Millstream Systems

Suna Bensch Frank Drewes
Department of Computing Science, Umeå University,
S–901 87 Umeå, Sweden
{suna,drewes}@cs.umu.se

Abstract

We introduce Millstream systems, a mathematical framework in the tradition of the Theory of Computation that uses logic to formalize the interfaces between different aspects of language, the latter being described by any number of independent modules. Unlike other approaches that serve a similar goal, Millstream systems neither presuppose nor establish a particular linguistic theory or focus, but can be instantiated in various ways to accomodate different points of view.

UMINF 09.21 ISSN 0348-0542

1 Introduction

We introduce Millstream systems, a mathematical framework for the description of language that makes it possible to formalize and reason about the interaction and interdependency of different aspects of language, such as phonology, morphology, syntax, and semantics. Our approach is motivated and inspired by contemporary linguistic theories that refrain from the idea these linguistical levels are intrinsically ordered, the output of one linguistical level providing the input for the next one in a hierarchical fashion. In [Sad91, Jac97, Jac02, Jac07], Sadock and Jackendoff promote the view that these "levels" instead correspond to autonomous modules that work simultaneously and are linked with each other through interfaces that describe their interaction and interdependency. Both authors argue that the human way of processing language is more adequately described by such a non-hierarchical approach, as the human brain seems to store and process different informational levels (including the linguistical levels) in parallel, at the same time linking them according to certain rules in order to create a whole that is more than the sum of its parts.

Many grammatical formalisms that are studied in modern computational linguistics establish interfaces between different aspects of language. Some of the most well-known formalisms of this kind are the Head-Driven Phrase Structure Grammar (HPSG), the Lexical Functional Grammar (LFG), and the Combinatory Categorical Grammar (CCG).[1] Each of these focuses on certain aspects of language (usually including at least syntax and semantics), and provides some type of mathematical mechanism for expressing the interfaces between them. Both HPSG and LFG use logical formalisms (feature logic and linear logic, resp.) for this purpose.

The goal of this article is to propose a formalism that provides a generic framework for studying and implementing various notions of interfaces, including those mentioned above. We start out from the view advocated by Sadock and Jackendoff, aiming at a formalism that is able to capture an arbitrary number of linguistic aspects. To achieve this, we assume that every single aspect, such as the syntax of a language, is modeled as a tree language. The particular formalisms used to describe these tree languages do not matter; they can be chosen freely. We call them the modules of the Millstream system. For example, a module may be a tree adjoining grammar, a finite-state tree automaton, a

[1] For introductions see, e.g., [PS94, Ste00, Dal01].

dependency grammar, a corpus, human input, etc. Given any number of such modules, the Millstream system links them by logical interfaces. The modules need not be of the same nature, since they are kept as individual units acting as "black boxes". The way in which the individual module works is of no interest for the Millstream system. As a consequence, our concept is able to capture derivational as well as non-derivational approaches. In fact, even though we generally assume the modules to define some kind of tree language, even this assumption is not crucial at all. For example, one could just as well consider modules that yield graphs.

The central component of a Millstream system is the *interface*. Suppose that a Millstream system has k modules. Roughly speaking, the interface consists of interface rules in the form of logical expressions that establish links between the (nodes of the) trees $t_1, \ldots, t_k$ that are generated by the individual modules. Thus, a valid combination of trees is not just any collection of trees $t_1, \ldots, t_k$ generated by the k modules. It also includes, between these structures, interconnecting links that represent their relationships and that must follow the rules expressed by the interface. Grammaticality, in terms of a Millstream system, means that the individual structures must be valid (as defined by the modules) and are linked in such a way that all interface rules are logically satisfied. A Millstream system can thus be considered to perform independent concurrent derivations of autonomous modules, enriched by an interface that establishes links between the outputs of the modules. In particular, only outputs that can correctly be linked represent a consistent overall result; the remaining ones are discarded.

To obtain a framework in which different types of interface specifications can be studied, we do not presuppose a particular logical formalism. Similar to the modules of a Millstream system, which may be of any sort, we allow to consider any type of logic, as long as it has a certain minimum expressiveness needed to make statements about trees and links in between them. As a consequence, it should be possible to translate formalisms such as HPSG, LFG, and CCG to specific types of Millstream systems. Millstream system could then be used to study what these theories have in common (and what distinguishes them), to establish theoretical results that hold for all of them, and to implement them.

We note here that the combination of logic and Formal Language Theory has a long and fruitful tradition in the Theory of Computation. An early result is the famous Bchi-Elgot-Trakhtenbrot Theorem that characterizes the regular languages using monadic second-order (MSO) logic [Büc60, Elg61, Tra62]. Later, this result was extended in numerous ways, for example to regular tree languages [TW68, Don70] and to string and tree transducers [EM99, EH01, EM03]. In the theory of graph grammars, MSO-definable graph languages and transductions play an immensely important role; see, e.g., [Cou90, Cou97]. See [Tho97] for an overview of the relation between automata theory and logic, and [Lib04] for a more general introduction to finite model theory.

To the best of our knowledge, Millstream systems are based on new ideas that cannot be found in the literature. Contemporary Formal Language Theory offers a few other models that allow us to combine several grammatical devices and make them "cooperate", such as *cooperating distributed grammar systems* and *parallel communicating grammar systems* [CVDKP94, DPR97, PS89, CVS98]. However, readers who are familiar with these notions will notice in the further course of this paper that Millstream systems differ quite fundamentally from these types of systems. See also Section 6 for a brief discussion of the differences.

In the future, we intend to study Millstream systems from a variety of different perspectives, including their application to problems in Computational Linguistics. The current paper is mainly devoted to their introduction and motivation. We hope to be able to convince the reader of the potential of Millstream systems by outlining, in a necessarily simplified example, a Millstream system that formalizes one particular instance (taken from [Jac97, Jac02, Jac07]) of an interface between the morphophonology, syntax, and semantics of the English language. In a separate section, we also discuss some observations related to Formal Language Theory that indicate interesting theoretical problems.

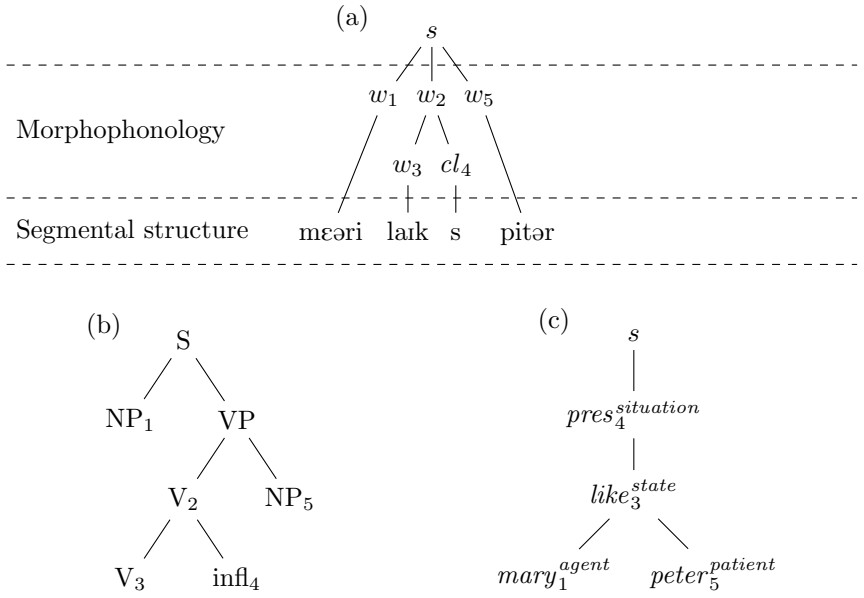

Figure 1: Phonological, syntactical and semantical structure of *Mary likes Peter*

We think that solutions of these problems would not only be of interest formal language theorists, but also for those whose focus is on applications in Computational Linguistics.

The remainder of the paper is organized as follows. We continue this introduction by giving a linguistical example which illustrates the non-hierarchical view of linguistic levels that determine the structure of a sentence. In Section 2, we provide the preliminaries and definitions in order to give the notational framework of this paper. In Section 3, we define Millstream systems and give a first example from Formal Language Theory. Section 4 contains the example mentioned above, formalizing the lexicon and the interface rules in [Jac97, Jac02, Jac07]. Section 5 contains further examples and remarks related to Formal Language Theory. Finally, Section 6 concludes the paper and sketches forthcoming mathematical issues to be addressed within the framework of Millstream systems.

As mentioned above, Millstream systems are inspired by the linguistic theories of Jackendoff and Sadock, and in particular their notion of interfaces. Our aim is to introduce Millstream systems as a flexible mathematical tool that allows us to formalize and implement theories of this kind, and to investigate their computational properties. In the rest of this section, we discuss an example that illustrates the linguistic ideas that have motivated our approach. However, the reader should carefully avoid identifying Millstream systems with the linguistical theories and setup used in this example, in which we mainly follow the presentation and terminology by Jackendoff [Jac02]. The only thing that is of real importance is the conceptual idea of having autonomously acting modules linked and constrained by interfaces.

Figure 1 shows the phonological, syntactical and semantical structure, depicted as trees (a), (b) and (c), respectively of the sentence *Mary likes Peter*. Trees are defined formally in the next section, for the time being we assume the reader to be familiar with the general notion of a tree as used in linguistics and computer science.

The segmental structure in the phonological tree (a) is the basic pronunciation of the sentence *Mary likes Peter*, where each symbol represents a speech sound. This string of speech sound symbols is structured into phonological words by morphophonolgy. The morphophonological structure in our example consists of the three full phonological words w_1, w_3, and w_5, namely mɛəri, laık and pitər, respectively. Notice that the inflection s is not a phonological word but a clitic. Clitis are attached to adjacent phonological words

in order to form larger phonological constituents, i.e. phonological words. The syntactical tree (b) divides the sentence S into a noun phrase NP_1 and a verb phrase VP. The verb phrase VP is divided into an inflected verb V_2 and a noun phrase NP_5. The inflected verb V_2 consists of its uninflected form V_3 and its inflection $infl_4$, which refers, in our example, to the grammatical features present tense and third person singular. Notice that words such as *Mary* and *likes* are not depicted in the syntactic tree as it is usually done when illustrating the syntactical structure of a natural language sentence in terms of trees; the underlying idea is that words are phonological categories, not syntactical ones. The semantical tree (c) depicts the semantical constituents. In our example, *like* is a function of type *state* and takes two arguments, namely *mary* and *peter* which are of type *agent* and *patient*, respectively. The structure of the sentence *Mary likes Peter* is not only the collection of its phonological, syntactical and semantical structures, but also includes the relationships between categories in these tree structures. These links between the trees (a), (b), and (c) are represented by the indices, equal indices meaning that the nodes correspond to each other (i.e., are linked). The morphophonological category w_1, for example, is coindexed with the noun phrase NP_1 in the syntactical tree and with the conceptual constituent $mary_1^{agent}$ in the semantical tree. This illustrates that w_1, NP_1, and $mary_1^{agent}$ are the corresponding morphophonological, syntactical and semantical representations of *Mary*, respectively. But there are also correspondences that concern only the phonological and syntactical trees, excluding the semantical tree. For example, the inflected word V_2 in the syntactical structure corresponds to the phonological word w_2, but has no link to the semantical structure whatsoever. These correspondences between the tree structures are established by the interface via interface rules. The three main interface rules proposed in [Jac97, Jac02, Jac07] are given below. Rule 1 links the phonological and syntactical structures and rules 2 and 3 link the syntactical and semantical structures:

1. The linear order of morphophonological units (the leaves of the morphophonological tree) corresponds to the linear order of the corresponding syntactical units.
2. A syntactic head corresponds to a semantic function, while its syntactic arguments correspond to the arguments of the semantic function.
3. The syntactic subject of a transitive verb corresponds to the first argument of a semantic function and the syntactic object corresponds to the second argument of that semantic function.

In our example, interface rule 1 is satisfied since the linearly ordered morphophonological words $w_1\, w_3\, cl_4\, w_5$ in tree (a) correspond to the linearly ordered syntactical words $NP_1\, V_3\, infl_4\, NP_5$ in tree (b). Interface rules 2 and 3 are satisfied, because the syntactic head V_3 in tree (b) corresponds to the semantic function $like_3^{state}$ in tree (c) and the syntactic arguments NP_1 and NP_5 of V_3 correspond respectively to the arguments $mary_1^{agent}$ and $peter_5^{patient}$ of $like_3^{state}$.

In fact, there is another, huge but finite, interface rule. It is represented by the lexicon, a storage of morphemes. A lexical entry for a morpheme typically consists of a phonological, a syntactical, and a semantical entry for that morpheme. Figure 2 depicts simplified lexical entries, as used in our example, for *Mary, Peter, like* and *s*, of which each has a phonological, syntactical and semantical entry. The phonological entry for a morpheme provides its pronunciation and its morphophonological structure. The semantical entry for *like*, for example, represents the semantical constituent *like* as a function that takes two (obligatory) arguments, namely the variables X and Y. The syntactical entry for *s* provides the syntactical structure for inflection to represent the third person singular in present tense. Note that most of the nodes in a lexical entry carry an index. These indices establish the correspondences between the phonological, syntactical and semantical constituents, when these are embedded in larger tree structures. Thus, the lexicon as a whole functions as a fourth interface rule linking the phonological, syntactical, and semantical

	Phonological entry	Syntactical entry	Semantical entry
Lexical entry for *Mary*	w_q | mɛəri	NP_q	$mary_q^{agent}$
Lexical entry for *Peter*	w_q | pitər	NP_q	$peter_q^{patient}$
Lexical entry for *like*	w_q | laɪk	V_q	$like_q^{state}$ / \ X Y
Lexical entry for *-s*	w_p / \ w_q cl_r | *-s*	V_p / \ V_q infl_r	$pres_r^{situation}$

Figure 2: Lexical entries for *Mary, Peter, like* and *s*.

structures.

As mentioned earlier, for a sentence to be grammatical, not only the interface rules must be satisfied. In addition, all its tree structures must individually be valid. For example, consider the ungrammatical sentence **Mary like Peter*. Since the subject-verb agreement is violated in this sentence, it has no valid syntactical structure. In other words, the syntactical module will not provide us with any tree representing this sentence. The sentence does, however, have a valid phonological structure, a tree t expressing that **Mary like Peter* consists of three full phonological words (see Figure 3). Hence, the phonological module, if considered on its own, may accept t as a valid phonological tree. Nevertheless, t will not occur in any correctly linked triple of phonological, syntactical, and semantical trees, simply because an appropriate syntactical counterpart is missing (even if a semantical might be found).

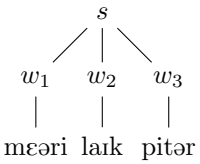

Figure 3: The phonological structure of **Mary like Peter*.

2 Mathematical Preliminaries

The set of non-negative integers is denoted by $\mathbb{N}$, and $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$. For $k \in \mathbb{N}$, we let $[k] = \{1, \ldots, k\}$. For a set S, the set of all nonempty finite sequences (or strings) over S is denoted by S^+; if the empty sequence ϵ is included, we write S^*. As usual, $A_1 \times \cdots \times A_k$ denotes the Cartesian product of sets $A_1, \ldots, A_k$. The transitive and reflexive closure of a binary relation $\Rightarrow \subseteq A \times A$ on a set A is denoted by $\Rightarrow^*$.

2.1 Trees and Tree Generators

A *ranked alphabet* is a finite set Σ of pairs (f, k), where f is a symbol and $k \in \mathbb{N}$ is its *rank*. We denote (f, k) by $f^{(k)}$, or simply by f if k is understood or of lesser importance.

Further, we let $\Sigma^{(k)} = \{f^{(n)} \in \Sigma \mid n = k\}$.

We define trees over Σ in one of the standard ways, by identifying the nodes of a tree t with sequences of natural numbers. Intuitively, such a sequence shows that path from the root of the tree to the node in question. In particular, the root is the empty sequence ϵ. As an example, the second child of the first child of the root would be the node 1 2, and its label in Σ would be $t(1\,2)$. The rank of this label is required to coincide with the number of children of the node.

Formall, the set T_Σ of trees over Σ consists of all mappings $t\colon V(t) \to \Sigma$ (called trees) with the following properties:

- The set $V(t)$, whose elements are the *nodes* of t, is a finite and non-empty prefix-closed subset of $\mathbb{N}_+^*$. Thus, for every node $vi \in V(t)$ (where $i \in \mathbb{N}_+$), its *parent* v is in $V(t)$ as well.
- For every node $v \in V(t)$, if $t(v) = f^{(k)}$, then $\{i \in \mathbb{N} \mid vi \in V(t)\} = [k]$. In other words, the *children* of v are $v1, \ldots, vk$.

Let $t \in \mathrm{T}_\Sigma$ be a tree. The *root* of t is the node ϵ. For every node $v \in V(t)$, the *subtree of t rooted at v* is denoted by t/v. It is defined by $V(t/v) = \{v' \in \mathbb{N}^* \mid vv' \in V(t)\}$ and, for all $v' \in V(t/v)$, $(t/v)(v') = t(vv')$. We shall denote a tree t as $f[t_1, \ldots, t_k]$ if $t(\epsilon) = f^{(k)}$ and $t/i = t_i$ for $i \in [k]$. In the special case where $k = 0$ (i.e., $V(t) = \{\epsilon\}$), the brackets may be omitted, thus denoting t as f. For a set S of trees, the set of all trees of the form $f[t_1, \ldots, t_k]$ such that $f^{(k)} \in \Sigma$ and $t_1, \ldots, t_k \in S$ is denoted by $\Sigma(S)$.

For a tuple $T \in \mathrm{T}_\Sigma^k$, we let $V(T)$ denote the set $\{(i, v) \mid i \in [k] \text{ and } v \in V(t_i)\}$. Thus, $V(T)$ is the disjoint union of the sets $V(t_i)$. Furthermore, we let $V(T, i)$ denote the ith component of this disjoint union, i.e., $V(T, i) = \{i\} \times V(t_i)$ for all $i \in [k]$.

A *tree language* is a subset of T_Σ, for a ranked alphabet Σ, and a *Σ-tree generator* (or simply *tree generator*) is any sort of formal device G that determines a tree language $L(G) \subseteq \mathrm{T}_\Sigma$. A typical sort of tree generator, which we will use in our examples, is the regular tree grammar.

Definition 2.1 (regular tree grammar [Bra69]) A *regular tree grammar* is a tuple $G = (N, \Sigma, R, S)$ consisting of

- disjoint ranked alphabets N and Σ of *nonterminals* and *terminals*, where $N = N^{(0)}$,
- a finite set R of rules of the form $A \to r$, where $A \in N$ and $r \in \mathrm{T}_{\Sigma \cup N}$, and
- an *initial nonterminal* $S \in N$.

Given trees $t, t' \in \mathrm{T}_{\Sigma \cup N}$, there is a *derivation step* $t \Rightarrow t'$ if t' is obtained from t by replacing a single occurrence of a nonterminal A with r, where $A \to r$ is a rule in R. The *regular tree language generated by G* is

$$L(G) = \{t \in \mathrm{T}_\Sigma \mid S \overset{*}{\Rightarrow} t\}.$$

It is well known that a string language L is context-free if and only if there is a regular tree language L', such that $L = \mathrm{yield}(L')$. Here, $\mathrm{yield}(L') = \{\mathrm{yield}(t) \mid t \in L'\}$ denotes the set of all yields of trees in L', the yield $\mathrm{yield}(t)$ of a tree t being the string obtained by reading its leaves from left to right.

Although we are going to use regular tree grammars in our illustrating examples, the reader should keep in mind that, in general, we use a very wide notion of tree generators. In particular, tree generators need not be grammatical devices. Any formalism that yields a set of trees may be used as a tree generator. This includes, for instance, automata, logical formulas, systems of equations, tree corpora, and, in fact, even human input.

2.2 Trees as Logical Structures

As mentioned in the introduction, the interface of a Millstream system (which will be defined in the next section) uses logical expressions for establishing relationships between the trees generated by the modules of the Millstream system. To make the trees generated by the individual modules accessible to logic, they have to be converted into logical structures. We now define such a logical representation of trees, which is fairly standard (see, e.g., [Lib04]).

Throughout the rest of this paper, let Λ denote any type of predicate logic that allows us to make use of n-ary predicates symbols. We indicate the arity of predicate symbols in the same way as the rank of symbols in ranked alphabets, i.e., by writing $P^{(n)}$ if P is a predicate symbol of arity n. The set of all well-formed formulas in Λ without free variables (i.e., the set of sentences of Λ) is denoted by F_Λ. If S is a set, we say that a predicate symbol $P^{(n)}$ is S-typed if it comes with an associated type $(s_1, \ldots, s_n) \in S^n$. We write $P\colon s_1 \times \cdots \times s_n$ to specify the type of P.

Recall that an n-ary predicate ψ on D is a function $\psi\colon D^n \to \{\text{true}, \text{false}\}$. Alternatively, ψ can be viewed as a subset of D^n, namely the set of all $(d_1, \ldots, d_n) \in D^n$ such that $\psi(d_1, \ldots, d_n) = \text{true}$. We will use both these views, selecting whichever is more convenient in a given situation. Given a (finite) set $\mathcal{P}$ of predicate symbols, a logical structure $\langle D; (\psi_P)_{P\in\mathcal{P}}\rangle$ consists of a set D called the domain and, for each $P^{(n)} \in \mathcal{P}$, a predicate $\psi_P \subseteq D^n$. If an existing structure Z is enriched with additional predicates $(\psi_P)_{P\in\mathcal{P}'}$ (where $\mathcal{P} \cap \mathcal{P}' = \emptyset$), we denote the resulting structure by $\langle Z; (\psi_P)_{P\in\mathcal{P}'}\rangle$. In this paper, we will only consider structures with finite domains.

To represent (tuples of) trees as logical structures, consider a ranked alphabet Σ, and let r be the maximum rank of symbols in Σ. A tuple $T = (t_1, \ldots, t_k) \in \mathrm{T}_\Sigma^k$ will be represented by the structure

$$|T| = \langle V(T); (V_i)_{i\in[k]}, (\mathrm{lab}_g)_{g\in\Sigma}, (\downarrow_i)_{i\in[r]}\rangle$$

consisting of the domain $V(T)$ and the predicates $V_i^{(1)}$ ($i \in [k]$), $\mathrm{lab}_g^{(1)}$ ($g \in \Sigma$) and $\downarrow_i^{(2)}$ ($i \in [r]$). The predicates are given as follows:

- For every $i \in [k]$, $V_i = V(T, i)$. Thus, $V_i(d)$ expresses that d is a node in t_i (or, to be precise, that d represents a node of t_i in the disjoint union $V(T)$).
- For every $g \in \Sigma$, $\mathrm{lab}_g = \{(i, v) \in V(T) \mid i \in [k] \text{ and } t_i(v) = g\}$. Thus, $\mathrm{lab}_g(d)$ expresses that the label of d is g.
- For every $j \in [r]$, $\downarrow_j = \{((i, v), (i, vj)) \mid i \in [k] \text{ and } v, vj \in V(t_i)\}$. Thus, $\downarrow_j(d, d')$ expresses that d' is the jth child of d in one of the trees $t_1, \ldots, t_k$. In the following, we write $d \downarrow_j d'$ instead of $\downarrow_j(d, d')$.

Note that, in the definition of $|T|$, we have blurred the distinction between predicate symbols and their interpretation as predicates, because this interpretation is fixed. In the following, especially in intuitive explanations, we shall sometimes also identify the logical structure $|T|$ with the tuple T it represents.

Example 2.2 As an example, let $\Sigma = \{f^{(2)}, g^{(2)}, f^{(1)}, a^{(0)}\}$, and consider the pair $T = (t_1, t_2)$ with $t_1 = g[f[a, g[a]]]$ and $t_2 = g[f[a], a]$. Then the domain of $|T|$ is the set $\{(1, \epsilon), (1, 1), (1, 11), (1, 12), (1, 121)\} \cup \{(2, \epsilon), (2, 1), (2, 11), (2, 2)\}$. With $x = (1, 12)$ and $y = (1, 121)$, predicates that hold are, e.g., $V_1(x)$, $\mathrm{lab}_g(x)$, and $x \downarrow_1 y$. Using ordinary first-order predicate logic, we can for instance express that, for every leaf in t_1 that is the first child of another node, there is a leaf in t_2 also being a first child, such that both parents carry the same label:

$$\begin{aligned}\forall x, y\colon\ & V_1(x) \wedge x \downarrow_1 y \wedge \mathrm{lab}_a(y)\\ & \to \exists x', y'\colon V_2(x') \wedge x' \downarrow_1 y' \wedge \mathrm{lab}_a(y') \wedge (\mathrm{lab}_f(x) \leftrightarrow \mathrm{lab}_f(x')).\end{aligned}$$

By choosing the second a in t_1 (i.e., the node $(1, 121)$) as y, one can check that this formula is *not* satisfied in $|T|$, because no a in t_2 is the first child of a g.

To end this section, let us note that the particular representation of trees as logical structures defined above is not specifically important. Any other reasonable representation could be used as a basis for Millstream systems, as introduced in the next section.

3 Millstream Systems

We are now going to define Millstream systems. For this, we first formalize our notion of interfaces. The idea is that a tuple $T = (t_1, \ldots, t_k)$ of trees, represented as $|T|$, is augmented with additional *interface links* that are subject to logical conditions. An interface may contain finitely many different kinds of interface links. Formally, the collection of all interface links of a given kind is viewed as a predicate. The names of the predicates are called interface symbols. Each interface symbol is given a type that indicates which trees it is intended to link with each other. For example, if we want to have ternary links called TIE, each linking a node of t_1 with a node of t_3 and a node of t_4, we use the interface symbol TIE: $1 \times 3 \times 4$. This interface symbol would then be interpreted as a predicate $\psi_{\text{TIE}} \subseteq V(T, 1) \times V(T, 3) \times V(T, 4)$. Each triple in ψ_{TIE} would thus be an interface link of type TIE that links a node in $V(t_1)$ with a node in $V(t_3)$ and a node in $V(t_4)$.

Definition 3.1 (interface) Let Σ be a ranked alphabet. An *interface on* T^k_Σ ($k \in \mathbb{N}$) is a pair $INT = (\mathcal{I}, \Phi)$, such that

- $\mathcal{I}$ is a finite set of $[k]$-typed predicate symbols called *interface symbols*, and
- Φ is a finite set of formulas in F_Λ that may, in addition to the fixed vocabulary of Λ, contain the predicate symbols in $\mathcal{I}$ and those occurring in the structures $|T|$ ($T \subseteq \mathrm{T}^k_\Sigma$). These formulas are called *interface conditions*.

A *well-formed configuration* (w.r.t. INT) is a structure $C = \langle |T|; (\psi_I)_{I \in \mathcal{I}} \rangle$, such that

- $T \subseteq \mathrm{T}^k_\Sigma$,
- for each I: $i_1 \times \cdots \times i_l$ in $\mathcal{I}$, $\psi_I \subseteq V(T, i_1) \times \cdots \times V(T, i_l)$, and
- C satisfies the interface conditions in Φ (if each symbol $I \in \Psi$ is interpreted as ψ_I).

Note that several interfaces can always be combined into one. For this, one first renames the interface symbols to avoid name conflicts. Afterwards, one simply takes the union of the sets of interface symbols and of the sets of interface conditions to obtain the combined interface.

We are now ready to state the definition of Millstream systems. Such a system consists of k tree generators, called the modules of the Millstream system, and an interface. The modules yield the "raw material", k-tuples of trees that are linked together by the interface (thereby sorting out those which cannot be linked as required).

Definition 3.2 (Millstream system) Let Σ be a ranked alphabet and $k \in \mathbb{N}$. A *Millstream system* (MS, for short) is a system $MS = (M_1, \ldots, M_k; INT)$ consisting of Σ-tree generators $M_1, \ldots, M_k$, called the *modules* of MS, and an interface INT on T^k_Σ. The language $L(MS)$ generated by MS is the set of all well-formed configurations $\langle |T|; (\psi_I)_{I \in \mathcal{I}} \rangle$ such that $T \in L(M_1) \times \cdots \times L(M_k)$.

Sometimes, it is convenient to have a notation for the tuples of trees in well-formed configurations that discards the links. Further, one may even want to consider only some of the trees in these tuples. For this, if MS is as above and $1 \leq i_1 < \cdots < i_l \leq k$, we define the notation

$$L^{M_{i_1} \times \cdots \times M_{i_l}}(MS) = \{(t_{i_1}, \ldots, t_{i_l}) \mid \langle |(t_1, \ldots, t_k); (\psi_I)_{I \in \mathcal{I}}| \rangle \in L(MS)\}.$$

Example 3.3 Let us discuss a Millstream system MS whose modules are two regular tree grammars, namely M_1 and M_2. If we consider the yields of the generated trees, M_1 describes the string language $\{a,b\}^+$, i.e., $\{yield(t) \mid t \in L(M_1)\} = \{a,b\}^+$, each tree in $L(M_1)$ being a simple left comb. The module M_2 is similar, except that the generated binary trees are right combs. In detail,

$$M_1 = (N, \Sigma, R_1, S) \quad \text{and} \quad M_2 = (N, \Sigma, R_2, S),$$

where $N = \{S\}$, $\Sigma = \{f^{(2)}, f^{(1)}, a^{(0)}, b^{(0)}\}$, and

$$\begin{aligned} R_1 &= \{S \to f[S,a], S \to f[S,b], S \to a, S \to b\}, \\ R_2 &= \{S \to f[a,S], S \to f[b,S], S \to a, S \to b\}. \end{aligned}$$

The interface of MS uses ordinary first-order predicate logic. The interface conditions imposed on the pairs of trees generated by M_1 and M_2 express that the trees should be mirror images of each other. This is achieved by using a single kind of interface link, given by the interface symbol MIRR: 1×2. Recall that the type of MIRR indicates that its links are binary, each linking a node of the first tree with one of the second. The interface conditions provide more specific conditions, telling us which collections of links are admissible. To ensure the mirror-image property, we use the following interface conditions.

- The root nodes of the two trees generated are linked:

  $$\forall x, y\colon root_1(x) \wedge root_2(y) \to \text{MIRR}(x,y).$$

  Here, we make use of the abbreviation $root_i(x) \equiv V_i(x) \wedge \nexists y\colon y \downarrow_1 x$ that expresses that x is the root of tree i.

- Only nodes that carry the same label are linked:

  $$\begin{aligned} \forall x, y\colon \text{MIRR}(x,y) \to\ & (\text{lab}_a(x) \wedge \text{lab}_a(y)) \\ & \vee (\text{lab}_b(x) \wedge \text{lab}_b(y)) \\ & \vee (\text{lab}_f(x) \wedge \text{lab}_f(y)). \end{aligned}$$

  Note that, together with the first formula, this implies that the root symbols of the trees are equal.

- If two binary nodes are linked, then the first child of the first is linked with the second child of the second, and vice versa:

  $$\begin{aligned} \forall x, y, c_1, c_2, c_1', c_2'\colon\ & \text{MIRR}(x,y) \wedge (x \downarrow_1 c_1) \wedge (x \downarrow_2 c_2) \wedge (y \downarrow_1 c_1') \wedge (y \downarrow_2 c_2') \\ & \to \text{MIRR}(c_1, c_2') \wedge \text{MIRR}(c_2, c_1'). \end{aligned}$$

What do the well-formed configurations in $L(MS)$ look like? By definition, they are the structures $\langle t_1, t_2; \psi_{\text{MIRR}}\rangle$, where $t_1 \in L(M_1)$, $t_2 \in L(M_2)$, and the links in $\psi_{\text{MIRR}} \subseteq V(t_1) \times V(t_2)$ satisfy the three formulas above.[2] One of these well-formed configurations is shown in Figure 4. Using the second and third interface conditions, it can easily be shown by induction that t_1/u is the mirror image of t_2/v for all $(u,v) \in \psi_{\text{MIRR}}$. Thus, by the first condition, it follows that $t_1 = t_1/\epsilon$ is the mirror image of $t_2 = t_2/\epsilon$. In other words, if we denote the mirror image of a tree t by $mirr(t)$, then

$$\begin{aligned} L^{M_1 \times M_2}(MS) &= \{(t_1, t_2) \in L(M_1) \times L(M_2) \mid t_1 = mirr(t_2)\} \\ &= \{(t, mirr(t)) \mid t \in L(M_1)\}. \end{aligned}$$

A few additional remarks may be instructive. First, note that the interface conditions given above do not ensure that the set of links between t_1 and t_2 is minimal. The reader may easily check that we may add arbitrary links between equally labeled leaves in t_1 and t_2 without violating the well-formedness of the configuration (owing to the fact that equally labeled leaves are mirror images of each other). If desired, this can be avoided

[2] Recall that we, as mentioned above, for better readability talk about $\langle t_1, t_2; \psi_{\text{MIRR}}\rangle$ rather than about $\langle |(t_1, t_2)|; \psi_{\text{MIRR}}\rangle$, as would formally be more correct. Consequently, we also view ψ_{MIRR} as a subset of $V(t_1) \times V(t_2)$.

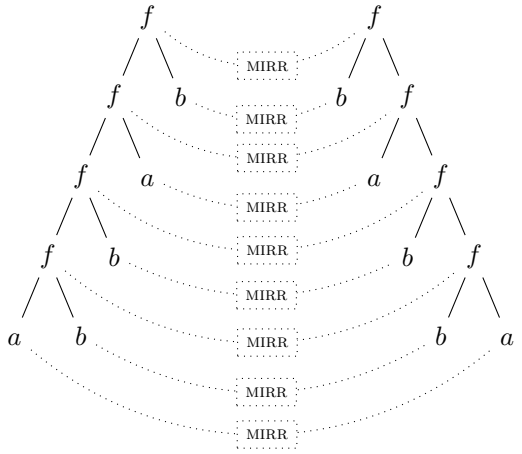

Figure 4: An element of $L(MS)$ in Example 3.3

by adding a fourth interface condition expressing the requirement that every node in t_1 is linked only once:

$$\forall x, y, y' \colon \textsc{mirr}(x, y) \wedge \textsc{mirr}(x, y') \rightarrow y = y'.$$

However, this solution assumes that Λ contains a built-in equality predicate. If we want to restrict ourselves to first-order logic without equality, the solution becomes slightly more complicated. We leave this as a small exercise to the reader.

Our second remark is that, while the equality $L^{M_i}(MS) = L(M_i)$ $(i \in [2])$ holds in this particular example, this is not true for Millstream systems in general. This is because, e.g., $L^{M_1}(MS)$ contains only those trees in $L(M_1)$ which, using an appropriate tree in $L(M_2)$ and an appropriate set of interface links between these trees, can be completed to a well-formed configuration. Note that this makes a lot of sense from a linguistical point of view. For example, if we are given syntactical and semantical modules M_1 and M_2, then a sentence may be syntactically correct (i.e., it is represented by a tree in $L(M_1)$), but is nevertheless ruled out if it does not have a reasonable semantical interpretation.

The third thing to be noted is that the interface ensures the mirror-image property not only for the combs generated by M_1 and M_2, but even for arbitrary pairs of trees over Σ. Thus, replacing M_1 and M_2 by any other modules M_1' and M_2' over Σ, the interface would "accept" only those pairs $(t_1, t_2) \in L(M_1') \times L(M_2')$ in which the tree t_1 is the mirror image of the tree t_2. In fact, since M_1 and M_2 generate trees of a very special structure, for these two modules the mirror-image property may be ensured by an interface using fewer links: the reader may wish to modify the interface conditions above in such a way that no f-labeled nodes but only leaves of the two trees are linked, yielding well-formed configurations as shown in Figure 5.

Finally, let us note that the interface used in this example is very strict in the sense that, for each tree $t_1 \in \mathrm{T}_\Sigma$, there is exactly one tree $t_2 \in \mathrm{T}_\Sigma$, such that t_1 and t_2 can be linked according to the interface conditions. In other words, in every well-formed configuration $\langle |(t_1, t_2)|; \psi_{\textsc{mirr}} \rangle$, the trees t_1 and t_2 determine each other uniquely. In general, interface conditions may, of course, tie the trees generated by the modules in a much weaker manner (or, in the extreme case, not at all).

The reader should note that, intentionally, Millstream systems are not a priori "generative". Even less so, they are "derivational" by nature. This is because there is no predefined notion of derivation that allows us to create well-formed configurations by means of a stepwise (though typically nondeterministic) procedure. In fact, there cannot be one, unless we make specific assumptions regarding the way in which the modules work, but also regarding the logic Λ and the form of the interface conditions that may be used.

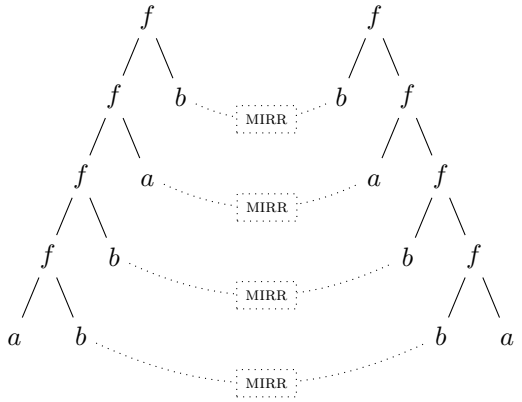

Figure 5: Alternative well-formed configurations that could have been used in Example 3.3

Similarly, as mentioned in the introduction, there is no predefined order of importance or priority among the modules. However, as the previous example illustrates, the situation may be different in specific cases. In the example, any of the two modules may be taken to be the "main" one. Intuitively, every derivation of this module extends in a unique manner to a derivation of a well-formed configuration. Whenever a rule is applied to a nonterminal of the main tree, this triggers the application of a rule to the corresponding nonterminal of the second tree. One of the key observations is that the correct links can be established immediately between the nodes in the right-hand sides of the rules that have been applied.

4 A Natural Language Example: Formalizing the Lexicon and Interface Rules

Let us now see how the interface between the morphophonological, syntactical, and semantical trees discussed in the introduction can be formalized using the notions introduced in the previous section. The discussion in this section focuses on the interface and disregards the modules; that is, we assume that suitable modules M_1, M_2, and M_3 are given, defining sets of linguistically acceptable morphophonological, syntactical, and semantical trees.

For the discussion below, we let Λ be first-order predicate logic enriched by a binary built-in predicate $\leq$ such that $u \leq v$ holds in $|T|$ if and only if u and v are nodes of the same tree in T and u occurs not later than v in the pre-order traversal of the tree.

The first question to be answered when turning the example discussed in the introduction into a Millstream system is which interface symbols one should use. At first sight, it may seem natural to use a ternary interface symbol CORRESPONDS: $1 \times 2 \times 3$. However, then *every* link would have to link three nodes, one from each tree. As we saw in the introduction, this is not always appropriate, since a node in the syntactical tree may correspond to a node in the phonological tree, whereas it does not have a corresponding node in the semantical tree. Therefore, we have to split the correspondences into two binary ones, using interface symbols MORPH: 1×2 and SEM: 2×3.

Let us now discuss the (formalization of the) lexicon. For this, consider a triple (t_1, t_2, t_3) consisting of a morphophonological, a syntactical, and a semantical tree. As discussed in the introduction, the lexicon is one of the mechanisms that help to establish appropriate links between t_1 and t_2 (of type MORPH) on the one hand, and t_2 and t_3 (of type SEM) on the other hand. Unfortunately, the linguistic literature that we are aware of does not provide mathematically precise definitions of how this happens. Therefore, one of the tasks of this section is to propose such a definition. Note, however, that it is not our goal to capture all possible forms, aspects, and uses of lexicons in Computational

Linguistics. Instead, we want to provide a reasonably neat formalization of the simple type of lexicon described in the introduction, for the purpose of illustrating the usefulness of interfaces in Millstream systems.

The most important questions our formalization has to answer is *where* in the trees lexicon entries are required to match and *how* matching is exactly defined. We answer these questions as follows.

1. Let us call a symbol in Σ an *anchor symbol* if it occurs at the root of one of the three components of some lexicon entry. Then, for every node $v_1 \in V(t_1)$ such that $t_1(V_1)$ an anchor symbol, one of the lexicon entries has to match at v_1, v_2, v_3, for some nodes $v_2 \in V(t_2)$ and $v_3 \in V(t_3)$. Similar requirements are imposed for each $v_2 \in V(t_2)$ and each $v_3 \in V(t_3)$ labeled by an anchor symbol. In other words, there must not be any nodes labeled by anchor nodes at which no lexicon entry matches.

2. Now, consider a set $\mathit{links} \subseteq V(t_1) \times V(t_2) \cup V(t_2) \times V(t_3)$ of links, as indicated by common indices in Figure 1. (Later, *links* will correspond to the union of the sets of MORPH and SEM links.) Given $v_1 \in V(t_1), v_2 \in V(t_2), v_3 \in V(t_3)$, we say that a lexicon entry (l_1, l_2, l_3) *matches* at v_1, v_2, v_3 if the following hold.

   (a) For each $i \in [3]$, the variables in l_i (if any; cf. the semantical entry for *like* in Figure 2, which contains the variables X and Y) can be substituted in such a way that l_i becomes t_i/v_i up to the indices of nodes in l_i.
   Intuitively, this means that the subtree of t_i rooted at v_i has the structure and labeling given by l_i.

   (b) For all $\{i, j\} \in \{\{1, 2\}, \{2, 3\}\}$ and every node $v_i' \in V(l_i)$, there is a link $(v_i v_i', u) \in \mathit{links}$ for exactly those $u \in V(t_j)$ that can be written as $u = v_j v_j'$ for a node $v_j' \in V(l_j)$ carrying the same index as v_i'.
   Thus, intuitively, the images of the nodes of l_1, l_2, l_3 in t_1, t_2, t_3 carry exactly those links which are indicated by the indices in l_1, l_2, l_3. In particular, there must not exist any additional links connecting a node in the image of l_i with nodes outside those images.

We are now going to express the conditions above as first-order formulas. In these formulas, we use the standard abbreviations $\bigwedge_{i \in I} \varphi_i$ and $\bigvee_{i \in I} \varphi_i$ to denote the conjunction and disjunction, resp., of finitely many formulas φ_i. Here, I is supposed to be a finite set and φ_i is a formula for every $i \in I$. For instance, if $I = \{a, b, c\}$, then $\bigwedge_{i \in I} \varphi_i$ stands for $\varphi_a \wedge \varphi_b \wedge \varphi_c$. In the special case where $I = \emptyset$, we let $\bigwedge_{i \in I} \varphi_i$ and $\bigvee_{i \in I} \varphi_i$ stand for true and false, resp.

To make formulas more readable, we shall also frequently give names to certain subformulas that we define separately (using italic font to distinguish these abbreviations from our fixed vocabulary). In fact, we have already made use of this technique in Example 3.3, where we used the abbreviation $\mathit{root}_i(x)$.

Now, let $\Sigma' \subseteq \Sigma$ be the set of anchor symbols of the lexicon (see the first item above) and let LEX be the (finite) set of all lexicon entries. Then the lexicon is formalized by the interface condition

$$\forall x\colon \mathit{anchor}(x) \to \exists y, z : \bigvee_{\mathit{lex} \in \mathrm{LEX}} (\mathit{match}_{\mathit{lex}}(x, y, z) \vee \mathit{match}_{\mathit{lex}}(y, x, z) \vee \mathit{match}_{\mathit{lex}}(y, z, x)).$$

Here, $\mathit{anchor}(x) \equiv \bigvee_{\sigma \in \Sigma'} \mathrm{lab}_\Sigma(x)$ expresses that x is labeled by one of the anchor symbols, and $\mathit{match}_{\mathit{lex}}(x, y, z)$ is a formula (defined below) expressing that *lex* matches at the nodes x, y, and z.

For the definition of $\mathit{match}_{\mathit{lex}}(x, y, z)$, let VAR be the set of all variables occurring in

the lexicon. For $(l_1, l_2, l_3) \in \text{LEX}$, the formula $match_{(l_1,l_2,l_3)}(x, y, z)$ is given as follows:

$$
\begin{aligned}
&match_{(l_1,l_2,l_3)}(x_1, x_2, x_3) \equiv \\
&\quad V_1(x_1) \wedge V_2(x_2) \wedge V_3(x_3) \\
&\quad \wedge \bigwedge_{i \in [3]} \bigwedge_{\substack{v \in N(l_i) \\ l_i(v) \notin VAR}} symb_{i,v,l_i(v)}(x_i) \\
&\quad \wedge \bigwedge_{\substack{i \in [3] \\ v_i \in N(l_i)}} \forall x, y \colon (path_{v_i}(x_i, x) \wedge links(x, y)) \leftrightarrow \bigvee_{\substack{j \in [3] \setminus \{i\} \\ v_j \in N(l_j)}} (path_{v_j}(x_j, y) \wedge eq\_ind_{l_i,l_j,v_i,v_j})
\end{aligned}
$$

Here, a couple of subformulas are used:

- $symb_{i,v,\sigma}(x_i)$ expresses that the node $x_i v$ is in t_i and is labeled by σ. This is easily expressed. For example, if $i = 1, v = 1\,2$, and $\sigma = cl$, then
$$symb_{i,v,\sigma}(x_1) \equiv \exists z, z' : (x_1 \downarrow_1 z) \wedge (z \downarrow_2 z') \wedge lab_{Cl}(z').$$
- $links(x, y)$ expresses that x and y are linked:
$$links(x, y) \equiv \text{MORPH}(x, y) \vee \text{MORPH}(y, x) \vee \text{SEM}(x, y) \vee \text{SEM}(y, x).$$
- $path_{v_i}(x_i, x)$ expresses that, if we start at x_i and follow the path determined by v_i, then we end up at x. Again, this just requires to write a formula that states that there exist $|v_i| - 1$ nodes leading from x_i to x as given by v_i at hand. For instance, for $v_i = 2\,3\,1$,
$$path_{v_i}(x_i, x) \equiv \exists z, z' : (x_i \downarrow_2 z) \wedge (z \downarrow_3 z') \wedge (z' \downarrow_1 x).$$
In the special case where $v_i = \epsilon$, we set $path_{v_i}(x_i, x) \equiv x_i = x$. (The reader might wish to think about expressing this without the use of equality.)
- Finally, $same\_index_{l_i,l_j,v_i,v_j}$ expresses that the nodes v_i and v_j of l_i and l_j, respectively, carry the same index. Thus, the formula equal to the constant true if this is the case (for the given choice of l_i, l_j, v_i and v_j) and is equal to false otherwise.

The lexicon alone does not suffice to determine the correct correspondence between (morpho-)phonology, syntax, and semantics. One aspect that is neglected is order. This can be seen, for example, by noticing that the indices 1 and 6 in Figure 1 could be interchanged. The lexicon would still approve the configuration, although *Peter* and *Mary* have been mixed up. Similar, but somewhat more complicated problems occur when looking at the correspondence between syntax and semantics. The approach suggested in [Jac02] addresses these issues by means of the interface rules 1, 2, and 3 mentioned in the introduction. Below, we turn them into interface conditions to be added to our interface. We repeat the interface rules for reading convenience, and combine rules 2 and 3 into one.

Interface rule 1: The linear order of morphophonological units (the leaves of the morphophonological tree) corresponds to the linear order of the corresponding syntactical units.

$$\forall x, y, x', y' \colon m\_unit(x) \wedge m\_unit(y) \wedge \text{MORPH}(x, x') \wedge \text{MORPH}(y, y') \wedge x \leq y \rightarrow x' \leq y'.$$

Here, the auxiliary formula $m\_unit(x)$ expresses that x is a morphological unit, meaning that it is the parent of a leaf of the phonological tree (as the leaves of this tree correspond to the segmental structure). In the interface condition above, there is no need for $m\_unit(x)$ and $m\_unit(y)$ to check that x and y are nodes of the phonological tree, as this is already guaranteed by $\text{MORPH}(x, x') \wedge \text{MORPH}(y, y')$. Thus, it suffices to define

$$m\_unit(x) \equiv \exists x' \colon x \downarrow_1 x' \wedge \nexists x'' \colon x' \downarrow_1 x''.$$

Interface rules 2 and 3: A syntactic head corresponds to a semantic function, while its syntactic arguments correspond to the arguments of the semantic function. If the verb is

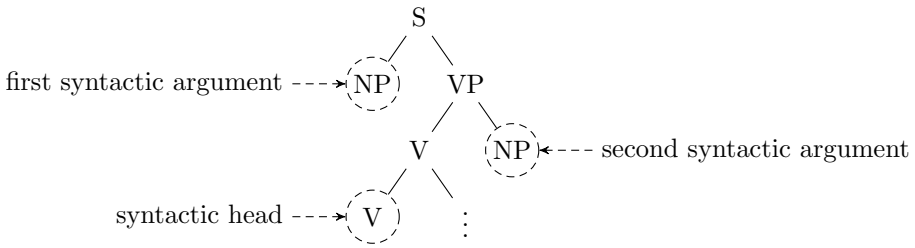

Figure 6: The pattern relating a syntactic head to its syntactic arguments

transitive, its syntactic subject corresponds to the first argument of a semantic function and its syntactic object corresponds to the second argument of that semantic function.

Below, we only consider the case where the syntactic head is a transitive verb in active voice. Note that the distinction between the transitive and intransitive cases is easy, because (our formalization of) the lexicon ensures that a transitive verb is linked, via a SEM-link, with a binary semantic function, whereas an intransitive verb is linked with a unary semantic function. Thus, it suffices to distinguish between unary and binary semantic functions.

In our interface condition, we have to identify a syntactic head together with its syntactic arguments and then make sure that the SEM-links satisfy the rule(s) above. For the case of transitive verbs in active voice, an instance of the pattern we have to look for is found in Figure 6.

Thus, we have to look for nodes x that is labeled by V (our syntactic head) and linked to a node x' that is labeled by a binary semantic function. Then we have to check that the children of x' are linked with the syntactic arguments of x as in Figure 6. This yields the interface condition

$$\begin{aligned}\forall x, x', x'_1, x'_2\colon\ & \mathrm{lab}_{\mathrm{V}}(x) \wedge \textsc{sem}(x, x') \wedge x' \downarrow_1 x'_1 \wedge x' \downarrow_2 x'_2 \\ & \rightarrow \exists x_1, x_2\colon \bigwedge_{i\in[2]} \mathrm{lab}_{\mathrm{NP}}(x_i) \wedge \textsc{sem}(x_i, x'_i) \wedge \mathit{syn\_args}(x, x_1, x_2),\end{aligned}$$

where

$$\begin{aligned}\mathit{syn\_args}(x, x_1, x_2) \equiv \exists y, y_1, y_2\colon & \mathrm{lab}_{\mathrm{S}}(y) \wedge \mathrm{lab}_{\mathrm{VP}}(y_1) \wedge \mathrm{lab}_{\mathrm{V}}(y_2) \\ & \wedge y \downarrow_1 x_1 \wedge y \downarrow_2 y_1 \wedge y_1 \downarrow_1 y_2 \wedge y_2 \downarrow_1 x \wedge y_1 \downarrow_2 x_2\end{aligned}$$

To finish this discussion, let us extend the example by a genitive construction: *Jill's daughter likes Peter*, which has the syntax tree displayed in Figure 7. Here, the genitive

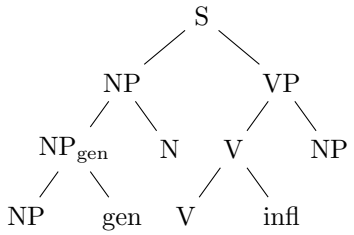

Figure 7: The syntax tree of *Jill's daughter likes Peter*

case turns N into a syntactic head corresponding to a unary semantic function (*the daughter of*) whose argument is given by the NP (*Jill*). The corresponding variant of the interface condition above looks is

$$\begin{aligned}\forall x, x', x'_1\colon\ & \mathrm{lab}_{\mathrm{N}}(x) \wedge \textsc{sem}(x, x') \wedge x' \downarrow_1 x'_1 \\ & \rightarrow \exists x_1\colon \mathrm{lab}_{\mathrm{NP}}(x_1) \wedge \textsc{sem}(x_1, x'_1) \wedge \mathit{syn\_arg}(x, x_1),\end{aligned}$$

where $\mathit{syn\_arg}(x, x_1) \equiv \exists y, y_1\colon \mathrm{lab}_{\mathrm{NP}}(y) \wedge \mathrm{lab}_{\mathrm{NP_{gen}}}(y_1) \wedge y \downarrow_1 y_1 \wedge y \downarrow_2 x \wedge y_1 \downarrow_1 x_1$.

5 Further Examples and Remarks Related to Formal Language Theory

The purpose of this section is to indicate, by means of examples and easy observations, that Millstream systems are not only linguistically well motivated, but also worth studying from the point of view of computer science, most notably regarding their algorithmic and language-theoretic properties. While this kind of study is beyond the scope of the current article, part of our future research on Millstream systems will be devoted to such questions. Our hope and expectation is that, supported by the results of these theoretical studies, but also by practical implementations, Millstream systems will turn out to be a valuable tool for understanding, formalizing, and solving problems in natural language processing.

Let us start by a simple example in the style of Example 3.3.

Example 5.1 Again, let Λ be ordinary first-order logic with equality. Consider the Millstream system MS over the ranked alphabet $\Sigma = \{\circ^{(2)}, a^{(0)}, b^{(0)}, c^{(0)}, d^{(0)}\}$ that consists of

- two identical modules $M_1 = M_2$ that simply generate T_Σ (e.g., using the regular tree grammar with the single nonterminal S and the rules[3] $S \to \circ[S, S] \mid a \mid b \mid c \mid d$) and
- a single interface symbol $\textsc{bij}\colon 1 \times 2$ with the interface conditions

$$\begin{aligned}
&\forall x\colon \mathrm{lab}_{\{a,b,c,d\}}(x) \leftrightarrow \exists y\colon \textsc{bij}(x,y) \vee \textsc{bij}(y,x),\\
&\forall x,y,z\colon (\textsc{bij}(x,y) \wedge \textsc{bij}(x,z) \vee \textsc{bij}(y,x) \wedge \textsc{bij}(z,x)) \to y = z,\\
&\forall x,y\colon \textsc{bij}(x,y) \to \bigvee_{z\in\{a,b,c,d\}} (\mathrm{lab}_z(x) \wedge \mathrm{lab}_z(y)).
\end{aligned}$$

The first interface condition expresses that all and only the leaves of both trees are linked. The second expresses that no leaf is linked with two or more leaves. In effect, this amounts to saying that BIJ is a bijection between the leaves of the two trees. The third interface condition expresses that this bijection is label preserving. Altogether, this amounts to saying that the yields of the two trees are permutations of each other; see Figure 8.

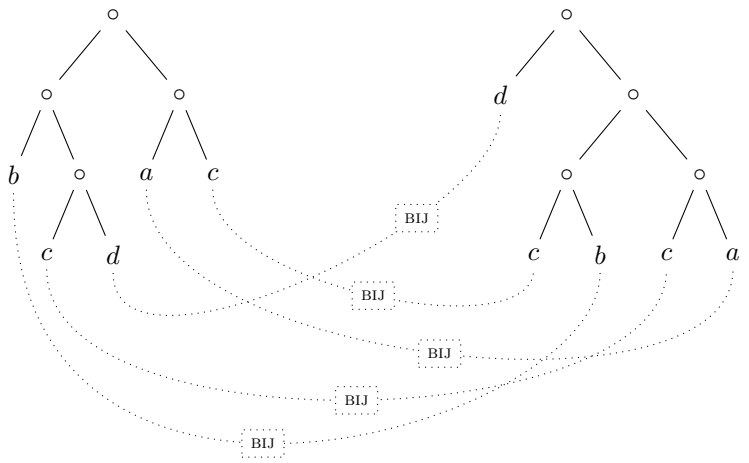

Figure 8: An element of $L(MS)$ in Example 5.1

Now, let us replace the modules by slightly more interesting ones. For a string w over $\{A, B, a, b, c, d\}$, let $\underline{w}$ denote any tree over $\{\circ^{(2)}, A^{(0)}, B^{(0)}, a^{(0)}, b^{(0)}, c^{(0)}, d^{(0)}\}$ such that $\mathrm{yield}(\underline{w}) = w$. (For example, we may choose $\underline{w}$ to be the left comb whose leaf symbols are

[3] As usual, $A \to r \mid r'$ stands for $A \to r,\ A \to r'$.

given by w.) Let the Millstream system MS' be defined as MS, but using the modules $M_1' = (\{A, B, C, D\}, \Sigma, R_1, A)$ and $M_2' = (\{A, B\}, \Sigma, R_2, A)$ with the following rules:

$$\begin{array}{lcl} R_1' & = & \{A \to \underline{aA} \mid \underline{aB},\ B \to \underline{bB} \mid \underline{bC},\ C \to \underline{cC} \mid \underline{cD},\ D \to \underline{dD} \mid \underline{d}\}, \\ R_2' & = & \{A \to \underline{acA} \mid \underline{acB},\ B \to \underline{bdB} \mid \underline{bd}\}. \end{array}$$

Thus, M_1' and M_2' are the "standard" grammars (written as regular tree grammars) that generate the regular languages $\{a^k b^l c^m d^n \mid k, l, m, n \geq 1\}$ and $\{(ac)^m (bd)^n \mid m, n \geq 1\}$. The interface makes sure that $L^{M_1' \times M_2'}(MS')$ contains only those pairs of trees t_1, t_2 in which $\text{yield}(t_1)$ is a permutation of $\text{yield}(t_2)$. As a consequence, it follows that $\text{yield}(L^{M_1'}(MS)) = \{a^m b^n c^m d^n \mid m, n \geq 1\}$.

The next example discusses how top-down tree transductions can be implemented as Millstream systems.

Example 5.2 (top-down tree transduction) Recall that a *tree transduction* is a binary relation $\tau \subseteq \mathrm{T}_\Sigma \times \mathrm{T}_{\Sigma'}$, where Σ and Σ' are ranked alphabets. The set of trees that a tree $t \in \mathrm{T}_\Sigma$ is transformed into is given by $\tau(t) = \{t' \in \mathrm{T}_{\Sigma'} \mid (t, t') \in \tau\}$. Obviously, every Millstream system of the form $MS = (M_1, M_2; INT)$ defines a tree transduction, namely $L^{M_1 \times M_2}(MS)$.

Let us consider a very simple instance of a deterministic top-down tree transduction τ (see, e.g., [GS84, GS97, FV98, Dre06, CDG+07] for definitions and references regarding top-down tree transductions), where $\Sigma = \Sigma' = \{f^{(2)}, g^{(2)}, a^{(0)}\}$. We transform a tree $t \in \mathrm{T}_\Sigma$ into the tree obtained from t by interchanging the subtrees of all top-most fs (i.e., of all nodes that are labeled with f and do not have a predecessor that is labeled with f as well) and turning the f at hand into a g. To accomplish this, a top-down tree transducer would use two states, say SWAP and COPY to traverse the input tree from the top down, starting in state SWAP. Whenever an f is reached in this state, its subtrees are interchanged and the traversal continues in parallel on each of the subtrees in state COPY. The only purpose of this state is to copy the input to the output without changing it. Formally, this would be expressed by the following term rewrite rules, viewing the states as symbols of rank 1:

$$\begin{array}{rcl} \textsc{swap}[f[x_1, x_2]] & \to & g[\textsc{copy}[x_2], \textsc{copy}[x_1]], \\ \textsc{copy}[f[x_1, x_2]] & \to & f[\textsc{copy}[x_1], \textsc{copy}[x_2]], \\ \textsc{swap}[g[x_1, x_2]] & \to & g[\textsc{swap}[x_1], \textsc{swap}[x_2]], \\ \textsc{copy}[g[x_1, x_2]] & \to & g[\textsc{copy}[x_1], \textsc{copy}[x_2]], \\ \textsc{swap}[a] & \to & a, \\ \textsc{copy}[a] & \to & a. \end{array}$$

(We hope that these rules are intuitive enough to be understood even by readers who are unfamiliar with top-down tree transducers, as giving the formal definition of top-down tree transducers would be out of the scope of this article.)

We mimic the behaviour of the top-down tree transducer using a Millstream system with interface symbols SWAP: 1×2 and COPY: 1×2. Since the modules simply generate T_Σ, they are not explicitly discussed. The idea behind the interface is that an interface link labeled $q \in \{\textsc{swap}, \textsc{copy}\}$ links a node v in the input tree with a node v' in the output tree if the simulated computation of the tree transducer reaches v in state q, resulting in node v' in the output tree.

First, we specify that the initial state is SWAP, which simply means that the roots of the two trees are linked by a SWAP link:

$$\forall x, y\colon \mathit{root}_1(x) \wedge \mathit{root}_2(y) \to \textsc{swap}(x, y),$$

where root_i is defined as in Example 3.3. The next interface condition corresponds to the first rule of the simulated top-down tree transducer:

$$\begin{array}{l} \forall x, y, x_1, x_2\colon \textsc{swap}(x, y) \wedge \mathrm{lab}_f(x) \wedge x \downarrow_1 x_1 \wedge x \downarrow_2 x_2 \\ \qquad \to \mathrm{lab}_g(y) \wedge \exists y_1, y_2\colon y \downarrow_1 y_1 \wedge y \downarrow_2 y_2 \wedge \textsc{copy}(x_1, y_2) \wedge \textsc{copy}(x_2, y_1). \end{array}$$

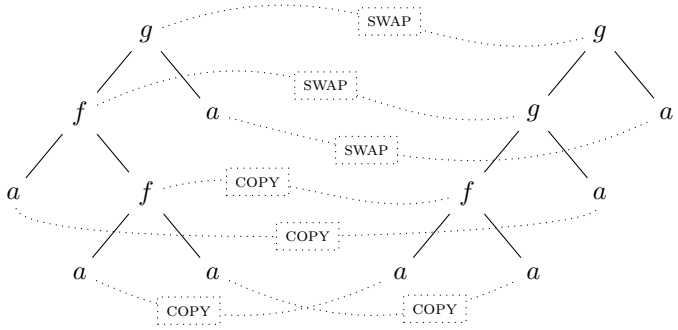

Figure 9: An element of $L(MS)$ in Example 5.2

In a similar way, the remaining rules are turned into interface conditions:

$$\begin{aligned}
&\forall x, y, x_1, x_2 \colon \textsc{copy}(x, y) \wedge \mathrm{lab}_f(x) \wedge x \downarrow_1 x_1 \wedge x \downarrow_2 x_2 \\
&\qquad \rightarrow \mathrm{lab}_f(y) \wedge \exists y_1, y_2 \colon y \downarrow_1 y_1 \wedge y \downarrow_2 y_2 \wedge \textsc{copy}(x_1, y_1) \wedge \textsc{copy}(x_2, y_2), \\
&\forall x, y, x_1, x_2 \colon \textsc{swap}(x, y) \wedge \mathrm{lab}_g(x) \wedge x \downarrow_1 x_1 \wedge x \downarrow_2 x_2 \\
&\qquad \rightarrow \mathrm{lab}_g(y) \wedge \exists y_1, y_2 \colon y \downarrow_1 y_1 \wedge y \downarrow_2 y_2 \wedge \textsc{swap}(x_1, y_1) \wedge \textsc{swap}(x_2, y_2), \\
&\forall x, y, x_1, x_2 \colon \textsc{copy}(x, y) \wedge \mathrm{lab}_g(x) \wedge x \downarrow_1 x_1 \wedge x \downarrow_2 x_2 \\
&\qquad \rightarrow \mathrm{lab}_g(y) \wedge \exists y_1, y_2 \colon y \downarrow_1 y_1 \wedge y \downarrow_2 y_2 \wedge \textsc{copy}(x_1, y_1) \wedge \textsc{copy}(x_2, y_2), \\
&\forall x, y \colon \textsc{swap}(x, y) \wedge \mathrm{lab}_a(x) \rightarrow \mathrm{lab}_a(y), \\
&\forall x, y \colon \textsc{copy}(x, y) \wedge \mathrm{lab}_a(x) \rightarrow \mathrm{lab}_a(y).
\end{aligned}$$

One of the elements of $L(MS)$ is shown in Figure 9. It should not be difficult to see that, indeed, $L^{M_1 \times M_2}(MS) = \tau$.

Extending the previous example, one can easily see that all top-down and bottom-up tree transductions can be turned into Millstream systems in a natural way. A similar remark holds for many other types of tree transductions known from the literature. Most notably, monadic second-order definable tree transductions (see the references mentioned in the introduction) can be expressed as Millstream systems. Since the mentioned types of tree transductions are well studied, and much is known about their algorithmic properties, future research on Millstream systems should investigate the relationship between different types of tree transductions and Millstream systems in detail. In particular, it should be tried to formulate requirements on the interface conditions that can be used to obtain characterizations of various classes of tree transductions.

We note here that results of this type would be interesting not only from a purely mathematical point of view, but also for applications in natural language processing. For example, tree transducers have turned out to be a valuable tool in machine translation [KG05, MK06, GKM08] and may also play an important role in many other areas of natural language processing. For example, in the setting of Section 4, one may say that natural language understanding is concerned with the interface between the syntactical and semantical components of the Millstream system sketched in that section. The goal of natural language understanding can then be expressed as follows: Given a tree generated by the syntactical module, construct a semantic tree and suitable interface links, such that a well-formed configuration is obtained.[4] In practice, rather than trying to solve this problem in one step, one would first simplify the syntactical tree by applying transformations that turn it into some kind of normal form. This can nicely be formalized by a Millstream system by adding, in between the syntactical module M_{syn} and the semantical module M_{sem}, a number of intermediary modules $M_1, \ldots, M_k$, such that $L^{M_{i-1} \times M_i}(MS)$ is a simplifying tree transduction for every $i \in [k]$ (where $M_0 = M_{\mathrm{syn}}$). Then the original problem has been reduced to handling the step from M_k to M_{sem}. A nice side effect

[4]Depending on the situation, one may or may not decide to neglect other components, such as the morphophonological one.

of using a Millstream system for this purpose is that the relation between the different steps is explictly available, represented by the interface links in the resulting well-formed configurations.

6 Conclusions

Millstream systems, as introduced in this article, are formal devices that allow to model situations in which several tree-generating modules are interconnected by logical interfaces. As mentioned in the introduction, the most prominent models of Formal Language Theory that also allow to define systems of grammatical devices working together are *cooperating distributed grammar systems* and *parallel cooperating grammar systems* (CD and PC grammar systems, respectively).

There are several fundamental differences between grammar systems (of either type) and Millstream systems. The modules in a Millstream system are viewed as black boxes that do not need to be grammatical formalisms. In fact, even human input could be used as a module. In contrast, the definition of grammar systems relies on the availability of a derivation relation, which is used to define a derivation relation of the system as a whole. For this, a notion of communication between the component grammars is used. Thus, in contrast to Millstream systems, the interaction of the components is interwoven with their derivation. Both types of grammar systems generate languages whose elements are strings. In the case of CD grammar systems, the individual grammars work on a common string that, in the end, becomes the string generated. In a PC grammar system, the grammars work on their own local string each, but there is a "master component" managing the "master string". In contrast, Millstream systems keep the trees generated by their modules as separate entities (augmented with links as specified in the interface).

Despite these differences (or rather: because of them), future work should try to establish formal relationships between grammar systems and suitably restricted versions of Millstream systems. The same holds for other types of systems known from Formal Language Theory. In particular, it could be very interesting to try to identify types of Millstream systems that correspond to monadic second-order definable tree transductions.

Future work should also investigate questions of the kind indicated at the end of Section 3. This includes properties that make it possible to obtain well-formed configurations in a generative way and algorithms to complete an incomplete configuration in such a way that the resulting configuration belongs to $L(MS)$. In fact, the second question, which can be regarded as a generalization of the first, is of central importance. Possible applications include natural language understanding (where, e.g., a syntactic tree is given and an appropriate semantical tree is sought; see the previous section) and language generation (where, conversely, a semantical tree is given and a corresponding syntax and/or phonological tree is sought).

Certainly, much more research is necessary to firmly establish the usefulness of Millstream systems from the point of view of Computational Linguistics and natural language processing. One of the things needed is an efficient implementation of Millstream systems, in order to be able to experiment with nontrivial examples. While this work should, to the extent possible, be application independent, it will also be necessary to seriously attempt to formalize and implement linguistical theories as Millstream systems. This includes exploring various such theories with respect to their appropriateness. While the example in Section 4 indicates one possible direction this may take, it includes far too little detail to be conclusive. Nevertheless, we hope that the example has convinced the reader of the potential of Millstream systems for formalizing and, eventually, implementing linguistical theories.

In the introduction, it was mentioned that it should be possible to translate formalisms such as HPSG, LFG, and CCG into Millstream systems. Such a translation would give rise to Millstream systems similar to the example in Section 4, but much more elaborate, including specific types of modules. To gain further insight into the usefulness and limita-

tions of Millstream systems for Computational Linguistics, future work should work out this translation in detail.

Acknowledgment We thank Dot and Danie van der Walt for providing us with a calm and relaxed atmosphere at Millstream Guest House in Stellenbosch (South Africa), where the first ideas around Millstream systems were born in Spring 2009.

References

[Bra69] Walter S. Brainerd. Tree generating regular systems. *Information and Control*, 14:217–231, 1969.

[Büc60] Richard J. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

[CDG+07] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Christof Löding, Denis Lugiez, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications*. Internet publication available at `http://tata.gforge.inria.fr`, 2007. Release October 2007.

[Cou90] Bruno Courcelle. Graph rewriting: an algebraic and logic approach. In J. van Leuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 193–242. Elsevier, Amsterdam, 1990.

[Cou97] Bruno Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations*, chapter 5, pages 313–400. World Scientific, Singapore, 1997.

[CVDKP94] Erzsébet Csuhaj-Varjú, Jürgen Dassow, Jozef Kelemen, and Gheorghe Păun. *Grammar Systems: a Grammatical Approach to Distribution and Cooperation*, volume 5 of *Topics in Computer Mathematics*. Gordon and Breach, 1994.

[CVS98] Erzsébet Csuhaj-Varjú and Arto Salomaa. Networks of language processors: Parallel communicating systems. *Bulletin of the EATCS*, 66:122–138, 1998.

[Dal01] Mary Dalrymple. *Lexical Functional Grammar*, volume 34 of *Syntax and Semantics*. Academic Press, 2001.

[Don70] John E. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4:406–451, 1970.

[DPR97] Jürgen Dassow, Gheorghe Păun, and Grzegorz Rozenberg. Grammar systems. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*. Vol. 2: *Linear Modeling: Background and Application*, chapter 4, pages 155–213. Springer, 1997.

[Dre06] Frank Drewes. *Grammatical Picture Generation – A Tree-Based Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[EH01] Joost Engelfriet and Henrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic*, 2:216–254, 2001.

[Elg61] Calvin C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the AMS*, 98:21–52, 1961.

[EM99] Joost Engelfriet and Sebastian Maneth. Macro tree transducers, attribute grammars, and MSO definable tree translations. *Information and Computation*, 154:34–91, 1999.

[EM03] Joost Engelfriet and Sebastian Maneth. Macro tree translations of linear size increase are MSO definable. *SIAM Journal on Computing*, 32:950–1006, 2003.

[FV98] Zoltán Fülöp and Heiko Vogler. *Syntax-Directed Semantics: Formal Models Based on Tree Transducers*. Springer, 1998.

[GKM08] Jonathan Graehl, Kevin Knight, and Jonathan May. Training tree transducers. *Computational Linguistics*, 34(3):391–427, 2008.

[GS84] Ferenc Gécseg and Magnus Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.

[GS97] Ferenc Gécseg and Magnus Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*. Vol. 3: *Beyond Words*, chapter 1, pages 1–68. Springer, 1997.

[Jac97] Ray Jackendoff. *The Architecture of the Language Faculty*. The MIT Press, Cambridge, Massachusetts, 1997.

[Jac02] Ray Jackendoff. *Foundations of Language: Brain, Meaning, Grammar, Evolution*. Oxford University Press, Oxford, 2002.

[Jac07] Ray Jackendoff. *Language, consciousness, culture: essays on mental structure*. The MIT Press, Cambridge, Massachusetts, 2007.

[KG05] Kevin Knight and Jonathan Graehl. An overview of probabilistic tree transducers for natural language processing. In Alexander F. Gelbukh, editor, *Proc. 6th Intl. Conf. on Computational Linguistics and Intelligent Text Processing (CICLing 2005)*, volume 3406 of Lecture Notes in Computer Science, pages 1–24. Springer, 2005.

[Lib04] Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[MK06] Jonathan May and Kevin Knight. Tiburon: A weighted tree automata toolkit. In Oscar H. Ibarra and Hsu-Chun Yen, editors, *Proc. 11th Intl. Conf. on Implementation and Application of Automata (CIAA 2006)*, volume 4094 of Lecture Notes in Computer Science, pages 102–113. Springer, 2006.

[PS89] Gheorghe Păun and Lila Sântean. Parallel communicating grammar systems: The regular case. *Annals of the University of Bucharest. Series Mathematics-Informatics*, 38:55–63, 1989.

[PS94] Carl Pollard and Ivan Sag. *Head-Driven Phrase Structure Grammar*. Chicago University Press, 1994.

[Sad91] Jerrold Sadock. *Autolexical Syntax - A Theory of Parallel Grammatical Representations*. The University of Chicago Press, Chicago & London, 1991.

[Ste00] Mark Steedman. *The Syntactic Process (Language, Speech, and Communication)*. MIT Press, 2000.

[Tho97] Wolfgang Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*. Vol. 3: *Beyond Words*, chapter 7, pages 389–455. Springer, 1997.

[Tra62] Boris A. Trakhtenbrot. Finite automata and the logic of one-place predicates. *Siberian Mathematical Journal*, 3:103–131, 1962. In Russian. English translation in *AMS Translations*, Series 2, 59:23–55, 1966.

[TW68] James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision-problem of second-order logic. *Mathematical Systems Theory*, 2:57–81, 1968.