Polynomially-parsable Unification Grammars

Hadas Peled
Department of Computer Science
University of Haifa, Israel
hadas.peled@gmail.com

Shuly Wintner
Department of Computer Science
University of Haifa, Israel
shuly@cs.haifa.ac.il

Abstract

Unification grammars (UG) are a grammatical formalism that underlies several contemporary linguistic theories, including Lexical-functional Grammar and Head-driven Phrase-structure Grammar. UG is an especially attractive formalism because of its expressivity, which facilitates the expression of complex linguistic structures and relations. Formally, UG is Turing-complete, generating the entire class of recursively enumerable languages. This expressivity, however, comes at a price: the universal recognition problem is undecidable for arbitrary unification grammars. We define a constrained version of UG that guarantees efficient processing, while allowing the expression of complex linguistic structures. We do so by proving that the constrained formalism is equivalent to Range Concatenation Grammar, a formalism that generates exactly the class of languages recognizable in deterministic polynomial time. We thus obtain a grammatical formalism that is on one hand highly expressive, and on the other efficient to compute with.

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, *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 poly-

nomial 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; Bertsch and Nederhof, 2001; Kallmeyer et al., 2009), a restricted version of UG that is equivalent to RCG (along with an efficient conversion procedure) is guaranteed to have an efficient recognition algorithm.

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.

We begin in Section 2 by setting up notation and describing related work; specifically, we recall the definitions of (typed) unification grammars, restricted versions thereof, and range concatenation grammars. In Section 3 we define a restricted version of UG, such that constrained grammars can be simulated by an equivalent RCG. The mapping of constrained UG grammars to RCG is given in Section 4. In Section 5 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. Section 6 sketches a proof of the correctness of the two mappings. We conclude with suggestions for future research.

2 Related work

We set up notation in this section for the two formalisms we focus on, namely Typed Unification Grammars (Section 2.1) and Range Concatenation Grammars (Section 2.3). In addition, in Section 2.2 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.

2.1 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* TYPES *and a set of features* FEATS*, an* ***appropriateness specification*** *is a partial function $Approp$: TYPES $\times$ FEATS $\rightarrow$ TYPES, such that:*

- *for every $f \in$ FEATS, let $T_f = \{t \in \text{TYPES} \mid Approp(t, f)\downarrow\}$; 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 $f \in$ FEATS, $Approp\,(t, f) \uparrow$.*

Definition 3 (Type signatures)**.** *A **type signature** is a quadruple $\langle$TYPES, $\sqsubseteq$, FEATS, $Approp\rangle$, where $\langle$TYPES, $\sqsubseteq\rangle$ is a type hierarchy and $Approp$: TYPES $\times$ FEATS $\rightarrow$ TYPES is an appropriateness specification.*

In this work we use the LKB notation (Copestake, 1999, 2002) 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:

TYPES $= \{t_1, t_2, t_3, t_4\}$, 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 $\bot \sqsubseteq t_1$, $t_1 \sqsubseteq t_2$, $\bot \sqsubseteq t_3$, $t_3 \sqsubseteq t_4$, $Approp(t_2, f_1) = t_3$, and $Approp(t_2, f_2) = t_4$.

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 $\delta : Q \times$ FEATS $\rightarrow Q$ specifying the arcs such that every node $q \in Q$ is accessible from $\bar{q}$ and a total function $\theta : Q \rightarrow$ 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 PATHS $=$ FEATS*.

Definition 5 (Paths)**.** *The **paths** of a feature graph A are $\Pi\,(A) = \{\pi \in$ 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 $\delta\,(\bar{q}, \pi_1) = \delta\,(\bar{q}, \pi_2)$ implying $val\,(\pi_1) = val\,(\pi_2)$.*

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 $h : Q_1 \rightarrow Q_2$, 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 5 (page 11) 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 element is its **head** and the rest of the elements are the rule's **body**. We adopt a convention of depicting rules with an arrow ($\rightarrow$) 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 (page 16) depicts a unification grammar and specifies the language it generates.

2.2 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 the efficiency of parsing:

Non-reentrant unification grammars A unification grammar is *non-reentrant* if its rules include 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; and

- Additional 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 than 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 constrained versions of UG 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.)

2.3 Range Concatenation Grammars

Range Concatenation Grammars (RCG) is a syntactic formalism that was introduced by Boullier (1998a). Fundamental to RCG is the notion of *ranges*, pairs of integers denoting occurrences of substrings in an input 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) define 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*

$$\psi_0 \rightarrow \psi_1 ... \psi_j ... \psi_m$$

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. Expecting Definition 20, 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:*

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

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* $(i, j), 0 \leq i \leq j \leq n$, *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****:* $R_w^k = \{\langle \rho_1, \ldots \rho_k\rangle \mid \rho_i \in R_w, 1 \leq i \leq k\}$.

Let $w = a_1 \ldots a_n$ be an input string. Let $w_1 = a_1 \ldots a_i$, $w_2 = a_{i+1} \ldots a_j$ and $w_3 = a_{j+1} \ldots a_n$ 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 $\vec{\rho} = \rho_1, \ldots, \rho_i, \ldots, \rho_p$ is a vector of ranges, by definition $w^{\vec{\rho}}$ denotes the tuple of strings $w^{\rho_1}, \ldots, w^{\rho_i}, \ldots, w^{\rho_p}$.

Definition 14 (Concatenation of ranges)**.** ***Range concatenation*** *is defined by* $w^{\langle i_1..j_1\rangle} \cdot w^{\langle i_2..j_2\rangle} = w^{\langle i_1..j_2\rangle}$ *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 following 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. $A(w^{\langle g..h\rangle}, w^{\langle i..j\rangle}, w^{\langle k..l\rangle}) \rightarrow B(w^{\langle g+1..h\rangle}, w^{\langle i+1..j-1\rangle}, w^{\langle k..l-1\rangle})$ *is an instantiation of the clause* $A(aX, bYc, Zd) \rightarrow B(X, Y, Z)$ *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 than *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* $p = \psi_0 \rightarrow \psi_1 \ldots \psi_m$, *where* $m \geq 0$, *be a clause in* P. *Let* $\alpha \in \{T \cup V\}^*$ *be an argument of some predicate* $\psi_i, 0 \leq i \leq m$. *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* $w \in T^*$, *the set of* ***instantiated predicates*** *is*

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

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

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

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

As is customary in rewriting systems, the language of an RCG grammar is defined 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* $(IP_{G,w})^*$*:*

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

if and only if

$$A_0\left(\vec{\rho_0}\right) \rightarrow A_1\left(\vec{\rho_1}\right) \ldots A_m\left(\vec{\rho_m}\right) \in IC_{G,w}.$$

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 $w = a_1 \ldots a_n$ *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

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

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

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

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

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

To demonstrate the operation of the grammar, we list a derivation of the string $aaaaa$ *with* G_{prime}*:*

$(clause3)$	$S(aaaaa)$	$\rightarrow$	$A(aa, aaaaa, aa, aaaaa)\, eq(aa, aa)$
$(clause8)$	$eq(aa, aa)$	$\rightarrow$	$eq(a, a)$
$(clause8)$	$eq(a, a)$	$\rightarrow$	$eq(\epsilon, \epsilon)$
$(clause8)$	$eq(\epsilon, \epsilon)$	$\rightarrow$	ϵ
$(clause4)$	$A(aa, aaaaa, aa, aaaaa)$	$\rightarrow$	$A(a, aaaa, aa, aaaaa)$
$(clause4)$	$A(a, aaaa, aa, aaaaa)$	$\rightarrow$	$A(\epsilon, aaa, aa, aaaaa)$
$(clause5)$	$A(\epsilon, aaa, aa, aaaaa)$	$\rightarrow$	$A(aa, aaa, aa, aaaaa)$
$(clause4)$	$A(aa, aaa, aa, aaaaa)$	$\rightarrow$	$A(a, aa, aa, aaaaa)$
$(clause4)$	$A(a, aa, aa, aaaaa)$	$\rightarrow$	$A(\epsilon, a, aa, aaaaa)$
$(clause5)$	$A(\epsilon, a, aa, aaaaa)$	$\rightarrow$	$A(aa, a, aa, aaaaa)$
$(clause4)$	$A(aa, a, aa, aaaaa)$	$\rightarrow$	$A(a, \epsilon, aa, aaaaa)$
$(clause7)$	$A(a, \epsilon, aa, aaaaa)$	$\rightarrow$	$NonEmpty(a)\, Len2(aa)$
$(clause10)$	$NonEmpty(a)$	$\rightarrow$	ϵ
$(clause11)$	$Len2(aa)$	$\rightarrow$	ϵ

3 Restricted Typed Unification Grammars

Our goal is to define a restricted version of TUG whose grammars can be converted 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.

3.1 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, 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*

$$\begin{aligned} bi\_list\_\text{TYPES} &= \{\textit{terminal}, \textit{node}, \textit{null}, \textit{ne\_node}, \textit{bi\_list}, \textit{elist}, \textit{ne\_bi\_list}\} \\ bi\_list\_\text{FEATS} &= \{\text{CURR}, \text{PREV}, \text{NEXT}, \text{HEAD}, \text{TAIL}\} \end{aligned}$$

Then ***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 5 (bi_list)**.** *As an example of a TFS representing a bidirectional list of terminals, consider A (Figure 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 $\boxed{3}$.*

$$A = \begin{bmatrix} \textit{ne\_bi\_list} \\ \text{HEAD} : \boxed{1} \begin{bmatrix} \textit{ne\_node} \\ \text{CURR} : a \\ \text{PREV} : \textit{null} \\ \text{NEXT} : \boxed{2} \begin{bmatrix} \textit{ne\_node} \\ \text{CURR} : b \\ \text{PREV} : \boxed{1} \\ \text{NEXT} : \boxed{3} \begin{bmatrix} \textit{ne\_node} \\ \text{CURR} : b \\ \text{PREV} : \boxed{2} \\ \text{NEXT} : \textit{null} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{TAIL} : \boxed{3} \end{bmatrix}$$

Figure 1: An example TFS representing a bidirectional list of terminals

A TFS $A = \boxed{\textit{bi\_list}}$ (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 5 is $\langle a, b, b\rangle$.

Let A_1 and A_2 be two bi_lists. The **concatenation** 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.

Example 6. *(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 7. *(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 $\langle a, a, b\rangle$.*

3.2 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 is used as the super-type of every TFS occurring 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 4.1.

Definition 23 (Restricted signature)**.** *A type signature $\langle$TYPES, 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,*
  - *Approp (start, INPUT_{start}) = bi_list, and*
  - *Approp (start, f) $\uparrow$ for every $f \neq \text{INPUT}_{start}$;*
- *Any type t, such that:*
  - *main $\sqsubseteq$ t;*

– *if main* $\overset{\circ}{\sqsubset}$ *t:*
  * $Approp\,(t, \text{INPUT}_t) = bi\_list$*;*
  * *for every* $\text{F} \neq \text{INPUT}_t$*, if* $Approp\,(t, \text{F})\downarrow$*, then* $Approp\,(t, \text{F}) = bi\_list$*.*
– *if t'* $\overset{\circ}{\sqsubset}$ *t and t'* $\neq$ *main,* $\{\text{F} \mid Approp\,(t, \text{F})\downarrow\} = \{\text{F} \mid Approp\,(t', \text{F})\downarrow\}$*;*

- *No other types are allowed in* TYPES.

Example 8. *(Restricted typed signature). Let*

$$\text{TYPES} = \left\{ \begin{array}{l} main, cat, start, np, v, \\ terminal, lamb, Rachel, Jacob, \ldots \end{array} \right\} \bigcup bi\_list\_\text{TYPES}$$

$$\text{FEATS} = \{\text{INPUT}_{cat}, \text{INPUT}_v, \text{SUBCAT}\} \bigcup bi\_list\_\text{FEATS}$$

Then the following typed signature (unioned with the bi_list signature as of Definition 22) is restricted:

main
  cat INPUT$_{cat}$*: bi_list*
    start
    np
    v
      v_np
      v_np_s
terminal
  lamb
  Rachel
  Jacob
  ...

3.3 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 9 (Restricted TFS)**.** *Given the following signature fragment:*

main
  counter INPUT$_{counter}$*: bi_list* COUNT*: bi_list*
terminal
  a
  b

The following TFS is restricted:

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

3.4 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 10. *(Restricted $\mathcal{L}(b)$).*

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

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

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

3.5 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 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:*

$$\begin{bmatrix} t_0 \\ \text{INPUT}_{t_0} : A_1 \cdot \ldots \cdot A_k \\ \text{F}_1^0 : B_1 \\ \vdots \\ \text{F}_n^0 : B_{n_0} \end{bmatrix} \rightarrow \begin{bmatrix} t_1 \\ \text{INPUT}_{t_1} : A_1 \\ \text{F}_1^1 : C_1^1 \\ \vdots \\ \text{F}_{n_1}^1 : C_{n_1}^1 \end{bmatrix} \ldots \begin{bmatrix} t_k \\ \text{INPUT}_{t_k} : A_k \\ \text{F}_1^k : Ci_1^k \\ \vdots \\ \text{F}_{n_k}^k : C_{n_k}^k \end{bmatrix}$$

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_j^i 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 12. *In the following restricted rule, the feature values of the mother are obtained by concatenating the feature values of the daughters:*

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

3.6 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* $\langle \text{TYPES}, \sqsubseteq, \text{FEATS}, Approp \rangle$ *is* ***restricted typed UG (RTUG)*** *if:*

- *The type signature* $\langle \text{TYPES}, \sqsubseteq, \text{FEATS}, Approp \rangle$ *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.*

3.7 Examples of Restricted TUG

We demonstrate two RTUG grammars, one for a formal language and one for a small fragment of a natural language. The grammar G_{abc} 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 13. *($a^n b^n c^n$) Figure 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} of 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 3 demonstrates a derivation tree for the string "aabbcc" with this grammar.*

Example 14 (Long distance dependencies)**.** *Figures 4, 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:*

Signature The signature includes the bi_list_signature, as defined in Definition 22, and in addition:

main

simple

counter INPUT_{count}:*bi_list* COUNT:*bi_list*

at

bt

ct

start INPUT_{start}:*bi_list*

terminal

a

b

c

Lexicon

$$a \rightarrow \begin{bmatrix} at \\ \text{INPUT}_{count} : \langle a \rangle \\ \text{COUNT} : \langle a \rangle \end{bmatrix}, \quad b \rightarrow \begin{bmatrix} bt \\ \text{INPUT}_{count} : \langle b \rangle \\ \text{COUNT} : \langle a \rangle \end{bmatrix}, \quad c \rightarrow \begin{bmatrix} ct \\ \text{INPUT}_{count} : \langle c \rangle \\ \text{COUNT} : \langle a \rangle \end{bmatrix}$$

Rules

$$\begin{bmatrix} start \\ \text{INPUT}_{start} : \boxed{1} \cdot \boxed{2} \cdot \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} at \\ \text{INPUT}_{count} : \boxed{1} bi\_list \\ \text{COUNT} : \boxed{1} \end{bmatrix} \begin{bmatrix} bt \\ \text{INPUT}_{count} : \boxed{2} bi\_list \\ \text{COUNT} : \boxed{1} \end{bmatrix} \begin{bmatrix} ct \\ \text{INPUT}_{count} : \boxed{3} bi\_list \\ \text{COUNT} : \boxed{1} \end{bmatrix}$$

$$\begin{bmatrix} at \\ \text{INPUT}_{count} : a \cdot \boxed{1} \\ \text{COUNT} : a \cdot \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} at \\ \text{INPUT}_{count} : \langle a \rangle \\ \text{COUNT} : \langle a \rangle \end{bmatrix} \begin{bmatrix} at \\ \text{INPUT}_{count} : \boxed{1} bi\_list \\ \text{COUNT} : \boxed{2} bi\_list \end{bmatrix}$$

$$\begin{bmatrix} bt \\ \text{INPUT}_{count} : b \cdot \boxed{1} \\ \text{COUNT} : a \cdot \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} bt \\ \text{INPUT}_{count} : \langle b \rangle \\ \text{COUNT} : \langle a \rangle \end{bmatrix} \begin{bmatrix} bt \\ \text{INPUT}_{count} : \boxed{1} bi\_list \\ \text{COUNT} : \boxed{2} bi\_list \end{bmatrix}$$

$$\begin{bmatrix} ct \\ \text{INPUT}_{count} : c \cdot \boxed{1} \\ \text{COUNT} : a \cdot \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} ct \\ \text{INPUT}_{count} : \langle c \rangle \\ \text{COUNT} : \langle a \rangle \end{bmatrix} \begin{bmatrix} ct \\ \text{INPUT}_{count} : \boxed{1} bi\_list \\ \text{COUNT} : \boxed{2} bi\_list \end{bmatrix}$$

Figure 2: An RTUG, G_{abc}, generating the language $a^n b^n c^n$

- *start;*
- *np for noun phrases;*
- *vp for verb phrases;*
- *v for verbs with no subcategorized complement;*

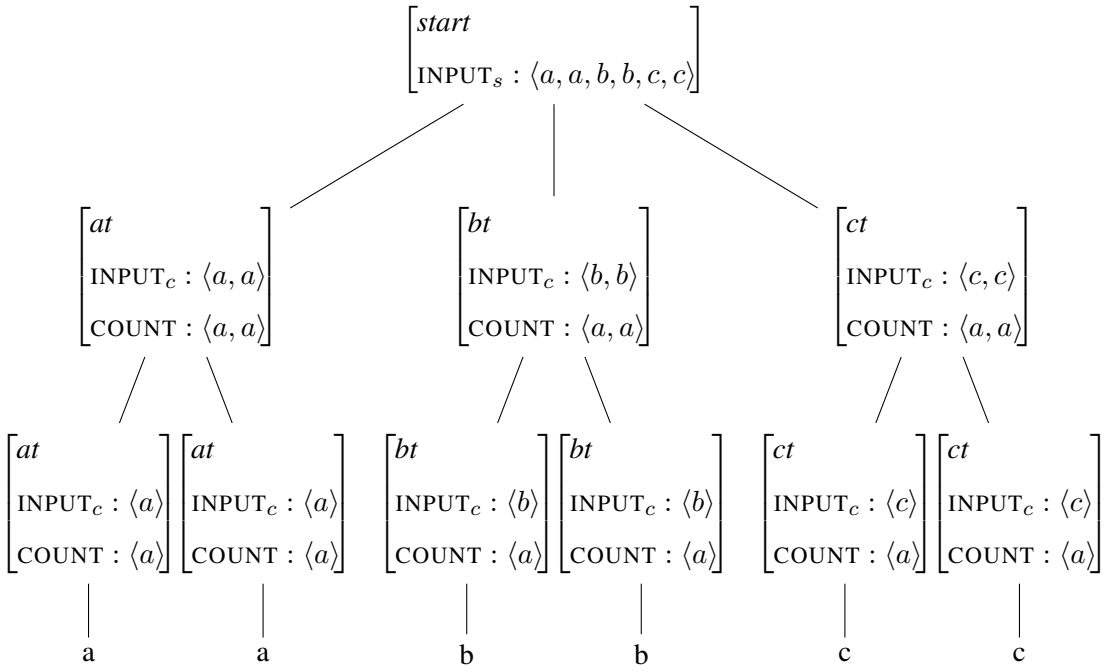

Figure 3: A derivation tree for the string *"aabbcc"*

- *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 6 demonstrates a derivation tree of the string Laban wondered whom Jacob loves *with this grammar.*

4 Simulation of RTUG by RCG

Let $G_{ug} = \langle \mathcal{R}, A_s, \mathcal{L}\rangle$ be an RTUG over a restricted type signature $\langle \textsc{Types}, \sqsubseteq, \textsc{Feats}, Approp\rangle$. 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

Signature

main

start	INPUT_{start} : *bi_list*	
v	INPUT_{v} : *bi_list*	
v_subcat	INPUT_{vs} : *bi_list*	
	v_np	
	v_s	
np	INPUT_{np} : *bi_list*	
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*

terminal

Jacob
Rachel
Laban
whom
loves
wondered
...

Lexicon

$$\textit{loves} \rightarrow \begin{bmatrix} \textit{v\_np} \\ \text{INPUT}_v : \langle \textit{loves} \rangle \end{bmatrix}, \quad \textit{wondered} \rightarrow \begin{bmatrix} \textit{v\_s} \\ \text{INPUT}_v : \langle \textit{wondered} \rangle \end{bmatrix}$$

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

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

Figure 4: An RTUG, $G_{longdist}$

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 $\langle \text{TYPES}, \sqsubseteq, \text{FEATS}, Approp \rangle$. The **RCG mapping of** G_{tug}, denoted by **TUG2RCG** (G_{tug}), is an RCG $G_{rcg} = \langle N, T, V, S \rangle$, such that:*

- $T = \{t \mid \textit{terminal} \sqsubset t\}$*;*
- $N = \{N_t \mid \textit{main} \sqsubset t\}$*. For each* $N_t \in N$*,* $ar\,(N_t) = |\{f \mid Approp\,(t, f)\downarrow\}|$*;*
- *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 $p \in P$, either:*
  - *p is a unification clause, as in Definition 31 below, or*
  - *there is a rule $r \in \mathcal{R}$, such that p is part of rule2clause (r), as in Definition 32 below, or*
  - *there is a lexical entry $l \in \mathcal{L}$, such that p is part of rule2clause (l), as in Definition 33 below.*

Rules

start_rule :

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

queue_rule :

(2) $\begin{bmatrix} s_q \\ \text{INPUT}_{s_q} : \boxed{1} \cdot \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} np_q \\ \text{INPUT}_{np} : \boxed{1} \textit{bi\_list} \end{bmatrix} \begin{bmatrix} s\_slash \\ \text{INPUT}_{s\_slash} : \boxed{2} \textit{bi\_list} \\ \text{SLASH}_s : \boxed{1} \end{bmatrix}$

slash_rules :

(3) $\begin{bmatrix} s\_slash \\ \text{INPUT}_{s\_slash} : \boxed{1} \cdot \boxed{2} \\ \text{SLASH}_s : \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} np \\ \text{INPUT}_{np} : \boxed{1} \textit{bi\_list} \end{bmatrix} \begin{bmatrix} vp\_slash \\ \text{INPUT}_{vp\_slash} : \boxed{2} \textit{bi\_list} \\ \text{SLASH}_{vp} : \boxed{3} \end{bmatrix}$

(4) $\begin{bmatrix} vp\_slash \\ \text{INPUT}_{vp\_slash} : \boxed{1} \textit{bi\_list} \\ \text{SLASH}_{vp} : \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} v\_np \\ \text{INPUT}_{vs} : \boxed{1} \textit{bi\_list} \end{bmatrix}$

(5) $\begin{bmatrix} vp\_slash \\ \text{INPUT}_{vp\_slash} : \boxed{1} \cdot \boxed{2} \\ \text{SLASH}_{vp} : \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} v\_s \\ \text{INPUT}_{vs} : \boxed{1} \textit{bi\_list} \end{bmatrix} \begin{bmatrix} s\_slash \\ \text{INPUT}_{s\_slash} : \boxed{2} \textit{bi\_list} \\ \text{SLASH}_s : \boxed{3} \textit{bi\_list} \end{bmatrix}$

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

vp_rules :

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

(8) $\begin{bmatrix} vp \\ \text{INPUT}_{vp} : \boxed{1} \cdot \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} v\_s \\ \text{INPUT}_{vs} : \boxed{1} \textit{bi\_list} \end{bmatrix} \begin{bmatrix} start \\ \text{INPUT}_{start} : \boxed{2} \textit{bi\_list} \end{bmatrix}$

(9) $\begin{bmatrix} vp \\ \text{INPUT}_{vp} : \boxed{1} \cdot \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} v\_s \\ \text{INPUT}_{vs} : \boxed{1} \textit{bi\_list} \end{bmatrix} \begin{bmatrix} s_q \\ \text{INPUT}_{s_q} : \boxed{2} \textit{bi\_list} \end{bmatrix}$

(10) $\begin{bmatrix} vp \\ \text{INPUT}_{vp} : \boxed{1} \cdot \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} v\_np \\ \text{INPUT}_{vs} : \boxed{1} \textit{bi\_list} \end{bmatrix} \begin{bmatrix} np \\ \text{INPUT}_{np} : \boxed{2} \textit{bi\_list} \end{bmatrix}$

Figure 5: An RTUG, $G_{longdist}$ (continued)

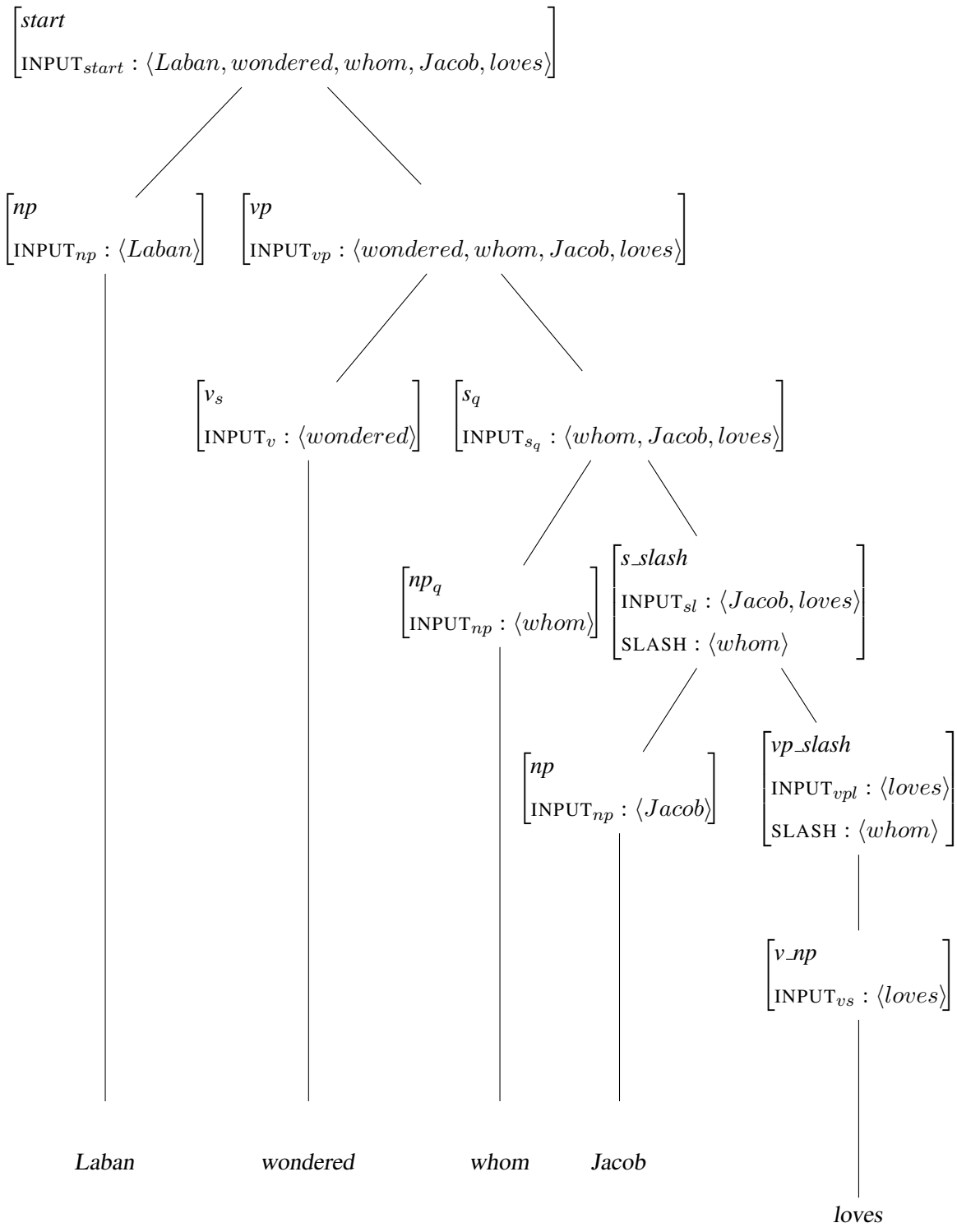

Figure 6: A derivation tree of the string *Laban wondered whom Jacob loves*

4.1 Mapping of type signatures 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. First, we set the number of variables, f, needed for expressing RCG clauses; this is the number of features appropriate for t_{max}, the type with the most appropriate features. We then define an RCG clause for each pair of types that have a common subtype.

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 $\sqsubset t_1$, and t_1 and t_2 have $k \leq f$ features, the following clause is included in P:*

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

Example 15. *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}:

$$\begin{array}{lcl} \textit{counter}\,(X_1, X_2) & \rightarrow & \textit{at}\,(X_1, X_2) \\ \textit{counter}\,(X_1, X_2) & \rightarrow & \textit{bt}\,(X_1, X_2) \\ \textit{counter}\,(X_1, X_2) & \rightarrow & \textit{ct}\,(X_1, X_2) \end{array}$$

4.2 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:*

$$A_0 \to A_1 \dots A_n$$

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 $\textsc{Tags}(r) = \{\boxed{1}, \dots, \boxed{d}\}$*. 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:

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

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 $N_s(\alpha_1, \dots, \alpha_k)$, 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 16. *The following UG rule*

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

is simulated by the following RCG clause:

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

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.*

4.3 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:*

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

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:*

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

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

Example 17. *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:

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

Example 18. *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* $np \overset{\circ}{\sqsubset} np\_pl$. *Then* l *is mapped to the following RCG clauses:*

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

$$np\_sg\,(sheep, X_1) \rightarrow \epsilon$$

$$np\_pl\,(sheep, X_1) \rightarrow \epsilon$$

4.4 Examples

We demonstrate the mapping of RTUG to equivalent RCGs on the two grammars presented in Section 3.7.

Example 19. *(*$a^n b^n c^n$*) Here is the RCG mapping of* G_{abc} *that was presented in Example 13.* $a^n b^n c^n$ *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:*

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

The clauses obtained from the lexicon are:

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

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:

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

Example 20 (Long distance dependencies)**.** *The RCG mapping TUG2RCG $(G_{longdist})$ (see Example 14) is:*

Unification clauses

$$\begin{array}{lll} v\_subcat\,(X) & \rightarrow & v\_s\,(X) \\ v\_subcat\,(X) & \rightarrow & v\_np\,(X) \end{array}$$

Lexicon clauses

$$\begin{array}{lll} v\_np\,(\mathit{loves}) & \rightarrow & \epsilon \\ v\_s\,(\mathit{wondered}) & \rightarrow & \epsilon \\ np\,(\mathit{Jacob}) & \rightarrow & \epsilon \\ np\,(\mathit{Rachel}) & \rightarrow & \epsilon \\ np\,(\mathit{Laban}) & \rightarrow & \epsilon \\ np_q\,(\mathit{whom}) & \rightarrow & \epsilon \end{array}$$

Rule clauses

$$\begin{array}{llllll} start\_rule: & (1) & start\,(XY) & \rightarrow & np\,(X)\,vp\,(Y) \\ \\ queue\_rules: & (2) & s_q\,(XY) & \rightarrow & np_q\,(X)\,s\_slash\,(Y,X) \\ \\ slash\_rules: & (3) & s\_slash\,(XY,Z) & \rightarrow & np\,(X)\,vp\_slash\,(Y,Z) \\ & (4) & vp\_slash\,(X,Y) & \rightarrow & v\_np\,(X) \\ & (5) & vp\_slash\,(XY,Z) & \rightarrow & v\_s\,(X)\,s\_slash\,(Y,Z) \\ & (6) & s\_slash\,(X,Y) & \rightarrow & vp\,(X) \\ \\ vp\_rules: & (7) & vp\,(X) & \rightarrow & v\,(X) \\ & (8) & vp\,(XY) & \rightarrow & v\_s\,(X)\,start\,(Y) \\ & (9) & vp\,(XY) & \rightarrow & v\_s\,(X)\,s_q\,(Y) \\ & (10) & vp\,(XY) & \rightarrow & v\_np\,(X)\,np\,(Y) \end{array}$$

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

(1)	*start* (*Laban wondered whom Jacob loves*)	$\rightarrow$	*np* (*Laban*) *vp* (*wondered whom Jacob loves*)
	np (*Laban*)	$\rightarrow$	ϵ
(9)	*vp* (*wondered whom Jacob loves*)	$\rightarrow$	*v_s* (*wondered*) s_q (*whom Jacob loves*)
	v_s (*wondered*)	$\rightarrow$	ϵ
(2)	s_q (*whom Jacob loves*)	$\rightarrow$	np_q (*whom*) *s_slash* (*Jacob loves,whom*)
	np_q (*whom*)	$\rightarrow$	ϵ
(3)	*s_slash* (*Jacob loves,whom*)	$\rightarrow$	*np* (*Jacob*) *vp_slash* (*loves,whom*)
	np (*Jacob*))	$\rightarrow$	ϵ
(4)	*vp_slash* (*loves,whom*)	$\rightarrow$	*v_np* (*loves*)
	v_np (*loves*)	$\rightarrow$	ϵ

5 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 $\langle$TYPES$, \sqsubseteq,$ FEATS$, Approp \rangle$, such that:*

Type signature *In addition to the types that are part of* every *RTUG (Definition 23), the signature of G_{ug} includes the following types:*

- *A type S' such that: main $\overset{\circ}{\sqsubset}$ S' and $Approp$ (S', 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";*
  - *$Approp$ (S", INPUT$_{S''}$) = bi_list; and*
  - *$Approp$ (S", 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 (Definition 23). During the entire derivation phase, the input feature of the TFSs is always empty (elist).
- *For every RCG non-terminal $A \in N$ such that $ar(A) = k$, there is a type A $\in$* TYPES *such that:*
  - *main $\overset{\circ}{\sqsubset}$ A;*
  - *$Approp$ (A, INPUT$_A$) = bi_list;*
  - *$Approp$ (A, ARG$_A$) = bi_list; and*

– *for every* i, $1 \leq i \leq k$, $Approp\left(A, \text{ARG}_A^i\right) = bi\_list$.

- *For every RCG terminal* $\alpha \in T$ *there is a type* $\alpha \in$ TYPES *such that terminal* $\overset{\circ}{\sqsubset}\ \alpha$, *and* α *is featureless.*

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

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

Start symbol

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

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

- *The start rule is of the form:*

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

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

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

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

- *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* $r = clause2rule\,(p)$; *see Definition 35 below.*

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

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

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*

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

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 defined as follows: A_i is a TFS of type N_i with $k_i + 1$ features of type bi_list:*

$$A_i = \begin{bmatrix} N_i \\ \text{INPUT}_{N_i} : \textit{elist} \\ \text{ARG}1_{N_i} : B_1^i \\ \vdots \\ \text{ARG}k_{iN_i} : B_{k_i}^i \end{bmatrix}$$

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 $p = N_0(\alpha_1, \ldots, \alpha_k) \rightarrow \epsilon$, then clause2rule (p) is the following rule:

$$\begin{bmatrix} N_0 \\ \text{INPUT}_{N_0} : \textit{elist} \\ \text{ARG}1_{N_0} : B_1 \\ \vdots \\ \text{ARG}k_{N_0} : B_k \end{bmatrix} \rightarrow \epsilon$$

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 $\alpha \in T \bigcup V^*$. Then* ***arg2feat*** *(α), its* ***bi_list mapping****, is a TFS of type bi_list defined as follows:*

- *if $\alpha = \epsilon$, then arg2feat (α) = 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 (δ);*
- *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 (δ).*

Example 21. *(bi_list Mapping of RCG arguments) Let $\alpha = aX_1X_2$, such that $a \in T$ and $X_1, X_2 \in V$, then*

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

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

Example 22 (G_{prime})**.** *We demonstrate how G_{prime} of Example 4 is mapped 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:

main
- *start* INPUT$_{start}$*: bi_list*
- *S'* INPUT$_{S'}$*: bi_list*
- *S"* INPUT$_{S''}$*: bi_list* ARG$_{S''}$*: bi_list*
- *A* INPUT$_{A}$*: bi_list* ARG1_{A}*: bi_list* ARG2_{A}*: bi_list* ARG3_{A}*: bi_list* ARG4_{A}*: bi_list*
- *eq* INPUT$_{eq}$*: bi_list* ARG1_{eq}*: bi_list* ARG2_{eq}*: bi_list*
- *NonEmpty* INPUT$_{NonEmpty}$*: bi_list* ARG$1_{NonEmpty}$*: bi_list*
- *Len2* INPUT$_{Len2}$*: bi_list* ARG1_{Len2}*: bi_list*
- *MinLen2* INPUT$_{MinLen2}$*: bi_list* 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*

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

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

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

$$\begin{bmatrix} S'' \\ \text{INPUT}_{S''} : \textit{elist} \\ \text{ARG}_{S''} : \boxed{1}\textit{bi\_list} \cdot \langle a \rangle \cdot \boxed{2}\textit{bi\_list} \end{bmatrix} \rightarrow \begin{bmatrix} A \\ \text{INPUT}_A : \textit{elist} \\ \text{ARG}1_A : \boxed{1} \\ \text{ARG}2_A : \boxed{1} \cdot \langle a \rangle \cdot \boxed{2} \\ \text{ARG}3_A : \boxed{1} \\ \text{ARG}3_A : \boxed{1} \cdot \langle a \rangle \cdot \boxed{2} \end{bmatrix} \begin{bmatrix} eq \\ \text{INPUT}_{eq} : \textit{elist} \\ \text{ARG}1_{eq} : \boxed{1} \\ \text{ARG}2_{eq} : \boxed{2} \end{bmatrix}$$

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

$$\begin{bmatrix} A \\ \text{INPUT}_A : \textit{elist} \\ \text{ARG}1_A : \langle a \rangle \cdot \boxed{1}\textit{bi\_list} \\ \text{ARG}2_A : \langle a \rangle \cdot \boxed{2}\textit{bi\_list} \\ \text{ARG}3_A : \boxed{3}\textit{bi\_list} \\ \text{ARG}4_A : \boxed{4}\textit{bi\_list} \end{bmatrix} \rightarrow \begin{bmatrix} A \\ \text{INPUT}_A : \textit{elist} \\ \text{ARG}1_A : \boxed{1} \\ \text{ARG}2_A : \boxed{2} \\ \text{ARG}3_A : \boxed{3} \\ \text{ARG}4_A : \boxed{4} \end{bmatrix}$$

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

$$\begin{bmatrix} A \\ \text{INPUT}_A : \textit{elist} \\ \text{ARG}1_A : \textit{elist} \\ \text{ARG}2_A : \boxed{2}\textit{bi\_list} \\ \text{ARG}3_A : \boxed{3}\textit{bi\_list} \\ \text{ARG}4_A : \boxed{4}\textit{bi\_list} \end{bmatrix} \rightarrow \begin{bmatrix} A \\ \text{INPUT}_A : \textit{elist} \\ \text{ARG}1_A : \boxed{3} \\ \text{ARG}2_A : \boxed{2} \\ \text{ARG}3_A : \boxed{3} \\ \text{ARG}4_A : \boxed{4} \end{bmatrix}$$

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

$$\begin{bmatrix} A \\ \text{INPUT}_A : \textit{elist} \\ \text{ARG}1_A : \boxed{1}\textit{bi\_list} \\ \text{ARG}2_A : \textit{elist} \\ \text{ARG}3_A : \langle a \rangle \cdot \boxed{3}\textit{bi\_list} \\ \text{ARG}4_A : \boxed{4}\textit{bi\_list} \end{bmatrix} \rightarrow \begin{bmatrix} A \\ \text{INPUT}_A : \textit{elist} \\ \text{ARG}1_A : \boxed{3} \\ \text{ARG}2_A : \boxed{4} \\ \text{ARG}3_A : \boxed{3} \\ \text{ARG}4_A : \boxed{4} \end{bmatrix} \begin{bmatrix} \textit{NonEmpty} \\ \text{INPUT}_{NonEmpty} : \textit{elist} \\ \text{ARG}1_{NonEmpty} : \boxed{1} \end{bmatrix} \begin{bmatrix} \textit{MinLen2} \\ \text{INPUT}_{MinLen2} : \textit{elist} \\ \text{ARG}1_{MinLen2} : \boxed{3} \end{bmatrix}$$

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

$$\begin{bmatrix} A \\ \text{INPUT}_A : elist \\ \text{ARG}1_A : \boxed{1}\, bi\_list \\ \text{ARG}2_A : elist \\ \text{ARG}3_A : \boxed{3}\, bi\_list \\ \text{ARG}4_A : \boxed{4}\, bi\_list \end{bmatrix} \rightarrow \begin{bmatrix} NonEmpty \\ \text{INPUT}_{NonEmpty} : elist \\ \text{ARG}1_{NonEmpty} : \boxed{1} \end{bmatrix} \begin{bmatrix} Len2 \\ \text{INPUT}_{Len2} : elist \\ \text{ARG}1_{Len2} : \boxed{3} \end{bmatrix}$$

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

$$\begin{bmatrix} eq \\ \text{INPUT}_{eq} : elist \\ \text{ARG}1_{eq} : \langle a \rangle \cdot \boxed{1}\, bi\_list \\ \text{ARG}2_{eq} : \langle a \rangle \cdot \boxed{2}\, bi\_list \end{bmatrix} \rightarrow \begin{bmatrix} eq \\ \text{INPUT}_{eq} : elist \\ \text{ARG}1_{eq} : \boxed{1} \\ \text{ARG}2_{eq} : \boxed{2} \end{bmatrix}$$

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

$$\begin{bmatrix} eq \\ \text{INPUT}_{eq} : elist \\ \text{ARG}1_{eq} : elist \\ \text{ARG}2_{eq} : elist \end{bmatrix} \rightarrow \epsilon$$

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

$$\begin{bmatrix} NonEmpty \\ \text{INPUT}_{NonEmpty} : elist \\ \text{ARG}1_{NonEmpty} : \langle a \rangle \cdot \boxed{1}\, bi\_list \end{bmatrix} \rightarrow \epsilon$$

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

$$\begin{bmatrix} Len2 \\ \text{INPUT}_{Len2} : elist \\ \text{ARG}1_{Len2} : \langle a, a \rangle \end{bmatrix} \rightarrow \epsilon$$

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

$$\begin{bmatrix} MinLen2 \\ \text{INPUT}_{MinLen2} : elist \\ \text{ARG}1_{MinLen2} : \langle a \rangle \cdot \boxed{1}\, bi\_list \end{bmatrix} \rightarrow \begin{bmatrix} NonEmpty \\ \text{INPUT}_{NonEmpty} : elist \\ \text{ARG}1_{NonEmpty} : \boxed{1} \end{bmatrix}$$

6 Proof of correctness (sketch)

In this section we sketch the proof of the main result of this work, namely that $\mathcal{L}_{RTUG} = \mathcal{L}_{RCG}$. We first prove that $\mathcal{L}_{RCG} \subseteq \mathcal{L}_{RTUG}$, by proving the correctness of the *RCG2TUG* mapping (Definition 34), and then that $\mathcal{L}_{RTUG} \subseteq \mathcal{L}_{RCG}$, by proving the correctness of the *TUG2RCG* mapping (Definition 30). While the proofs are technical, they are not difficult. We only provide the outline in this section; readers interested in the full details are referred to Peled (2011).

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

Example 23 (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 $w \in \text{WORDS}^*$, 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.

Example 24 (Instantiated TFS)**.** *Let A be a TFS over the signature presented in Example 13, 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.

Example 25. *(Rule instantiation) Let*

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

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).

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

6.2 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 $G_{tug} =$ *RCG2TUG* (G), and show first that $L(G) \subseteq L(G_{tug})$, and then that $L(G_{tug}) \subseteq L(G)$.

First we establish the commutativity of string instantiation and *bi_list* instantiation with *arg2feat* (Definition 35). See the commutative diagram below:

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

Then we establish the commutativity of instantiation (of predicates and of TFSs), with *pred2tfs* (Definition 35). See the commutative diagram below:

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

We then prove that $L(G) \subseteq L(G_{tug})$ by showing that if $w \in L(G)$, then $w \in L\left(G_{tug_{|w}}\right)$, implying that $w \in L(G_{tug})$. $L(G_{tug}) \subseteq L(G)$ is established by showing 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 $w \in L(G_{tug})$.

6.3 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 choose $G_{rcg} =$ *TUG2RCG* (G), (Definition 30), and show first that $L(G) \subseteq L(G_{rcg})$, and then that $L(G_{rcg}) \subseteq L(G)$.

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. 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. A predicate that is subsumed by no other predicate is called **a maximum predicate**.

Example 26. *Consider $G_{longdist}$ and TUG2RCG $(G_{longdist})$ of Example 14:*

- *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.*

We establish that string instantiation and *bi_list* instantiation commute with *feat2arg* (Definition 32). See the following commutative diagram:

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

We then address 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). See the following commutative diagram:

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

Example 27. *Consider the following fragment of the signature of $G_{longdist}$, repeated from Example 14:*

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:*

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

$$\begin{array}{lcll} \textit{tfs2pred}\,(A) & = & \varphi & = & v\_subcat\,(X)\,, \quad X \in V \\ \textit{tfs2pred}\,(IA) & = & \psi & = & v\_np\,(loves) \end{array}$$

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

$$\begin{array}{lcl} \textit{v\_subcat}\,(X) & \rightarrow & \textit{v\_np}\,(X) \\ \textit{v\_subcat}\,(X) & \rightarrow & \textit{v\_s}\,(X) \end{array}$$

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

We then prove that if *tfs2pred* (A) subsumes *tfs2pred* (B), then $A \sqsubseteq B$. See the following commutative diagram:

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

Finally, 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 $w \in L\,(G)$. $L\,(G_{rcg}) \subseteq L\,(G)$ is established by showing that if $w \in L\left(G_{rcg_{|_w}}\right)$, then $w \in L\,(G)$.

7 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. 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.

RTUG is a highly restricted variant of unification grammars, allowing features of a single type only, bi-directional lists of terminals. This fact makes the development of grammars in this formalism rather difficult. Compared to other highly restricted versions of UG, *One-reentrant unification grammars* and *PLPATR* (Section 2.2), 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, therefore, would be a formalism combining the benefits of RTUG and one-reentrant UG (or PLPATR). In this combined formalism, feature structures would 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 formalism would facilitate the design of natural language grammars, allowing simple implementation of linguistic phenomena like agreement, while at the same time guaranteeing efficient recognition time.

References

G. Edward Barton, Jr., Robert C. Berwick, and Eric Sven Ristad. The complexity of LFG. In G. Edward Barton, Jr., Robert C. Berwick, and Eric Sven Ristad, editors, *Computational Complexity and Natural Language*, Computational Models of Cognition and Perception, chapter 4, pages 103–114. MIT Press, Cambridge, MA, 1987.

Eberhard Bertsch and Mark-Jan Nederhof. On the complexity of some extensions of rcg parsing. In *Proceedings of the Seventh International Workshop on Parsing Technologies (IWPT-2001)*. Tsinghua University Press, October 2001. ISBN 7-302-04925-4.

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.

Ann Copestake. The (new) LKB system. Technical report, Stanford University, September 1999.

Ann Copestake. *Implementing Typed Feature Structure Grammars*. CSLI Publications, Stanford, 2002.

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, New York, NY, 2012. URL `http://cl.haifa.ac.il/ug`.

Efrat Jaeger, Nissim Francez, and Shuly Wintner. Unification grammars and off-line parsability. *Journal of Logic, Language and Information*, 14(2):199–234, 2005.

Mark Johnson. *Attribute-Value Logic and the Theory of Grammar*, volume 16 of *CSLI Lecture Notes*. CSLI, Stanford, California, 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)*, 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.

Hadas Peled. Polynomially-parsable unification grammars. Master's thesis, Department of Computer Science, University of Haifa, November 2011.