,(n_3, v_2) & \rightarrow & \epsilon \\
(clause9) & \mathcal{V}_s\,(n_2n_3n_1, \epsilon) & \rightarrow & \epsilon
\end{array}$$

1.4 Structure of the thesis

In Chapter 2 we define a restricted version of UG, such that constrained grammars can be simulated by an equivalent RCG. This mapping is given in Chapter 3; in Chapter 4 we show a reverse mapping of an arbitrary RCG to an equivalent restricted UG, thereby establishing the equivalence between the two classes of languages generated by the two formalisms. Chapter 5 presents a sketch of a proof of the correctness of the two mappings. We conclude with suggestions for future research.

Chapter 2

Restricted Typed Unification Grammars

We are looking for a restricted version of TUG that would facilitate conversion of restricted grammars to RCG. RCG clauses consist of predicates whose arguments are parts of the input string, and nothing more. An RCG derivation starts with the input word, and in each derivation step, substrings of the strings associated with the mother of the clause are passed to the daughters, until reaching, in the end of the derivation, the empty word.

Our motivation in the design of the restricted TUG is to have the unification rules simulate RCG, where feature values simulate RCG arguments. We thus define restrictions over unification grammars, such that feature values can only contain representations of parts of the input string, and nothing more. In addition, we want UG derivations to simulate RCG derivations, such that in every UG rule, the feature values of the daughters can only contain parts of the feature values of the mother.

2.1 Representing Lists of Terminals with TFSs

RCG arguments are strings of terminals and variables, where in each derivation step, these strings can be split or concatenated. In order to manipulate strings and substrings thereof with UG, we define an infrastructure for handling bi-directional lists of terminal symbols with TFSs. While the formal definitions are suppressed for brevity (see Appendix A), we list below some of the main concepts we need, in an informal way. First, we manipulate *lists* of terminal symbols. Each such list consists of *nodes*. A *bi_list node* is a TFS with three features:

- CURR which includes the actual value of the node of type *terminal*;
- PREV which points to the previous node in the list;
- NEXT which points to the next node in the list.

Then, the list itself is represented as a TFS with two features:

- HEAD which points to the first node of the list;
- TAIL which points to the last node of the list.

For every type $t \in$ TYPES we define the following bi_list infrastructure types:

- *node*, the super-type of bi_list nodes,
- *null*, such that *node* $\overset{\circ}{\sqsubset}$ *null*, the type of empty nodes.
- *ne_node*, such that *node* $\overset{\circ}{\sqsubset}$ *ne_node*, the type of non-empty nodes.
- *bi_list*, the super-type of bi_lists,
- *elist*, such that *bi_list* $\overset{\circ}{\sqsubset}$ *elist*, the type of empty bi_lists.
- *ne_bi_list*, such that *bi_list* $\overset{\circ}{\sqsubset}$ *ne_bi_list*, the type of non-empty bi_lists.

Definition 22 (bi_list signature)**.** *Let*

$$\begin{aligned} bi_list_\text{TYPES} &= \left\{ \textit{terminal}, \textit{node}, \textit{null}, \textit{ne_node}, \textit{bi_list}, \textit{elist}, \textit{ne_bi_list} \right\} \\ bi_list_\text{FEATS} &= \{\text{CURR}, \text{PREV}, \text{NEXT}, \text{HEAD}, \text{TAIL}\} \end{aligned}$$

bi_list signature $= \langle bi_list_\text{TYPES}, bi_list_\text{FEATS}, \sqsubseteq\rangle$ *is defined as follows:*

terminal
node
    null
    ne_node CURR*: terminal* PREV*: node* NEXT*: node*
bi_list
    elist
    ne_bi_list HEAD*: node* TAIL*: node*

Example 6 (bi_list)**.** *As an example of a TFS representing a bidirectional list of terminals, consider A (Figure 2.1). The terminals are a and b (in other words, terminal $\sqsubseteq a$ and terminal $\sqsubseteq b$). A represents the list $\langle a, b, b\rangle$. Note that:*

- *the length of A is 3;*
- *the* 1*-node of A is* $\boxed{1}$*, the* 2*-node is* $\boxed{2}$ *and the* 3*-node is* $\boxed{3}$*;*

A TFS $A = \left[\textit{bi_list}\right]$ (that is, neither an *elist* nor a *ne_bi_list*), is called an **implicit bi_list**.

We use the notation of $\langle a_1, \ldots, a_m\rangle$ for a bi_list TFS whose length is m, where for every i, $1 \leq i \leq m$, a_i is a TFS of type *terminal*, the CURR value of the i-th node of the bi_list TFS. For example, the list notation of the *bi_list* of Example 6 is $\langle a, b, b\rangle$.

Let A_1 and A_2 be two bi_lists. The **concatenation** operation of A_1 and A_2, denoted by $A_1 \cdot A_2$, produces a new bi_list which contains the nodes of A_1, concatenated with the nodes of A_2.

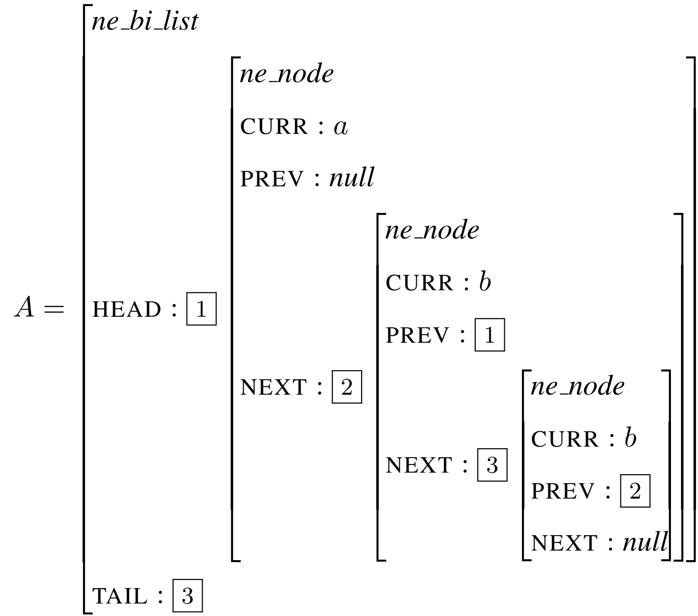


Figure 2.1: An example TFS representing a bidirectional list of terminals

Example 7. *(bi_list concatenation) Let terminal, a, b be three types in* TYPES*, such that terminal $\sqsubseteq$ a, and terminal $\sqsubseteq$ b. Let $A_1 = \langle a, a, a \rangle$ and $A_2 = \langle b, a \rangle$ be two bi_list TFSs of type terminal. Then $A_1 \cdot A_2 = \langle a, a, a, b, b \rangle$.*

Let A be a bi_list. The **sublists** of A are all the bi_lists whose nodes are ordered subsets of the nodes of A.

Example 8. *(sublists) Let $A = \langle a, a, b \rangle$. Then its sublists are: elist, $\langle a \rangle$, $\langle b \rangle$, $\langle a, a \rangle$, $\langle a, b \rangle$ and $\langle a, a, b \rangle$.*

2.2 Restricted type signatures

We begin by defining restrictions over the type signature. We constrain the set TYPES to consist of the following types only:

- all the types in $bi_list_$TYPES, as defined in Definition 22.
- A type for every word $\alpha \in$ WORDS, such that $\mathcal{L}(\alpha) \neq \emptyset$.
- A type *main* which will be used as the super-type of every TFS ocurring at the top level of any grammar rule.
- Subtypes of *main*. Such types can include only features of type *bi_list*. These types are called **main types**.
- Any main type must include one feature of type *bi_list* which is used to encode a substring of the input word. This feature is called *the input feature*.

- A main type *start* which is the type of the start FS A_s. Since in RCG the start predicate always has one argument only, containing the input word, the type *start* only has one feature, the input feature.

In addition, we require that for every two main types t and t', such that $t \sqsubseteq t'$, t' have no additional features over the ones it inherits from t. The motivation for this restriction is explained in Section 31.

Definition 23 (Restricted signature)**.** *A type signature* $\langle \textsc{Types}, \textsc{Feats}, \sqsubseteq \rangle$ *is restricted if* TYPES *includes exactly the following types:*

- *all the types in* bi_list_TYPES*, as defined in Definition 22.*
- *A type terminal such that* $\bot \overset{\circ}{\sqsubset}$ *terminal and terminal is featureless.*
- *Every word* $\alpha \in$ WORDS *is also a type in* TYPES*, such that terminal* $\overset{\circ}{\sqsubset} \alpha$*, and* α *is featureless.* α *is an* ***explicit*** *type of terminal*
- *A type main, such that* $\bot \overset{\circ}{\sqsubset}$ *main and main is featureless.*
- *A type start such that:*
  - *main* $\sqsubseteq$ *start and*
  - $Approp\,(start, \textsc{input}_{start}) = bi_list$*, and*
  - $Approp\,(start, f) \uparrow$ *for every* $f \neq \textsc{input}_{start}$.
- *Any type t, such that:*
  - *main* $\sqsubseteq$ *t;*
  - *if main* $\overset{\circ}{\sqsubset}$ *t:*
    * $Approp\,(t, \textsc{input}_t) = bi_list$*;*
    * *for every* $\textsc{f} \neq \textsc{input}_t$*, if* $Approp\,(t, \textsc{f}) \downarrow$*, then* $Approp\,(t, \textsc{f}) = bi_list$*.*
  - *if t'* $\overset{\circ}{\sqsubset}$ *t and t'* $\neq$ *main,* $\{\textsc{f} \mid Approp\,(t, \textsc{f}) \downarrow\} = \{\textsc{f} \mid Approp\,(t', \textsc{f}) \downarrow\}$
- *No other types are allowed in* TYPES.

Example 9. *(Restricted typed signature). Let*

$$\begin{aligned} \textsc{Types} &= \left\{ \begin{array}{l} main, cat, start, np, v, \\ terminal, lamb, Rachel, Jacob, \ldots \end{array} \right\} \bigcup bi_list_\textsc{Types} \\ \textsc{Feats} &= \{\textsc{input}_{cat}, \textsc{input}_v, \textsc{subcat}\} \bigcup bi_list_\textsc{Feats} \end{aligned}$$

Then the following typed signature is restricted:

main

  cat INPUT$_{cat}$*: bi_list*

    start

    np

    v

      v_np

      v_np_s

terminal

  lamb

  Rachel

  Jacob

  ...

assuming it also includes bi_list signature as defined in Definition 22.

2.3 Restricted TFS

For the following discussion, fix a restricted type signature $\langle \text{TYPES}, \sqsubseteq, \text{FEATS}, Approp \rangle$.

Definition 24 (main TFS)**.** *A TFS A of type t is a* main *TFS if main* $\sqsubseteq t$.

Definition 25 (Restricted TFS)**.** *A main TFS A is **restricted** if all its feature values are of type bi_list.*

Example 10 (Restricted TFS)**.** *Given the following signature fragment:*

main

  counter INPUT$_{counter}$*: bi_list* COUNT*: bi_list*

terminal

  a

  b

The following TFS is restricted:

$$\begin{bmatrix} counter \\ \text{INPUT}_{counter} : \langle a, b, a \rangle \\ \text{COUNT} : \langle a, a, a \rangle \end{bmatrix}$$

2.4 Restricted lexicon

Definition 26 (Restricted lexicon)**.** *A lexicon $\mathcal{L}$ is **restricted** if for every $\alpha \in$ WORDS, for every $A \in \mathcal{L}(\alpha)$, A is a restricted TFS, whose input feature contains exactly α.*

Example 11. *(Restricted $\mathcal{L}(b)$).*

$$b \rightarrow \begin{bmatrix} bt \\ \text{INPUT}_{bt} : \langle b \rangle \\ \text{COUNT} : \langle a \rangle \end{bmatrix}$$

Example 12. *(Restricted $\mathcal{L}(tell)$).*

$$tell \rightarrow \begin{bmatrix} v_np_s \\ \text{INPUT}_{v} : \langle tell \rangle \end{bmatrix}$$

2.5 Restricted rules

In this section we define restrictions over the *rules* of a restricted TUG. For every rule in $\mathcal{R}$ we require that both the mother and the daughters be restricted TFS. In addition, we add restrictions over the feature values of these TFS:

- First, we require that the value of the input feature of the mother be the concatenation of the input feature values of the daughters, in the order they occur in the rule. The motivation for this constraint is the nature of derivation in RCG: the string associated with the mother of an RCG clause is also obtained by concatenating the strings associated with the daughters. Our input features simulate such strings, hence the constraint.
- In addition, we require that every feature value of the daughters contain nothing but a sublist of some feature value of the mother.

Definition 27 (Restricted rule)**.** *A set of unification rules $\mathcal{R}$ is **restricted** if for every $r \in \mathcal{R}$, r is of the form:*

$$\begin{bmatrix} t_0 \\ \text{INPUT}_{t_0} : A_1 \cdot \ldots \cdot A_k \\ \text{F}^0_1 : B_1 \\ \vdots \\ \text{F}^0_n : B_{n_0} \end{bmatrix} \rightarrow \begin{bmatrix} t_1 \\ \text{INPUT}_{t_1} : A_1 \\ \text{F}^1_1 : C^1_1 \\ \vdots \\ \text{F}^1_n : C^1_{n_1} \end{bmatrix} \cdots \begin{bmatrix} t_k \\ \text{INPUT}_{t_k} : A_k \\ \text{F}^k_1 : Ci^k_1 \\ \vdots \\ \text{F}^k_n : C^k_{n_k} \end{bmatrix}$$

such that:

- *Each TFS in r is restricted.*
- *The value of the feature INPUT_{t_0} of the mother is the concatenation of the values of the INPUT_{t_i} features, for every i, $1 \leq i \leq k$.*
- *For every i, $1 \leq i \leq k$, for every j, $1 \leq j \leq n_i$, C^i_j is either:*
  - *a sublist of B_l for some $1 \leq l \leq n$;*

– *a sublist of* $A_1 \cdot \ldots \cdot A_k$.

Example 13. *In the following restricted rule, the feature values of the mother are obtained by concatenating the feature values of the daughters:*

$$\begin{bmatrix} bt \\ \textsc{INPUT}_{bt} : b \cdot \boxed{1} \\ \textsc{COUNT} : a \cdot \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} bt \\ \textsc{INPUT}_{bt} : \langle b \rangle \\ \textsc{COUNT} : \langle a \rangle \end{bmatrix} \begin{bmatrix} bt \\ \textsc{INPUT}_{bt} : \boxed{1} bi_list \\ \textsc{COUNT} : \boxed{2} bi_list \end{bmatrix}$$

2.6 Restricted Unification Grammars

Definition 28 (Restricted typed UG)**.** *A typed unification grammar* $G = \langle \mathcal{R}, A_s, \mathcal{L} \rangle$ *over a type signature* $\langle \textsc{Types}, \sqsubseteq, \textsc{Feats}, Approp \rangle$ *is* ***restricted typed UG (RTUG)*** *if:*

- *The type signature* $\langle \textsc{Types}, \sqsubseteq, \textsc{Feats}, Approp \rangle$ *is restricted (Definition 23).*
- *The set of rules* $\mathcal{R}$ *is restricted (Definition 27).*
- *The start symbol* A_s *is a restricted TFS of type start.*
- $\mathcal{L}$ *is restricted (Definition 26).*

Definition 29 (RTULs)**.** *If G is an RTUG, then* $L(G)$ *is a* ***restricted typed unification language (RTUL)****. Let* $\mathcal{L}_{RTUG}$ *be the class of RTULs.*

2.7 Examples of Restricted TUG

We demonstrate here two RTUG grammars, one for a formal language and one for a small fragment of a natural language. The grammar $a^n b^n c^n$] generates a formal language that is trans-context-free; $G_{longdist}$ is a grammar of basic sentence structure in several natural languages, demonstrating a naïve account of verb sub-categorization and long distance dependency phenomena.

Example 14. *(*$a^n b^n c^n$*) Figure 2.2 depicts an RTUG,* G_{abc}*, generating the language* $a^n b^n c^n$*. The grammar is inspired by the (untyped) unification grammar* G_{abc}*, presented in Francez and Wintner (2012, chapter 6). It has three simple main types: at, bt and ct, derived from the supertype counter. Each of them counts the length of a string of* a, b *and* c *symbols, respectively. Counting is done in unary base, by the feature* COUNT*, where a string of length* n *is derived by a bi_list of* n a*-s. We use* a *for counting and not* t*, as in the example of Francez and Wintner (2012), because the value of* COUNT *must be a sublist of the input word. The start rule has three daughters, for counting the* a*-s,* b*-s and* c*-s. Note that the value of* COUNT *of each of the daughters must be reentrant with the value of the input feature of the first daughter. In other words, the number of* b*-s and* c*-s must be equal to the number of* a*-s in the input word. Figure 2.3 demonstrates a derivation tree for the string "aabbcc" with this grammar.*

Signature Like any restricted signature, it includes the bi_list_signature, as defined in Definition 22, and in addition:

main
- *simple*
  - *counter* $\textsc{input}_{count}$:*bi_list* $\textsc{count}$:*bi_list*
    - at
    - bt
    - ct
  - *start* $\textsc{input}_{start}$:*bi_list*

terminal
- a
- b
- c

Lexicon

$$a \to \begin{bmatrix} at \\ \textsc{input}_{count} : \langle a \rangle \\ \textsc{count} : \langle a \rangle \end{bmatrix}, \quad b \to \begin{bmatrix} bt \\ \textsc{input}_{count} : \langle b \rangle \\ \textsc{count} : \langle a \rangle \end{bmatrix}, \quad c \to \begin{bmatrix} ct \\ \textsc{input}_{count} : \langle c \rangle \\ \textsc{count} : \langle a \rangle \end{bmatrix}$$

Rules

$$\begin{bmatrix} start \\ \textsc{input}_{start} : \boxed{1} \cdot \boxed{2} \cdot \boxed{3} \end{bmatrix} \to$$

$$\begin{bmatrix} at \\ \textsc{input}_{count} : \boxed{1} bi_list \\ \textsc{count} : \boxed{1} \end{bmatrix} \begin{bmatrix} bt \\ \textsc{input}_{count} : \boxed{2} bi_list \\ \textsc{count} : \boxed{1} \end{bmatrix} \begin{bmatrix} ct \\ \textsc{input}_{count} : \boxed{3} bi_list \\ \textsc{count} : \boxed{1} \end{bmatrix}$$

$$\begin{bmatrix} at \\ \textsc{input}_{count} : a \cdot \boxed{1} \\ \textsc{count} : a \cdot \boxed{2} \end{bmatrix} \to \begin{bmatrix} at \\ \textsc{input}_{count} : \langle a \rangle \\ \textsc{count} : \langle a \rangle \end{bmatrix} \begin{bmatrix} at \\ \textsc{input}_{count} : \boxed{1} bi_list \\ \textsc{count} : \boxed{2} bi_list \end{bmatrix}$$

$$\begin{bmatrix} bt \\ \textsc{input}_{count} : b \cdot \boxed{1} \\ \textsc{count} : a \cdot \boxed{2} \end{bmatrix} \to \begin{bmatrix} bt \\ \textsc{input}_{count} : \langle b \rangle \\ \textsc{count} : \langle a \rangle \end{bmatrix} \begin{bmatrix} bt \\ \textsc{input}_{count} : \boxed{1} bi_list \\ \textsc{count} : \boxed{2} bi_list \end{bmatrix}$$

$$\begin{bmatrix} ct \\ \textsc{input}_{count} : c \cdot \boxed{1} \\ \textsc{count} : a \cdot \boxed{2} \end{bmatrix} \to \begin{bmatrix} ct \\ \textsc{input}_{count} : \langle c \rangle \\ \textsc{count} : \langle a \rangle \end{bmatrix} \begin{bmatrix} ct \\ \textsc{input}_{count} : \boxed{1} bi_list \\ \textsc{count} : \boxed{2} bi_list \end{bmatrix}$$

Figure 2.2: An RTUG, G_{abc}, generating the language $a^n b^n c^n$

Example 15 (Long distance dependencies)**.** *Figures 2.4, 2.5 depict an RTUG, $G_{longdist}$. The grammar is inspired by the (untyped) unification grammar G_3, presented in Francez and Wintner (2012, chapter 5), with additional rules presented in Francez and Wintner (2012, section 5.6). $G_{longdist}$ reflects basic sentence structure in a natural language such as English, and demonstrates an account of verb sub-categorization and long distance dependency phenomena, producing sentences like "Jacob loved Rachel" and "Laban wondered whom Jacob loved". It has the following main types:*

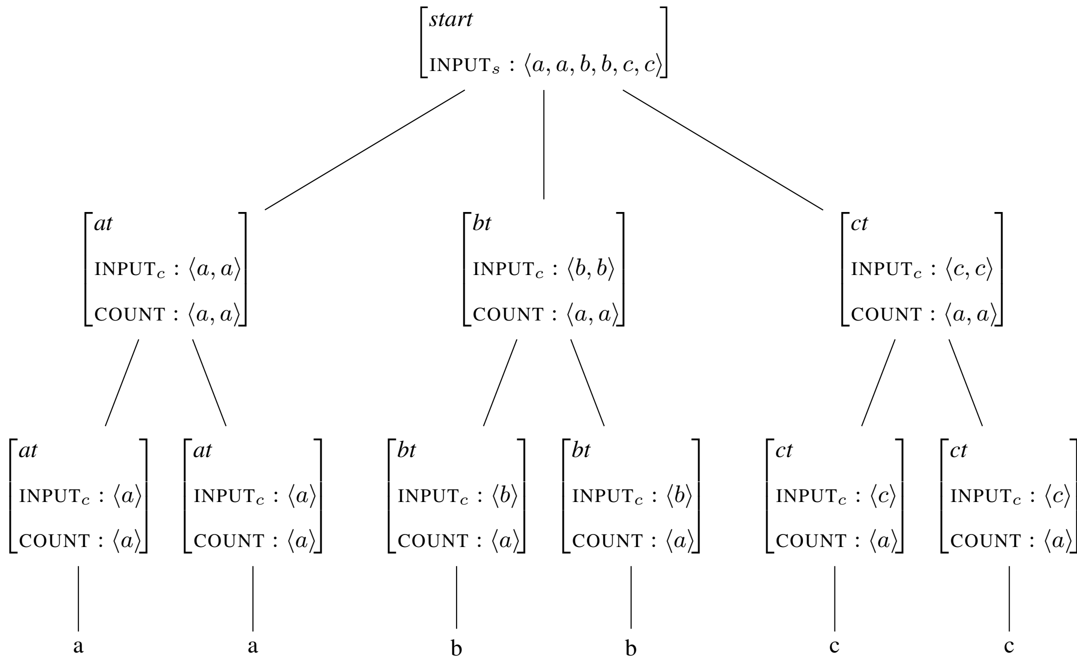


Figure 2.3: A derivation tree for the string *"aabbcc"*

- *start,*
- *np for noun phrases,*
- *vp for verb phrases,*
- *v for verbs with no subcategorized complement,*
- *v_subcat is a super type for verbs with subcategorized complement,*
- *v_np for verbs subcategorizing for a noun phrase complement, such as "loved",*
- *v_s for verbs subcategorizing for a sentence complement, such as "wondered",*
- *np_q for interrogative noun phrases, such as "whom",*
- *s_q for sentences that start with an interrogative noun phrase, realizing a transposed constituent, for example "whom Jacob loved."*
- *vp_{slash} for verb phrases in which a subcategorized complement is missing. vp_{slash} has an additional feature,* SLASH*, for the missing phrase.*
- *s_{slash} for sentences consisting of a noun phrase, followed by a slashed verb phrase. s_{slash} also has a* SLASH *feature for the missing element.*

Figure 2.6 demonstrates a derivation tree of the string "Laban wondered whom Jacob loves" with this grammar.

Signature *main* *terminal*

start		INPUT_{start} : *bi_list*		*Jacob*
v		INPUT_{v} : *bi_list*		*Rachel*
v_subcat		INPUT_{vs} : *bi_list*		*Laban*
	v_np			*whom*
	v_s			*loves*
np		INPUT_{np} : *bi_list*		*wondered*
np_q		INPUT_{np_q} : *bi_list*		...
s_q		INPUT_{s_q} : *bi_list*		
s_slash		INPUT_{s_slash} : *bi_list*	SLASH_{s} : *bi_list*	
vp		INPUT_{vp} : *bi_list*		
vp_slash		INPUT_{vp_slash} : *bi_list*	SLASH_{vp} : *bi_list*	

Lexicon

$$\textit{loves} \rightarrow \begin{bmatrix} \textit{v_np} \\ \text{INPUT}_v : \langle \textit{loves} \rangle \end{bmatrix}, \quad \textit{wondered} \rightarrow \begin{bmatrix} \textit{v_s}_q \\ \text{INPUT}_v : \langle \textit{wondered} \rangle \end{bmatrix}$$

$$\textit{Jacob} \rightarrow \begin{bmatrix} \textit{np} \\ \text{INPUT}_{np} : \langle \textit{Jacob} \rangle \end{bmatrix}, \quad \textit{Rachel} \rightarrow \begin{bmatrix} \textit{np} \\ \text{INPUT}_{np} : \langle \textit{Rachel} \rangle \end{bmatrix}$$

$$\textit{Laban} \rightarrow \begin{bmatrix} \textit{np} \\ \text{INPUT}_{np} : \langle \textit{Laban} \rangle \end{bmatrix}, \quad \textit{whom} \rightarrow \begin{bmatrix} \textit{np}_q \\ \text{INPUT}_{np_q} : \langle \textit{whom} \rangle \end{bmatrix}$$

Figure 2.4: An RTUG, $G_{longdist}$

Rules

$start_rule$:

(1)
$$\begin{bmatrix} \textit{start} \\ \text{INPUT}_{start} : \boxed{1} \cdot \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{np} \\ \text{INPUT}_{np} : \boxed{1}\textit{bi_list} \end{bmatrix} \begin{bmatrix} \textit{vp} \\ \text{INPUT}_{vp} : \boxed{2}\textit{bi_list} \end{bmatrix}$$

$queue_rule$:

(2)
$$\begin{bmatrix} s_q \\ \text{INPUT}_{s_q} : \boxed{1} \cdot \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{np}_q \\ \text{INPUT}_{np} : \boxed{1}\textit{bi_list} \end{bmatrix} \begin{bmatrix} \textit{s_slash} \\ \text{INPUT}_{s_slash} : \boxed{2}\textit{bi_list} \\ \text{SLASH}_{s} : \boxed{1} \end{bmatrix}$$

$slash_rules$:

(3)
$$\begin{bmatrix} \textit{s_slash} \\ \text{INPUT}_{s_slash} : \boxed{1} \cdot \boxed{2} \\ \text{SLASH}_{s} : \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{np} \\ \text{INPUT}_{np} : \boxed{1}\textit{bi_list} \end{bmatrix} \begin{bmatrix} \textit{vp_slash} \\ \text{INPUT}_{vp_slash} : \boxed{2}\textit{bi_list} \\ \text{SLASH}_{vp} : \boxed{3} \end{bmatrix}$$

(4)
$$\begin{bmatrix} \textit{vp_slash} \\ \text{INPUT}_{vp_slash} : \boxed{1}\textit{bi_list} \\ \text{SLASH}_{vp} : \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{v_np} \\ \text{INPUT}_{vs} : \boxed{1}\textit{bi_list} \end{bmatrix}$$

(5)
$$\begin{bmatrix} \textit{vp_slash} \\ \text{INPUT}_{vp_slash} : \boxed{1} \cdot \boxed{2} \\ \text{SLASH}_{vp} : \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{v_s} \\ \text{INPUT}_{vs} : \boxed{1}\textit{bi_list} \end{bmatrix} \begin{bmatrix} \textit{s_slash} \\ \text{INPUT}_{s_slash} : \boxed{2}\textit{bi_list} \\ \text{SLASH}_{s} : \boxed{3}\textit{bi_list} \end{bmatrix}$$

(6)
$$\begin{bmatrix} \textit{s_slash} \\ \text{INPUT}_{s_slash} : \boxed{1} \\ \text{SLASH}_{s} : \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{vp} \\ \text{INPUT}_{vp} : \boxed{1} \end{bmatrix}$$

vp_rules :

(7)
$$\begin{bmatrix} \textit{vp} \\ \text{INPUT}_{vp} : \boxed{1} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{v} \\ \text{INPUT}_{v} : \boxed{1}\textit{bi_list} \end{bmatrix}$$

(8)
$$\begin{bmatrix} \textit{vp} \\ \text{INPUT}_{vp} : \boxed{1} \cdot \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{v_s} \\ \text{INPUT}_{vs} : \boxed{1}\textit{bi_list} \end{bmatrix} \begin{bmatrix} \textit{start} \\ \text{INPUT}_{start} : \boxed{2}\textit{bi_list} \end{bmatrix}$$

(9)
$$\begin{bmatrix} \textit{vp} \\ \text{INPUT}_{vp} : \boxed{1} \cdot \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{v_s} \\ \text{INPUT}_{vs} : \boxed{1}\textit{bi_list} \end{bmatrix} \begin{bmatrix} s_q \\ \text{INPUT}_{s_q} : \boxed{2}\textit{bi_list} \end{bmatrix}$$

(10)
$$\begin{bmatrix} \textit{vp} \\ \text{INPUT}_{vp} : \boxed{1} \cdot \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{v_np} \\ \text{INPUT}_{vs} : \boxed{1}\textit{bi_list} \end{bmatrix} \begin{bmatrix} \textit{np} \\ \text{INPUT}_{np} : \boxed{2}\textit{bi_list} \end{bmatrix}$$

Figure 2.5: An RTUG, $G_{longdist}$ (continued)

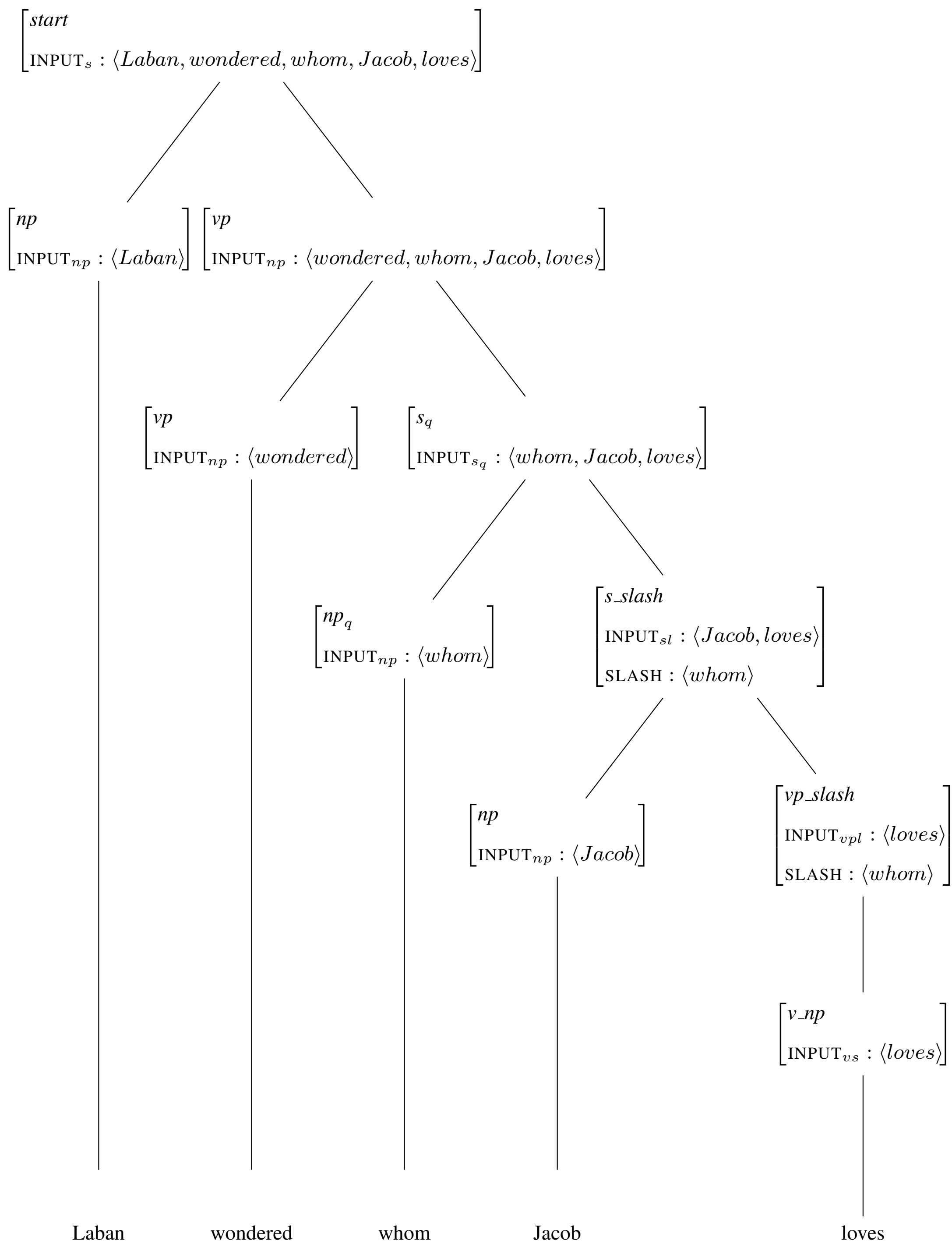


Figure 2.6: A derivation tree of the string *"Laban wondered whom Jacob loves"*

Chapter 3

Simulation of RTUG by RCG

Let $G_{ug} = \langle \mathcal{R}, A_s, \mathcal{L} \rangle$ be an RTUG over a restricted type signature $\langle \text{TYPES}, \sqsubseteq, \text{FEATS}, \mathit{Approp} \rangle$. In this section we show how to construct an RCG $G_{rcg} = (N, T, V, P, S)$ such that $L(G_{ug}) = L(G_{rcg})$.We show how to simulate each rule $r \in \mathcal{R}$ by an equivalent clause $p \in P$, where each main TFS is mapped to a predicate, whose name is the type of the TFS, and where the feature values are mapped to the predicate arguments. In addition, in order to simulate unification between TFSs, P also includes a set of *unification clauses* for every two types in TYPES that have a common upper bound. Also, for every rule $r \in \mathcal{R}$, if the type t of the mother TFS is not maximal, then for every type s that is subsumed by t, there is an additional clause in P, where the name of the predicate in the head is s.

Definition 30 (RCG mapping of RTUG)**.** *Fix an RTUG $G_{ug} = \langle \mathcal{R}, A_s, \mathcal{L} \rangle$ over a restricted type signature $\langle \text{TYPES}, \sqsubseteq, \text{FEATS}, \mathit{Approp} \rangle$. The* ***RCG mapping of*** *G_{tug}, denoted by* ***TUG2RCG*** *(G_{tug}), is an RCG $G_{rcg} = \langle N, T, V, S \rangle$, such that:*

- $T = \{t \mid \mathit{terminal} \sqsubset t\}$.
- $N = \{N_t \mid \mathit{main} \sqsubset t\}$. *For each $N_t \in N$, $ar(N_t) = |\{f \mid \mathit{Approp}(t, f) \downarrow\}|$.*
- *Let k be the maximal arity of any non-terminal $N_t \in N$; Let d be the maximal number of daughters in any rule $r \in \mathcal{R}$. Then $V = \{X_1, \ldots, X_{k \times d}\}$.*
- *for every $p \in P$, p is either:*
  - *a unification clause, as defined in Definition 31 below, or*
  - *there is a rule $r \in \mathcal{R}$, such that p is part of rule2clause (r), as defined in Definition 32 below, or*
  - *there is a lexical entry $l \in \mathcal{L}$, such that p is part of rule2clause (l), as defined in Definition 33 below.*

3.1 Mapping of type signature to RCG clauses

The type hierarchy of the signature is mapped to unification clauses in P that simulate the unification between every two types in TYPES that have a common upper-bound.

Definition 31 (Simulation of the type signature)**.** *Let $t_{max} \in$ TYPES be the subtype of main with the maximal number of appropriate features. Let f be the number of features appropriate for t_{max}. Let $V' = \{X_1, \ldots, X_f\} \subseteq V$ be a set of variables. Then, for every $t_1, t_2 \in$ TYPES such that:*

- $t_1 \sqsubset t_2$
- *main* $\sqsubset t_1$
- t_1 *and* t_2 *have* $k \leq f$ *features*

the following clause is included in P:

$$t_1\left(X_1, \ldots, X_k\right) \rightarrow t_2\left(X_1, \ldots, X_k\right)$$

Example 16. *The following type hierarchy:*

main
    counter INPUT$_{counter}$*: bi_list* COUNT*: bi_list*
        at
        bt
        ct

is simulated by the following unification clauses in G_{rcg}:

$$\begin{aligned} \textit{counter}\left(X_1, X_2\right) &\rightarrow \textit{at}\left(X_1, X_2\right) \\ \textit{counter}\left(X_1, X_2\right) &\rightarrow \textit{bt}\left(X_1, X_2\right) \\ \textit{counter}\left(X_1, X_2\right) &\rightarrow \textit{ct}\left(X_1, X_2\right) \end{aligned}$$

3.2 Mapping of UG rules to RCG clauses

We now show how UG rules are mapped to an RCG clause p, where:

- The mother of the rule is mapped to a predicate in the head of the clause.
- Every daughter of the rule is mapped to a predicate in the body of the clause.

The predicate mapping of a main TFS is as follows:

- The name of the predicate is the type of the TFS.

- The arity of the predicate is the number of features of the TFS.
- Each feature value of the TFS is mapped to an argument of the predicate.

In addition, if the type t of the mother of the rule is not maximal, then for every type s that is subsumed by t, there is an additional clause q, such that:

- The mother of the rule is mapped to a predicate in the head of q, whose name is s instead of t, and whose arguments are the same as the arguments of the predicate in the head of p.
- The body of the clause is the same as q.

Definition 32 (RCG clause mapping of a rule)**.** *Let $r \in \mathcal{R}$ be a unification rule of the form:*

$$A_0 \rightarrow A_1 \dots A_n$$

where for every i, $0 \leq i \leq n$, A_i is a main TFS of type t_i, that has k_i features. Let $\textsc{Tags}(r)$ *be the ordered set of tags in r, and let $d = |\textsc{Tags}(r)|$. Then $p \in P$ is the **RCG clause mapping of** r, denoted by **rule2clause** (r), and is defined as follows:*

- *A_0 is mapped to a predicate ψ_0 in the head of p.*
- *For every i, $1 \leq i \leq n$, A_i is mapped to a predicate ψ_i in the body of p.*

Assume, without lost of generality, that $\textsc{Tags}(r) = \left\{\boxed{1}, \dots, \boxed{d}\right\}$. *Let $V_p = \{X_1, \dots, X_d\}$ be an ordered set of RCG variables.*

Let A be a main TFS in r of the form:

$$A = \begin{bmatrix} t \\ \text{F}_1 : B_1 \\ \vdots \\ \text{F}_k : B_k \end{bmatrix}$$

*such that for every i, $1 \leq i \leq k$, B_i is a bi_list TFS. The **predicate mapping of** A, denoted by **tfs2pred** (A), is:*

$$\textit{tfs2pred}\,(A) = N_t\,(\alpha_1, \dots, \alpha_k)\,,$$

*where for every i, $1 \leq i \leq k$, $\alpha_i =$ feat2arg (B_i) is an **argument mapping of** B_i, defined as follows:*

- *If $B_i =$ elist, then feat2arg $(B_i) = \epsilon$.*
- *If $B_i = \langle a \rangle$, such that a is a terminal, then feat2arg $(B_i) = a$.*
- *If $B_i = \left[\textit{bi_list}\right]$, and B_i is marked with a tag $\boxed{l}$, then feat2arg $(B_i) = X_l$.*

- *If* $B_i = \langle a \rangle \cdot C$*, such that* a *is a terminal and* C *is a bi_list, then feat2arg* $(B_i) = a \cdot \delta$*, such that* $\delta =$ *feat2arg* (C)*.*

- *If* $B_i = C' \cdot C$*, such that* $C' = \left[\textit{bi_list} \right]$*, marked with a tag* $\boxed{l}$ *and* C *is a bi_list, then feat2arg* $(B_i) = X_l \cdot \delta$*, such that* $\delta =$ *feat2arg* (C)*.*

Let t *be the type of* A_0*. If* t *is not maximal, then for every type* s*, such that* $t \sqsubset s$*,* r *is mapped to an additional clause* $q \in P$*, such that:*

- *The head of the clause is a predicate of the form* $N_s(\alpha_1, \ldots, \alpha_k)$*, where for every* i*,* $1 \leq i \leq k$*,* $\alpha_i =$ *feat2arg* (B_i)*, and*

- *The body of the clause is the same as in rule2clause* (r)*.*

Example 17. *The following UG rule*

$$\begin{bmatrix} \textit{bt} \\ \text{INPUT}_{bt} : b \cdot \boxed{1}\textit{bi_list} \\ \text{COUNT} : \boxed{2}\textit{bi_list} \cdot \boxed{3}\textit{bi_list} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{bt} \\ \text{INPUT}_{bt} : \langle b \rangle \\ \text{COUNT} : \boxed{2} \end{bmatrix} \begin{bmatrix} \textit{bt} \\ \text{INPUT}_{bt} : \boxed{1} \\ \text{COUNT} : \boxed{3} \end{bmatrix}$$

is simulated by the following RCG clause:

$$\textit{bt}(bX_1, X_2X_3) \rightarrow \textit{bt}(b, X_2)\,\textit{bt}(X_1, X_3)$$

Observe that:

- *The* INPUT$_{bt}$ *feature value of the mother is a concatenation of a terminal and some implicit bi_list, so it is mapped to a concatenation of a terminal and a variable* (bX_1)*.*

- *The* COUNT *feature value of the mother is a concatenation of two implicit bi_list, so it is mapped to a concatenation of two variables* (X_2X3)*.*

- *The* INPUT$_{bt}$ *feature value of the first daughter contains only one terminal, so it is mapped to an argument that also contains the same terminal.*

- *The* COUNT *feature value of the first daughter is a sublist of the* COUNT *feature value of the mother, so it is mapped to an argument that reuses the same variable as the mother's.*

- *The feature values of the second daughter are both sublists of the feature values of the mother, so they are mapped to arguments that reuse the same variables as the mother's.*

3.3 Mapping the lexicon to RCG clauses

Each lexical entry is mapped to a clause in P, where the head of the clause is the predicate mapping of the pre-terminal, and the body of the clause is ϵ. In addition, if the type t of the pre-terminal is not

maximal, then for every type s that is subsumed by t, there is an additional clause in P, where the head of the clause is the predicate mapping of the pre-terminal, but its name is s instead of t.

Definition 33 (mapping of RTUG lexicon to RCG clauses)**.** *Let a be a word in* WORDS*, and $A \in \mathcal{L}(a)$, such that A is a main TFS of the form:*

$$A = \begin{bmatrix} t \\ \text{INPUT}_t : \langle a \rangle \\ \text{F}_1 : B_1 \\ \vdots \\ \text{F}_k : B_k \end{bmatrix}$$

and for every i, $1 \leq i \leq k$, B_i is a bi_list.

Then A is mapped to a clause p as follows:

$$p = N_t\left(a, \alpha_1, \ldots, \alpha_k\right) \rightarrow \epsilon,$$

where for every i, $1 \leq i \leq k$, $\alpha_i =$ feat2arg (B_i).

Let $t \sqsubset s$. A is also mapped to a clause q as follows:

$$p = N_s\left(a, \alpha_1, \ldots, \alpha_k\right) \rightarrow \epsilon,$$

where for every i, $1 \leq i \leq k$, $\alpha_i =$ feat2arg (B_i).

Example 18. *The following lexical entry*

$$b \rightarrow \begin{bmatrix} bt \\ \text{INPUT}_{bt} : \langle b \rangle \\ \text{COUNT} : \langle a \rangle \end{bmatrix}$$

is mapped to the following RCG clause:

$$bt\left(b, a\right) \rightarrow \epsilon$$

Example 19. *Let l be following lexical entry*

$$l = sheep \rightarrow \begin{bmatrix} np \\ \text{INPUT}_{np} : \langle sheep \rangle \end{bmatrix},$$

such that $np \overset{\circ}{\sqsubset} np_sg$ and $np \overset{\circ}{\sqsubset} np_pl$. l is mapped to the following RCG clauses:

$$np\left(sheep, X_1\right) \rightarrow \epsilon$$

$$np_sg\,(sheep, X_1) \to \epsilon$$

$$np_pl\,(sheep, X_1) \to \epsilon$$

3.4 Examples

We demonstrate the mapping of RTUG to equivalent RCGs on the two grammars presented in Section 2.7.

Example 20. *($a^n b^n c^n$) Here is the RCG mapping of G_{abc} that was presented in Example 14. $a^n b^n c^n$ language is very natural for RCG, and a direct implementation of an RCG grammar for it, which requires only 3 clauses, was demonstrated in Example 1. The RCG that is produced by our mapping is slightly more complicated. It has four non-terminals: start, which is the mapping of the type start, and at, bt and ct, which are the mappings of the types at, bt and ct, where the first argument is the mapping of the input feature and the second argument is the mapping of* COUNT *feature. We do not need a non-terminal mapping of the supertype counter, since there is no TFSs of this type in the grammar rules. For the same reason, there is no need in unification clauses. The clauses obtained from the rules are:*

$$\begin{array}{lll} (clause1) & \textit{start}\,(XYZ) & \to & \textit{at}\,(X,X)\,\textit{bt}\,(Y,X)\,\textit{ct}\,(Z,X) \\ (clause2) & \textit{at}\,(aX,aY) & \to & \textit{at}\,(a,a)\,\textit{at}\,(X,Y) \\ (clause3) & \textit{bt}\,(bX,aY) & \to & \textit{bt}\,(b,a)\,\textit{bt}\,(X,Y) \\ (clause4) & \textit{ct}\,(cX,aY) & \to & \textit{ct}\,(c,a)\,\textit{ct}\,(X,Y) \end{array}$$

The clauses obtained from the lexicon are:

$$\begin{array}{llll} (clause5) & \textit{at}\,(a,a) & \to & \epsilon \\ (clause6) & \textit{bt}\,(b,a) & \to & \epsilon \\ (clause7) & \textit{ct}\,(c,a) & \to & \epsilon \end{array}$$

Compare the grammar above with the grammar of Example 1, which generates the same language.

To demonstrate the operation of the grammar, we describe below a derivation of the string $aabbcc$ with this grammar:

$$\begin{array}{llll} (clause1) & \textit{start}\,(aabbcc) & \to & \textit{at}\,(aa,aa)\,\textit{bt}\,(bb,aa)\,\textit{ct}\,(cc,aa) \\ (clause2) & \textit{at}\,(aa,aa) & \to & \textit{at}\,(a,a)\,\textit{at}\,(a,a) \\ (clause5) & \textit{at}\,(a,a) & \to & \epsilon \\ (clause3) & \textit{bt}\,(bb,ab) & \to & \textit{bt}\,(b,a)\,\textit{bt}\,(b,a) \\ (clause6) & \textit{bt}\,(b,a) & \to & \epsilon \\ (clause4) & \textit{ct}\,(cc,aa) & \to & \textit{ct}\,(c,a)\,\textit{ct}\,(c,a) \\ (clause7) & \textit{ct}\,(c,a) & \to & \epsilon \end{array}$$

Example 21 (Long distance dependencies)**.** *Here is the RCG mapping of TUG2RCG ($G_{longdist}$), presented in Example 15:*

Unification clauses

$$\begin{aligned} \textit{v_subcat}\,(X) &\rightarrow \textit{v_s}\,(X) \\ \textit{v_subcat}\,(X) &\rightarrow \textit{v_np}\,(X) \end{aligned}$$

Lexicon clauses

$$\begin{aligned} \textit{v_np}\,(loves) &\rightarrow \epsilon \\ \textit{v_s}\,(wondered) &\rightarrow \epsilon \\ \textit{np}\,(Jacob) &\rightarrow \epsilon \\ \textit{np}\,(Rachel) &\rightarrow \epsilon \\ \textit{np}\,(Laban) &\rightarrow \epsilon \\ \textit{np}_q\,(whom) &\rightarrow \epsilon \end{aligned}$$

Rule clauses

$$\begin{array}{llllll} start_rule: & (1) & \textit{start}\,(XY) & \rightarrow & \textit{np}\,(X)\,\textit{vp}\,(Y) \\ \\ queue_rules: & (2) & \textit{s}_q\,(XY) & \rightarrow & \textit{np}_q\,(X)\,\textit{s_slash}\,(Y,X) \\ \\ slash_rules: & (3) & \textit{s_slash}\,(XY,Z) & \rightarrow & \textit{np}\,(X)\,\textit{vp_slash}\,(Y,Z) \\ & (4) & \textit{vp_slash}\,(X,Y) & \rightarrow & \textit{v_np}\,(X) \\ & (5) & \textit{vp_slash}\,(XY,Z) & \rightarrow & \textit{v_s}\,(X)\,\textit{s_slash}\,(Y,Z) \\ & (6) & \textit{s_slash}\,(X,Y) & \rightarrow & \textit{vp}\,(X) \\ \\ vp_rules: & (7) & \textit{vp}\,(X) & \rightarrow & \textit{v}\,(X) \\ & (8) & \textit{vp}\,(XY) & \rightarrow & \textit{v_s}\,(X)\,\textit{start}\,(Y) \\ & (9) & \textit{vp}\,(XY) & \rightarrow & \textit{v_s}\,(X)\,\textit{s}_q\,(Y) \\ & (10) & \textit{vp}\,(XY) & \rightarrow & \textit{v_np}\,(X)\,\textit{np}\,(Y) \end{array}$$

To demonstrate the operation of the grammar, we describe below a derivation of the string "Laban wondered whom Jacob loves" *with this grammar:*

$$\begin{array}{llll} (1) & \textit{start}\,(\textit{Laban wondered whom Jacob loves}) & \rightarrow & \textit{np}\,(Laban)\,\textit{vp}\,(\textit{wondered whom Jacob loves}) \\ & \textit{np}\,(Laban) & \rightarrow & \epsilon \\ (9) & \textit{vp}\,(\textit{wondered whom Jacob loves}) & \rightarrow & \textit{v_s}\,(wondered)\,\textit{s}_q\,(\textit{whom Jacob loves}) \\ & \textit{v_s}\,(wondered) & \rightarrow & \epsilon \\ (2) & \textit{s}_q\,(\textit{whom Jacob loves}) & \rightarrow & \textit{np}_q\,(whom)\,\textit{s_slash}\,(\textit{Jacob loves}, whom) \\ & \textit{np}_q\,(whom) & \rightarrow & \epsilon \\ (3) & \textit{s_slash}\,(\textit{Jacob loves}, whom) & \rightarrow & \textit{np}\,(Jacob)\,\textit{vp_slash}\,(loves, whom) \\ & \textit{np}\,(Jacob)) & \rightarrow & \epsilon \\ (4) & \textit{vp_slash}\,(loves, whom) & \rightarrow & \textit{v_np}\,(loves) \\ & \textit{v_np}\,(loves) & \rightarrow & \epsilon \end{array}$$

Chapter 4

Simulation of RCG by RTUG

In this section we define a reverse mapping that, given any RCG, yields a *restricted* UG whose language is identical. Since RCG derivations start with the whole input word, and terminate with empty clauses (ϵ rules), the UG simulation has 2 phases: in the first phase the UG derivation scans the input word and stores it in a TFS of type *bi_list*; the second phase starts with the *bi_list* that contains the whole input word, and simulates the RCG derivation, step by step, where in each step, like in the RCG, the *bi_list* is split to sub-lists or trimmed, until ϵ is obtained in all of the branches of the derivation tree. Crucially, the UG simulating an arbitrary RCG is *restricted*.

Definition 34 (TUG mapping of RCG)**.** *Let $G_{rcg} = (N, T, V, P, S)$ be an RCG. The **RTUG mapping of** G_{rcg}, denoted by **RCG2TUG** (G_{rcg}), is $G_{tug} = \langle \mathcal{R}, A_s, \mathcal{L} \rangle$, defined over a restricted type signature $\langle \text{TYPES}, \sqsubseteq, \text{FEATS}, Approp \rangle$, such that:*

Type signature *In addition to the types that are defined for* every *RTUG in Definition 23, the signature of G_{ug} includes the following types:*

- *A type S' such that:*
  - *main $\overset{\circ}{\sqsubset}$ S' and*
  - *$Approp\,(S', \text{INPUT}_{S'}) =$ bi_list.*

  S' is used for phase 1 of the derivation to collect the input word.
- *A type S" such that:*
  - *main $\overset{\circ}{\sqsubset}$ S" and*
  - *$Approp\,(S'', \text{INPUT}_{S''}) =$ bi_list, and*
  - *$Approp\,(S'', \text{ARG}_{S''}) =$ bi_list.*

  S" roots the second phase of the derivation, simulating the derivation steps of G_{rcg}. The input feature is added only to adhere to the restrictions of restricted type signatures, as defined in Definition 23. During the entire derivation phase, the input feature of the TFSs is always empty (elist).

- *For every RCG non-terminal $A \in N$ where $ar(A) = k$, there is a type $A \in$ TYPES, such that:*
  - *main $\overset{\circ}{\sqsubseteq}$ A and*
  - *$Approp(A, \text{INPUT}_A) =$ bi_list, and*
  - *$Approp(A, \text{ARG}_A) =$ bi_list, and*
  - *for every i, $1 \leq i \leq k$, $Approp\left(A, \text{ARG}^i_A\right) =$ bi_list.*
- *For every RCG terminal $\alpha \in T$ there is a type $\alpha \in$ TYPES such that terminal $\overset{\circ}{\sqsubseteq} \alpha$, and α is featureless.*

Lexicon *$\mathcal{L}(\alpha) = \{A\}$ if an only if $\alpha \in T$, and A is of the form:*

$$A = \begin{bmatrix} \textit{S'} \\ \text{INPUT}_{S'} : \langle\alpha\rangle \end{bmatrix}$$

Start symbol

$$A_s = \begin{bmatrix} \textit{start} \\ \text{INPUT}_{start} : \boxed{1}\textit{bi_list} \end{bmatrix}$$

rules *$\mathcal{R}$ includes the following rules:*

- *The start rule is of the form:*

$$\begin{bmatrix} \textit{start} \\ \text{INPUT}_{start} : \boxed{1}\textit{bi_list} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{S'} \\ \text{INPUT}_{S'} : \boxed{1} \end{bmatrix} \begin{bmatrix} \textit{S''} \\ \text{INPUT}_{S''} : \textit{elist} \\ \text{ARG}_{S''} : \boxed{1} \end{bmatrix}$$

- *To collect the input word in the first phase of the derivation, $\mathcal{R}$ always includes the following rules:*

$$\begin{bmatrix} \textit{S'} \\ \text{INPUT}_{S'} : \boxed{1} \cdot \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{S'} \\ \text{INPUT}_{S'} : \boxed{1}\langle\textit{terminal}\rangle \end{bmatrix} \begin{bmatrix} \textit{S'} \\ \text{INPUT}_{S'} : \boxed{2}\textit{bi_list} \end{bmatrix}$$

$$\begin{bmatrix} \textit{S'} \\ \text{INPUT}_{S'} : \textit{elist} \end{bmatrix} \rightarrow \epsilon$$

- *The second phase of the derivation is the actual simulation of G_{rcg} derivation steps. For this phase, for every clause $p \in P$ there is a rule $r \in \mathcal{R}$, such that $r =$ clause2rule(p), as defined in Definition 35 below.*

Definition 35 (Rules simulating of RCG clauses)**.** *Let p be a clause in P,*

$$p = \varphi_0 \to \varphi_1 \ldots \varphi_n$$

*such that for every i, $0 \leq i \leq n$, φ_i is a predicate with non-terminal N_i and arity k_i of the form $N_i\left(\alpha_1^i \ldots \alpha_{k_i}^i\right)$, and for every j, $1 \leq j \leq k_i$, $\alpha_j^i \in T \bigcup V^*$. Then $r \in \mathcal{R}$ is **the rule mapping of** p, denoted by $r =$ clause2rule (p), where*

$$r = A_0 \to A_1 \ldots A_n$$

*such that, for every i, $0 \leq i \leq n$, A_i is **the TFS mapping of predicate** φ_i, denoted pred2tfs (φ_i), and is defined as follows: A_i is a TFS of type N_i with $k_i + 1$ features of type bi_list:*

$$A_i = \begin{bmatrix} N_i \\ \text{INPUT}_{N_i} : \textit{elist} \\ \text{ARG1}_{N_i} : B_1^i \\ \vdots \\ \text{ARG}k_{i N_i} : B_{k_i}^i \end{bmatrix}$$

where for every j, $1 \leq j \leq k_i$, B_j^i is a TFS of type bi_list that is the mapping of the argument α_j^i as described in Definition 36 below. If φ_i is the start symbol $S(\alpha)$, then the type of A_i is S''.

If p is an ϵ clause of the form $p = N_0(\alpha_1, \ldots, \alpha_k) \to \epsilon$, then clause2rule (p) is the following rule:

$$\begin{bmatrix} N_0 \\ \text{INPUT}_{N_0} : \textit{elist} \\ \text{ARG1}_{N_0} : B_1 \\ \vdots \\ \text{ARG}k_{N_0} : B_k \end{bmatrix} \to \epsilon$$

where for every i, $1 \leq i \leq k$, B_i is a TFS of type bi_list that is the mapping of the argument α_i as described in Definition 36.

Definition 36 (*bi_list* mapping of RCG arguments)**.** *Let $\alpha \in T \bigcup V^*$. Its **bi_list mapping**, denoted **arg2feat** (α), is a TFS of type bi_list defined as follows:*

- *if $\alpha = \epsilon$, then arg2feat $(\alpha) =$ elist*
- *if $\alpha = a$, $a \in T$, then arg2feat $(\alpha) = \langle a \rangle$*
- *if $\alpha = X_l$, $X_l \in V$, then arg2feat $(\alpha) = \boxed{l}$ bi_list*

- *if* $\alpha = a \cdot \delta$, *where* $a \in T$ *and* $\delta \in T \bigcup V^*$, *then* $arg2feat(\alpha) = \langle a \rangle \cdot B$, *where* $B = arg2feat(\delta)$.
- *if* $\alpha = X_l \cdot \delta$, *where* $X_l \in V$ *and* $\delta \in T \bigcup V^*$, *then* $arg2feat(\alpha) = \boxed{l}\, bi_list \cdot B$, *where* $B = arg2feat(\delta)$.

Example 22. *(bi_list Mapping of RCG arguments) Let* $\alpha = aX_1X_2$, *such that* $a \in T$ *and* $X_1, X_2 \in V$, *then*

$$arg2feat(\alpha) = \langle a \rangle \cdot \boxed{1}\, bi_list \cdot \boxed{2}\, bi_list$$

where terminal $\overset{\circ}{\sqsubset}$ a.

4.1 Examples

We demonstrate now an RTUG mapping of the two RCG grammars that were presented in Section 1.3

Example 23 (G_{prime})**.** *Here we demostrate how to map* G_{prime} *grammar from Example 4 to an RTUG:*

The types in TYPES *are obtained from* G_{prime} *predicates, where:*

- *The types start, S', S", terminal, node and bi_list are fixed types, generated for every RTUG;*
- *The types* A, *eq*, $NonEmpty$, $Len2$ *and* $MinLen2$ *are mappings of* G_{prime} *non-terminals;*
- *The type* a *is a mapping of the only* G_{prime} *terminal,* a.

The complete type signature is depicted in Figure 4.1.

Since there is only one terminal in T, *the lexicon has only one entry:*

$$a \rightarrow \begin{bmatrix} \textit{S'} \\ \text{INPUT}_{S'} : \langle a \rangle \end{bmatrix}$$

$\mathcal{R}$ *includes the start rule and phase 1 rules, as defined in Definition 34. In addition,* $\mathcal{R}$ *includes the rule mappings of* G_{prime} *clauses, as follows:*

1. $S(aa) \rightarrow \epsilon$

$$\begin{bmatrix} \textit{S"} \\ \text{INPUT}_{S''} : \textit{elist} \\ \text{ARG}_{S''} : \langle a, a \rangle \end{bmatrix} \rightarrow \epsilon$$

2. $S(aaa) \rightarrow \epsilon$

$$\begin{bmatrix} \textit{S"} \\ \text{INPUT}_{S''} : \textit{elist} \\ \text{ARG}_{S''} : \langle a, a, a \rangle \end{bmatrix} \rightarrow \epsilon$$

- *main*
  - *start* — $\textsc{input}_{start}$: *bi_list*
  - *S'* — $\textsc{input}_{S'}$: *bi_list*
  - *S''* — $\textsc{input}_{S''}$: *bi_list*, $\textsc{arg}_{S''}$: *bi_list*
  - *A* — $\textsc{input}_{A}$: *bi_list*, $\textsc{arg}{1}_{A}$: *bi_list*, $\textsc{arg}{2}_{A}$: *bi_list*, $\textsc{arg}{3}_{A}$: *bi_list*, $\textsc{arg}{4}_{A}$: *bi_list*
  - *eq* — $\textsc{input}_{eq}$: *bi_list*, $\textsc{arg}{1}_{eq}$: *bi_list*, $\textsc{arg}{2}_{eq}$: *bi_list*
  - *NonEmpty* — $\textsc{input}_{NonEmpty}$: *bi_list*, $\textsc{arg}{1}_{NonEmpty}$: *bi_list*
  - *Len2* — $\textsc{input}_{Len2}$: *bi_list*, $\textsc{arg}{1}_{Len2}$: *bi_list*
  - *MinLen2* — $\textsc{input}_{MinLen2}$: *bi_list*, $\textsc{arg}{1}_{MinLen2}$: *bi_list*
- *terminal*
  - *a*
- *node*
  - *null*
  - *ne_node* — CURR:*terminal*, PREV:*node*, NEXT:*node*
- *bi_list*
  - *elist*
  - *ne_bi_list* — HEAD:*ne_node*, TAIL:*ne_node*

Figure 4.1: The type signature of G_{prime}

3. $S\,(XaY) \rightarrow A\,(X, XaY, X, XaY)\; eq\,(X, Y)$

$$\begin{bmatrix} \textit{S''} \\ \textsc{input}_{S''} : \textit{elist} \\ \textsc{arg}_{S''} : \boxed{1}\textit{bi_list} \cdot \langle a \rangle \cdot \boxed{2}\textit{bi_list} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{A} \\ \textsc{input}_{A} : \textit{elist} \\ \textsc{arg}{1}_{A} : \boxed{1} \\ \textsc{arg}{2}_{A} : \boxed{1} \cdot \langle a \rangle \cdot \boxed{2} \\ \textsc{arg}{3}_{A} : \boxed{1} \\ \textsc{arg}{3}_{A} : \boxed{1} \cdot \langle a \rangle \cdot \boxed{2} \end{bmatrix} \begin{bmatrix} \textit{eq} \\ \textsc{input}_{eq} : \textit{elist} \\ \textsc{arg}{1}_{eq} : \boxed{1} \\ \textsc{arg}{2}_{eq} : \boxed{2} \end{bmatrix}$$

4. $A\,(aX, aY, Z, W) \rightarrow A\,(X, Y, Z, W)$

$$\begin{bmatrix} \textit{A} \\ \textsc{input}_{A} : \textit{elist} \\ \textsc{arg}{1}_{A} : \langle a \rangle \cdot \boxed{1}\textit{bi_list} \\ \textsc{arg}{2}_{A} : \langle a \rangle \cdot \boxed{2}\textit{bi_list} \\ \textsc{arg}{3}_{A} : \boxed{3}\textit{bi_list} \\ \textsc{arg}{4}_{A} : \boxed{4}\textit{bi_list} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{A} \\ \textsc{input}_{A} : \textit{elist} \\ \textsc{arg}{1}_{A} : \boxed{1} \\ \textsc{arg}{2}_{A} : \boxed{2} \\ \textsc{arg}{3}_{A} : \boxed{3} \\ \textsc{arg}{4}_{A} : \boxed{4} \end{bmatrix}$$

5. $A(\epsilon, Y, Z, W) \rightarrow A(Z, Y, Z, W)$

$$\begin{bmatrix} \textit{A} \\ \text{INPUT}_A : \textit{elist} \\ \text{ARG1}_A : \textit{elist} \\ \text{ARG2}_A : \boxed{2}\textit{bi_list} \\ \text{ARG3}_A : \boxed{3}\textit{bi_list} \\ \text{ARG4}_A : \boxed{4}\textit{bi_list} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{A} \\ \text{INPUT}_A : \textit{elist} \\ \text{ARG1}_A : \boxed{3} \\ \text{ARG2}_A : \boxed{2} \\ \text{ARG3}_A : \boxed{3} \\ \text{ARG4}_A : \boxed{4} \end{bmatrix}$$

6. $A(X, \epsilon, aZ, W) \rightarrow A(Z, W, Z, W)\, NonEmpty(X)\, MinLen2(Z)$

$$\begin{bmatrix} \textit{A} \\ \text{INPUT}_A : \textit{elist} \\ \text{ARG1}_A : \boxed{1}\textit{bi_list} \\ \text{ARG2}_A : \textit{elist} \\ \text{ARG3}_A : \langle a \rangle \cdot \boxed{3}\textit{bi_list} \\ \text{ARG4}_A : \boxed{4}\textit{bi_list} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{A} \\ \text{INPUT}_A : \textit{elist} \\ \text{ARG1}_A : \boxed{3} \\ \text{ARG2}_A : \boxed{4} \\ \text{ARG3}_A : \boxed{3} \\ \text{ARG4}_A : \boxed{4} \end{bmatrix} \begin{bmatrix} \textit{NonEmpty} \\ \text{INPUT}_{NonEmpty} : \textit{elist} \\ \text{ARG1}_{NonEmpty} : \boxed{1} \end{bmatrix} \begin{bmatrix} \textit{MinLen2} \\ \text{INPUT}_{MinLen2} : \textit{elist} \\ \text{ARG1}_{MinLen2} : \boxed{3} \end{bmatrix}$$

7. $A(X, \epsilon, Z, W) \rightarrow NonEmpty(X)\, Len2(Z)$

$$\begin{bmatrix} \textit{A} \\ \text{INPUT}_A : \textit{elist} \\ \text{ARG1}_A : \boxed{1}\textit{bi_list} \\ \text{ARG2}_A : \textit{elist} \\ \text{ARG3}_A : \boxed{3}\textit{bi_list} \\ \text{ARG4}_A : \boxed{4}\textit{bi_list} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{NonEmpty} \\ \text{INPUT}_{NonEmpty} : \textit{elist} \\ \text{ARG1}_{NonEmpty} : \boxed{1} \end{bmatrix} \begin{bmatrix} \textit{Len2} \\ \text{INPUT}_{Len2} : \textit{elist} \\ \text{ARG1}_{Len2} : \boxed{3} \end{bmatrix}$$

8. $eq(aX, aY) \rightarrow eq(X, Y)$

$$\begin{bmatrix} \textit{eq} \\ \text{INPUT}_{eq} : \textit{elist} \\ \text{ARG1}_{eq} : \langle a \rangle \cdot \boxed{1}\textit{bi_list} \\ \text{ARG2}_{eq} : \langle a \rangle \cdot \boxed{2}\textit{bi_list} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{eq} \\ \text{INPUT}_{eq} : \textit{elist} \\ \text{ARG1}_{eq} : \boxed{1} \\ \text{ARG2}_{eq} : \boxed{2} \end{bmatrix}$$

9. $eq\,(\epsilon, \epsilon) \rightarrow \epsilon$

$$\begin{bmatrix} \textit{eq} \\ \text{INPUT}_{eq} : \textit{elist} \\ \text{ARG1}_{eq} : \textit{elist} \\ \text{ARG2}_{eq} : \textit{elist} \end{bmatrix} \rightarrow \epsilon$$

10. $NonEmpty\,(aX) \rightarrow \epsilon$

$$\begin{bmatrix} \textit{NonEmpty} \\ \text{INPUT}_{NonEmpty} : \textit{elist} \\ \text{ARG1}_{NonEmpty} : \langle a \rangle \cdot \boxed{1}\textit{bi_list} \end{bmatrix} \rightarrow \epsilon$$

11. $Len2\,(aa) \rightarrow \epsilon$

$$\begin{bmatrix} \textit{Len2} \\ \text{INPUT}_{Len2} : \textit{elist} \\ \text{ARG1}_{Len2} : \langle a, a \rangle \end{bmatrix} \rightarrow \epsilon$$

12. $MinLen2\,(aX) \rightarrow NonEmpty\,(X)$

$$\begin{bmatrix} \textit{MinLen2} \\ \text{INPUT}_{MinLen2} : \textit{elist} \\ \text{ARG1}_{MinLen2} : \langle a \rangle \cdot \boxed{1}\textit{bi_list} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{NonEmpty} \\ \text{INPUT}_{NonEmpty} : \textit{elist} \\ \text{ARG1}_{NonEmpty} : \boxed{1} \end{bmatrix}$$

We demonstrate the operation of the grammar by showing the derivation of the string "aaaaa". Figure 4.2 shows the root of the derivation tree; Figure 4.3 depicts the tree fragment that is responsible to collecting the input word; and Figures 4.4, 4.5 show the subtree of the second phase, the actual simulation of G_{prime}.

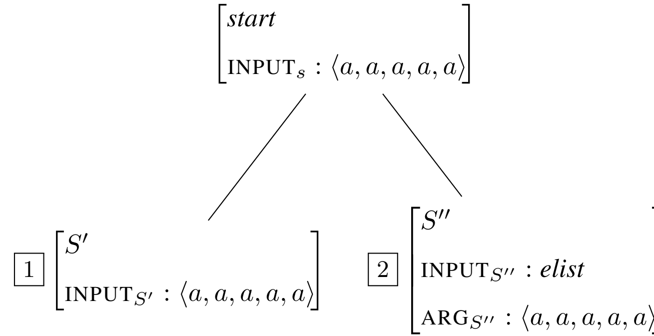


Figure 4.2: The root of the derivation tree

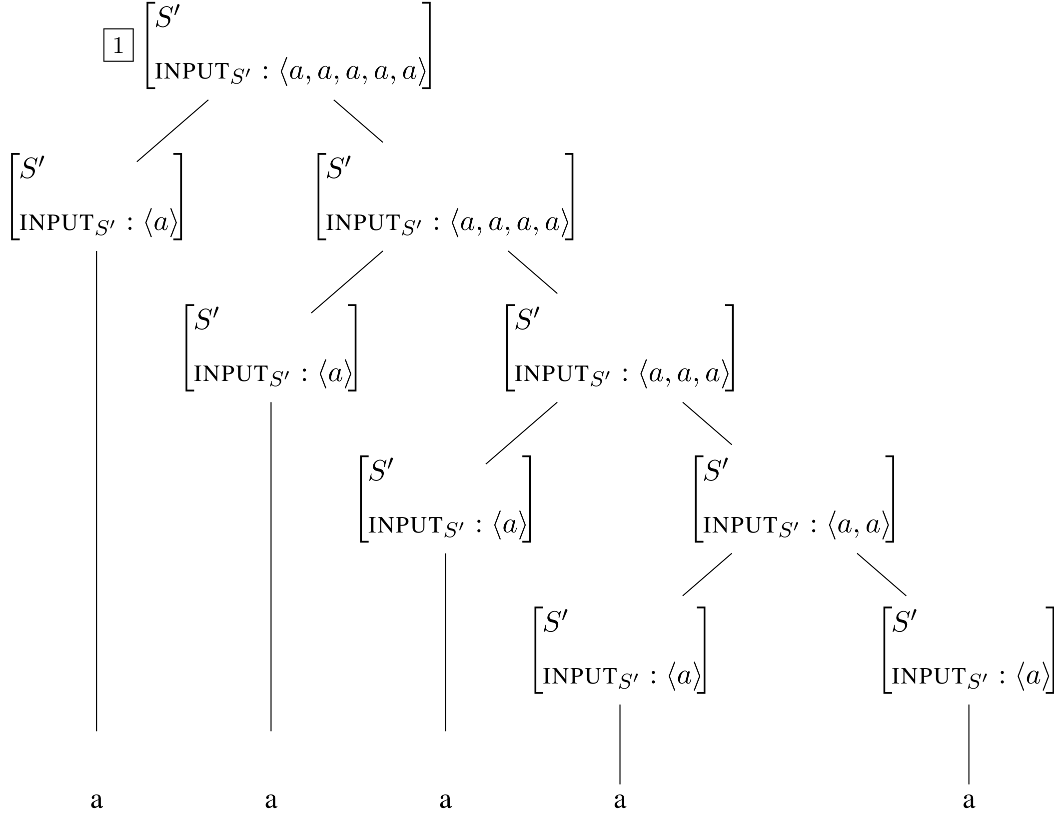


Figure 4.3: Derivation subtree showing how the input word is collected

Example 24 (Word scrambling). *We demonstrate how the grammar G_{SCR} of Example 5 is mapped to an RTUG. The types in* TYPES *are obtained from G_{SCR} predicates, where:*

- *The types start, S', S", terminal, node and bi_list are fixed types, generated for every RTUG;*
- *The types $\{\mathcal{N}, \mathcal{V}, h, \mathcal{N}^{+}\mathcal{V}^{+}, \mathcal{N}_s, \mathcal{V}_s, \mathcal{N}in\mathcal{V}^{+}, \mathcal{V}in\mathcal{N}^{+}\}$ are mappings of G_{SCR} non-terminals;*
- *The types $n_1, \ldots, n_l, v_1, \ldots, v_m$ are mappings of G_{SCR} terminals.*

The complete type signature is depicted in Figure 4.6.

The lexicon is:

$$n_1 \rightarrow \begin{bmatrix} \textit{S'} \\ \textsc{INPUT}_{S'} : \langle n_1 \rangle \end{bmatrix} \quad \ldots \quad n_l \rightarrow \begin{bmatrix} \textit{S'} \\ \textsc{INPUT}_{S'} : \langle n_l \rangle \end{bmatrix}$$

$$v_1 \rightarrow \begin{bmatrix} \textit{S'} \\ \textsc{INPUT}_{S'} : \langle v_1 \rangle \end{bmatrix} \quad \ldots \quad v_m \rightarrow \begin{bmatrix} \textit{S'} \\ \textsc{INPUT}_{S'} : \langle v_m \rangle \end{bmatrix}$$

$\mathcal{R}$ includes the start rule and phase 1 rules, as defined in Definition 34. In addition, $\mathcal{R}$ includes the rule mappings of G_{SCR} clauses, depicted in Figures 4.7, 4.8.

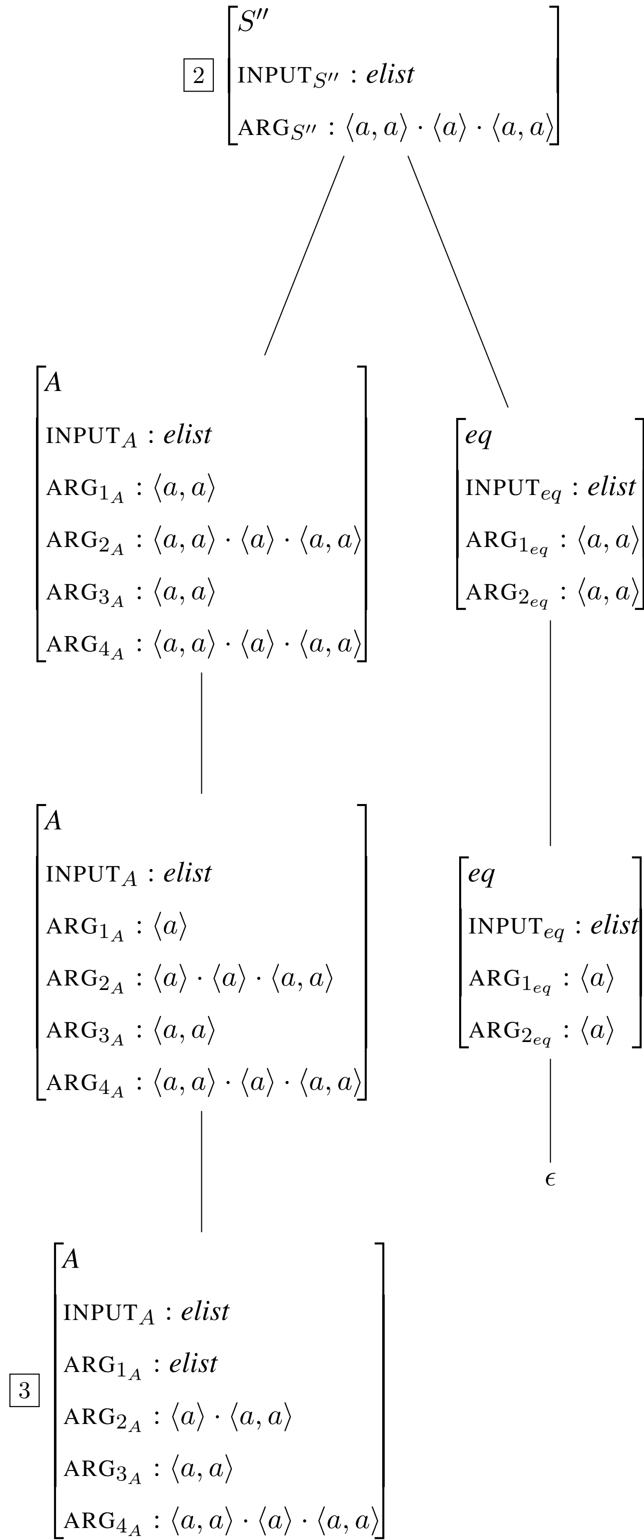


Figure 4.4: Subtree showing actual simulation of G_{prime} (part 1)

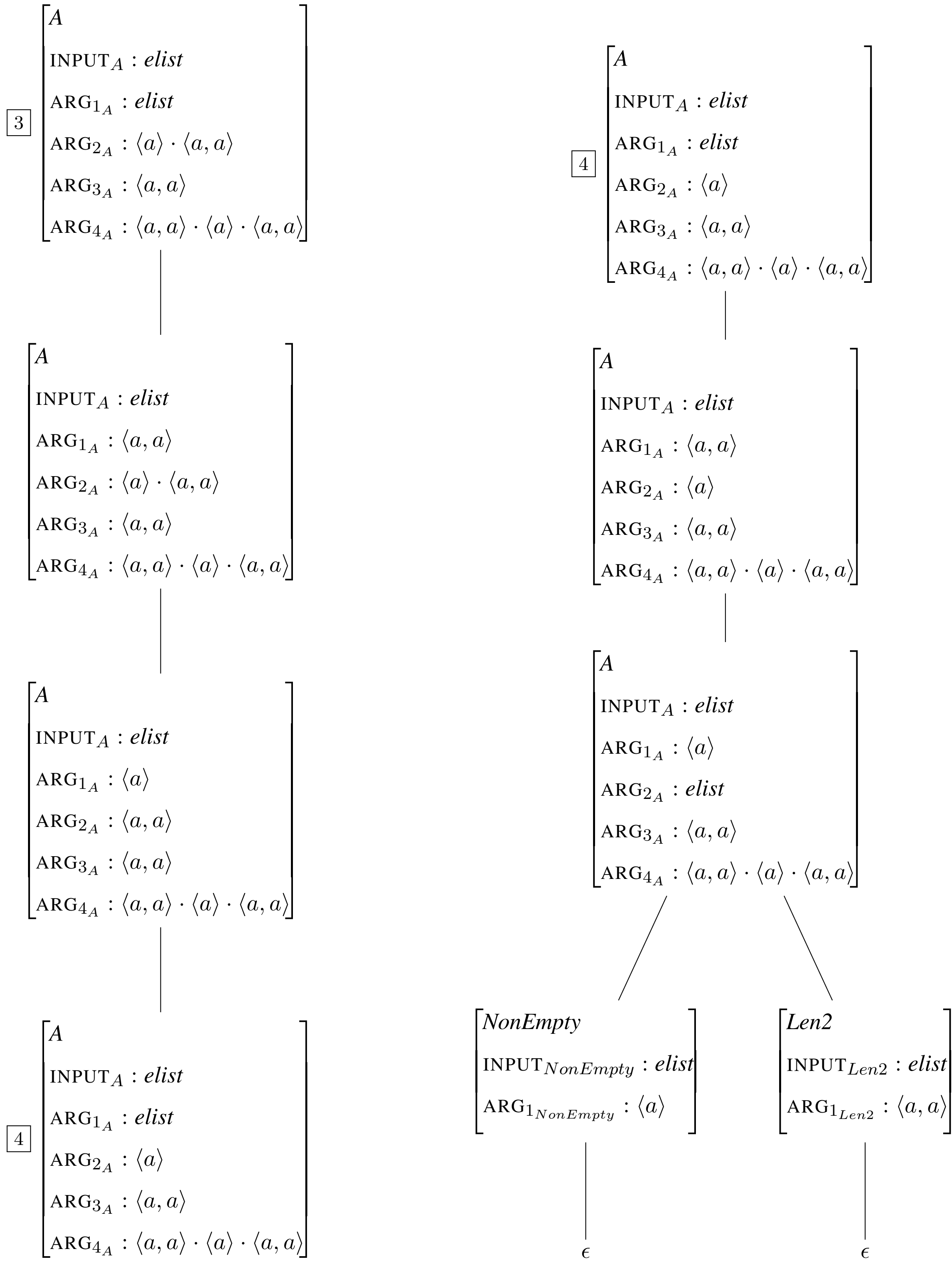


Figure 4.5: Subtree showing actual simulation of G_{prime} (part 2)

main

start	INPUT_{start}: *bi_list*		
S'	$\text{INPUT}_{S'}$: *bi_list*		
S"	$\text{INPUT}_{S''}$: *bi_list*	$\text{ARG}_{S''}$: *bi_list*	
$\mathcal{N}^+\mathcal{V}^+$	$\text{INPUT}_{\mathcal{N}^+\mathcal{V}^+}$: *bi_list*	$\text{ARG1}_{\mathcal{N}^+\mathcal{V}^+}$: *bi_list*	$\text{ARG2}_{\mathcal{N}^+\mathcal{V}^+}$: *bi_list*
$\mathcal{N}$	$\text{INPUT}_{\mathcal{N}}$: *bi_list*	$\text{ARG1}_{\mathcal{N}}$: *bi_list*	
$\mathcal{V}$	$\text{INPUT}_{\mathcal{V}}$: *bi_list*	$\text{ARG1}_{\mathcal{V}}$: *bi_list*	
h	INPUT_{h}: *bi_list*	ARG1_{h}: *bi_list*	ARG2_{h}: *bi_list*
$\mathcal{N}_s$	$\text{INPUT}_{\mathcal{N}_s}$: *bi_list*	$\text{ARG1}_{\mathcal{N}_s}$: *bi_list*	$\text{ARG2}_{\mathcal{V}_s}$: *bi_list*
$\mathcal{V}_s$	$\text{INPUT}_{\mathcal{V}_s}$: *bi_list*	$\text{ARG1}_{\mathcal{V}_s}$: *bi_list*	$\text{ARG2}_{\mathcal{V}_s}$: *bi_list*
$\mathcal{N}in\mathcal{V}^+$	$\text{INPUT}_{\mathcal{N}in\mathcal{V}^+}$: *bi_list*	$\text{ARG1}_{\mathcal{N}in\mathcal{V}^+}$: *bi_list*	$\text{ARG2}_{\mathcal{N}in\mathcal{V}^+}$: *bi_list*
$\mathcal{V}in\mathcal{N}^+$	$\text{INPUT}_{\mathcal{V}in\mathcal{N}^+}$: *bi_list*	$\text{ARG1}_{\mathcal{V}in\mathcal{N}^+}$: *bi_list*	$\text{ARG2}_{\mathcal{V}in\mathcal{N}^+}$: *bi_list*

terminal

- n_1
- $\vdots$
- n_l
- v_1
- $\vdots$
- v_m

node

- *null*
- *ne_node* CURR:*terminal* PREV:*node* NEXT:*node*

bi_list

- *elist*
- *ne_bi_list* HEAD:*ne_node* TAIL:*ne_node*

Figure 4.6: The type signature of G_{SCR}

To demonstrate the operation of the grammar, similary to Example 5, let the set of nouns be $\mathcal{N} = \langle n_1, n_2, n_3\rangle$, the set of verbs be $\mathcal{V} = \langle v_1, v_2\rangle$, and the mapping between $\mathcal{N}$ and $\mathcal{V}$ be:

$$\begin{bmatrix} h \\ \text{INPUT}_h : \textit{elist} \\ \text{ARG1}_h : \langle n_1\rangle \\ \text{ARG2}_h : \langle v_1\rangle \end{bmatrix} \to \epsilon, \quad \begin{bmatrix} h \\ \text{INPUT}_h : \textit{elist} \\ \text{ARG1}_h : \langle n_2\rangle \\ \text{ARG2}_h : \langle v_1\rangle \end{bmatrix} \to \epsilon, \quad \begin{bmatrix} h \\ \text{INPUT}_h : \textit{elist} \\ \text{ARG1}_h : \langle n_3\rangle \\ \text{ARG2}_h : \langle v_2\rangle \end{bmatrix} \to \epsilon$$

The derivation tree of the string $n_2 n_3 n_1 v_1 v_2$ is listed in Figures 4.9—4.11. We only demonstrate the second phase of the derivation, since the first phase (collecting the input string) is just the same as showed in Example 23 above.

$$(1)\quad \begin{bmatrix} S'' \\ \text{INPUT}_{S''} : \textit{elist} \\ \text{ARG}_{S''} : \boxed{1}\,\textit{bi_list} \end{bmatrix} \rightarrow \begin{bmatrix} \mathcal{N}^{+}\mathcal{V}^{+} \\ \text{INPUT}_{\mathcal{N}^{+}\mathcal{V}^{+}} : \textit{elist} \\ \text{ARG1}_{\mathcal{N}^{+}\mathcal{V}^{+}} : \boxed{1} \\ \text{ARG2}_{\mathcal{N}^{+}\mathcal{V}^{+}} : \boxed{1} \end{bmatrix}$$

$$(2)\quad \begin{bmatrix} \mathcal{N}^{+}\mathcal{V}^{+} \\ \text{INPUT}_{\mathcal{N}^{+}\mathcal{V}^{+}} : \textit{elist} \\ \text{ARG1}_{\mathcal{N}^{+}\mathcal{V}^{+}} : \boxed{1} \\ \text{ARG2}_{\mathcal{N}^{+}\mathcal{V}^{+}} : \boxed{2} \cdot \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} \mathcal{N} \\ \text{INPUT}_{\mathcal{N}} : \textit{elist} \\ \text{ARG1}_{\mathcal{N}} : \boxed{2} \end{bmatrix} \begin{bmatrix} \mathcal{N}^{+}\mathcal{V}^{+} \\ \text{INPUT}_{\mathcal{N}^{+}\mathcal{V}^{+}} : \textit{elist} \\ \text{ARG1}_{\mathcal{N}^{+}\mathcal{V}^{+}} : \boxed{1} \\ \text{ARG2}_{\mathcal{N}^{+}\mathcal{V}^{+}} : \boxed{3} \end{bmatrix}$$

$$(3)\quad \begin{bmatrix} \mathcal{N}^{+}\mathcal{V}^{+} \\ \text{INPUT}_{\mathcal{N}^{+}\mathcal{V}^{+}} : \textit{elist} \\ \text{ARG1}_{\mathcal{N}^{+}\mathcal{V}^{+}} : \boxed{1} \cdot \boxed{2} \cdot \boxed{3} \\ \text{ARG2}_{\mathcal{N}^{+}\mathcal{V}^{+}} : \boxed{2} \cdot \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} \mathcal{V} \\ \text{INPUT}_{\mathcal{V}} : \textit{elist} \\ \text{ARG1}_{\mathcal{V}} : \boxed{2} \end{bmatrix} \begin{bmatrix} \mathcal{N}_s \\ \text{INPUT}_{\mathcal{N}_s} : \textit{elist} \\ \text{ARG1}_{\mathcal{N}_s} : \boxed{1} \\ \text{ARG2}_{\mathcal{N}_s} : \boxed{2} \cdot \boxed{3} \end{bmatrix} \begin{bmatrix} \mathcal{V}_s \\ \text{INPUT}_{\mathcal{V}_s} : \textit{elist} \\ \text{ARG1}_{\mathcal{V}_s} : \boxed{1} \\ \text{ARG2}_{\mathcal{V}_s} : \boxed{2} \cdot \boxed{3} \end{bmatrix}$$

$$(4)\quad \begin{bmatrix} \mathcal{N}_s \\ \text{INPUT}_{\mathcal{N}_s} : \textit{elist} \\ \text{ARG1}_{\mathcal{N}_s} : \boxed{1} \cdot \boxed{2} \\ \text{ARG2}_{\mathcal{N}_s} : \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} \mathcal{N}in\mathcal{V}^{+} \\ \text{INPUT}_{\mathcal{N}in\mathcal{V}^{+}} : \textit{elist} \\ \text{ARG1}_{\mathcal{N}in\mathcal{V}^{+}} : \boxed{1} \\ \text{ARG2}_{\mathcal{N}in\mathcal{V}^{+}} : \boxed{3} \end{bmatrix} \begin{bmatrix} \mathcal{N}_s \\ \text{INPUT}_{\mathcal{N}_s} : \textit{elist} \\ \text{ARG1}_{\mathcal{N}_s} : \boxed{2} \\ \text{ARG2}_{\mathcal{N}_s} : \boxed{3} \end{bmatrix}$$

$$(5)\quad \begin{bmatrix} \mathcal{N}_s \\ \text{INPUT}_{\mathcal{N}_s} : \textit{elist} \\ \text{ARG1}_{\mathcal{N}_s} : \textit{elist} \\ \text{ARG2}_{\mathcal{N}_s} : \boxed{3} \end{bmatrix} \rightarrow \epsilon$$

$$(6)\quad \begin{bmatrix} \mathcal{N}in\mathcal{V}^{+} \\ \text{INPUT}_{\mathcal{N}in\mathcal{V}^{+}} : \textit{elist} \\ \text{ARG1}_{\mathcal{N}in\mathcal{V}^{+}} : \boxed{1} \\ \text{ARG2}_{\mathcal{N}in\mathcal{V}^{+}} : \boxed{2} \cdot \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} h \\ \text{INPUT}_{h} : \textit{elist} \\ \text{ARG1}_{h} : \boxed{1} \\ \text{ARG2}_{h} : \boxed{2} \end{bmatrix}$$

$$(7)\quad \begin{bmatrix} \mathcal{N}in\mathcal{V}^{+} \\ \text{INPUT}_{\mathcal{N}in\mathcal{V}^{+}} : \textit{elist} \\ \text{ARG1}_{\mathcal{N}in\mathcal{V}^{+}} : \boxed{1} \\ \text{ARG2}_{\mathcal{N}in\mathcal{V}^{+}} : \boxed{2} \cdot \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} \mathcal{N}in\mathcal{V}^{+} \\ \text{INPUT}_{\mathcal{N}in\mathcal{V}^{+}} : \textit{elist} \\ \text{ARG1}_{\mathcal{N}in\mathcal{V}^{+}} : \boxed{1} \\ \text{ARG2}_{\mathcal{N}in\mathcal{V}^{+}} : \boxed{3} \end{bmatrix}$$

Figure 4.7: The rules of G_{SCR} (part 1)

$$(8)\quad \begin{bmatrix} \mathcal{V}_s \\ \text{INPUT}_{\mathcal{V}_s} : \textit{elist} \\ \text{ARG1}_{\mathcal{V}_s} : \boxed{1} \\ \text{ARG2}_{\mathcal{V}_s} : \boxed{2} \cdot \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} \mathcal{V}in\mathcal{N}^+ \\ \text{INPUT}_{\mathcal{V}in\mathcal{N}^+} : \textit{elist} \\ \text{ARG1}_{\mathcal{V}in\mathcal{N}^+} : \boxed{2} \\ \text{ARG2}_{\mathcal{V}in\mathcal{N}^+} : \boxed{1} \end{bmatrix} \begin{bmatrix} \mathcal{V}_s \\ \text{INPUT}_{\mathcal{V}_s} : \textit{elist} \\ \text{ARG1}_{\mathcal{V}_s} : \boxed{1} \\ \text{ARG2}_{\mathcal{V}_s} : \boxed{3} \end{bmatrix}$$

$$(9)\quad \begin{bmatrix} \mathcal{V}_s \\ \text{INPUT}_{\mathcal{V}_s} : \textit{elist} \\ \text{ARG1}_{\mathcal{V}_s} : \boxed{1} \\ \text{ARG2}_{\mathcal{V}_s} : \textit{elist} \end{bmatrix} \rightarrow \epsilon$$

$$(10)\quad \begin{bmatrix} \mathcal{V}in\mathcal{N}^+ \\ \text{INPUT}_{\mathcal{V}in\mathcal{N}^+} : \textit{elist} \\ \text{ARG1}_{\mathcal{V}in\mathcal{N}^+} : \boxed{1} \\ \text{ARG2}_{\mathcal{V}in\mathcal{N}^+} : \boxed{2} \cdot \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} h \\ \text{INPUT}_{h} : \textit{elist} \\ \text{ARG1}_{h} : \boxed{2} \\ \text{ARG2}_{h} : \boxed{1} \end{bmatrix}$$

$$(11)\quad \begin{bmatrix} \mathcal{V}in\mathcal{N}^+ \\ \text{INPUT}_{\mathcal{V}in\mathcal{N}^+} : \textit{elist} \\ \text{ARG1}_{\mathcal{V}in\mathcal{N}^+} : \boxed{1} \\ \text{ARG2}_{\mathcal{V}in\mathcal{N}^+} : \boxed{2} \cdot \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} \mathcal{V}in\mathcal{N}^+ \\ \text{INPUT}_{\mathcal{V}in\mathcal{N}^+} : \textit{elist} \\ \text{ARG1}_{\mathcal{V}in\mathcal{N}^+} : \boxed{1} \\ \text{ARG2}_{\mathcal{V}in\mathcal{N}^+} : \boxed{3} \end{bmatrix}$$

$$(12)\quad \begin{bmatrix} \mathcal{N} \\ \text{INPUT}_{\mathcal{N}} : \textit{elist} \\ \text{ARG1}_{\mathcal{N}} : \langle n_1 \rangle \end{bmatrix} \rightarrow \epsilon \quad \ldots \quad \begin{bmatrix} \mathcal{N} \\ \text{INPUT}_{\mathcal{N}} : \textit{elist} \\ \text{ARG1}_{\mathcal{N}} : \langle n_l \rangle \end{bmatrix} \rightarrow \epsilon$$

$$(13)\quad \begin{bmatrix} \mathcal{V} \\ \text{INPUT}_{\mathcal{V}} : \textit{elist} \\ \text{ARG1}_{\mathcal{V}} : \langle v_1 \rangle \end{bmatrix} \rightarrow \epsilon \quad \ldots \quad \begin{bmatrix} \mathcal{V} \\ \text{INPUT}_{\mathcal{V}} : \textit{elist} \\ \text{ARG1}_{\mathcal{V}} : \langle v_m \rangle \end{bmatrix} \rightarrow \epsilon$$

$$(14)\quad \begin{bmatrix} h \\ \text{INPUT}_{h} : \textit{elist} \\ \text{ARG1}_{h} : \langle n_1 \rangle \\ \text{ARG2}_{h} : \langle v_1 \rangle \end{bmatrix} \rightarrow \epsilon \quad \ldots \quad \begin{bmatrix} h \\ \text{INPUT}_{h} : \textit{elist} \\ \text{ARG1}_{h} : \langle n_l \rangle \\ \text{ARG2}_{h} : \langle v_m \rangle \end{bmatrix} \rightarrow \epsilon$$

Figure 4.8: The rules of G_{SCR} (part 2)

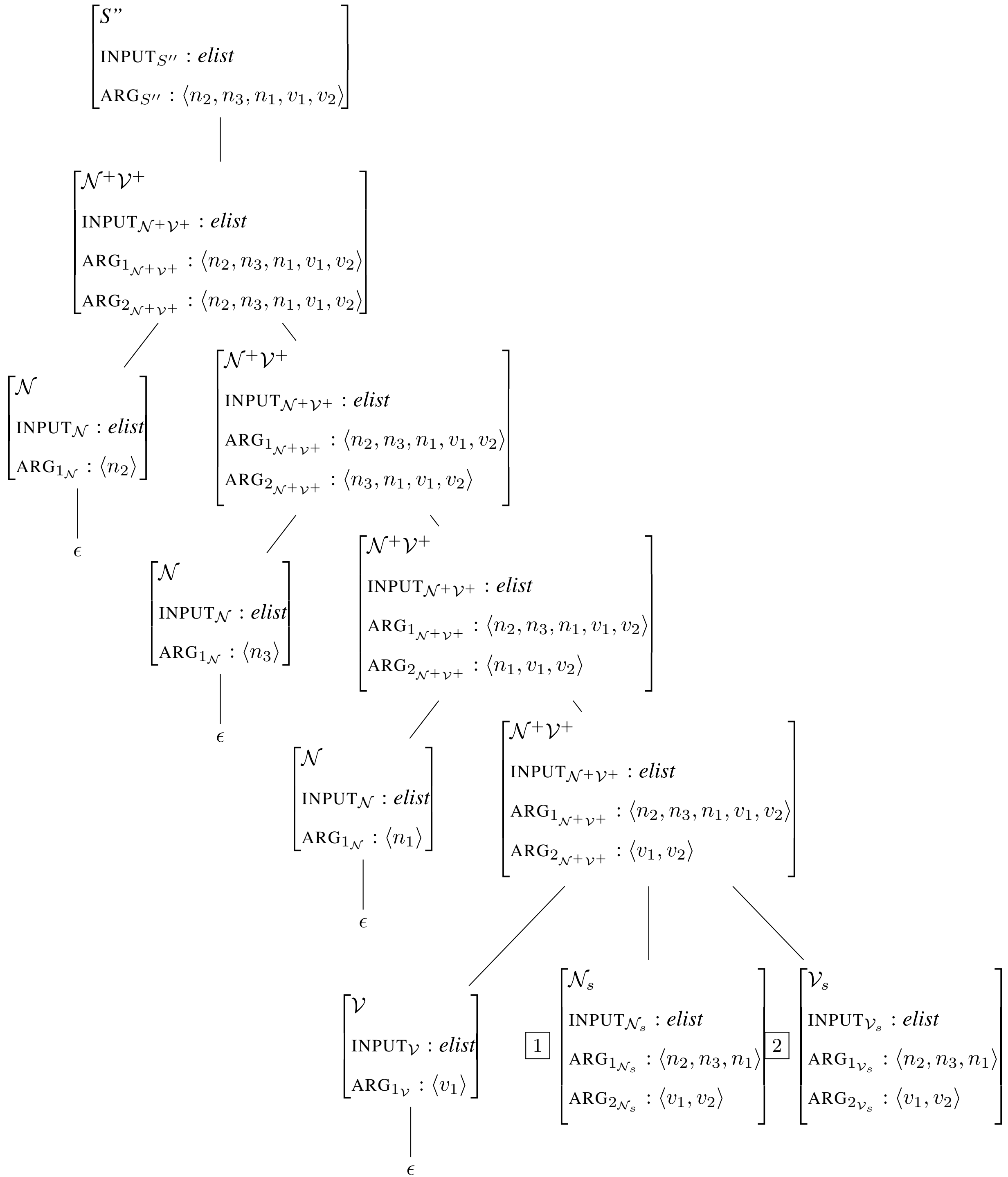


Figure 4.9: Derivation tree of the string $n_2 n_3 n_1 v_1 v_2$ (part 1)

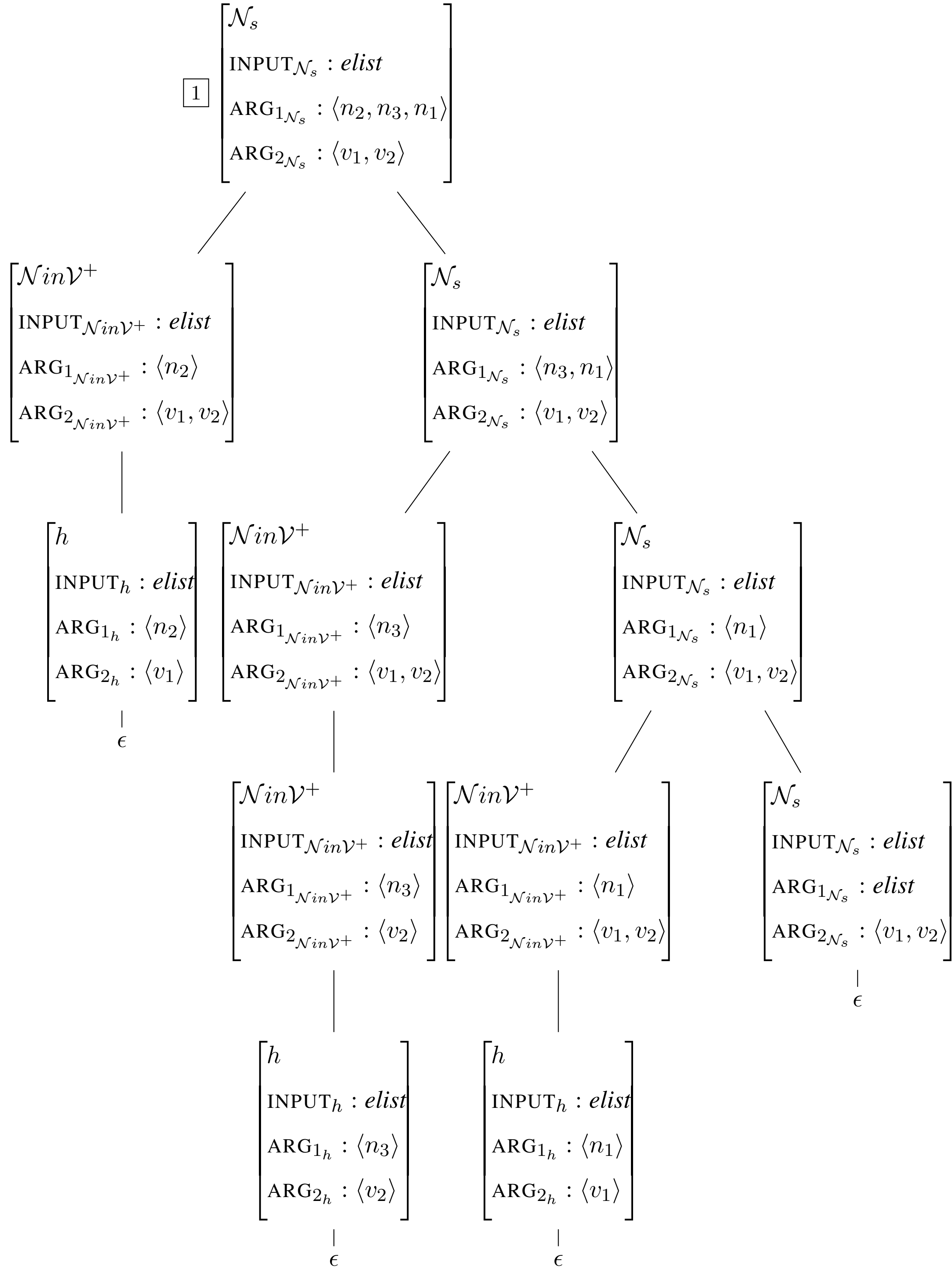


Figure 4.10: Derivation tree of the string $n_2 n_3 n_1 v_1 v_2$ (part 2)

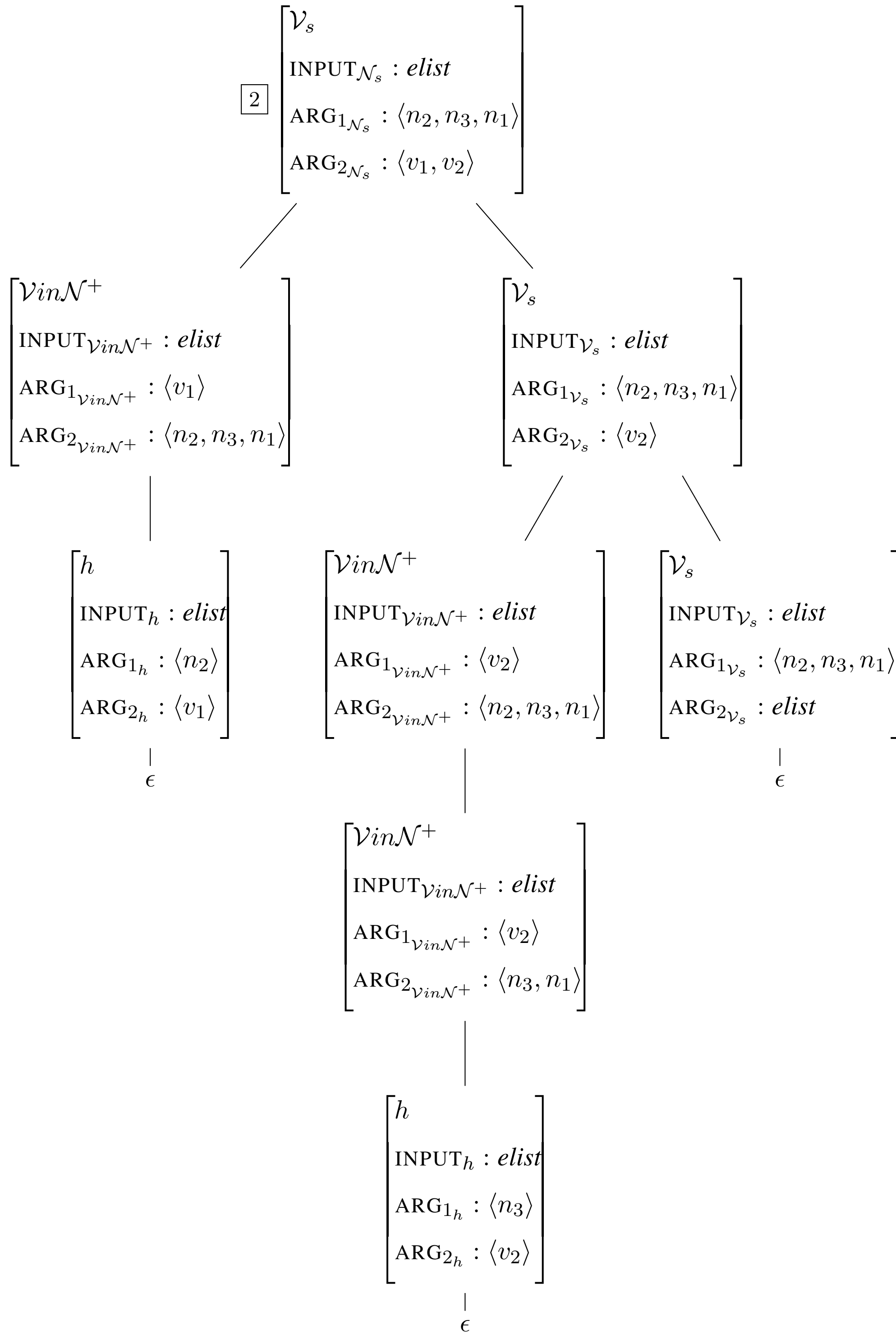


Figure 4.11: Derivation tree of the string $n_2 n_3 n_1 v_1 v_2$ (part 3)

Chapter 5

A sketch of the proof

In this section we show a sketch of the proof of the main result of this work, namely that $\mathcal{L}_{RTUG} = \mathcal{L}_{RCG}$. The full proof is deferred to Appendix B.

In order to prove that $\mathcal{L}_{RTUG} = \mathcal{L}_{RCG}$, we first prove that $\mathcal{L}_{RCG} \subseteq \mathcal{L}_{RTUG}$, by proving the correctness of *RCG2TUG* mapping (Definition 34), and then that $\mathcal{L}_{RTUG} \subseteq \mathcal{L}_{RCG}$, by proving the correctness of *TUG2RCG* mapping (Definition 30). To do so, we first define an intermediate grammar, called **the instantiated grammar**.

5.1 Instantiated grammar

As a technical aid, we first define, for a constrained unification grammar G, a set of *instantiated grammars*. Each grammar in this set is designed to generate at most one word. More precisely, the instantiated grammar $G_{|_w}$ is obtained from G by restricting it to a specific word w, such that $L(G_{|_w}) = \{w\}$ if $w \in L(G)$, and is empty otherwise. Crucially, while $G_{|_w}$ is a unification grammar, it is formally equivalent to a context-free grammar. We provide the informal description below, while the formal definitions are deferred to section B.1.

We start by defining **instantiated *bi_list*s, TFSs and rules**, in a similar way to instantiated predicates and clauses (Definition 15). Given a word $w \in \textsc{Words}^*$, a **list instantiation of** w is a TFS of type *bi_list* whose content is a substring of w (see Definition 56).

Example 25 (list instantiation)**.** *Let $w = abbb$ and $B = \langle a \rangle \cdot \boxed{1}$bi_list. $IB = \langle a, b, b \rangle$ is a list instantiation of B and w.*

Given a word $w \in \textsc{Words}^*$, and a main TFS A, **the instantiated TFS of A and w** is a maximally specific main TFS IA, such that $A \sqsubseteq IA$, and where the contents of the feature values are substrings of w (see Definition 57).

Example 26 (Instantiated TFS)**.** *Let A be a TFS over the signature presented in Example 14, and $w =$*

aabb. Let

$$A = \begin{bmatrix} \textit{counter} \\ \text{INPUT} : \boxed{1}\,\textit{bi_list} \\ \text{COUNTER} : \langle a \rangle \cdot \boxed{1} \end{bmatrix}$$

The following TFS, IA, is an instantiated TFS of A and w:

$$IA = \begin{bmatrix} \textit{at} \\ \text{INPUT} : \langle a \rangle \\ \text{COUNTER} : \langle a, a \rangle \end{bmatrix}$$

Given a word $w \in \text{WORDS}^*$, an RTUG G over S, and a rule $r \in \mathcal{R}$, r' is an **instantiated rule of** r **and** w, if r subsumes r', and every TFS of r' is an instantiated TFS of w (see Definition 58).

Example 27. *(Rule instantiation) Let*

$$r = \begin{bmatrix} \textit{at} \\ \text{INPUT}_{counter} : \boxed{1} \cdot \boxed{2} \\ \text{COUNT} : \langle a \rangle \cdot \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} \textit{at} \\ \text{INPUT}_{counter} : \boxed{1}\,\textit{bi_list} \\ \text{COUNT} : \langle a \rangle \end{bmatrix} \begin{bmatrix} \textit{bt} \\ \text{INPUT}_{counter} : \boxed{2}\,\textit{bi_list} \\ \text{COUNT} : \boxed{3}\,\textit{bi_list} \end{bmatrix},$$

and $w = aabb$. The following is an instantiated rule of r and w:

$$r' = \begin{bmatrix} \textit{at} \\ \text{INPUT}_{counter} : \langle a, b, b \rangle \\ \text{COUNT} : \langle a, b, b \rangle \end{bmatrix} \rightarrow \begin{bmatrix} \textit{at} \\ \text{INPUT}_{counter} : \langle a \rangle \\ \text{COUNT} : \langle a \rangle \end{bmatrix} \begin{bmatrix} \textit{bt} \\ \text{INPUT}_{counter} : \langle b, b \rangle \\ \text{COUNT} : \langle b, b \rangle \end{bmatrix},$$

where $\langle a \rangle$ is the list instantiation of $\boxed{1}$, and $\langle b, b \rangle$ is the list instantiation of $\boxed{2}$ and $\boxed{3}$ (see the definition of list instantiation above).

The set of all instantiated rules of G and w is **the instantiated grammar of** G **and** w, denoted by $G_{|w}$ (see Definition 57). For every RTUG G and $w \in \text{WORDS}^*$, $w \in L(G)$ if and only if $w \in L\left(G_{|w}\right)$ (Lemmas 1 and 2, Appendix B.1).

5.2 Direction 1: $\mathcal{L}_{RCG} \subseteq \mathcal{L}_{RTUG}$

For every RCG G, there exists an RTUG G_{tug}, such that $L(G) = L(G_{tug})$. Obviously, we choose $G_{tug} = \textit{RCG2TUG}(G)$, and show first that $L(G) \subseteq L(G_{tug})$, and then that $L(G_{tug}) \subseteq L(G)$. The formal proof is detailed in Appendix B.2.

First we show (Lemmas 3 and 4) the commutativity of string instantiation and *bi_list* instantiation

(Definition 56) with *arg2feat* (Definition 35). See the commutative diagram below:

$$\begin{array}{ccccc} & \alpha & \xrightarrow{\textit{arg2feat}} & A & = & \textit{arg2feat}\,(\alpha) \\ \text{string inst.} & \downarrow & & \downarrow & \text{list inst.} & \\ & u & \xrightarrow{\textit{arg2feat}} & B & = & \textit{arg2feat}\,(u) \end{array}$$

Then we show (Lemmas 5 and 7) the commutativity of instantiation (of predicates, Definition 15, and of TFSs, Definition 57) with *pred2tfs* (Definition 35). See the commutative diagram below:

$$\begin{array}{ccccc} & \varphi & \xrightarrow{\textit{pred2tfs}} & A & = & \textit{pred2tfs}\,(\varphi) \\ \text{pred. inst.} & \downarrow & & \downarrow & \text{TFS inst.} & \\ & \psi & \xrightarrow{\textit{pred2tfs}} & IA & = & \textit{pred2tfs}\,(\psi) \end{array}$$

Theorem 9 proves that $L(G) \subseteq L(G_{tug})$ by sowing that if $w \in L(G)$, then $w \in L\left(G_{tug_{|w}}\right)$, implies that $w \in L(G_{tug})$. Theorem 11 proves that $L(G_{tug}) \subseteq L(G)$ by sowing that if $w \in L\left(G_{tug_{|w}}\right)$, then $w \in L(G)$. Recall that $w \in L\left(G_{tug_{|w}}\right)$ if and only if $w \in L(G_{tug})$.

5.3 Direction 2: $\mathcal{L}_{RTUG} \subseteq \mathcal{L}_{RCG}$

Conversely, for every RTUG G, there exists an RCG G_{rcg}, such that $L(G) = L(G_{rcg})$. In a similar way to Direction 1 of the proof, we will choose $G_{rcg} = \textit{TUG2RCG}\,(G)$, as defined in Definition 30, and show first that $L(G) \subseteq L(G_{rcg})$, and then that $L(G_{rcg}) \subseteq L(G)$. The formal proof is detailed in Appendix B.3.

First, we define **a hierarchy over non-terminals and predicates** of RCG that is equivalent to the hierarchy over types and TFSs of RTUG: In general, given an RTUG G over a signature S, and an RCG $G_{rcg} = \textit{TUG2RCG}\,(G)$, we say that the non-terminal $N_t \in N$ **subsumes** the non-terminal $N_s \in N$, if the type t subsumes the type s in S (see Definition 60). We say that a predicate φ **subsumes** the predicate ψ, if the non-terminal of φ subsumes the non-terminal of ψ, and every argument of φ is a string instantiation of the corresponding argument of φ (see Definition 61). A predicate that is subsumed by no other predicate is called **a maximum predicate**.

Example 28. *Consider $G_{longdist}$ and TUG2RCG $(G_{longdist})$ of Example 15:*

- *v_subcat (X) subsumes v_np $(loves)$, because v_subcat $\sqsubseteq$ v_np;*
- *v_np is a maximum type and v_np $(loves)$ is a maximum predicate.*

Lemmas 14 and 15 show that string instantiation and *bi_list* instantiation (Definition 56) commute

with *feat2arg* (Definition 32). See the commutative diagram below:

$$\begin{array}{rccc} & B & \overset{\textit{feat2arg}}{\longrightarrow} & \alpha = \textit{feat2arg}\,(B) \\ \text{list inst.} & \downarrow & & \downarrow \text{ string inst.} \\ & C & \overset{\textit{feat2arg}}{\longrightarrow} & \rho = \textit{feat2arg}\,(C) \end{array}$$

Lemma 16 deals with the commutativity of instantiation and the mapping between TFSs and predicates. Unlike the previous direction, in a general RTUG a TFS A and its instantiated TFS IA can be of different types. In this case, we cannot claim that *tfs2pred* (IA) is an instantiated predicate of *tfs2pred* (A), since they may have different non-terminals. What we can claim, however, is that *tfs2pred* (A) **subsumes** *tfs2pred* (IA), as defined in Definition 61. See the commutative diagram below:

$$\begin{array}{rccc} & A & \overset{\textit{tfs2pred}}{\longrightarrow} & \varphi = \textit{tfs2pred}\,(A) \\ \text{TFS inst.} & \downarrow & & \downarrow \text{ subsumes} \\ & IA & \overset{\textit{tfs2pred}}{\longrightarrow} & \psi = \textit{tfs2pred}\,(IA) \end{array}$$

Example 29. *Consider the following fragment of the signature of $G_{longdist}$, repeated from Example 15:*

Signature

main

  v_subcat INPUT$_{vs}$:*bi_list*

    v_np

    v_s

...

Consider further the TFS

$$A = \begin{bmatrix} \textit{v_subcat} \\ \text{INPUT}_{vs} : \boxed{1}\,\textit{bi_list} \end{bmatrix}$$

Let $w = loves$, so the instantiated TFS of A and w is:

$$IA = \begin{bmatrix} \textit{v_np} \\ \text{INPUT}_{vs} : \langle loves \rangle \end{bmatrix}$$

$$\begin{array}{lclcl} \textit{tfs2pred}\,(A) & = & \varphi & = & v_subcat\,(X)\,, \quad X \in V \\ \textit{tfs2pred}\,(IA) & = & \psi & = & v_np\,(loves) \end{array}$$

Clearly, ψ is not an instantiated predicate of φ. However, given the unification clauses of the grammar

TUG2RCG $(G_{longdist})$*:*

$$\begin{array}{rcl} \textit{v_subcat}\,(X) & \rightarrow & \textit{v_np}\,(X) \\ \textit{v_subcat}\,(X) & \rightarrow & \textit{v_s}\,(X) \end{array}$$

we can see that v_subcat subsumes v_np and φ subsumes ψ.

Lemma 19 proves that if *tfs2pred* (A) subsumes *tfs2pred* (B), then $A \sqsubseteq B$. See the commutative diagram below:

$$\begin{array}{rccc} & A & \stackrel{\textit{tfs2pred}}{\longrightarrow} & \varphi = \textit{tfs2pred}\,(A) \\ \sqsubseteq & \downarrow & & \downarrow \text{subsumes} \\ & B & \stackrel{\textit{tfs2pred}}{\longrightarrow} & \psi = \textit{tfs2pred}\,(B) \end{array}$$

In Theorem 18 we prove that $L(G) \subseteq L(G_{rcg})$ by showing that if $w \in L\left(G_{|w}\right)$, then $w \in L(G_{rcg})$. Recall that $w \in L\left(G_{|w}\right)$ if and only if $w \in L(G)$. In Theorem 21 we prove that $L(G_{rcg}) \subseteq L(G)$ by showing that if $w \in L\left(G_{rcg_{|w}}\right)$, then $w \in L(G)$.

Chapter 6

Conclusions

The main contribution of this work is the definition of a restricted version of typed unification grammars, RTUG, which is polynomially-parsable. Furthermore, RTUG generates exactly the class of languages recognizable in deterministic polynomial time. We prove this result by showing a conversion algorithm between RTUG and Range Concatenation Grammar (RCG), a grammatical formalism that generates exactly the class of polynomially recognizable languages. In this work we also demonstrate RTUGs that generate formal languages, $a^n b^n c^n$ and a^{prime}, and RTUGs that describe natural languages phenomena, long distance dependencies and word scrambling.

But still, RTUG is a highly restricted UG, allowing features of a single type only, bi-directional lists of terminals. This fact makes the development of grammars in this formalism rather difficult. Comparing RTUG to other highly restricted versions of UG, **One-reentrant unification grammars** and **PLPATR** (see details in Section 1.2), RTUG rules and feature structures are very limited in the type of values their features are allowed to take. At the same time, RTUG imposes no constraints on grammar rule reentrancies. One-reentrant UG and PLPATR, on the other hand, do not limit the values of the features, while reentrancy is extremely limited. Both formalisms generate classes of languages that are strictly included in the class of polynomially recognizable languages (TAL and LCFRS). A possible extension of this work would therefore be a new formalism combining the benefits of RTUG and one-reentrant UG or PLPATR. In this combined formalism feature structures will allow features of type bi-directional lists of terminal, in which reentrancy is not limited, along with other features, with unlimited values, where reentrancy is limited, according to the constraints of one-reentrant UG or PLPATR. Such a combined formalism will facilitate the design of natural grammars, allowing simple implementation of linguistic phenomena like agreement, while at the same time, will add nothing to the expressivity of RTUG, thereby generating exactly the class of polynomially recognizable languages.

Bibliography

G. Edward Barton, Robert C. Berwick, and Eric S. Ristad. *Computational Complexity and Natural Language*. MIT Press, Cambridge, MA, USA, 1987. ISBN 0262022664.

Tilman Becker, Aravind K. Joshi, and Owen Rambow. Long-distance scrambling and tree adjoining grammars. In *Proceedings of the fifth conference on European chapter of the Association for Computational Linguistics*, pages 21–26, Stroudsburg, PA, USA, 1991. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/977180.977185. URL `http://dx.doi.org/10.3115/977180.977185`.

Pierre Boullier. A generalization of mildly context-sensitive formalisms. In *Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks*, pages 17–20, Philadelphia, 1998a. University of Pennsylvania.

Pierre Boullier. Proposal for a natural language processing syntactic backbones. Research Report 3342, INRIA-Rocquencourt, France, 1998b.

Pierre Boullier. Chinese numbers, MIX, scrambling, and range concatenation grammars. In *Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics*, pages 53–60, Morristown, NJ, USA, 1999. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/977035.977044.

Pierre Boullier. Range concatenation grammars. In John Carroll Harry Bunt and Giorgio Satta, editors, *New Developments in Parsing Technology*, pages 269–289. Springer, Netherlands, 2000.

Bob Carpenter. *The Logic of Typed Feature Structures*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1992.

Daniel Feinstein and Shuly Wintner. Highly constrained unification grammars. *Journal of Logic, Language and Information*, 17(3):345–381, 2008. URL `http://dx.doi.org/10.1007/s10849-008-9062-9`.

Nissim Francez and Shuly Wintner. *Unification Grammars*. Cambridge University Press, Cambridge, 2012.

Efrat Jaeger, Nissim Francez, and Shuly Wintner. Unification grammars and off-line parsability. *J. of Logic, Lang. and Inf.*, 14(2):199–234, 2005. ISSN 0925-8531. doi: http://dx.doi.org/10.1007/s10849-005-4511-1.

Mark Johnson. *Attribute-Value Logic and the Theory of Grammar*. CSLI lecture notes ; 16. Center for the Study of Language and Information, Stanford University, Stanford, CA, 1988.

Laura Kallmeyer, Wolfgang Maier, and Yannick Parmentier. An Earley parsing algorithm for range concatenation grammars. In *Joint conference of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing (ACL-IJCNLP 2009)*, Suntec Singapore, 2009. URL `http://hal.inria.fr/inria-00393980/en/`.

Bill Keller and David Weir. A tractable extension of linear indexed grammars. In *Proceedings of the seventh conference on European chapter of the Association for Computational Linguistics*, pages 75–82, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. doi: http://dx.doi.org/10.3115/976973.976985.

Appendix A

Representing bidirectional lists with TFSs

RCG arguments are strings of terminals and variables, where in each derivation step, these strings are being split or concatenated. In order to manipulate strings and substrings thereof with UG, we define an infrastructure for handling bi-directional lists with TFSs. This infrastructure includes:

- bi_list nodes which are TFSs with three features:
  - CURR which includes the actual value of the node;
  - PREV which points to the previous node in the list;
  - NEXT which points to the next node in the list.
- bi_list TFSs that represent bi-directional lists and have two features:
  - HEAD which points to the first node of the list;
  - TAIL which points to the last node of the list.

Definition 37 (Bi_list type)**.** *The type node is defined as follows:*

- $\bot \overset{\circ}{\sqsubset}$ *node, and node is featureless;*
- *node* $\overset{\circ}{\sqsubset}$ *null, and null is featureless;*
- *node* $\overset{\circ}{\sqsubset}$ *ne_node, and:*
  - $Approp\,(\mathit{ne_node}, \textsc{curr}) = $ *terminal;*
  - $Approp\,(\mathit{ne_node}, \textsc{prev}) = $ *node;*
  - $Approp\,(\mathit{ne_node}, \textsc{next}) = $ *node;*
  - $Approp\,(\mathit{ne_node}, f) \uparrow$*, for every* $f \notin \{\textsc{curr}, \textsc{prev}, \textsc{next}\}$

The type bi_list is defined as follows:

- $\bot \overset{\circ}{\sqsubset}$ *bi_list, and bi_list is featureless;*

- *bi_list* $\overset{\circ}{\sqsubset}$ *elist, and elist is featureless;*
- *bi_list* $\overset{\circ}{\sqsubset}$ *ne_bi_list, and:*
  - $Approp\,(ne_bi_list, \textsc{head}) = ne_node$*;*
  - $Approp\,(ne_bi_list, \textsc{tail}) = ne_node$*;*
  - $Approp\,(ne_bi_list, f)\uparrow$*, for every* $f \notin \{\textsc{head}, \textsc{tail}\}$

A.1 Restrictions over node TFSs and bi_list TFSs

In the following, we define restriction over node TFSs and bi_list TFSs, such that list operations like concatenation and sublist can be defined.

Definition 38 (Node TFS)**.** *A **node TFS** is a TFS whose type subsumes node. A TFS of type node is called an **implicit node**, and a TFS of type null or ne_node is called a **non-implicit node**.*

Definition 39 (Bi_list TFS)**.** *A **bi_list TFS** is a TFS whose type subsumes bi_list.*

In a non-implicit node, the value of the feature PREV must point to the previous node in the list, and the value of the feature NEXT must point to the next node in the list.

Definition 40 (Valid prev-node)**.** *Let A be a non-implicit node TFS. The TFS B is a **valid prev-node** of A if either:*

- *B is an implicit node, or*
- *B = null, or*
- *B is a non-implicit node, in which:*
  - *the value of* NEXT *is A, and*
  - *the value of* PREV *is a valid prev-node of B.*

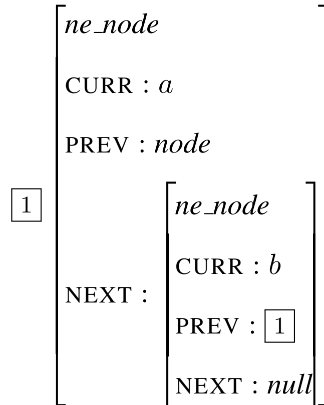
Definition 41 (Valid next-node)**.** *Let A be a non-implicit node TFS. The TFS B is a **valid next-node** of A if either:*

- *B is an implicit node, or*
- *B = null, or*
- *B is a non-implicit node, in which:*
  - *the value of* PREV *is A, and,*
  - *the value of* NEXT *is a valid next-node of B.*

Definition 42 (Valid node)**.** *a node TFS A is a **valid node** if either:*

- *A is an implicit node, or*
- *$A = null$, or*
- *A is a non-implicit node, in which:*
  - *the value of the feature* PREV *is a valid prev-node of A, and,*
  - *the value of the feature* NEXT *is a valid next-node of A*

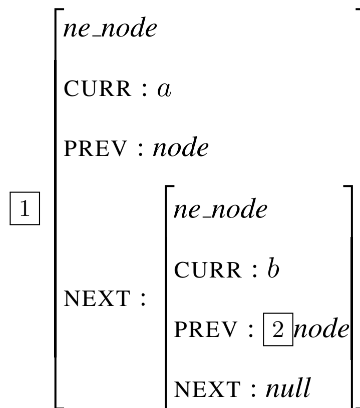
Example 30. *(valid nodes) Let terminal $\sqsubseteq a$ and terminal $\sqsubseteq b$. The following TFS is a valid node:*



Observe that:

- *The* PREV *value of* [1] *has an implicit node,*
- *The* NEXT *value of* [1] *has a non-implicit node whose* PREV *value points back to* [1].

The following TFS is not a valid node:



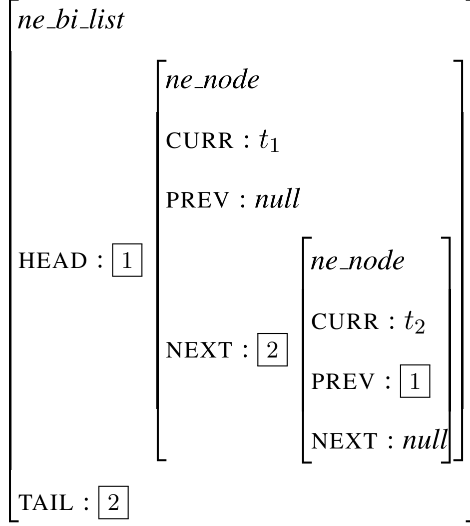
Observe that the NEXT *value of* [1] *has a non-implicit node whose* PREV *value points to a node other than* [1].

Definition 43 (Valid bi_list TFS)**.** *A TFS A is a **valid bi_list TFS** if either:*

- *$A = elist$, or*

- *A is of type ne_bi_list where:*
  - *the value of* HEAD*is a valid node whose* PREV *value is null.*
  - *the value of* TAIL *is a valid node whose* NEXT *value is null.*

Example 31. *(Valid bi_list TFS) Let t_1, t_2 be types in* TYPES*, such that terminal $\sqsubseteq t_1$ and terminal $\sqsubseteq t_2$. The following TFS is a valid bi_list TFS:*



A.2 Explicit bi_lists

In the following we define k-head-explicit (k-tail-explicit) bi_list, which is a bi_list whose first (last) k members are explicitly defined. This is useful for defining several operations on bi_lists.

Definition 44 (k-head-explicit node)**.** *k**-head-explicit node** is defined recursively as follows:*

- *A valid non-implicit node A is a **1-head-explicit node** if its* NEXT *value is an implicit node.*
- *A valid non-implicit node A is a k**-head-explicit node** if its* NEXT *value is a (k-1)-head-explicit node.*

Definition 45 (k-tail-explicit node)**.** *k**-tail-explicit node** is defined recursively as follows:*

- *A valid non-implicit node A is a **1-tail-explicit node** if its* PREV *value is an implicit node.*
- *A valid non-implicit node A is a k**-tail-explicit node** if its* PREV *value is a (k-1)-tail-explicit node.*

Definition 46 (k-head-explicit bi_list)**.** *A bi_list TFS A is a k**-head-explicit bi_list** if A is of type ne_bi_list, and its* HEAD *value is a k-head-explicit node.*

Definition 47 (k-tail-explicit bi_list)**.** *A bi_list TFS A is a k**-tail-explicit bi_list** if A is of type ne_bi_list, and its* TAIL *value is a k-tail-explicit node.*

Definition 48 (The i-node)**.** *Let A be a k-head-explicit bi_list, such that $k \geq 1$.*

- *the* 1***-node*** *of A is the value of* HEAD *feature of A.*
- *for every i, $2 \leq i \leq k$, the i**-node** of A is the value of the* NEXT *feature of the $(i-1)$-node of A.*

Explicit bi_list is a bi_list which all of its members are explicit nodes.

Definition 49 (Explicit bi_list)**.** *A bi_list TFS A is an **explicit bi_list** if:*

- *$A = elist$, or,*
- *A is a k-head-explicit bi_list for some $k \leq 1$, such that the feature value of* TAIL *is the k-node of A. We say that k is the **length** of A.*

Definition 50 (The i-element)**.** *Let A be a k-head-explicit bi_list. For every i, $1 \leq i \leq k$, the i**-element** of A is the value of the* CURR *feature of the i-node of A (which is a terminal).*

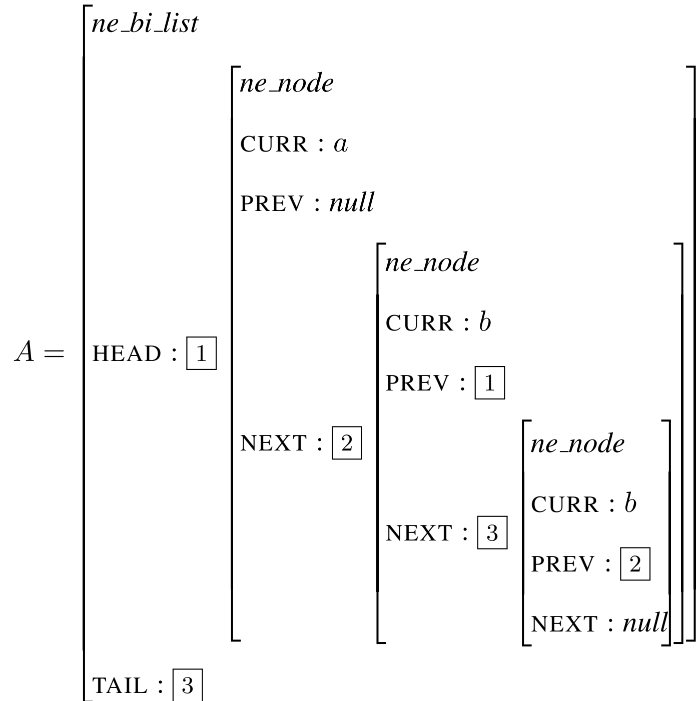
In the following, we sometimes use list notation ($\langle \ldots \rangle$) to describe TFSs of type *bi_list*.

Definition 51 (list notation)**.** *Let A be an explicit bi_list TFS, with length $k \geq 1$. Then A can be denoted as follows:*

$$A = \langle C_1, \ldots, C_k \rangle$$

where, for every i, $1 \leq i \leq k$, C_i is the i-element of A.

Example 32 (list notation of a *bi_list*)**.** *Let A be a bi_list TFS, such that terminal $\sqsubseteq$ a, and terminal $\sqsubseteq$ b:*



then:

- *the length of A is 3;*
- *the* 1*-node of A is* $\boxed{1}$*, the* 2*-node is* $\boxed{2}$ *and the* 3*-node is* $\boxed{3}$*;*
- *the* 1*-element of A is a, the* 2*-element is b and the* 3*-element is b;*
- *the list notation of A is $\langle a, b, b\rangle$;*

A.3 Bi_list operations

In the following we define several operations on bi_list TFSs.

Definition 52 (pop_head)**.** *Let A and B be two bi_list TFSs, such that the length of A is $k \geq 1$. $B = pop_head\,(A)$ if:*

- *if $k = 1$:*

$$A = \begin{bmatrix} \textit{ne_bi_list} \\ \text{HEAD} : \boxed{1}\textit{node} \\ \text{TAIL} : \boxed{1} \end{bmatrix}$$

  then

$$B = \textit{elist}$$

- *if $k \geq 2$*

$$A = \begin{bmatrix} \textit{ne_bi_list} \\ \text{HEAD} : \boxed{1}\begin{bmatrix} \textit{ne_node} \\ \text{CURR} : t_1 \\ \text{PREV} : \textit{null} \\ \text{NEXT} : \begin{bmatrix} \textit{ne_node} \\ \text{CURR} : t_2 \\ \text{PREV} : \boxed{1} \\ \text{NEXT} : \boxed{2}\textit{node} \end{bmatrix} \end{bmatrix} \\ \text{TAIL} : \boxed{3}\textit{node} \end{bmatrix}$$

where terminal $\sqsubseteq t_1$ *and terminal* $\sqsubseteq t_2$*, then*

$$B = \begin{bmatrix} \textit{ne_bi_list} & \\ \text{HEAD} : & \begin{bmatrix} \textit{ne_node} \\ \text{CURR} : t_2 \\ \text{PREV} : \textit{null} \\ \text{NEXT} : \boxed{2}\,\textit{node} \end{bmatrix} \\ \text{TAIL} : \boxed{3}\,\textit{node} & \end{bmatrix}$$

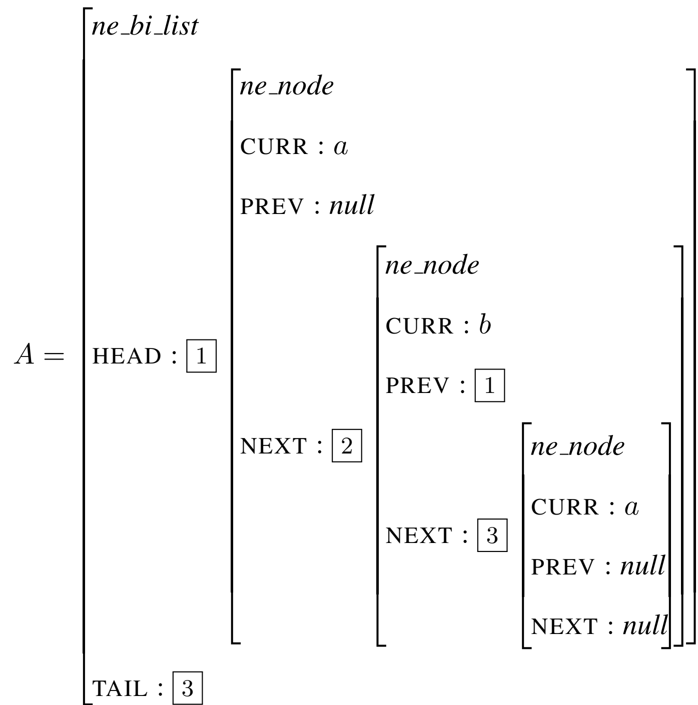
Definition 53 (concatenation of bi_lists)**.** *Let* A_1 *and* A_2 *be two bi_list TFSs, with length* k_1 *and* k_2*, respectively. A TFS bi_list* A *is a* ***concatenation*** *of* A_1 *and* A_2*, denoted by* $A = A_1 \cdot A_2$*, if:*

- *if* $A_2 =$ *elist, then* $A = A_1$*;*
- *if* $A_1 =$ *elist, then* $A = A_2$*;*
- *if* $A_1 \neq$ *elist and* $A_2 \neq$ *elist, we assume, without loss of generality, that the reentrancy tags of* A_1 *and* A_2 *are disjoint. Then* A *is a bi_list of t of length* $k = k_1 + k_2$ *such that:*
  - *for every* i*,* $1 \leq i \leq k_1 - 1$*, the* i*-node of* A *is the* i*-node of* A_1*.*
  - *the* k_1*-node of* A *is the* k_1*-node of* A_1*, whose* NEXT *feature points to the* 1*-node of* A_2*.*
  - *the* $k_1 + 1$*-node of* A *is the* 1*-node of* A_2*, whose* PREV *feature points to the* k_1*-node of* A_1*.*
  - *for every* j*,* $2 \leq j \leq k_2$*, the* $k_1 + j$*-node of* A *is the* j*-node of* A_2*.*

Example 33. *(concatenation of bi_list TFSs). Let* $\textit{terminal}, a, b \in \text{TYPES}$*, such that terminal* $\sqsubseteq a$ *and terminal* $\sqsubseteq b$*. Let* A_1 *and* A_2 *be two TFSs of type ne_bi_list as follows:*

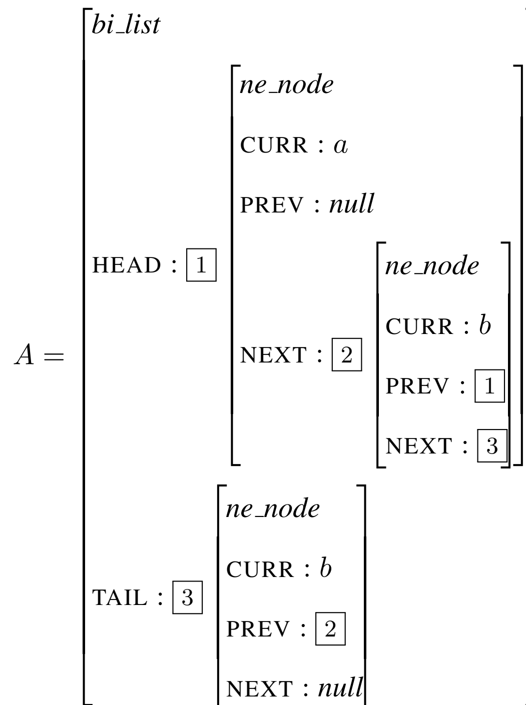
$$A_1 = \begin{bmatrix} \textit{ne_bi_list} & \\ \text{HEAD} : \boxed{1} & \begin{bmatrix} \textit{ne_node} & \\ \text{CURR} : a & \\ \text{PREV} : \textit{null} & \\ \text{NEXT} : \boxed{2} & \begin{bmatrix} \textit{ne_node} \\ \text{CURR} : b \\ \text{PREV} : \boxed{1} \\ \text{NEXT} : \textit{null} \end{bmatrix} \end{bmatrix} \\ \text{TAIL} : \boxed{2} & \end{bmatrix} \quad A_2 = \begin{bmatrix} \textit{ne_bi_list} & \\ \text{HEAD} : \boxed{3} & \begin{bmatrix} \textit{ne_node} \\ \text{CURR} : a \\ \text{PREV} : \textit{null} \\ \text{NEXT} : \textit{null} \end{bmatrix} \\ \text{TAIL} : \boxed{3} & \end{bmatrix}$$

Then $A = A_1 \cdot A_2$ *is:*



Definition 54 (sublist)**.** *Let* A *and* B *be two bi_list TFSs.* B *is a* ***sublist*** *of* A*, if there exist two bi_list TFSs,* C *and* D*, such that* $A = C \cdot B \cdot D$*.*

Example 34. *(list notation and sublists of a bi_list) Let* A *be a TFS of type bi_list:*



then the list notation of A is $A = \langle a, b, b \rangle$ and the sublists of A (in list notation) are:

$$\langle a, b, b \rangle \ , \langle a, b \rangle \ , \langle b, b \rangle \ , \langle a \rangle \ , \langle b \rangle \, , \textit{elist}.$$

Appendix B

Proofs of the main results

This section proves that the class of RTULs , $\mathcal{L}_{RTUG}$, is equivalent to the class RCLs, $\mathcal{L}_{RCG}$. First we prove that $\mathcal{L}_{RCG} \subseteq \mathcal{L}_{RTUG}$ (Section B.2), and then that $\mathcal{L}_{RTUG} \subseteq \mathcal{L}_{RCG}$ (Section B.3). To do so, we first define an *instantiated grammar*, $G_{|_w}$, over an RTUG G and a word w, such that $w \in L\left(G_{|_w}\right)$ if and only if $w \in L(G)$. We use instantiated grammars as intermediate grammars, to show that for a given RTUG G and a given word w , there is an RCG G_{RCG}, such that $w \in L(G_{RCG})$ if and only if $w \in L\left(G_{|_w}\right)$, hence, $w \in L(G)$, and vice versa.

B.1 Instantiated grammar

We start by defining **instantiated TFSs and rules**, in a similar way to instantiated predicates and clauses (Definition 15). Given a restricted signature S and a string $w \in \text{WORDS}^*$, an instantiated TFS of w is a maximally specific TFS over S, such that the content of every feature value of type *bi_list* is a substring of w (see the definition of *bi_list* content below). Given a word $w \in \text{WORDS}$, an RTUG G over S, and a rule $r \in \mathcal{R}$, r' is an **instantiated rule** of r and w, if r subsumes r', and every TFS of r' is an instantiated TFS of w. The set of all instantiated rules of G and w is **the instantiated grammar of** G **and** w, denoted by $G_{|_w}$.

Definition 55 (*bi_list* content)**.** *Let B be an explicit bi_list of length k (see Definition 49 in Appendix A). The **content of** B is $w = a_1 \dots a_k$, and B is the **list representation of** w, if for every i, $1 \leq i \leq k$, the i-element of B is a_i. If the content of B is w, we write $B = [w]$.*

Example 35. *(The content of bi_list) The content of $\langle a, a, b, b\rangle$ is $aabb$, and is written $[aabb]$.*

Definition 56 (list instantiation)**.** *Let B be a TFS of type bi_list and $w \in \text{WORDS}^*$. IB is a **list instantiation** of B and w if:*

- *IB is an explicit bi_list of type bi_list,*
- *$B \sqsubseteq IB$,*
- *the content of IB is a substring of w.*

Example 36 (list instantiation)**.** *Let $w = abbb$ and $B = \langle a \rangle \cdot \boxed{1} bi_list$. $IB = \langle a, b, b \rangle$ is a list instantiation of B and w.*

Given a word $w \in$ WORDS*, and a main TFS A, the instantiated TFS of A and w is a maximally specific main TFS IA, such that $A \sqsubseteq IA$, and where the contents of the feature values are substrings of w.

Definition 57 (Instantiated TFS)**.** *Let A be a main TFS over a restricted signature S and $w \in$ WORDS*. A main TFS IA is **an instantiated TFS of** A **and** w if:*

- $A \sqsubseteq IA$
- *IA is maximally specific.*
- *For every feature f of A, the value of the feature f of IA is a list instantiation of B and w.*

Example 37 (Instantiated TFS)**.** *Let A be a TFS over the signature presented in Example 14, and $w = aabb$.*

$$A = \begin{bmatrix} \textit{counter} \\ \text{INPUT} : \boxed{1} \textit{bi_list} \\ \text{COUNTER} : \langle a \rangle \cdot \boxed{1} \end{bmatrix}$$

The following TFS IA is an instantiated TFS of A and w:

$$A = \begin{bmatrix} \textit{at} \\ \text{INPUT} : \langle a \rangle \\ \text{COUNTER} : \langle a, a \rangle \end{bmatrix}$$

Definition 58 (Instantiated rule)**.** *Let G be an RTUG over a restricted signature S, $w \in$ WORDS*, and $r \in \mathcal{R}$, $r = A_0 \rightarrow A_1 \ldots A_m$. The rule $r' = IA_0 \rightarrow IA_1 \ldots IA_m$ is **an instantiated rule of r and** w if:*

- *r' is a restricted rule, as defined in Definition 27,*
- *for every i, $0 \leq i \leq m$, IA_i is an instantiated TFS of A_i and w,*
- *If two bi_lists of r, B^1 and B^2 are marked with the same tag $\boxed{l}$, then the corresponding list instantiation of B^1 and B^2 in r', IB^1 and IB^2 respectively, have the same content.*
- *r' has no reentrancy tags.*

Instantiated rules include only instantiated TFSs. Instantiated TFS are maximally specific (they subsume no TFS but themselves). This renders unification with such TFSs no more than identity check. Since unification is trivial, there is no distinction between type and token identity; in other words, reentrancies can be ignored.

Example 38. *(Rule instantiation) Let*

$$r = \begin{bmatrix} at \\ \text{INPUT}_{counter} : \boxed{1} \cdot \boxed{2} \\ \text{COUNT} : \langle a \rangle \cdot \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} at \\ \text{INPUT}_{counter} : \boxed{1} \textit{bi_list} \\ \text{COUNT} : \langle a \rangle \end{bmatrix} \begin{bmatrix} bt \\ \text{INPUT}_{counter} : \boxed{2} \textit{bi_list} \\ \text{COUNT} : \boxed{3} \textit{bi_list} \end{bmatrix},$$

and $w = aabb$. The following is an instantiated rule of r and w:

$$r' = \begin{bmatrix} at \\ \text{INPUT}_{counter} : \langle a, b, b \rangle \\ \text{COUNT} : \langle a, b, b \rangle \end{bmatrix} \rightarrow \begin{bmatrix} at \\ \text{INPUT}_{counter} : \langle a \rangle \\ \text{COUNT} : \langle a \rangle \end{bmatrix} \begin{bmatrix} bt \\ \text{INPUT}_{counter} : \langle b, b \rangle \\ \text{COUNT} : \langle b, b \rangle \end{bmatrix},$$

where $\langle a \rangle$ is the list instantiation of $\boxed{1}$, and $\langle b, b \rangle$ is the list instantiation of $\boxed{2}$ and $\boxed{3}$ (see the definition of list instantiation above).

Definition 59 (Instantiated grammar)**.** *Let $G = \langle \mathcal{R}, A_s, \mathcal{L} \rangle$ be an RTUG and $w \in \text{WORDS}^*$. $G_{|_w} = \left\langle \mathcal{R}_{|_w}, A_{s|_w}, \mathcal{L}_{|_w} \right\rangle$ is* ***The instantiated grammar of*** G ***and*** w *if:*

- *$r' \in \mathcal{R}_{|_w}$ if and only if r' is an instantiated rule of some rule $r \in \mathcal{R}$ and w;*
- $A_{s|_w} = \begin{bmatrix} S \\ \text{INPUT}_S : [w] \end{bmatrix}$;
- *$IA \in \mathcal{L}_{|_w}(u)$ if and only if $u \in \text{WORDS}$ is a substring of w, and IA is an instantiated TFS of A and w, for some $A \in \mathcal{L}(u)$.*

Observe that for a given word $w \in \text{WORDS}^*$ and a given rule $r \in \mathcal{R}$, there can be many instantiated rules of r and w. All of them are in $\mathcal{R}_{|_w}$.

By definition of instantiated rules (Definition 58), they have no reentrancies, hence $G_{|_w}$ has no reentrancies. Feinstein and Wintner (2008) show that UG without reentrancies is equivalent to a context-free grammar. Therefore, $G_{|_w}$ is a context-free grammar (CFG).

Note that the fact that $G_{|_w}$ is a CFG does not suggest that G itself is also a CFG. $G_{|_w}$ is created by instantiation of G to exactly one word, w. As we show in Lemma 2, the language of $G_{|_w}$ includes only w, if w is included in $L(G)$, and is empty otherwise.

Lemma 1. *Let G be an RTUG, $w \in \text{WORDS}^*$ and A a main TFS. Let u be a substring of w and $k \geq 0$. $A \overset{k}{\Rightarrow}_G u$ if and only if there is an instantiated TFS of A and w, IA, such that $IA \overset{*}{\Rightarrow}_{G_{|_w}} u$, and the value of the input feature of IA is a list representation of u.*

Proof. **Direction 1:** By induction on the length of the derivation path k:

Base: If $k = 1$, $A \overset{1}{\Rightarrow}_G u$, then $u \in \text{WORDS}$ and $A \in \mathcal{L}(u)$. By the definition of restricted lexicons, the value of the input feature of A is u, hence the value of the input feature of every instantiated TFS of A is u.

Step: Assume that the lemma holds for every $d \leq k$. We now prove the lemma for $k+1$: If $A \overset{k+1}{\Rightarrow}_G u$, then by definition of TUG derivation, there is a rule $r \in \mathcal{R}$:

$$A \rightarrow A_1 \ldots A_m, m \geq 1,$$

such that

$$A_1 \ldots A_m \overset{k}{\Rightarrow}_G u.$$

By definition of TUG derivation, for every i, $1 \leq i \leq m$, there exist u_i and d_i, such that:

$$A_i \overset{d_i}{\Rightarrow}_G u_i,$$

where $d_i \leq k$ and $u = u_1 \cdot \ldots \cdot u_m$. By the induction hypothesis, for every i, $1 \leq i \leq m$, there is an instantiated TFS of A_i and w, IA_i, such that $IA_i \overset{*}{\Rightarrow}_{G_{|w}} u_i$, and the value of the input feature of IA_i is a list representation of u_i. Then, by definition of rule instantiation, there is an instantiated rule of r and w:

$$IA \rightarrow IA_1 \ldots IA_m,$$

such that IA is an instantiated TFS of A and w. Hence, by the definition of TUG derivation, $IA \overset{*}{\Rightarrow}_{G_{|w}} u_1 \cdot \ldots \cdot u_m = u$. By definition of restricted rules, the value of the input feature of the mother is a concatenation of the input feature of the daughters, hence the value of the input feature of IA is the list representation of $u_1 \cdot \ldots \cdot u_m = u$.

Direction 2: By induction on the length of the derivation path k:

Base: If $k = 1$, $IA \overset{1}{\Rightarrow}_{G_{|w}} u$, then $u \in \textsc{Words}$ and $IA \in \mathcal{L}_{G_{|w}}(u)$. By the definition of $\mathcal{L}_{G_{|w}}$, $IA \in \mathcal{L}_{|w}(u)$ if there is a main TFS A, such that, IA is an instantiated TFS of A and w, and $A \in \mathcal{L}(u)$. Hence, $A \overset{*}{\Rightarrow}_G u$.

Step: Assume that the lemma holds for every $d \leq k$. We now prove the lemma for $k+1$: If $IA \overset{k+1}{\Rightarrow}_{G_{|w}} u$, then by definition of TUG derivation, there is a rule $r' \in \mathcal{R}$:

$$IA \rightarrow IA_1 \ldots IA_m, m \geq 1,$$

such that

$$IA_1 \ldots IA_m \overset{k}{\Rightarrow}_G u.$$

By definition of TUG derivation, for every i, $1 \leq i \leq m$, there exist u_i and d_i, such that:

$$IA_i \overset{d_i}{\Rightarrow}_{G_{|w}} u_i,$$

where $d_i \leq k$, and $u = u_1 \cdot \ldots \cdot u_m$. By the induction hypothesis, for every i, $1 \leq i \leq m$, there is a main TFS of A_i, such that IA_i is an instantiated TFS of A_i and w, and $A_i \overset{*}{\Rightarrow}_G u_i$.

By definition of rule instantiation, there is a rule r and w:

$$A \rightarrow A'_1 \ldots A'_m,$$

such that IA is an instantiated TFS of A and w, and for every i, $1 \leq i \leq m$, IA_i is an instantiated TFS of A'_i and w. Note that for every i, $1 \leq i \leq m$, A_i and A'_i can be different TFSs, but they both subsume IA_i, hence, they are unifiable. By definition of TUG derivation, if $A_i \stackrel{*}{\Rightarrow}_G u_i$, then $A'_i \stackrel{*}{\Rightarrow}_G u_i$. Hence, by the definition of TUG derivation, $A \stackrel{*}{\Rightarrow}_G u_1 \cdot \ldots \cdot u_m = u$.

□

Lemma 2. *Let G be an RTUG and $w \in$ WORDS*. $w \in L(G)$ if and only if $w \in L\left(G_{|_w}\right)$*

Proof. **Direction 1:** Assume $w \in L(G)$, hence $A_s \stackrel{*}{\Rightarrow}_G w$. From Lemma 1, there is an instantiated TFS of A_s and w, IA_s, such that $IA_s \stackrel{*}{\Rightarrow}_{G_{|_w}} w$, and the content of the input feature of IA_s is w. By definition of instantiated grammars (Definition 59), the start symbol of $G_{|_w}$ is an instantiated TFS of A_s whose input feature content is w. Hence, IA_s is the start symbol of $G_{|_w}$, and by definition of context-free language, if $IA_s \stackrel{*}{\Rightarrow}_{G_{|_w}} w$, $w \in L\left(G_{|_w}\right)$.

Direction 2: Assume $w \in L\left(G_{|_w}\right)$, hence there is a derivation path from the start symbol of $G_{|_w}$, IA_s to w. By definition of instantiated grammars IA_s is an instantiated TFS of A_s and w. From Lemma 1, $IA_s \stackrel{*}{\Rightarrow}_{G_{|_w}} w$ only if there is a TFS A, such that IA_s is an instantiated TFS of A and w, and $A \stackrel{*}{\Rightarrow}_G w$. By definition of TUG derivation, since A and A_s have a common upper bound IA_s, $A_s \stackrel{*}{\Rightarrow}_G w$. Hence, by definition of the language of TUG, $w \in L(G)$.

□

B.2 Direction 1: $\mathcal{L}_{RCG} \subseteq \mathcal{L}_{RTUG}$

This section proves that for every RCG G_{rcg}, there exists an RTUG G_{tug}, such that $L(G_{rcg}) = L(G_{tug})$.

Obviously, we choose $G_{tug} = \mathit{RCG2TUG}(G_{rcg})$, and show that their languages coincide. For the following discussion fix an RCG $G = \langle N, T, V, P, S \rangle$, and let $\mathit{RCG2TUG}(G) = G_{tug} = \langle \mathcal{R}, A_s, \mathcal{L} \rangle$, defined over the signature $\langle$TYPES, $\sqsubseteq$, FEATS, *Approp*, WORDS$\rangle$.

B.2.1 $L(G) \subseteq L(G_{tug})$

We prove that $L(G) \subseteq L(G)_{tug}$ by showing that if $w \in L(G)$, then $w \in L\left(G_{tug_{|_w}}\right)$. We already proved in Lemma 2 that $w \in L\left(G_{tug_{|_w}}\right)$ implies $w \in L(G_{tug})$.

The following two lemmas claim that string instantiation and *bi_list* instantiation (Definition 56))

commute with *arg2feat* (Definition 35). See the commutative diagram below:

$$\begin{array}{ccccc} & \alpha & \xrightarrow{\textit{arg2feat}} & A & = \quad \textit{arg2feat}\,(\alpha) \\ \text{string inst.} & \downarrow & & \downarrow & \text{list inst.} \\ & u & \xrightarrow{\textit{arg2feat}} & B & = \quad \textit{arg2feat}\,(u) \end{array}$$

Example 39. *Let $w = abcd$, $\alpha = aXc, X \in V$ and $u = abc$. Observe that u is a string instantiation of α, where the terminal b is assigned to the variable X.*

$$A = \textit{arg2feat}\,(\alpha) = \begin{bmatrix} \textit{ne_bi_list} & \\ \text{HEAD}: & \begin{bmatrix} \textit{ne_node} \\ \text{CURR}: a \\ \text{PREV}: \textit{elist} \\ \text{NEXT}: \boxed{1}\,\textit{bi_list} \end{bmatrix} \\ \text{TAIL}: & \begin{bmatrix} \textit{ne_node} \\ \text{CURR}: c \\ \text{PREV}: \boxed{1} \\ \text{NEXT}: \textit{elist} \end{bmatrix} \end{bmatrix}$$

$$B = \textit{arg2feat}\,(u) = \begin{bmatrix} \textit{ne_bi_list} & \\ \text{HEAD}: \boxed{2} & \begin{bmatrix} \textit{ne_node} & \\ \text{CURR}: a & \\ \text{PREV}: \textit{elist} & \\ \text{NEXT}: \boxed{3} & \begin{bmatrix} \textit{ne_node} \\ \text{CURR}: b \\ \text{PREV}: \boxed{2} \\ \text{NEXT}: \boxed{4} \end{bmatrix} \end{bmatrix} \\ \text{TAIL}: \boxed{4} & \begin{bmatrix} \textit{ne_node} \\ \text{CURR}: c \\ \text{PREV}: \boxed{3} \\ \text{NEXT}: \textit{elist} \end{bmatrix} \end{bmatrix}$$

Observe that B is a list instantiation of A and w, since we can write $A = \langle a \rangle \cdot D \cdot \langle c \rangle$, D is a bi_list TFS, and $B = \langle a \rangle \cdot \langle b \rangle \cdot \langle c \rangle$.

Lemma 3. *Let $w \in T^*$, u be a substring of w, and $\alpha \in \{T \cup V\}^*$. Let $A = \mathit{arg2feat}\,(\alpha)$, and $B = \mathit{arg2feat}\,(u)$. If u is a string instantiation of α, then B is a list instantiation of A and w.*

Proof. By induction on the length of α:

Base:

- If $\alpha = \epsilon$, then, by definition of string instantiation, $u = \epsilon$. By definition of *arg2feat* , $A =$ *elist*, and also $B =$ *elist*. Hence, B is a list instantiation of A and w.
- If $\alpha = a \in T$, then, by definition of string instantiation, $u = a$. By definition of *arg2feat*, $A = \langle a \rangle = B$. Hence, B is a list instantiation of A and w.
- If $\alpha = X \in V$ then, by definition of string instantiation, any substring of w is a string instantiation of w, and in particular u. By definition of *arg2feat*, $A =$ *bi_list*, and $B = [u]$. By definition of list instantiation, any explicit *bi_list* whose content is a substring of w is a list instantiation of A, in particular B.

Step: Assume that the lemma holds for every α whose length $\leq k$. We now prove it for α of length $k + 1$:

- If $\alpha = a \cdot \beta$, such that $a \in T$ and $\beta \in \{T \cup V\}^*$ is of length k, then, by definition of *arg2feat*, $A = \langle a \rangle \cdot D$, such that $D = \mathit{arg2feat}\,(\beta)$. By definition of predicate instantiation, $u = a \cdot \delta$, such that δ is a string instantiation of β. By definition of *arg2feat*, $B = \langle a \rangle \cdot E$, such that E is an explicit *bi_list* whose content is δ. From the induction hypothesis, E is a list instantiation of D, and so B is a list instantiation of A and w.
- If $\alpha = X \cdot \beta$, such that $X \in V$ and $\beta \in \{T \cup V\}^*$ is oflength k, then, by definition of *arg2feat*, $A = H \cdot D$, such that H is an implicit bi_list and $D = \mathit{arg2feat}\,(\beta)$. By definition of string instantiation, $u = a \cdot \delta$, such that $a \in T$ and δ is a string instantiation of β. By definition of *arg2feat*, $B = \langle a \rangle \cdot E$, such that E is an explicit *bi_list* whose content is δ. From the induction hypothesis, E is a list instantiation of D. By definition of list instantiation, $\langle a \rangle$ is a list instantiation of E, and so, B is a list instantiation of A and w.

□

Lemma 4. *Let $w \in T^*$, u be a substring of w, and $\alpha \in \{T \cup V\}^*$. Let $A = \mathit{arg2feat}\,(\alpha)$, and $B = \mathit{arg2feat}\,(u)$. If B is a list instantiation of A and w, then u is a string instantiation of α.*

Proof. By induction on the length of α:

Base:

- If $\alpha = \epsilon$, then, by definition of *arg2feat*, $A =$ *elist*. By definition of list instantiation, $B = A =$ *elist*. By definition of *arg2feat*, $u = \epsilon$. Hence, u is a string instantiation of α.
- If $\alpha = a \in T$, then, by definition of *arg2feat*, $A = \langle a \rangle$. By definition of list instantiation, $B = A = \langle a \rangle$. By definition of *arg2feat*, $u = a$. Hence, u is a string instantiation of α.

- If $\alpha = X \in V$ then, then, by definition of *arg2feat*, $A =$ *bi_list*. By definition of list instantiation, B can be any explicit *bi_list* whose content is a substring of w. By definition of *arg2feat*, u is the content of B, hence u is a substring of w, and by definition of string instantiation, any substring of w, and in particular u, is a string instantiation of α.

Step: Assume that the lemma holds for every α whose length $\leq k$. We now prove it for α of length $k+1$:

- If $\alpha = a \cdot \beta$, such that $a \in T$ and $\beta \in \{T \cup V\}^*$ is of length k, then, by definition of *arg2feat*, $A = \langle a \rangle \cdot D$, such that $D = \textit{arg2feat}(\beta)$. By definition of list instantiation, $B = \langle a \rangle \cdot E$, such that E is a list instantiation of D. By definition of *arg2feat*, $u = a \cdot \delta$, such that δ is the content of E, and a substring of w. From the induction hypothesis, δ is a string instantiation of β, and so u is a string instantiation of α.
- If $\alpha = X \cdot \beta$, such that $X \in V$ and $\beta \in \{T \cup V\}^*$ is of length k, then, by definition of *arg2feat*, $A = H \cdot D$, such that H is an implicit bi_list and $D = \textit{arg2feat}(\beta)$. By definition of list instantiation, $B = \langle a \rangle \cdot E$, such that E is a list instantiation of D. By definition of *arg2feat*, $u = a \cdot \delta$, such that $a \in T$ and δ is the content of E, and a substring of w. From the induction hypothesis, δ is a string instantiation of β. By definition of string instantiation, any substring of w, and in particular a, is a string instantiation of X and w. Hence u is a string instantiation of α.

□

The following two lemmas claim that instantiation (of predicates, Definition 15, and of TFSs, Definition 57) commutes with *pred2tfs* (Definition 35).See the commutative diagram below:

$$\begin{array}{cccccc} & \varphi & \stackrel{\textit{pred2tfs}}{\longrightarrow} & A & = & \textit{pred2tfs}(\varphi) \\ \text{pred. inst.} & \downarrow & & \downarrow & \text{TFS inst.} & \\ & \psi & \stackrel{\textit{pred2tfs}}{\longrightarrow} & IA & = & \textit{pred2tfs}(\psi) \end{array}$$

The following lemma claims that given $w \in T^*$ and a predicate φ, if ψ is a predicate instantiation of φ and w, then *pred2tfs* (ψ) is a TFS instantiation of *pred2tfs* (φ) and w.

Example 40. *Let $w = abac$, and $\varphi = N(XY, aZ)$ be a predicate such that $X, Y, Z \in V$. An instantiated predicate of φ and w is $\psi = N(ab, ac)$. The TFS A is pred2tfs (φ):*

$$A = \begin{bmatrix} N \\ \text{INPUT}_N : \textit{elist} \\ \text{ARG}_1 : \boxed{1}\textit{bi_list} \cdot \boxed{2}\textit{bi_list} \\ \text{ARG}_2 : \langle a \rangle \cdot \boxed{3}\textit{bi_list} \end{bmatrix}.$$

The TFS IA is pred2tfs (ψ):

$$IA = \begin{bmatrix} N \\ \textsc{input}_N : \textit{elist} \\ \textsc{arg}_1 : \langle a, b \rangle \\ \textsc{arg}_2 : \langle a, c \rangle \end{bmatrix}.$$

Clearly, IA is an instantiated TFS of A and w.

Lemma 5. *Let $w \in T^*$. For every predicate $\varphi = N(\alpha_1, \ldots, \alpha_h)$, such that for every i, $1 \leq i \leq h$, $\alpha_i \in \{T \cup V\}^*$, and for every instantiated predicate $\psi = N(\rho_1, \ldots, \rho_h) \in IP_{G,w}$, $IA =$ pred2tfs (ψ) is an instantiated TFS of $A =$ pred2tfs (φ) and w.*

Proof. By Definition 35, *pred2tfs* (φ) is of the form:

$$A = \begin{bmatrix} N \\ \textsc{input}_N : \textit{elist} \\ \textsc{arg}_1 : B_1 \\ \vdots \\ \textsc{arg}_h : B_h \end{bmatrix}$$

where for every i, $1 \leq i \leq h$, $B_i =$ *arg2feat* (α_i). *pred2tfs* (ψ) is of the form:

$$IA = \begin{bmatrix} N \\ \textsc{input}_N : \textit{elist} \\ \textsc{arg}_1 : C_1 \\ \vdots \\ \textsc{arg}_h : C_h \end{bmatrix}$$

where for every i, $1 \leq i \leq h$, $C_i =$ *arg2feat* (ρ_i). By Definition 36, each C_i is an explicit bi_list whose content is ρ_i. By definition of string instantiation (Definition 15) ρ_i is a substring of w. From Lemma 3 for every i, $1 \leq i \leq h$, C_i is a list instantiation of B_i. Hence, IA is an instantiated TFS of A and w. $\square$

In a similar way to Lemma 5, The following lemma claims that instantiation and of TFSs commutes with *pred2tfs*, but in the other direction: Given $w \in T^*$ and a main TFS A, if φ is a predicate such that $A =$ *pred2tfs* (φ), and IA is a TFS instantiation of A and w, and if ψ is a predicate such that $IA =$ *pred2tfs* (ψ), then ψ is a predicate instantiation of φ and w.

Example 41. *Let $w = abac$, and A be a main TFS:*

$$A = \begin{bmatrix} t \\ \text{INPUT}_t : \textit{elist} \\ \text{ARG}_1 : \boxed{1}\textit{bi_list} \cdot \boxed{2}\textit{bi_list} \\ \text{ARG}_2 : \langle a \rangle \cdot \boxed{3}\textit{bi_list} \end{bmatrix}.$$

IA is an instantiated TFS of A and w:

$$IA = \begin{bmatrix} t \\ \text{INPUT}_t : \textit{elist} \\ \text{ARG}_1 : \langle a, b \rangle \\ \text{ARG}_2 : \langle a, c \rangle \end{bmatrix}.$$

$\varphi = t(XY, aZ)$ is a predicate such that $A = \textit{pred2tfs}(\varphi)$. $\psi = t(ab, ac)$ is a predicate such that $IA = \textit{pred2tfs}(\psi)$. Clearly ψ is an instantiated predicate of φ and w.

Lemma 6. *Let $w \in T^*$. For every main TFS A, and for every instantiated TFS of A and w, IA, if φ is a predicate such that $A = \textit{pred2tfs}(\varphi)$, and ψ is a predicate such that $IA = \textit{pred2tfs}(\psi)$, then $\psi \in IP_{G,w}$ is an instantiated predicate of φ.*

Proof. If A is of the form:

$$A = \begin{bmatrix} t \\ \text{INPUT}_t : \textit{elist} \\ \text{ARG}_1 : B_1 \\ \vdots \\ \text{ARG}_h : B_h \end{bmatrix},$$

then, from Definition 57, IA is of the form:

$$IA = \begin{bmatrix} t \\ \text{INPUT}_t : \textit{elist} \\ \text{ARG}_1 : IB_1 \\ \vdots \\ \text{ARG}_h : IB_h \end{bmatrix},$$

where for every i, $1 \leq i \leq h$, IB_i is a list instantiation of B_i and its content is u_i, a substring of w. By the definition of *pred2tfs*, Definition 35:

$$\varphi = t(\alpha_1, \ldots \alpha_h),$$

where for every i, $1 \le i \le h$, $B_i = \textit{arg2feat}\,(\alpha_i)$, and

$$\psi = t\,(\rho_1, \ldots \rho_h)\,,$$

where for every i, $1 \le i \le h$, ρ_i is the content of IB_i and, by definition of list instantiation, it is a substring of w. From Lemma 4, for every i, $1 \le i \le h$, ρ_i is a string instantiation of α_i. Hence, ψ is an instantiated predicate of φ. □

Lemma 7. *Let $w \in T^*$. For every instantiated predicate $\psi \in IP_{G,w}$, if $\psi \overset{k}{\Rightarrow}_{G,w} \epsilon$ (see k-derivation, Definition 19), then pred2tfs $(\psi) \overset{*}{\Rightarrow}_{G_{tug_{|w}}} \epsilon$.*

Proof. By induction on the length of the derivation path:

Base: If $k = 1$, then $\psi \overset{1}{\Rightarrow}_{G,w} \epsilon$. By the definition of RCG derivation, there is an instantiated clause in $IC_{G,w}$;

$$\psi \to \epsilon.$$

By Definition 18, there is a non-instantiated clause in P:

$$\varphi \to \epsilon,$$

where ψ is an instantiated predicate of φ. By Definition 35 , there is a rule in $\mathcal{R}$:

$$\textit{pred2tfs}\,(\varphi) \to \epsilon.$$

From Lemma 5, *pred2tfs* (ψ) is an instantiated TFS of *pred2tfs* (φ) and w. By definition of rule instantiation, there is an instantiated rule in $G_{tug_{|w}}$ of the form:

$$\textit{pred2tfs}\,(\psi) \to \epsilon.$$

Hence,

$$\textit{pred2tfs}\,(\psi) \overset{1}{\Rightarrow}_{G_{tug_{|w}}} \epsilon.$$

Step: Assume that Lemma 7 holds for every $d \le k$. We now prove the lemma for $k + 1$. Assume that

$$\psi \overset{k+1}{\Rightarrow}_{G,w} \epsilon,$$

By Definition 19, there is an instantiated clause in $IC_{G,w}$:

$$\psi \to \psi_1 \ldots \psi_m,$$

where for every i, $1 \le i \le m$, $\psi_i \in IP_{G,w}$, and there is a $d_i \le k$, such that,

$$\psi_i \overset{d_i}{\Rightarrow}_{G,w} \epsilon.$$

From Lemma 5, there is an instantiated rule in $G_{tug_{|w}}$:

$$(*)\, \textit{pred2tfs}\,(\psi) \rightarrow \textit{pred2tfs}\,(\psi_1) \ldots \textit{pred2tfs}\,(\psi_m)\,.$$

From the induction hypothesis, for every i, $1 \leq i \leq m$,

$$\textit{pred2tfs}\,(\psi_i) \overset{*}{\Rightarrow}_{G_{tug_{|w}}} \epsilon.$$

Hence, we can derive ϵ from every daughter of $(*)$, and obtain

$$\textit{pred2tfs}\,(\psi) \overset{*}{\Rightarrow}_{G_{tug_{|w}}} \epsilon.$$

□

Lemma 8. *For every $w \in T^*$, there is a derivation path in $G_{tug_{|w}}$:*

$$\begin{bmatrix} S' \\ \textsc{input}_{S'} : [w] \end{bmatrix} \overset{*}{\Rightarrow} w.$$

Proof. By induction on the length of w:

Base:

- If $w = \epsilon$, by Definition 34, there is a rule in $\mathcal{R}$:

$$\begin{bmatrix} S' \\ \textsc{input}_{S'} : \textit{elist} \end{bmatrix} \rightarrow \epsilon$$

Hence,

$$\begin{bmatrix} S' \\ \textsc{input}_{S'} : \textit{elist} \end{bmatrix} \overset{*}{\Rightarrow} \epsilon$$

- If $w = a \in T$, by Definition 34, there is an entry in $\mathcal{L}$:

$$a \rightarrow \begin{bmatrix} S' \\ \textsc{input}_{S'} : \langle a \rangle \end{bmatrix}$$

Hence,

$$\begin{bmatrix} S' \\ \textsc{input}_{S'} : \langle a \rangle \end{bmatrix} \overset{1}{\Rightarrow} a$$

Step: Assume that Lemma 8 holds for every w, such that $|w| \leq k$. We now prove the lemma for $w' = a \cdot w$, such that $a \in T$ and $|w| \leq k$:

- From the induction hypothesis:

$$(1)\ \begin{bmatrix} S' \\ \text{INPUT}_{S'} : [w] \end{bmatrix} \overset{*}{\Rightarrow} w,$$

$$(2)\ \begin{bmatrix} S' \\ \text{INPUT}_{S'} : \langle a \rangle \end{bmatrix} \overset{*}{\Rightarrow} a.$$

- By Definition 34, $\mathcal{R}$ always includes the following rule:

$$\begin{bmatrix} S\text{'} \\ \text{INPUT}_{S'} : \boxed{1} \cdot \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} S\text{'} \\ \text{INPUT}_{S'} : \boxed{1}\, \langle \textit{terminal} \rangle \end{bmatrix} \begin{bmatrix} S\text{'} \\ \text{INPUT}_{S'} : \boxed{2}\, \textit{bi_list} \end{bmatrix}$$

Hence, there is an instantiated rule in $G_{tug|_{w'}}$:

$$\begin{bmatrix} S\text{'} \\ \text{INPUT}_{S'} : \boxed{1} \cdot \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} S\text{'} \\ \text{INPUT}_{S'} : \boxed{1}\, \langle a \rangle \end{bmatrix} \begin{bmatrix} S\text{'} \\ \text{INPUT}_{S'} : \boxed{2}\, [w] \end{bmatrix}.$$

Applying this instantiated rule to (1) and (2), we obtain:

$$\begin{bmatrix} S\text{'} \\ \text{INPUT}_{S'} : \boxed{1} \cdot \boxed{2} \end{bmatrix} \Rightarrow \begin{bmatrix} S\text{'} \\ \text{INPUT}_{S'} : \boxed{1}\, \langle a \rangle \end{bmatrix} \begin{bmatrix} S\text{'} \\ \text{INPUT}_{S'} : \boxed{2}\, [w] \end{bmatrix} \overset{*}{\Rightarrow} a \cdot w.$$

□

Theorem 9. *For every RCG G, if $G_{tug} = \mathit{RCG2TUG}\,(G)$, then $L\,(G) \subseteq L\,(G_{tug})$*

Proof. Let $w \in L\,(G)$. Hence, by Definition 20, there is a $k \geq 1$, such that:

$$S\,(w) \overset{k}{\Rightarrow}_{G,w} \epsilon$$

- By Definition 35,

$$\textit{pred2tfs}\,(S\,(w)) = \begin{bmatrix} S'' \\ \text{INPUT}_{S''} : \textit{elist} \\ \text{ARG}_{S''} : [w] \end{bmatrix}.$$

From Lemma 7, there is a derivation path:

$$(1)\ \begin{bmatrix} S'' \\ \text{INPUT}_{S''} : \textit{elist} \\ \text{ARG}_{S''} : [w] \end{bmatrix} \overset{*}{\Rightarrow} \epsilon$$

- From Lemma 8:

$$(2)\begin{bmatrix} S' \\ \\ \text{INPUT}_{S'}:[w] \end{bmatrix} \stackrel{*}{\Rightarrow} w.$$

- By Definition 34, the start rule of G_{tug} is

$$\begin{bmatrix} S \\ \\ \text{INPUT}_{S}:\boxed{1}\textit{bi_list} \end{bmatrix} \rightarrow \begin{bmatrix} S' \\ \\ \text{INPUT}_{S''}:\boxed{1} \end{bmatrix}\begin{bmatrix} S'' \\ \text{INPUT}_{S''}:\textit{elist} \\ \text{ARG}_{S''}:\boxed{1} \end{bmatrix}$$

Then, for every $w \in L\left(G\right)$, there is an instantiated rule in $G_{tug_{|w}}$:

$$\boxed{2}\begin{bmatrix} S \\ \\ \text{INPUT}_{S}:\boxed{1}\,[w] \end{bmatrix} \rightarrow \begin{bmatrix} S' \\ \\ \text{INPUT}_{S''}:\boxed{1} \end{bmatrix}\begin{bmatrix} S'' \\ \text{INPUT}_{S''}:\textit{elist} \\ \text{ARG}_{S''}:\boxed{1} \end{bmatrix}.$$

By applying this rule to (1) and (2), we obtain the following derivation sequence:

$$\boxed{2}\begin{bmatrix} S \\ \\ \text{INPUT}_{S}:\boxed{1}\,[w] \end{bmatrix} \Rightarrow \begin{bmatrix} S' \\ \\ \text{INPUT}_{S''}:\boxed{1} \end{bmatrix}\begin{bmatrix} S'' \\ \text{INPUT}_{S''}:\textit{elist} \\ \text{ARG}_{S''}:\boxed{1} \end{bmatrix} \stackrel{*}{\Rightarrow} w \begin{bmatrix} S'' \\ \text{INPUT}_{S''}:\textit{elist} \\ \text{ARG}_{S''}:\boxed{1} \end{bmatrix} \stackrel{*}{\Rightarrow} w.$$

Since $\boxed{2}$ is an instantiated TFS of the start symbol A_s and w, $A_s \stackrel{*}{\Rightarrow} w$, and so $w \in L\left(G_{tug}\right)$.

□

B.2.2 $L\left(G_{tug}\right) \subseteq L\left(G\right)$

We prove that $L\left(G_{tug}\right) \subseteq L\left(G\right)$ by showing that if $w \in L\left(G_{tug}\right)$, then $w \in L\left(G_{tug_{|w}}\right)$. By Lemma 2, this implies $w \in L\left(G\right)$. Recall that for every $w \in \text{WORDS}^*$, $G_{tug_{|w}}$ is context-free.

Lemma 10. *For every $w \in T^*$ and for every $k \geq 1$, if there is a derivation path in $G_{tug_{|w}}$:*

$$A_0 \stackrel{k}{\Rightarrow}_{G_{tug_{|w}}} A_1 \dots A_m,$$

then there is a derivation path in G:

$$\varphi_0 \stackrel{*}{\Rightarrow}_{G,w} \varphi_1 \cdots \varphi_m,$$

such that for every i, $0 \leq i \leq m$, $\varphi_i \in IP_{G,w}$, and $A_i = \textit{pred2tfs}\left(\varphi_i\right)$.

Proof. We prove the lemma by induction on the length of derivation path:

Base: If $k = 1$, $A_0 \overset{1}{\Rightarrow} A_1 \ldots A_m$, then there is a rule $r \in G_{tug_{|w}}$ of the form:

$$r = A_0 \rightarrow A_1 \ldots A_m$$

By Definition 57, there is a rule $r' \in G_{tug}$ of the form:

$$r' = B_0 \rightarrow B_1 \ldots B_m$$

where for every i, $0 \leq i \leq m$, A_i is an instantiated TFS of B_i and w.

By Definition 35, there is a clause $p \in P$, such that $r' = \textit{clause2rule}\,(p)$, of the form:

$$p = \psi_0 \rightarrow \psi_1 \ldots \psi_m,$$

such that for for every i, $0 \leq i \leq m$, $B_i = \textit{pred2tfs}\,(\psi_i)$.

From Lemma 6, there is an instantiated clause of p and w, $q \in IC_{G,w}$, such that $r = \textit{clause2rule}\,(q)$, of the form:

$$q = \varphi_0 \rightarrow \varphi_1 \ldots \varphi_m,$$

such that for for every i, $0 \leq i \leq m$, $A_i = \textit{pred2tfs}\,(\varphi_i)$. Hence,

$$\varphi_0 \overset{*}{\Rightarrow}_{G,w} \varphi_1 \ldots \varphi_m.$$

Strep: Assume that Lemma 10 holds for every $d \leq k$. Now we prove the lemma for $k+1$. Assume that there is a derivation path in $G_{tug_{|w}}$:

$$A_0 \overset{k+1}{\Rightarrow}_{G_{tug_{|w}}} A_1 \ldots A_m.$$

By definition of derivation in CFG, there is a rule in $G_{tug_{|w}}$:

$$A_0 \rightarrow B_1 \ldots B_l,$$

such that for every i, $1 \leq i \leq l$, B_i is an instantiated TFS, and

$$B_i \overset{d_i}{\Rightarrow}_{G_{tug_{|w}}} \Gamma_i,$$

where $d_i \leq k$ and Γ_i is a string of instantiated TFS, such that $\bigcup_{1 \leq i \leq l} \Gamma_i = A_1 \ldots A_m$.

Then there is an instantiated clause in $IC_{G,w}$:

$$\varphi_0 \rightarrow \psi_1 \ldots \psi_l,$$

such that $A_0 = \textit{pred2tfs}\,(\varphi_0)$, and for every i, $0 \leq i \leq l$, $B_i = \textit{pred2tfs}\,(\psi_i)$. From the induction

hypothesis, for every i, $0 \leq i \leq l$, there is a derivation path in G:

$$\psi_i \stackrel{*}{\Rightarrow}_{G,w} \Phi_i,$$

where Φ_i is a string of predicates, such that $\bigcup_{1 \leq i \leq l} \Phi_i = \varphi_1 \ldots \varphi_m$, where for every j, $1 \leq j \leq m$, $A_j = \textit{pred2tfs}\left(\varphi_j\right)$. Hence,

$$\varphi_0 \rightarrow \psi_1 \ldots \psi_l \stackrel{*}{\Rightarrow}_{G,w} \varphi_1 \cdots \varphi_l,$$

and by definition of RCG derivation (Definition 19),

$$\varphi_0 \stackrel{*}{\Rightarrow}_{G,w} \varphi_1 \cdots \varphi_l$$

□

Theorem 11. *For every RCG G, if* $G_{tug} = \textit{RCG2TUG}\left(G\right)$*, then* $L\left(G_{tug}\right) \subseteq L\left(G\right)$.

Proof. By Lemma 2, if $w \in L\left(G_{tug}\right)$, then $w \in G_{tug_{|w}}$. By the definition of derivation and the definition of $G_{tug_{|w}}$, there is a derivation path in $G_{tug_{|w}}$:

$$\begin{bmatrix} S'' \\ \text{INPUT}_{S''} : \textit{elist} \\ \text{ARG}_{S''} : [w] \end{bmatrix} \stackrel{*}{\Rightarrow}_{G_{tug_{|w}}} \epsilon.$$

Then, from Lemma 10, there is also a derivation path in G:

$$S\left(w\right) \stackrel{*}{\Rightarrow}_{G,w} \epsilon.$$

By Definition 20, $w \in L\left(G\right)$. □

B.3 Direction 2: $\mathcal{L}_{RTUG} \subseteq \mathcal{L}_{RCG}$

This section proves that for every RTUG G_{tug}, there exists an RCG G_{rcg}, such that $L\left(G_{rtug}\right) = L\left(G_{rcg}\right)$.

Unlike the previous section, in a general RTUG signature, subsumption may hold between main types. Hence if A is a main TFS and IA is an instantiated TFS of A and some $w \in \text{WORDS}^*$, then the type of IA may be different from the type of A (recall that by definition of an instantiated TFS, it must be maximally specific). For this case we define below a hierarchy over non-terminals and predicates of RCG that is equivalent to the hierarchy over types and TFSs of RTUG.

In a similar way to Direction 1 of the proof (Section B.2), we will choose $G_{rcg} = \textit{TUG2RCG}\left(G_{tug}\right)$, as defined in Definition 30, and show that their languages coincide. For the following discussion

fix an RTUG $G = \langle \mathcal{R}, A_s, \mathcal{L} \rangle$, defined over the typed signature $\langle \text{TYPES}, \sqsubseteq, \text{FEATS}, \textit{Approp} \rangle$, and let $\textit{TUG2RCG}\,(G) = G_{rcg} = \langle N, T, V, P, S \rangle$.

B.3.1 Predicate subsumption

We define a hierarchy over non-terminals and predicates of RCG, in a similar way to the type hierarchy and TFS subsumption of TUG.[1] In general, given an RTUG G over a signature S, and an RCG $G_{rcg} = \textit{TUG2RCG}\,(G)$, we say that the non-terminal $N_t \in N$ subsumes the non-terminal $N_s \in N$, if the type t subsumes the type s in S. We say that a predicate φ subsumes the predicate ψ, if the non-terminal of φ subsumes the non-terminal of ψ, and every argument of φ is a string instantiation of the corresponding argument of φ. Note that the definition of predicate subsumption is much tighter then the definition of TFS subsumption.

For the following discussion, fix an RTUG G over a restricted typed signature $S = \langle \text{TYPES}, \text{FEATS}, \sqsubseteq, \textit{Approp} \rangle$, and an RCG $G_{rcg} = \langle N, T, V, P, S \rangle = \textit{TUG2RCG}\,(G)$, such that $G_{rcg} = \textit{TUG2RCG}\,(G)$.

Definition 60 (Non-terminal hierarchy)**.** *Let N_t and N_s be two non-terminals in N, such that $ar\,(N_t) = ar\,(N_s)$. N_t **subsumes** N_s if and only if the type $t \in$ TYPES subsumes the type $s \in$ TYPES. If the non-terminal N_s subsumes no other non-terminal, we say that N_s is a **maximum non-terminal**.*

Definition 61 (Predicate subsumption)**.** *Let $\varphi = t\,(\alpha_1, \ldots, \alpha_h)$ and $\psi = s\,(\rho_1, \ldots, \rho_h)$ be two predicates. φ **subsumes** ψ if and only if:*

- *t subsumes s, and*
- *for every i, $1 \leq i \leq h$, ρ_i is a string instantiation of α_i.*

*ψ is a **maximum predicate** if it subsumes no other predicate but itself.*

Definition 62 (Non-terminal subsumption)**.** *Let φ be a predicate of type t, such that t subsumes s. The **non-terminal subsumption of** φ **and** s is the predicate γ whose non-terminal is s, and whose arguments are the same as φ's.*

Lemma 12. *Let*

$$\varphi = t\,(\alpha_1, \ldots, \alpha_h)\,.$$

Let

$$\psi = s\,(\rho_1, \ldots, \rho_h)\,.$$

Let γ be the non-terminal subsumption of φ and s:

$$\gamma = s\,(\alpha_1, \ldots, \alpha_h)\,.$$

If φ subsumes ψ, then ψ is an instantiated predicate of γ.

[1] Recall that each non-terminal N_t in *TUG2RCG* (G) corresponds to a type t in G.

Proof. By definition of predicate subsumption, for every i, $1 \leq i \leq h$, ρ_i is a string instantiation of α_i. Hence, by definition of predicate instantiation, ψ is a predicate instantiation of γ. □

Example 42. *Consider $G_{longdist}$ and TUG2RCG $(G_{longdist})$ of Example 15:*

- *v_subcat (X) subsumes v_np $(loves)$, because v_subcat $\sqsubseteq$ v_np;*
- *v_np is a maximum type and v_np $(loves)$ is a maximum predicate.*

Lemma 13. *Let*

$$\varphi = t\left(\alpha_1, \ldots, \alpha_h\right),$$

let

$$\gamma = t\left(\rho_1, \ldots, \rho_h\right),$$

be an instantiated predicate of φ, and let

$$\psi = s\left(\rho_1, \ldots, \rho_h\right).$$

If φ subsumes ψ, then $\gamma \overset{*}{\Rightarrow}_{G_{rcg},w} \psi$.

Proof. By definition of predicate subsumption, t subsumes s. By definition of (the unification clauses of) *TUG2RCG*:

$$t\left(X_1, \ldots, X_h\right) \overset{*}{\Rightarrow} s\left(X_1, \ldots, X_h\right), X_1, \ldots, X_h \in V$$

Hence, $\gamma \overset{*}{\Rightarrow}_{G_{rcg},w} \psi$. □

B.3.2 $L(G) \subseteq L(G_{rcg})$

Like the previous section, we prove that $L(G) \subseteq L(G_{rcg})$ by showing that if $w \in L(G)$, then $w \in G_{|w}$, and then $w \in L(G_{rcg})$.

The following lemmas claim that string instantiation and *bi_list* instantiation (Definition 56) commute with *feat2arg* (Definition 32). See the commutative diagram below:

$$\begin{array}{ccc} B & \overset{\textit{feat2arg}}{\longrightarrow} & \alpha = \textit{feat2arg}\,(B) \\ \text{list inst.} \downarrow & & \downarrow \text{string inst.} \\ C & \overset{\textit{feat2arg}}{\longrightarrow} & \rho = \textit{feat2arg}\,(C) \end{array}$$

If B is a TFS of type *bi_list*, its **length** is:.

- 0 if $B = elist$,
- 1 if B is an implicit *bi_list*, or if B is an explicit *bi_list* with only one element, and
- k, $k \geq 1$, if B is a concatenation of k *bi_list* of length 1 (see the definition of concatenation of bidirectional lists, Definition 53).

Lemma 14. *Let $w \in \text{WORDS}^*$, B be a TFS of type bi_list and C be an explicit bi_list whose content is u, a substring of w. Let $\alpha = \textit{feat2arg}\,(B)$ and $\rho = \textit{feat2arg}\,(C)$. If ρ is a string instantiation of α, then C is a list instantiation of B and w.*

Proof. We prove the lemma by induction on the length of B.

Base:
- If $B = elist$, then by the definition of *feat2arg*, $\alpha = \epsilon$. By definition of string instantiation $\rho = \epsilon$, and by definition of *feat2arg*, $C = elist$. Hence C is a list instantiation of B and w.
- If $B = \langle a \rangle, a \in \text{WORDS}$, then by the definition of *feat2arg*, $\alpha = a$. By the definition of string instantiation, $\rho = a$, and by the definition of *feat2arg*, $C = \langle a \rangle$. Hence C is a list instantiation of B and w.
- If B is an implicit *bi_list*, then by the definition of *feat2arg*, $\alpha = X$, for some $X \in V$. By definition of string instantiation $\rho = u$, such that u is any substring of w. By the definition of *feat2arg*, C is an explicit *bi_list* whose content u. By definition of list instantiation, any explicit *bi_list* whose content is a substring of w is a string instantiation of B and w. Specifically, C is such.

Step: Assume that the lemma holds for every B of length $\leq k$. We now prove it for B of length $k + 1$:

- If $B = \langle a \rangle \cdot D$, $a \in T$ and D is a *bi_list* of length $\leq k$, then by the definition of *feat2arg*, $\alpha = a \cdot \beta$, such that $\beta = \textit{feat2arg}\,(D)$. By definition of string instantiation, $\rho = a \cdot \mu$, where μ is a string instantiation of β, and $a \cdot \mu$ is a substring of w. By definition of *feat2arg*, $C = \langle a \rangle \cdot H$, such that $H = [\mu]$. From the induction hypothesis, H is a list instantiation of D and w, hence, C is a list instantiation of B and w.
- If $B = D \cdot E$, such that D is an implicit *bi_list* and E is a *bi_list* of length $\leq k$, then, by the definition of *feat2arg*, $\alpha = X \cdot \beta$, where $X \in V$ and $\beta = \textit{feat2arg}\,(E)$. By definition of string instantiation, $\rho = u \cdot \mu$, where μ is a string instantiation of β, and $u \cdot \mu$ is a substring of w. By *feat2arg* definition, $C = Q \cdot H$, such that $Q = [u]$, and $H = [\mu]$. From the induction hypothesis, H is a list instantiation of E and w. By definition of list instantiation, any explicit *bi_list* (in particular, Q) whose content is a substring of w is a list instantiation of D. Hence, C is a list instantiation of B and w.

$\square$

Lemma 15. *Let $w \in \text{WORDS}^*$, B be a TFS of type bi_list and C be an explicit bi_list whose content is u, a substring of w. Let $\alpha = \textit{feat2arg}\,(B)$ and $\rho = \textit{feat2arg}\,(C)$. If C is a list instantiation of B and w, then ρ is a string instantiation of α.*

Proof. We prove the lemma by induction on the length of B.

Base:
- If $B = elist$, then by the definition of *feat2arg*, $\alpha = \epsilon$. By definition of list instantiation, $C = elist$, and by the definition of *feat2arg*, $\rho = \epsilon = \alpha$, hence ρ is a string instantiation of α.

- If $B = \langle a \rangle, a \in$ WORDS, then by the definition of *feat2arg*, $\alpha = a$. By definition of list instantiation, $C = \langle a \rangle$, and by the definition of *feat2arg*, $\rho = a = \alpha$, hence ρ is a string instantiation of α.
- If B is an implicit *bi_list*, then by the definition of *feat2arg*, $\alpha = X$, for some $X \in V$. By definition of list instantiation, C is an explicit *bi_list* whose content ρ, is a substring of w. By definition of string instantiation, any substring of w is a string instantiation of X, hence ρ is a string instantiation of α.

Step: Assume that the lemma holds for every B of length $\leq k$. We now prove it for B of length $k+1$:

- If $B = \langle a \rangle \cdot D$, $a \in T$ and D is a *bi_list* of length $\leq k$, then by the definition of *feat2arg*, $\alpha = a \cdot \beta$, such that $\beta = \textit{feat2arg}\,(D)$. By definition of list instantiation, $C = \langle a \rangle \cdot E$, such that E is a list instantiation of D and w, and by the definition of *feat2arg*, $\rho = a \cdot \delta$, such that δ is the content of E. From the induction hypothesis, δ is a string instantiation of β, hence ρ is a string instantiation of α.
- If $B = D \cdot E$, such that D is an implicit *bi_list* and E is a *bi_list* of length $\leq k$, then, by the definition of *feat2arg*, $\alpha = X \cdot \beta$, where $X \in V$ and $\beta = \textit{feat2arg}\,(E)$. By definition of list instantiation, $C = F \cdot H$, where H is a list instantiation of E and F is some explicit *bi_list*. By the definition of *feat2arg*, $\rho = \delta \cdot \lambda$, where δ is the content of F. From the induction hypothesis, λ is a string instantiation of β. By definition of string instantiation, any substring of w is a string instantiation of X, and in particular δ. Hence ρ is a string instantiation of α.

□

The following lemma deals with the commutativity of instantiation and the mapping between TFSs and predicates. Unlike the previous section, in a general RTUG a TFS A and its instantiated TFS IA can be of different types. In this case, we cannot claim that *tfs2pred* (IA) is an instantiated predicate of *tfs2pred* (A), since they may have different non-terminals. What we can claim, however, is that *tfs2pred* (A) **subsumes** *tfs2pred* (IA), as defined in Definition 61. See the commutative diagram below:

$$\begin{array}{rccl} & A & \stackrel{\textit{tfs2pred}}{\longrightarrow} & \varphi = \textit{tfs2pred}\,(A) \\ \text{TFS inst.} & \downarrow & & \downarrow \text{subsumes} \\ & IA & \stackrel{\textit{tfs2pred}}{\longrightarrow} & \psi = \textit{tfs2pred}\,(IA) \end{array}$$

Example 43. *Consider the following fragment of the signature of $G_{longdist}$, repeated from Example 15:*

Signature

main

v_subcat INPUT$_{vs}$:*bi_list*

v_np

v_s

...

Consider further the TFS

$$A = \begin{bmatrix} \textit{v_subcat} \\ \text{INPUT}_{vs} : \boxed{1}\textit{bi_list} \end{bmatrix}$$

Let $w = loves$*, so the instantiated TFS of* A *and* w *is:*

$$IA = \begin{bmatrix} \textit{v_np} \\ \text{INPUT}_{vs} : \langle loves \rangle \end{bmatrix}$$

$$\begin{array}{lllll} \textit{tfs2pred}\,(A) & = & \varphi & = & \textit{v_subcat}\,(X)\,, \quad X \in V \\ \textit{tfs2pred}\,(IA) & = & \psi & = & \textit{v_np}\,(loves) \end{array}$$

Clearly, ψ *is not an instantiated predicate of* φ*. However, given the unification clauses of the grammar TUG2RCG* $(G_{longdist})$*:*

$$\begin{array}{lll} \textit{v_subcat}\,(X) & \rightarrow & \textit{v_np}\,(X) \\ \textit{v_subcat}\,(X) & \rightarrow & \textit{v_s}\,(X) \end{array}$$

we can see that v_subcat subsumes v_np and φ *subsumes* ψ*.*

Lemma 16. *Let* $w \in \text{WORDS}^*$*. Let* A *be a main TFS* A*, and* IA *be an instantiated TFS of* A *and* w*. If* $\varphi = \textit{tfs2pred}\,(A)$*, and* $\psi = \textit{tfs2pred}\,(IA)$*, then* φ *subsumes* ψ*.*

Proof. If:

$$A = \begin{bmatrix} t \\ \text{ARG}_1 : B_1 \\ \vdots \\ \text{ARG}_h : B_h \end{bmatrix},$$

then by definition of instantiated TFS (Definition 57):

$$IA = \begin{bmatrix} s \\ \text{ARG}_1 : IB_1 \\ \vdots \\ \text{ARG}_h : IB_h \end{bmatrix},$$

where $t \sqsubseteq s$, and for every i, $1 \leq i \leq h$, IB_i is a list instantiation of B_i whose content is ρ_i, a substring of w. By the definition of *tfs2pred* (Definition 32):

$$\varphi = N_t\,(\alpha_1, \ldots \alpha_h)\,,$$

where for every i, $1 \le i \le h$, $\alpha_i =$ *feat2arg* (B_i), and

$$\psi = N_s (\rho_1, \dots \rho_h) .$$

By definition of non-terminal subsumption (Definition 60), N_t subsumes N_s. From Lemma 15, for every i, $1 \le i \le h$, ρ_i is a string instantiation of α_i. Hence, by definition of predicate subsumption (Definition 61), φ subsumes ψ. $\square$

The last step of this direction is to prove that if $w \in L\left(G_{|w}\right)$, then $w \in L\left(G_{rcg}\right)$, or in other words, that $A_{s|_w} \overset{*}{\Rightarrow}_{G_{|w}} w$ implies $S(w) \overset{*}{\Rightarrow}_{G_{rcg},w} \epsilon$. To prove it we need one more lemma that claims that if $IA \overset{k}{\Rightarrow} G_{|w}u$, where IA is **any instantiated TFS** and u is **any substring of** w, then *tfs2pred* $(AI) \overset{*}{\Rightarrow}_{G_{rcg},w} \epsilon$. Recall from Lemma 1 that u is the content of the input feature of IA. Hence, by the definition of *tfs2pred*, u is the first argument of *tfs2pred* (IA).

Lemma 17. *Let $w \in L(G)$, IA be an instantiated TFS, u be a substring of w, and $\varphi =$ tfs2pred (IA). Let $k \ge 1$. If $IA \overset{k}{\Rightarrow}_{G_{|w}} u$, then $\varphi \overset{*}{\Rightarrow}_{G_{rcg},w} \epsilon$.*

Proof. Let t be the type of IA. By definition of instantiated TFS, t is maximal (it subsumes no other type). By definition of *tfs2pred*, the non-terminal of φ is N_t.

By induction on the length of the derivation path k:

Base: If $k = 1$, then $u \in$ WORDS, and by definition of derivation,

$$IA \rightarrow u \in G_{|w}.$$

Hence, $IA \in \mathcal{L}_{|w}(u)$. By definition of $\mathcal{L}_{|w}$, There is a TFS A, such that IA is an instantiated TFS of A and w, and $A \in \mathcal{L}(u)$. Let s be the type of A, and $\psi =$ *tfs2pred* (A). By definition of *tfs2pred*, the non-terminal of φ is N_s, and N_s subsumes N_t. Let γ be the non-terminal subsumption of ψ and N_t, hence the non-terminal of γ is N_t and its arguments are the same as ψ's. By definition of *TUG2RCG*, there is a clause in P

$$\psi \rightarrow \epsilon,$$

If N_s is maximal, then $N_s = N_t$, hence $\gamma = \psi$. Otherwise, since $N_s \sqsubseteq N_t$, by definition of *TUG2RCG*, there is a clause in P:

$$\gamma \rightarrow \epsilon.$$

From Lemma 16, ψ subsumes φ. From Lemma 12, φ is a predicate instantiation of γ. Hence, by definition of $IC_{G_{rcg},w}$, there is an instantiated clause:

$$\varphi \rightarrow \epsilon.$$

hence, by definition of derivation, $\varphi \overset{*}{\Rightarrow}_{G_{rcg},w} \epsilon$.

Step: Assume that Lemma 17 holds for every $d \leq k$. We now prove the lemma for $k+1$: If $IA \overset{k+1}{\Rightarrow}_{G_{|w}} u$, then by definition of derivation, there is a rule in $G_{|w}$ of the form:

$$IA \to IA_1 \dots IA_m,$$

where for every i, $1 \leq i \leq m$, IA_i is an instantiated TFS, such that

$$IA_i \overset{d_i}{\Rightarrow}_{G_{|w}} u_i, d_i \leq k$$

and $u = u_1 \dots u_m$. By the induction hypothesis, for every i, $1 \leq i \leq m$, if $\varphi_i = \textit{tfs2pred}\,(IA_i)$, then there is a derivation path

$$\varphi_i \overset{*}{\Rightarrow}_{G_{rcg},w} \epsilon.$$

By definition of instantiated grammar, there is a rule in $\mathcal{R}$:

$$A \to A_1 \dots A_m,$$

where IA is an instantiated TFS of A and w, and for every i, $1 \leq i \leq m$, IA_i is an instantiated TFS of A_i and w. Let s be the type of A, and $\psi = \textit{tfs2pred}\,(A)$. By definition of *tfs2pred*, the non-terminal of φ is N_s, and N_s subsumes N_t. Let γ be the non-terminal subsumption of ψ and N_t. By definition of *TUG2RCG*, there is a clause in P

$$\psi \to \psi_1 \dots \psi_m,$$

such that for every i, $1 \leq i \leq m$, $\psi_i = \textit{tfs2pred}\,(A_i)$. If N_s is maximal, then $N_s = N_t$, hence $\gamma = \psi$. Otherwise, since $N_s \sqsubseteq N_t$, by definition of *TUG2RCG*, there is a clause in P:

$$\gamma \to \psi_1 \dots \psi_m.$$

From Lemma 12, φ is an instantiated TFS of γ, hence, by definition of $IC_{G_{rcg},w}$, there is an instantiated clause:

$$\varphi \to \lambda_1 \dots \lambda_m,$$

such that for every i, $1 \leq i \leq m$, let λ_i is a predicate instantiation of ψ_i. From Lemma 16, for every i, $1 \leq i \leq m$, ψ_i subsumes φ_i, hence, from Lemma 13, $\lambda_i \overset{*}{\Rightarrow}_{G_{rcg},w} \varphi_i$. Hence,

$$\varphi \overset{*}{\Rightarrow}_{G_{rcg},w} \varphi_1 \dots \varphi_m \overset{*}{\Rightarrow}_{G_{rcg},w} \epsilon.$$

□

Theorem 18. *For every RTUG G, if $G_{rcg} = \textit{TUG2RCG}\,(G)$, then $L\,(G) \subseteq L\,(G_{rcg})$.*

Proof. If $w \in L\,(G)$, then from Lemma 2, $w \in L\left(G_{|w}\right)$. Hence, there is a derivation path in $G_{|w}$ from

the start symbol to w:

$$\begin{bmatrix} \textit{start} \\ \\ \text{INPUT}_{start} : [w] \end{bmatrix} \stackrel{*}{\Rightarrow}_{G_{|w}} w,$$

From Lemma 17, there is a derivation path in G_{rcg}:

$$S(w) \stackrel{*}{\Rightarrow}_{G_{rcg},w} \epsilon.$$

By definition of the language of G_{rcg}, Definition 20, $w \in L(G_{rcg})$. □

B.3.3 $L(G_{rcg}) \subseteq L(G)$

In this section we prove that $L(G_{rcg}) \subseteq L(G)$. Hence, if $w \in L(G_{rcg})$, then $w \in L(G)$.

In this section we prove it directly on G, without using $G_{|w}$, because here we deal with commutativity of **subsumption** of TFSs and predicates, rather then **instantiation** as in the previous sections. What we claim in Lemma 19 below is that if *tfs2pred*(A) subsumes *tfs2pred*(B), then $A \sqsubseteq B$. See the commutative diagram below:

$$\begin{array}{ccccc} & A & \stackrel{\textit{tfs2pred}}{\longrightarrow} & \varphi = \textit{tfs2pred}(A) & \\ \sqsubseteq & \downarrow & & \downarrow & \text{subsumes} \\ & B & \stackrel{\textit{tfs2pred}}{\longrightarrow} & \psi = \textit{tfs2pred}(B) & \end{array}$$

Lemma 19. *Let $w \in T^*$, A and B be main TFSs, φ = tfs2pred(A), and ψ = tfs2pred(B). If φ subsumes ψ, then $A \sqsubseteq B$.*

Proof. Let:

$$A = \begin{bmatrix} t \\ \\ \text{ARG}_1 : C_1 \\ \vdots \\ \text{ARG}_h : C_h \end{bmatrix},$$

where for every i, $1 \leq i \leq h$, C_i is a TFS of *bi_list* type. By definition of *tfs2pred*:

$$\varphi = N_t(\alpha_1, \ldots, \alpha_h),$$

where for every i, $1 \leq i \leq h$, $\alpha_i =$ *feat2arg*(C_i). By definition of predicate subsumption,

$$\psi = N_s(\rho_1, \ldots, \rho_h),$$

where N_t subsumes N_s, and for every i, $1 \leq i \leq h$, ρ_i is a string instantiation of α_i and a substring of

w. By definition of *tfs2pred*,

$$B = \begin{bmatrix} s \\ \\ \text{ARG}_1 : D_1 \\ \vdots \\ \\ \text{ARG}_h : D_h \end{bmatrix},$$

where for every i, $1 \leq i \leq h$, $\rho_i = \textit{arg2feat}\,(D_i)$. By definition of non-terminal subsumption, $t \sqsubseteq s$. From Lemma 14, for every i, $1 \leq i \leq h$, if ρ_i is a string instantiation of α_i, then D_i is a list instantiation of C_i. Hence, by definition of TFS subsumption, $A \sqsubseteq B$. □

The following lemma claims that if there is a main TFS A, such that *tfs2pred* (A) is an instantiated predicate whose first argument is u and there is a derivation path in G_{rcg} from *tfs2pred* (A) to ϵ, then there is a derivation path in G from A to u. This lemma is one step before showing that given a main TFS A, such that $A_s \sqsubseteq A$, and the value of the input feature of A is w, if there is a derivation path in G_{rcg} from *tfs2pred* (A) to ϵ (hence $w \in L\,(G_{rcg})$), there is a derivation path in G from A to w (and hence $w \in L\,(G)$).

Lemma 20. *Let $w \in T^*$ and u be a substring of w. Let φ be an instantiated predicate in $IP_{G_{rcg},w}$, such that the first argument of φ is u. Let A be a main TFS, such that $\varphi = \textit{tfs2pred}\,(A)$, and let $k \geq 1$. If $\varphi \overset{k}{\Rightarrow}_{G_{rcg},w} \epsilon$, then $A \overset{*}{\Rightarrow}_G u$.*

Proof. By definition of *feat2arg*, if the first argument of φ is u, then the value of the input feature of A is an explicit *bi_list* whose content is u. We prove the lemma by induction on the length of the derivation path, k:

Base: If $k = 1$, $\varphi \overset{1}{\Rightarrow}_{G_{rcg},w} \epsilon$. By definition of RCG derivation, there is an instantiated clause in $IC_{G_{rcg},w}$:

$$\varphi \rightarrow \epsilon.$$

By definition of instantiated clauses, there is a clause in P:

$$\gamma \rightarrow \epsilon,$$

such that φ is an instantiated predicate of γ. Let N_t be the non-terminal of φ and γ. Hence, by definition of *TUG2RCG*, there is a clause $p \in P$:

$$\psi \rightarrow \epsilon,$$

where the non-terminal of ψ is N_s, and γ is the non-terminal subsumption of ψ and N_t (in the trivial case, $N_s = N_t$, and $\psi = \gamma$). Let B be a TFS such that $\psi = \textit{tfs2pred}\,(B)$. Clearly, ψ subsumes φ, hence, from Lemma 19, $B \sqsubseteq A$. By definition of instantiated TFS, the value of the input feature of A is a list instantiation of the value of the input feature of B. By definition of *TUG2RCG*, p can be in P in the following cases:

1. there is a rule in $\mathcal{R}$ of the form:

$$B \to \epsilon,$$

or,

2. there is a word $u' \in$ WORDS, such that $B \in \mathcal{L}(u')$.

1. If there is a rule in $\mathcal{R}$:

$$B \to \epsilon,$$

by definition of UG derivation, if $B \sqsubseteq A$, then $A \overset{*}{\Rightarrow}_G \epsilon$. By definition of restricted rules, the value of the input feature of both A and B is *elist*, hence $u = \epsilon$, and $A \overset{*}{\Rightarrow}_G u$.

2. If there is a word $u' \in$ WORDS, such that $B \in \mathcal{L}(u')$, then by definition of restricted lexicon, the content of the input feature of B is u'. Hence, by definition of list instantiation, the content of the input feature of A is $u = u'$. By definition of UG derivation, if $B \sqsubseteq A$, then $A \overset{*}{\Rightarrow}_G u$.

Step: Assume that the lemma holds for every $d \leq k$. We now prove it for $k + 1$: If $\varphi \overset{k+1}{\Rightarrow}_{G_{rcg},w} \epsilon$, then by definition of RCG derivation, there is an instantiated clause in $IC_{G_{rcg},w}$:

$$\varphi \to \varphi_1 \dots \varphi_m,$$

where for every i, $1 \leq i \leq m$, $\varphi_i \overset{d_i}{\Rightarrow}_{G_{rcg},w} \epsilon, d_i \leq k$. From the induction hypothesis, if $\varphi_i =$ *tfs2pred* (A_i), then there is a substring of w, u_i, such that $A_i \overset{*}{\Rightarrow}_G u_i$. By definition of instantiated clauses, there is a clause in P:

$$\gamma \to \gamma_1 \dots \gamma_m,$$

such that φ is an instantiated predicate of γ, and for every i, $1 \leq i \leq m$, φ_i is an instantiated predicate of γ_i. Let t be the non-terminal of γ and φ. Hence, by definition of *TUG2RCG*, there is a clause $p \in P$:

$$\psi \to \gamma_1 \dots \gamma_m,$$

where the non-terminal of ψ is s, and γ is the non-terminal subsumption of ψ and t (in the trivial case, $s = t$, and $\psi = \gamma$). Let B be a TFS such that $\psi =$ *tfs2pred* (B). By definition of *RCG2TUG*, there is a rule in $\mathcal{R}$:

$$B \to C_1 \dots C_m,$$

where for every i, $1 \leq i \leq m$, $\gamma_i =$ *tfs2pred* (C_i). From Lemma 19, for every i, $1 \leq i \leq m$, if γ_i subsumes φ_i, then $C_i \sqsubseteq A_i$. By definition of UG derivation, if $A_i \overset{*}{\Rightarrow}_G u_i$, then $C_i \overset{*}{\Rightarrow}_G u_i$. Hence,

$$B \overset{*}{\Rightarrow}_G u_1 \dots u_m = u'$$

. From Lemma 19, $B \sqsubseteq A$, hence,

$$A \overset{*}{\Rightarrow}_G u'.$$

From Lemma 1, u' is the content of the input feature of A, hence, by definition of *feat2arg*, the first argument of φ is $u' = u$.

□

Theorem 21. *For every RTUG G, if $G_{rcg} = TUG2RCG(G)$, then $L(G_{rcg}) \subseteq L(G)$.*

Proof. If $w \in L(G_{rcg})$, then by definition of $L(G_{rcg})$, there is a derivation path:

$$S(w) \stackrel{*}{\Rightarrow}_{G_{rcg},w} \epsilon.$$

From Lemma 20, there is a derivation path in G:

$$\begin{bmatrix} start \\ \text{INPUT}_{start} : [w] \end{bmatrix} \stackrel{*}{\Rightarrow}_G w.$$

Clearly,

$$A_s = \begin{bmatrix} start \\ \text{INPUT}_{start} : \boxed{1}\,\textit{bi_list} \end{bmatrix} \sqsubseteq \begin{bmatrix} start \\ \text{INPUT}_{start} : [w] \end{bmatrix}$$

Hence, $A_s \stackrel{*}{\Rightarrow}_G w$, and $w \in L(G)$. □