Graph Transformation for Incremental Natural Language Analysis[☆]

Suna Bensch[a], Frank Drewes[a], Helmut Jürgensen[b], Brink van der Merwe[c]

[a]*Deptartment of Computing Science, Umeå University, Sweden*
[b]*Department of Computer Science, Western University, London, Canada*
[c]*Department of Computer Science, Stellenbosch University, South Africa*

Abstract

Millstream systems have been proposed as a non-hierarchical method for modelling natural language. Millstream configurations represent and connect multiple structural aspects of sentences. We present a method by which the Millstream configurations corresponding to a sentence are constructed. The *construction* is incremental, that is, it proceeds as the sentence is being read and is complete when the end of the sentence is reached. It is based on graph transformations and a lexicon which associates words with graph transformation rules that implement the incremental construction process. Our main result states that, for an effectively nonterminal-bounded reader $\mathcal{R}$ and a Millstream system MS based on monadic second-order logic, the correctness of $\mathcal{R}$ with respect to MS can be checked: it is decidable whether all graphs generated by $\mathcal{R}$ belong to the language of configurations specified by MS.

Keywords: graph transformation; hyperedge replacement; natural language analysis; reader; Millstream system

1. Introduction

Millstream systems simultaneously model several aspects of language structure in a parallel and co-ordinated way [4, 5]. A Millstream configuration of a sentence represents the analysis of that sentence with respect to those aspects, including appropriate links between the analyses. As aspects to be considered, morphology, syntax and semantics come to mind immediately. However, other aspects can be modelled as well. An important point is that the separation of aspects can lead to simple models for each of them; the connections between the models, the links, are established by, hopefully, also simple conditions. While the formal notions developed in this paper as well as the results obtained are

[☆]This article is a revised and extended version of [8].

Email addresses: `suna@cs.umu.se` (Suna Bensch), `drewes@cs.umu.se` (Frank Drewes), `hjj@csd.uwo.ca` (Helmut Jürgensen), `abvdm@cs.sun.ac.za` (Brink van der Merwe)

independent of the number and types of linguistic aspects considered, we illustrate them by rather small examples that cover only syntax and semantics – and even these in a very restricted way neglecting many linguistic details – because our aim is to convey the principles and the potential of our approach rather than to present a full-blown implementation of a system for linguistic analysis of sentences. Nevertheless, the implementation of such a system is one of the long-term goals of this research, and we hope that our presentation shows that such an implementation would be both desirable and possible.

Various psycholinguistic and cognitive neuroscience-based studies (see [35] for example) show that humans do not postpone the analysis of an utterance or sentence until it is complete; they rather start to process the sentence immediately when they have heard or read the first words or parts of words. Along these lines, we present results regarding the incremental syntactic and semantic analysis of natural language sentences using Millstream systems as part of our ongoing work on this formalism for the description and analysis of language.

Incremental language processing is an intensively studied topic both in the context of compiler construction for programming languages and in the context of natural language parsing. Of the vast literature related to incremental parsing we mention only a few selected studies: Work on incremental LR-parsing for programming languages by Ghezzi and Mandrioli [18] and by Wagner and Graham [36]; studies of incremental parsing for natural languages using various grammar models and various computing paradigms by Beuck et al. [9], Costa et al. [11, 10], Hassan et al. [23], Huang and Sagae [25], Lane and Henderson [29], Nivre [30] and Wu et al. [37]. In these and similar studies one contructs a structural representation of an utterance, a sentence, or a program by building partial structures as one progresses reading or hearing the input and by combining them or rejecting already constructed structures. The structural representation is intended to reflect all relevant aspects as described by a single formal grammar. In his 1960 paper *Grammar for the Hearer* [24], Hockett discusses the natural understanding of spoken language and the implied constraints on parsing models. What Hockett calls a "hearer" would be called a "reader" in our setting.

We propose that various linguistic levels like phonology, morphology, syntax and semantics should be considered simultaneously and not successively. Hence, we base our work on Millstream systems [4, 5], a generic mathematical framework for the description of natural language. These systems describe linguistic aspects such as syntax and semantics in parallel by separate modules and provide the possibility to express the relation between the aspects by so-called interfaces. Roughly speaking, a Millstream system[1] consists of a finite number of *modules* each of which describes a linguistic aspect and an *interface* which describes the dependencies between these aspects. The modules need not be of the same mathematical nature: one aspect might be adequately modelled by

[1]The term *Millstream system* refers to the place at which the notion was created; thus, it has no direct connection to language theory.

a context-free grammar while, for another aspect, a Montague grammar might be preferable. Each module defines a tree language which describes one linguistic aspect in isolation. The interface establishes links between the trees given by the modules, thus turning unrelated trees into a meaningful whole called a *configuration* and filtering out analyses which make sense with respect to some linguistic aspects, but not all of the ones modelled.

In contrast, if one were to use a single type of grammar to model all aspects simultaneously, the resulting construct would be unmanageable, as is well known and can be seen in the admirable attempt of [28] to model German.

Consider – for simplicity – a Millstream system containing only two modules, a syntactic and a semantic one, which model the syntax and semantics of a natural language. A configuration of the Millstream system consisting of two trees with links between them represents an analysis of a sentence. An obvious question is how to construct such a configuration from a given sentence. Such a procedure would be a step towards automatic language understanding based on Millstream systems. This paper continues the work begun in [7], where we proposed to use graph transformation for that purpose. We mimic the incremental language processing performed by humans to construct a Millstream configuration by a step-by-step procedure while reading the words of a sentence from left to right[2]. The idea is that the overall structure of a sentence is built incrementally, word by word. With each word, one or more lexicon entries are associated. These lexicon entries are graph transformation rules whose purpose it is to construct an appropriate configuration.

For a sentence like *Mary loves Peter*, for example, we first apply a lexicon entry corresponding to *Mary.* This results in a partial configuration representing the syntactic, semantic and interface structure of the word. We continue by applying a lexicon entry for *loves*, which integrates the syntactic, semantic and interface structure of this word into the configuration. Thus, after the second step, we have obtained a partial configuration representing *Mary loves.* Finally, the structure representing *Peter* is integrated into the configuration, resulting in the Millstream configuration for the entire sentence.

We call such a sequence of graph transformation steps a *reading* of the sentence. The graph transformation system itself, which consists mainly of the lexicon, is called a *reader.* Since words can appear in different contexts, alternative lexicon entries for one and the same word may co-exist. In general, this may result in nondeterminism or even ambiguity; the former occurs when two or more rules are applicable, but only one will finally lead to a complete reading; the latter arises when two or more readings of the sentence are possible. These effects are inevitable as they are caused by inherent properties of natural language. In many situations, however, only one lexicon entry will be applicable because its left-hand side requires certain partial structures to be present, to which the new part is added. This corresponds to the situation of a human reader who has already seen part of the sentence and can thus rule out certain

[2]Instead of “left to right” one might prefer to say “in their spoken (or natural) order.”

lexicon entries associated with the next word.

Given a reader that is supposed to construct configurations of a Millstream system *MS*, an obvious question to ask is whether the reader yields correct configurations, that is, whether the configurations it constructs are indeed configurations of *MS*. The main (formal) result of this paper is Corollary 1 which states that, under certain conditions, this question is decidable for so-called regular MSO Millstream systems, that is, systems in which the modules are regular tree grammars (or, equivalently, finite tree automata) and the interface conditions are expressed in monadic second-order (MSO) logic. In other words, given a regular MSO Millstream system *MS* and a reader satisfying the conditions mentioned, one can determine effectively whether all readings yield correct configurations of *MS*.

Our graph transformation rules use a special case of the well-known double-pushout (DPO) approach [15]. Since general DPO rules give rise to Turing-complete graph transformation systems, we try to avoid them. One of the most prominent special cases of the DPO approach is hyperedge replacement, a context-free type of graph transformation invented independently by Bauderon and Courcelle [3] and Habel and Kreowski [21]; see also [12, 20, 16, 13]. Our formal model of *readers* is based on an extension of hyperedge replacement that allows the replacement of several hyperedges simultaneously. Moreover, the replacement may depend on the context. This type of rules was inspired by *contextual hyperedge replacement* as proposed in [14] for applications in software refactoring. However, in that paper only single hyperedges are replaced and the context consists of isolated nodes.

We hope that the use of graph transformation in natural language processing can pave the way for incorporating the semantic dimension and other aspects in the linguistic analysis of sentences. In fact, we are not the only authors who emphasize the potential of graph transformation in natural language processing. Another recent paper that comes to exactly the same conclusion from the point of view of statistical machine translation is [27], where the use of synchronous hyperedge replacement grammars for semantically informed machine translation is proposed.

Our paper is structured as follows. In Section 2 we introduce the formal background about Millstream systems and the representation of Millstream configurations as graphs. We then introduce the model of readers in Section 3 and explain their mode of operation by means of a small example. In Section 4 we consider the correctness of such readers and prove the main result of the paper. Next, in Section 5, we explain the linguistic relevance of (nonterminal-bounded) readers, by considering structures occurring in natural language, such as unlimited embedding, anaphoric reference and *wh*-dependencies. We discuss the results and the next steps of this ongoing research in Section 6.

2. Millstream Configurations as Graphs

Throughout the paper, $\mathbb{N}$ denotes the set of non-negative integers. For a set S, S^* denotes the set of all finite sequences (or strings) of elements of S. For

$w \in S^*$, $|w|$ denotes the length of w, and $[w]$ denotes the set of all elements of S occurring in w, that is, $[w]$ is the smallest set A such that $w \in A^*$. As usual, if $\Rightarrow$ is a binary relation, then $\Rightarrow^*$ denotes its reflexive and transitive closure.

We first define the general type of graphs considered. For modelling convenience, we choose to work with hypergraphs in which the hyperedges, but not the nodes, are labelled. In an ordinary edge-labelled directed graph, edges are triples (u, v, a) consisting of a start node u, an end node v, and a label a. If we want to be able to consider multiple parallel edges with identical labels, we can accomplish this by saying that a graph consists of finite sets V and E of nodes and edges, respectively, an attachment function $\mathsf{att}\colon E \to V^2$ and a labelling function $\mathsf{lab}\colon E \to \Sigma$, where Σ is the set of labels. Hypergraphs generalize this one step further, by allowing a hyperedge to be attached to any sequence of nodes rather than just a pair, that is, the attachment is turned into a function $\mathsf{att}\colon E \to V^*$. Note that this includes hyperedges which are attached to only a single node or to no node at all.

Often it turns out to be convenient to assume that the label of a hyperedge determines the number of attached nodes. For this reason, the edge labels will be taken from a *ranked alphabet* Σ, meaning that Σ is a finite set of symbols such that every $a \in \Sigma$ has a *rank* $\mathsf{rk}(a) \in \mathbb{N}$. In the sequel, we use the terms *alphabet, graph,* and *edge* to mean *ranked alphabet, hypergraph,* and *hyperedge,* respectively.

The precise definition follows.

Definition 1 (Graph)**.** *Let Σ be an alphabet. A Σ-*graph *or simply* graph *is a quadruple $(V, E, \mathsf{att}, \mathsf{lab})$ consisting of*

- *a finite set V of* nodes*,*
- *a finite set E of* edges*,*
- *an* attachment *$\mathsf{att}\colon E \to V^*$, and*
- *an* edge labelling *$\mathsf{lab}\colon E \to \Sigma$ such that $\mathsf{rk}(\mathsf{lab}(e)) = |\mathsf{att}(e)|$ for all $e \in E$.*

If they are not explicitly named, the components of a graph G are referred to as $V_G, E_G, \mathsf{att}_G, \mathsf{lab}_G$. By $\mathcal{G}_\Sigma$ we denote the class of all Σ-graphs.

A subgraph of $G = (V, E, \mathsf{att}, \mathsf{lab})$ is a graph $(V', E', \mathsf{att}', \mathsf{lab}')$ such that $V' \subseteq V$, $E' \subseteq E$, and att' and lab' are the restrictions of att and lab, respectively, to E'.

Example 1 (Graph)**.** *Figure 1 shows a graph G consisting of nodes $v_1, \ldots, v_5$ and edges $e_1, \ldots, e_4$. The edges are drawn as rectangles with their labels inside. Lines, sometimes called* tentacles *in the literature, connect the edge with its attached nodes. For e_2, for instance, we have $\mathsf{lab}_G(e_2) = c$ and $\mathsf{att}_G(e_2) = v_5v_3v_2v_4$. Thus, the rank of c is assumed to be 4. The edges e_3 and e_4 are ordinary edges. Normally they would be drawn as arrows pointing from v_4 to v_3. In the sequel, graphs will be drawn in a less cluttered way, using suitable conventions about how to draw different types of edges and leaving out information which is irrelevant or can be inferred from the context.*

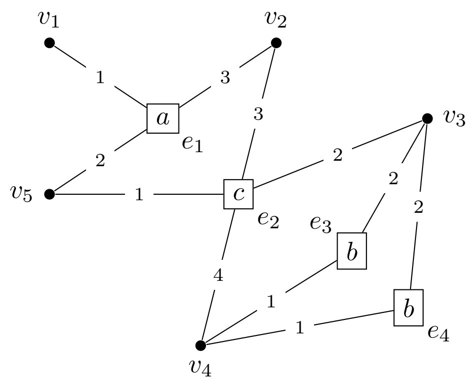


Figure 1: A graph.

An *isomorphism* between graphs G and H is a pair of bijective mappings $m_V\colon V_G \to V_H$ and $m_E\colon E_G \to E_H$ that preserves attachments and labels, that is, $m_V(\mathsf{att}_G(e)) = \mathsf{att}_H(m_E(e))$ and $\mathsf{lab}_G(e) = \mathsf{lab}_H(m_E(e))$ for all $e \in E_G$; here m_V is canonically extended to a homomorphism of V_G^* into V_H^*. If such an isomorphism between G and H exists, the two graphs are said to be isomorphic. We point out here that we do not wish to distinguish between isomorphic graphs even though, for technical convenience, all constructions will be given in terms of operations on concrete graphs. Since we generally identify isomorphic graphs, we do not explicitly mention that statements such as "the graph G is uniquely determined" or "there are only finitely many graphs of size k in $\mathcal{G}_\Sigma$" are intended to be read with the implicit addition "up to isomorphism".

A *term graph* G is a graph that represents a formal expression over a set Σ_t of function symbols. As usual, each symbol a in such an expression has an arity k which determines the number of subexpressions it requires. The arity k of $a \in \Sigma_\mathrm{t}$ is also indicated by denoting a as $a_{(k)}$. To represent an expression over Σ_t as a graph G, each $a_{(k)}$ is regarded as a label of rank $k+1$. If an edge $e \in E_G$ has the attachment $v_1 \cdots v_k v$ we say that $v_1, \ldots, v_k$ are the *source nodes* of e and v is its *target node*. Intuitively, in a term graph the target node of an edge labelled with a represents an expression $a(\cdots)$ the direct subexpressions of which are represented by the source nodes of the edge. Essentially, term graphs have already been proposed in [2], where they were called DOAGs (directed ordered acyclic graphs). Another closely related concept is the notion of a *jungle*; cf. [22].

To make the notion of term graphs precise, consider a ranked alphabet Σ_t such that $\mathsf{rk}(a) \geq 1$ for all $a \in \Sigma_\mathrm{t}$. For a graph G and nodes $u, v \in V_G$, there is a *path from u to v* if there are nodes $u = v_0, \ldots, v_n = v$ such that, for every $i \in \{1, \ldots, n\}$, E_G contains an edge that has v_{i-1} among its source nodes and v_i as its target node.

Definition 2 (Term graph). *Let Σ_t be a ranked alphabet such that* $\mathsf{rk}(a) \geq 1$ *for all $a \in \Sigma_\mathrm{t}$. A* term graph over Σ_t *is a Σ_t-graph G such that*

1. *every node $v \in V_G$ is the target node of a unique edge $e \in E_G$,*

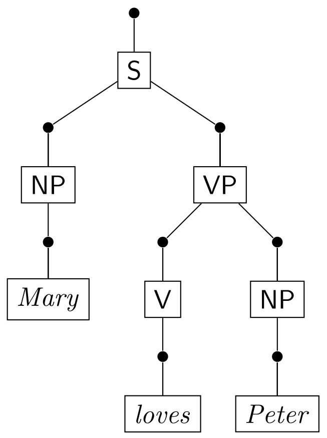
2. *there is a node $r \in V_G$, called the* root *of G, which is not a source node of any edge in E_G, and*

3. *for every node $v \in V_G$, there is a path from v to the root of G.*

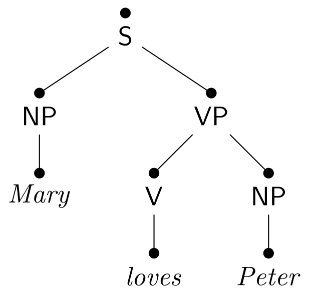
For the sake of clarity, let us define the expression represented by a term graph G. If $r \in V_G$ is the root of G then G represents the expression $\mathsf{term}(G) = \mathsf{term}_r(G)$. Here $\mathsf{term}_v(G)$ is defined as follows for all $v \in V_G$: If v is the target node of $e \in E_G$, $\mathsf{lab}_G(e) = a$ and $\mathsf{att}_G(e) = v_1 \cdots v_k v$, then

$$\mathsf{term}_v(G) = a(\mathsf{term}_{v_1}(G), \dots, \mathsf{term}_{v_k}(G)).$$

Note that distinct occurrences of equal subexpressions in $\mathsf{term}(G)$ may be shared in G, that is, represented by the same node. We call a term graph G a tree if it does not involve sharing. More precisely, a term graph G is a tree if for, all edges $e, e' \in E_G$ with $\mathsf{att}_G(e) = u_1 \cdots u_k u$ and $\mathsf{att}_G(e') = v_1 \cdots v_l v$, $u_i = v_j$ for some $i \in \{1, \dots, k\}$ and $j \in \{1, \dots, l\}$ implies $e = e'$ and $i = j$.



(a) A tree. The target node of each edge is the one on top.



(b) A more condensed representation of the tree in (a).

Figure 2: Pictorial representation of a term graph without sharing, that is, a tree.

Figure 2 depicts a tree over $\mathsf{S}_{(3)}$, $\mathsf{VP}_{(3)}$, $\mathsf{NP}_{(2)}$, $\mathsf{V}_{(2)}$, $\mathit{Mary}_{(1)}$, $\mathit{loves}_{(1)}$, $\mathit{Peter}_{(1)}$. We do not specify the order of the source nodes of edges explicitly, but use the convention that they should be read from left to right. Target nodes are drawn above their respective edges. To save space and make drawings of term graphs more comprehensible, we use the more usual and less redundant drawing style shown in Figure 2(b) in the sequel. The reader should, however, keep in mind that nodes are unlabelled; the labels drawn underneath the nodes are edge labels rather than node labels.

Let us now turn to Millstream configurations. In this paper, we represent them in a way which is suitable for graph transformation. Therefore, we define a configuration to consist of k term graphs – one for each linguistic aspect considered – with links between them. The links are represented by edges. They indicate relations between the nodes and may belong to different categories, that is, they may carry different labels. A typical link establishes a relation between two nodes, where one belongs to one term graph and the other one belongs to another one. In general, links may have an arbitrary arity, and may also link two or more nodes belonging to the same term graph.

The alphabets used to build Millstream configurations contain two types of edges: term edges and links, respectively. Term edges are those edges of which the term graphs consist, and links are used to establish connections between nodes in the term graphs.

Definition 3 (Millstream alphabet and configuration)**.** *Let $k \in \mathbb{N}$.*

1. *A* Millstream alphabet *is a ranked alphabet Σ partitioned into two disjoint alphabets Σ_t and Σ_l of* term symbols *and* link symbols*, respectively.*

Let $G \in \mathcal{G}_\Sigma$.

2. *Edges of G labelled with symbols in Σ_t and Σ_l are called* term edges *and* links*, respectively.*

3. *A* Millstream configuration of dimension k *(over Σ) is a graph $G \in \mathcal{G}_\Sigma$ such that the deletion of all links from G results in a disjoint union of k term graphs.*

Note that, for a Millstream configuration G and given link symbol σ of arity l, the set of all links labelled with σ represents an l-ary relation on the nodes of the configuration.

Since we are not going to make use of the formal definition of Millstream systems here, we explain only informally what a Millstream system in the sense of [5] is. Let $k \geq 1$ and let Σ be a Millstream alphabet.

A Millstream system *MS* over Σ is a $k+1$-tuple $MS = (M_1, \ldots, M_k; \Phi)$, consisting of k *modules* and a logical *interface* Φ. The modules are any formal devices that specify sets $L(M_1), \ldots, L(M_k)$ of term graphs over Σ_t. For example, classical tree automata can be used as modules. The interface consists of a set of closed logical formulæ which express properties of Millstream configurations. A Millstream configuration the underlying term graphs of which are $G_1, \ldots, G_k$ belongs to the language $L(MS)$ if $G_1 \in L(M_1), \ldots, G_k \in L(M_k)$ and, in addition, the configuration as a whole satisfies Φ.[3]

In this paper, we consider the class of *MSO Millstream systems*, where 'MSO' stands for 'monadic second-order'. By definition, an MSO Millstream system is

[3]See [12, Section 5.3.3] for a precise definition of how a graph G can be seen as a logical structure $|G|_2$ whose universe is the union of the sets of nodes and edges of the graph, and whose predicates determine the labels and attachments of the edges.

a Millstream system in which the k modules are closed formulæ $\Phi_1, \ldots, \Phi_k$ in MSO logic that define sets $L(\Phi_1), \ldots, L(\Phi_k)$ of term graphs, and the interface is an MSO formula as well.[4] The reader should note that it is easy to express in MSO logic that a graph is a term graph. Therefore, we may assume without loss of generality that only graphs that are indeed term graphs satisfy Φ_i, for $i \in \{1, \ldots, k\}$.

A special case of MSO Millstream systems (with respect to descriptional capacity) are the regular MSO Millstream systems studied in [4, 5, 6, 7]. These are Millstream systems in which the modules are regular tree grammars (or finite state tree automata) and the interface is an MSO formula. Since the regular tree languages are exactly the MSO-definable tree languages, MSO Millstream systems are more general only in so far as their modules can generate term graphs instead of trees.

There are two reasons for this generalization. The first is that such sharing often occurs in linguistic modelling, for example when a functional entity of a sentence is referred to several times. The second reason is that sharing is a natural concept to use when working with graph transformation. As mentioned earlier, the k term graphs in a Millstream configuration are supposed to represent k individual linguistic aspects of one and the same sentence. Millstream systems that are sophisticated enough to be of practical linguistic relevance may be expected to cover several linguistic aspects and contain links of various kinds and arities. As mentioned in the introduction, the discussion of such examples is beyond the scope and purpose of this paper. It would, furthermore, lead to incomprehensible figures. Therefore, the Millstream configurations in the examples of this paper will consist of two term graphs each, representing a simplified syntactic and semantic analysis of a sentence, and there will be only one link symbol. This link symbol is of rank 2, and it connects nodes across the two term graphs. Therefore, we draw links as unlabelled lines connecting the nodes in question. Furthermore, to distinguish links from term edges, the former are depicted as dashed lines.

With these conventions, a configuration looks as shown in Figure 3. It consists of two term graphs which represent the extremely simplified syntactic and semantic structures of the sentence *Mary loves Peter*. The term symbols of the semantic part should be interpreted as functions of a many-sorted algebra. The sorts of the algebra are the semantic domains of interest, and the evaluation of a term graph G yields an element of one of these sorts by evaluating $\mathsf{term}(G)$ in the usual manner. In the semantic tree shown in the figure, we assume that Mary and Peter are (interpreted as) functions without arguments, that is, constants, returning elements of the sort *name*. The function person takes a name as its argument and returns, say, an element of the domain *person*. Finally, loving is a function that takes two persons as arguments and returns an element of

[4] From the point of view of descriptive power, the modules are of course negligible in this case, because the formulæ Φ_i can be conjunctively added to the interface. However, modularization makes the construction of Millstream systems easier.

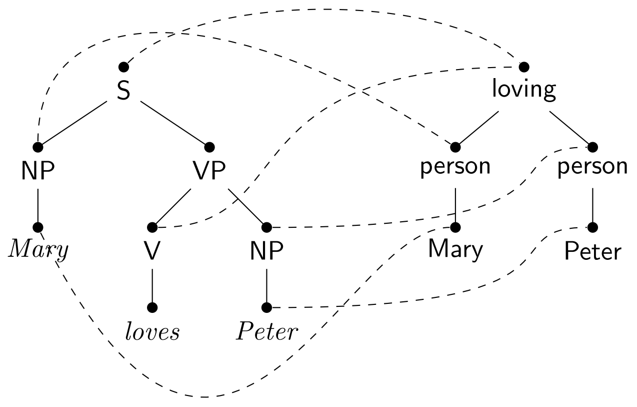


Figure 3: A sample configuration that relates a syntactic and a semantic tree.

the domain *state*, namely the state that the first argument (commonly called the *agent*) loves the second (the *patient*). The links establish correspondences between nodes in the two trees showing, for example, that the verb of the sentence corresponds to the function loving whose two arguments correspond to the two noun phrases of the sentence. We note here that this correspondence must respect the different thematic roles (agent and patient) of the noun phrases, thus making sure that the agent is really the first argument of the semantic function. Otherwise, the *state* object obtained by evaluating the semantic tree would express the wrong thing.

In realistic settings, one would use more elaborate term graphs on both the syntactic and the semantic sides. For example, one might decompose the verb into its stem *love* and the inflection *s* indicating present tense. Semantically, one would at least add a node above the current root. This node would be a function taking a *state* as its input and turning it into a *situation* in the present, that is, it would reflect the temporal information provided by the inflection. That node would then be linked with the inflection that gives rise to it (see, e.g., [26]). This slightly more elaborate configuration is shown in Figure 4. However, since we primarily want to convey the idea behind our way of constructing configurations, we will stick to the more simplified type of configurations shown in Figure 3 for the examples discussed throughout the rest of this paper.

3. Building Configurations Incrementally by Graph Transformation

We are given a sentence. We want to "understand" it. Technically, more precisely and more modestly, we want to turn this sentence into a "correct" Millstream configuration, if possible. What is a correct Millstream cofiguration for a given sentence and how can we find it? These questions are addressed in the present section of this paper. As described above, a Millstream system contains k modules, one for each of the k trees in a configuration. Furthermore, we are

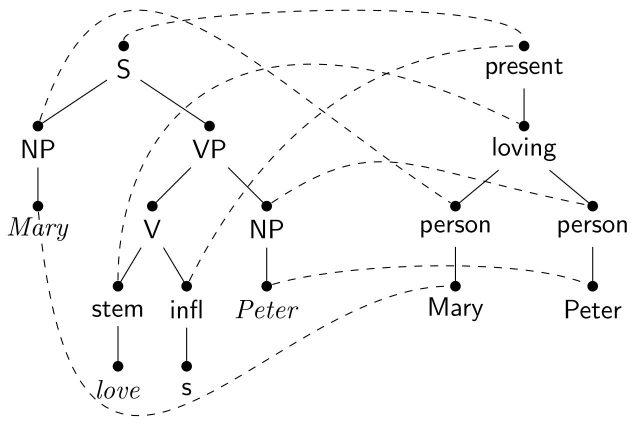


Figure 4: A slightly less simplified configuration.

given a logical *interface* describing which configurations – consisting of k term graphs generated by the modules and a set of links between them – are considered to be correct. Here we investigate how configurations can be built "from scratch" along a sentence using graph transformation. We use a lexicon which associates words with graph transformation rules. Consider the very simple sentence *Mary loves Peter.* The construction of a configuration for this sentence consists of applying three rules in succession, the first one being associated with *Mary*, the second one with *loves*, and the third one with *Peter.* In general, this process is nondeterministic.

We use graph transformation in the sense of the so-called double pushout (DPO) approach [15] with injective occurrence morphisms. A *DPO rule* or simply a *rule* r consists of three graphs, where the one in the middle is a subgraph of the others: $r = (L \supseteq K \subseteq R)$ (also called span in the literature). The rule applies to a given graph G if

1. L is isomorphic to a subgraph of G (for simplicity, we assume that the isomorphism is the identity) and

2. no edge in $E_G \setminus E_L$ is attached to a node in $V_L \setminus V_K$.

The application of r is defined up to isomorphism. It is best described in terms of concrete graphs, using the following two-step process. We first create the graph D from G which is obtained by removing all nodes and edges from it that are in L but not in K. Then, we obtain the result H by adding all nodes and edges to D that are in R but not in K. Thus, the middle component K, which is commonly called the *glueing graph* is not affected by the rule, but rather used to "glue" the new nodes and edges in the right-hand side R to the existing graph. The second condition for applicability, commonly known as the *dangling condition*, ensures well-formedness, as it makes sure that the deletion of nodes does not result in so-called dangling edges, that is, edges with an undefined attachment. The application of r to G, yielding H, is denoted by $G \Rightarrow_r H$. If R is a set of graph transformation rules and $G \Rightarrow_r H$ for some $r \in R$, we denote this fact by $G \Rightarrow_R H$.

Compared to general DPO rules, our lexicon rules are quite restricted. We use a ranked alphabet N of nonterminal labels to indicate "construction sites" in a partial configuration. Lexicon rules never delete nodes or ordinary edges, but only nonterminals. The following definition makes these notions of nonterminals and rules precise.

Definition 4 (Lexicon rule)**.** *Let Σ and N be a Millstream alphabet and a ranked alphabet of nonterminal labels, respectively, such that N is disjoint from Σ. A* nonterminal *of a graph G is an edge $e \in E_G$ such $\mathsf{lab}_G(e) \in N$.*

A lexicon rule over Σ and N *is a rule $r = (L \supseteq K \subseteq R)$ over $\Sigma \cup N$ that satisfies the following requirements:*

1. *$L \in \mathcal{G}_{\Sigma \cup N} \setminus \mathcal{G}_\Sigma$,*
2. *K is the graph obtained from L by deleting all nonterminals, and*
3. *for every edge $e \in E_R \setminus E_K$,*

$$[\mathsf{att}_R(e)] \subseteq \bigcup \{[\mathsf{att}_L(e')] \mid e' \in E_L, \mathsf{lab}_L(e') \in N\} \cup (V_R \setminus V_K).$$

Since the glueing graph K of a lexicon rule $r = (L \supseteq K \subseteq R)$ is uniquely determined by the left-hand side L, it is not necessary to mention it explicitly. We will therefore denote lexicon rules by $L ::= R$ in the sequel. Note that, since $V_K = V_L$, the dangling condition is automatically satisfied, that is, r applies to every subgraph which is isomorphic to L.

Let us briefly discuss the requirements on lexicon rules in Definition 4.

The first one requires that L contains at least one nonterminal. This requirement should intuitively be reasonable. It guarantees that a graph that does not contain any nonterminals is indeed terminal, that is, no lexicon rule applies to it. However, it is in fact not difficult to see that readers, as defined in Definition 5 below, do not gain any additional power if the requirement is dropped. This can be seen by adding a new nonterminal label Z of rank 0 and turning every lexicon rule $L ::= R$ with $L \in \mathcal{G}_\Sigma$ into the two rules $L + Z ::= R + Z$ and $L + Z ::= R$, where '$+\,Z$' denotes the addition of a new nonterminal labelled Z to a graph.

By the second requirement, the application of a lexicon rule removes the nonterminals in its left-hand side, but nothing else.

The third requirement states that new edges added to the configuration – term edges, links, and nonterminals – can only be attached to new nodes and nodes that were attached to one of the replaced nonterminals. In other words, the left-hand side except the nonterminals and their attached nodes is nothing but context the presence of which is required for the rule to be applicable. This context is neither changed by deleting part of it, nor are new edges attached to its nodes. This reflects the intuitive view that nonterminals indicate "construction sites", that is, parts of the configuration that are still under development.

It is worth recalling that, strictly speaking, we work with abstract graphs rather than concrete ones. In particular, derivation steps are defined only up to isomorphism. This means that we can safely select any concrete graph G as the representative of the corresponding abstract graph, that is, of the isomorphism class of all graphs that are isomorphic to G. We exploit this fact in order to simplify some constructions, using the following technical assumption.

Technical Assumption. *Consider a derivation step $G \Rightarrow_r H$ by a lexicon rule $r = (L ::= R)$. From now on, we assume that H is constructed from G in such a way that*

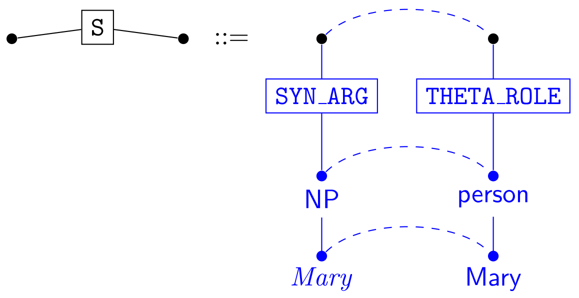
(a) the nonterminals in the image of L in G are removed from G, yielding a graph $D \subseteq G$, and

(b) fresh copies of the nodes and edges in R that are not in L are added to D.

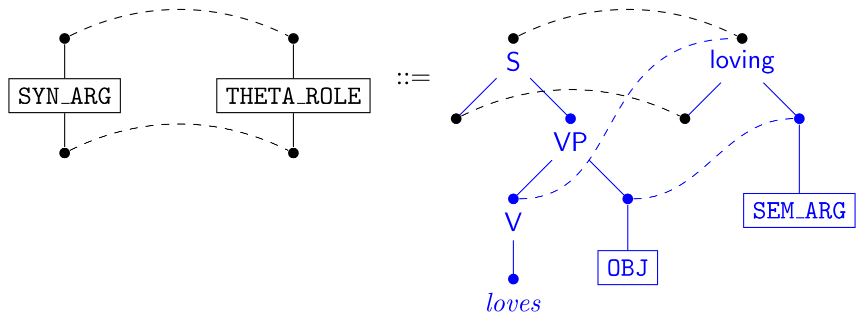
Here, fresh *means that these nodes and edges do not appear in G and, when a derivation is considered, not in earlier graphs of that derivation either. Thus, the nodes and edges in G are still present in H, except for nonterminals that have been replaced, and all nodes and edges that have been added are fresh copies of those in R.*

Figure 5 shows sample lexicon rules for *Mary*, *loves*, and *Peter*. Here, and in the sequel, a lexicon rule associated with a particular word w is referred to as a *lexicon entry for w*. The nonterminals are depicted in typewriter font surrounded by boxes, the term edges and links are drawn as before. The nonterminals in Figure 5 indicate which syntactic or semantic roles are missing. By deleting the nonterminals on the left-hand side of a lexicon rule we obtain the glueing graph, that is, the context required in a partial configuration for the lexicon rule to be applicable. This glueing graph is drawn in black on the right-hand side of a lexicon entry. The nodes, term edges and links drawn in blue[5] on the right-hand side are "added" around the glueing graph, thus replacing the nonterminals on the left-hand side. By replacing nonterminals the syntactic or semantic roles around the glueing graph are specified. Starting with the start graph – on the left-hand side of the lexicon entry for *Mary* – and applying the three lexicon entries in Figure 5 in the order in which the words appear in the sentence *Mary loves Peter* takes us to the configuration in Figure 3. The nonterminal labels `SYN_ARG` and `THETA_ROLE` on the right-hand side of the lexicon entry for *Mary* indicate that, after application of this rule, the syntactic and semantic roles of *Mary* are not yet specified: whether *Mary* is the syntactic subject or object (`SYN_ARG`) and semantic agent or patient (`THETA_ROLE`) has to be specified during the reading of sentence. Note also that the complete lexicon should contain another entry similar to the one in (a), but with *Mary* and

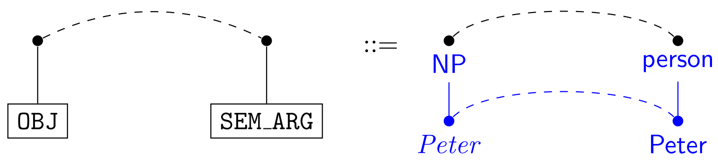
[5]In case of grey scale printouts the blue appears grey.



(a) Lexicon entry for *Mary*.



(b) Lexicon entry for *loves*.



(c) Lexicon entry for *Peter*.

Figure 5: Lexicon entries for *Mary*, *loves*, and *Peter*; the left-hand side of the lexicon entry for *Mary* is the (unique) start graph.

Mary being replaced by *Peter* and Peter, respectively. Similarly, there should be a variant of (c) for the name Mary. This would make it possible to read the sentence *Peter loves Mary.*[6] Before we discuss such a reading of a sentence and a slightly more complex example we define the notion of a *reader* formally.

Definition 5 (Reader)**.** *A* reader *is a tuple* $\mathcal{R} = (\Sigma, N, W, \Lambda, S)$ *consisting of*

- *a Millstream alphabet* Σ*,*
- *a ranked alphabet* N *of nonterminal labels that is disjoint from* Σ*,*
- *a finite set* W *of words, the input words,*
- *a mapping* Λ*, called the* lexicon*, that assigns, to every* $w \in W$*, a finite set* $\Lambda(w)$ *of lexicon rules over* Σ *and* N*, and*
- *a finite set* $S \subseteq \mathcal{G}_{\Sigma \cup N}$ *of start graphs.*

Let $u = w_1 w_2 \cdots w_n \in W^*$ *with* $w_1, w_2, \ldots, w_n \in W$*. A* reading *of* u *by* $\mathcal{R}$ *is a derivation*

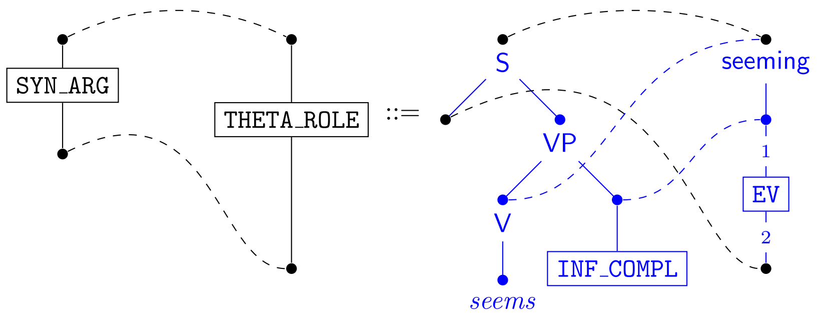
$$G_0 \Rightarrow_{\Lambda(w_1)} G_1 \Rightarrow_{\Lambda(w_2)} \cdots \Rightarrow_{\Lambda(w_n)} G_n$$

such that $G_0 \in S$ *and* $G_n \in \mathcal{G}_\Sigma$*. In this case,* G_n *is said to be the* result *of this reading. The set of all* Σ*-graphs resulting from readings of* u *is denoted by* $\mathcal{R}(u)$*, and* $L(\mathcal{R}) = \bigcup_{u \in W^*} \mathcal{R}(u)$*.*

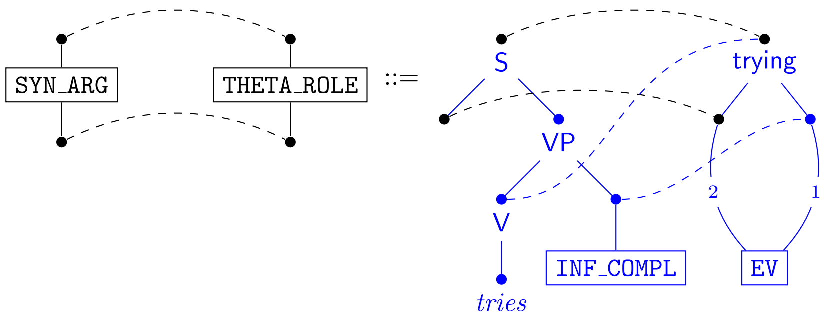
When considering readers as defined, we refer to a rule in $\Lambda(w)$ also as a *lexicon entry for w*. For $u = w_1 \cdots w_n$ and $(\Sigma \cup N)$-graphs $G_0, \ldots, G_n$, a derivation of the form $G_0 \Rightarrow_{\Lambda(w_1)} G_1 \Rightarrow_{\Lambda(w_2)} \cdots \Rightarrow_{\Lambda(w_n)} G_n$ may be abbreviated as $G_0 \Rightarrow^*_{\Lambda(u)} G_n$.

Let us discuss a reading of a sentence in a slightly more interesting case than the one above. We consider English sentences involving the verbs *to seem* and *to try*; they are raising and control verbs, respectively. In sentences like *Mary seems to sleep*, the noun *Mary* is the syntactic subject of *seems*, but it is not its semantic argument. In contrast, in the sentence *Mary tries to sleep*, *Mary* is both the syntactic and the semantic argument of *tries* as well as the semantic argument of *to sleep*. For the sake of simplicity, we assume that these sentences have the same syntactic structure, that is, that they differ only with respect to their semantic structure. Figure 9 shows Millstream configurations corresponding to the sentences. In the first configuration, the fact that *Mary* is the semantic argument of both *tries* and *to sleep* is represented by sharing rather than duplicating the corresponding substructures. Figure 6 shows the lexicon entries needed to be able to handle these sentences. In addition, the lexicon entry for *Mary* given in Figure 5(a) is needed. The nonterminal labels `INF_COMPL` and `EV` are abbreviations for *infinitival complement* and *eventual event*. Let us consider the reading of the sentence *Mary tries to sleep*. We start

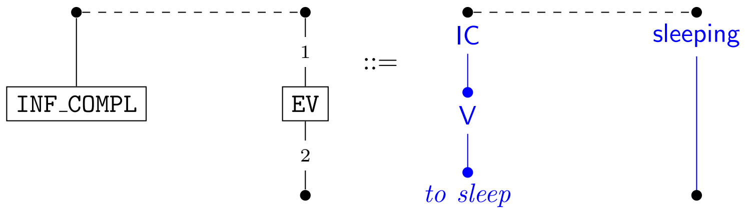
[6]From an implementation point of view, such situations should of course be handled by rule templates.



(a) Lexicon entry for *seems*.



(b) Lexicon entry for *tries*.



(c) Lexicon entry for *to sleep*.

Figure 6: Lexicon entries for handling *Mary seems to sleep* and *Mary tries to sleep*.

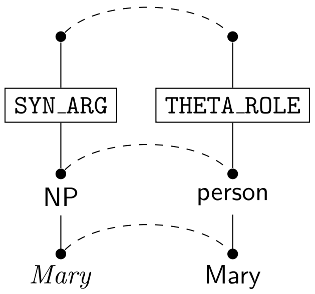


Figure 7: Partial configuration after having read *Mary*.

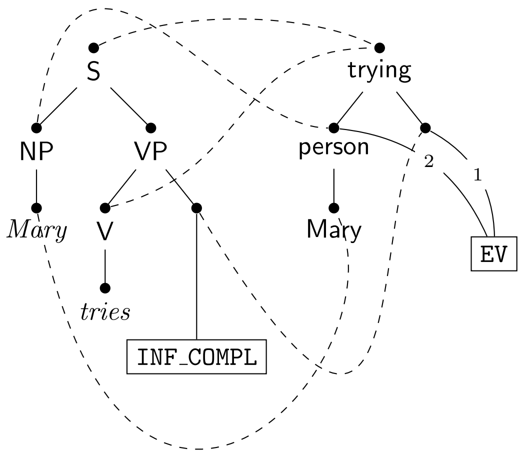
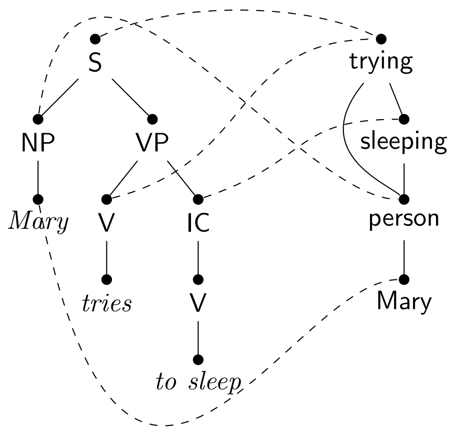


Figure 8: Partial configuration after having read *Mary tries*.

with the start graph on the left-hand side in Figure 5a. Reading *Mary*, as the first word in the sentence, we apply the lexicon entry in Figure 5a leading to the partial Millstream configuration shown in Figure 7.

The nonterminal labels `SYN_ARG` and `THETA_ROLE` in Figure 7 illustrate that the syntactic and semantic roles of *Mary* are not determined yet. Now we read *tries* and apply the lexicon rule in Figure 6b leading to the partial configuration shown in Figure 8. Here, the syntactic and semantic roles of *Mary* have been specified as the syntactic subject and the first semantic argument (agent), respectively. The nonterminal labels `INF_COMPL` and `EV` tell us that on the syntactic side an infinitival complement and on the semantic side an "eventual event" is missing. Reading the word *to sleep* enables the lexicon rule in Figure 6c. Note that the glueing graph or context obtained by deleting the nonterminals on the left-hand side of the lexicon entry for *to sleep* occurs in the partial configuration of *Mary tries* in Figure 8. Applying the lexicon rule for *to sleep* in Figure 6c after having read *to sleep* in the sentence *Mary tries to sleep*,



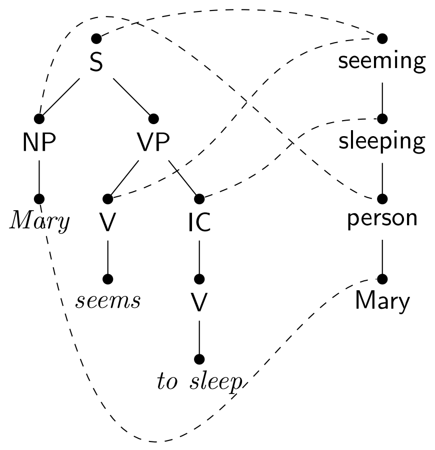


Figure 9: Configurations representing the sentences *Mary tries to sleep* and *Mary seems to sleep*.

replaces the nonterminals labelled `INF_COMPL` and `EV` in the partial configuration in Figure 8 and results in the final Millstream configuration depicted in the upper half of Figure 9. Via a similar reading, using the lexicon entry for *seems* instead of the one for *tries*, the sentence *Mary seems to sleep* gives rise to the configuration in the lower half of Figure 9.

In the context of a reader $\mathcal{R} = (\Sigma, N, W, \Lambda, S)$ let us call a graph $G \in \mathcal{G}_{\Sigma \cup N}$ *straight* if the sequence of attached nodes of each nonterminal in E_G does not contain repetitions, that is, it consists of pairwise distinct nodes. A reader is said to be in *normal form,* if all left-hand and right-hand sides of lexicon rules and all graphs in S are straight. Note that all nonterminal graphs derived by a reader in normal form are straight. The following lemma states that, without loss of generality, all readers can be assumed to be in normal form. This is similar to [20, Theorem I.4.6], where such a result is proved for hyperedge replacement grammars.

Lemma 1. *For every reader $\mathcal{R} = (\Sigma, N, W, \Lambda, S)$ one can construct a reader $\mathcal{R}'$ in normal form such that $\mathcal{R}'(u) = \mathcal{R}(u)$ for all $u \in W^*$.*

Proof. As usual, a partition of a set X is a set $\mathcal{X}$ of disjoint nonempty subsets of X such that $\bigcup_{Y \in \mathcal{X}} Y = X$. In particular, the unique partition of $\emptyset$ is $\emptyset$.

From $\mathcal{R}$ we construct $\mathcal{R}' = (\Sigma, \underline{N}, W, \underline{\Lambda}, \underline{S})$ as follows.

Consider a nonterminal label $A \in N$ and let k be its rank. For each partition π of the set $\{1, 2, \ldots, k\}$, let A_π be a new non-terminal of rank $|\pi|$. Let

$$\underline{N} = \{A_\pi \mid A \in N, \pi \text{ is a partition of } \{1, 2, \ldots, \mathsf{rk}(A)\}\}$$

be the set of such new nonterminals.

Next, for each graph $G \in \mathcal{G}_{\Sigma \cup N}$, we construct the graph $\underline{G} \in \mathcal{G}_{\Sigma \cup \underline{N}}$ as follows.

Consider a nonterminal $e \in E_G$ and assume that $\mathsf{att}_G(e) = v_1 \cdots v_k$. On the set $\{1, 2, \ldots, k\}$ consider the equivalence relation $\equiv$ defined by

$$i \equiv j \text{ if and only of } v_i = v_j.$$

Let π be the partition defined by $\equiv$. Now, let $\mathsf{lab}_{\underline{G}}(e) = \mathsf{lab}_G(e)_\pi$. From $v_1 \cdots v_k$ remove all nodes v_i such that i is not minimal in its block with respect to π. Define $\mathsf{att}_{\underline{G}}(e)$ to be the resulting sequence of nodes. The graph $\underline{G}$ obtained in this way is straight. Moreover, the function that maps G to $\underline{G}$ is a bijection between the graphs in $\mathcal{G}_{\Sigma \cup N}$ and the straight graphs in $\mathcal{G}_{\Sigma \cup \underline{N}}$.

Now, define $\mathcal{R}' = (\Sigma, \underline{N}, W, \underline{\Lambda}, \underline{S})$ by extending this construction canonically to sets of graphs, lexicon rules, and lexica. Thus $\mathcal{R}'$ is in normal form. Moreover, for every lexicon rule $r = (L ::= R)$ over Σ and N and all $G, H \in \mathcal{G}_{\Sigma \cup N}$, one has $G \Rightarrow_r H$ if and only if $\underline{G} \Rightarrow_{\underline{r}} \underline{H}$. By induction on the length of u this yields the statement of the lemma.

Thus, whenever convenient, we assume in the sequel that a reader is in normal form.

When designing readers, it is useful to be able to make some basic assumptions about the order of the words in the input sentence. The following result makes this possible, as it states that the set of input sentences a reader can process can be restricted by intersection with a regular language.

Theorem 1. *Let $\mathcal{R} = (\Sigma, N, W, \Lambda, S)$ be a reader. For every regular language $L \subseteq W^*$, one can effectively construct a reader $\mathcal{R}' = (\Sigma, N', W, \Lambda', S')$ such that, for every input sentence $u \in W^*$,*

$$\mathcal{R}'(u) = \begin{cases} \mathcal{R}(u) & \text{if } u \in L \\ \emptyset & \text{otherwise.} \end{cases}$$

Proof. Let $A = (W, Q, \delta, q_0, F)$ be a deterministic finite automaton (DFA) accepting L, where Q with $Q \cap N = \emptyset$ is the set of states, $\delta \colon Q \times W \to Q$ is the transition function, q_0 is the initial state, and F is the set of final states. One constructs $\mathcal{R}'$ as follows: Let $N' = N \cup Q$; define the rank of each $q \in Q$ to be zero; add a nonterminal labelled with q_0 to the graphs in S to obtain S'.

Now, $\Lambda' = \Lambda'_0 \cup \Lambda'_1$ is defined as follows. For $w \in W$ and every lexicon entry $r \in \Lambda(w)$, Λ'_0 contains a lexicon entry r^q for every $q \in Q$; each entry r^q results from adding single edges labelled q and $\delta(q, w)$ to the left-hand and right-hand sides of r, respectively. Moreover, with w and q as before, Λ'_1 consists of lexicon entries $r^{q,\mathrm{f}}$ for each q with $\delta(q, w) \in F$; these entries are obtained by adding q to the left-hand side of r and keeping the right-hand side of r unchanged.

Let $u \in W^*$, $G_0 \in S$, and $G \in \mathcal{G}_{\Sigma \cup N}$. Let G'_0 be the graph in S' corresponding to G_0 in S. By induction on the length of u one proves that $G_0 \Rightarrow^*_{\Lambda(u)} G$ if and only if $G'_0 \Rightarrow^*_{\Lambda'_0(u)} G'$, where G' is obtained from G by adding an edge labelled $\delta(q_0, u)$ to it, where δ is extended to W^* in its second argument in the usual manner. Moreover, a reading terminates if and only if a rule $r^{q,\mathrm{f}} \in \Lambda'_1$ is applied, that is, the state reached in A is a final state. Together these facts imply the claim of the theorem.

An important question is: how to prove the correctness of a reader with respect to a given Millstream system. In other words, one would like to guarantee that the language generated by the reader is equal to the language of configurations of the Millstream system. In the next section of this paper we investigate this issue.

4. Correctness of Readers

Throughout this section, we consider nonterminal-bounded readers in normal form. Consider a reader $\mathcal{R} = (\Sigma, N, W, \Lambda, S)$ with $R = \bigcup_{w \in W} \Lambda(w)$, and let $l \in \mathbb{N}$. A derivation $G_0 \Rightarrow_R G_1 \Rightarrow_R \cdots \Rightarrow_R G_n$ is *l-nonterminal-bounded* if none of the graphs G_i for $i \in \{0, \ldots, n\}$ contains more than l nonterminal edges. The reader $\mathcal{R}$ is *l-nonterminal-bounded* or, briefly, *nonterminal-bounded* if all derivations $G_0 \Rightarrow_R G_1 \Rightarrow_R \cdots \Rightarrow_R G_n$ with $G_0 \in S$ are l-nonterminal-bounded. We say that a reader is *effectively nonterminal-bounded* if it is nonterminal bounded and an explicit bound l as above is known.

In this section we show that the languages of effectively nonterminal-bounded readers are context-free graph languages – more precisely, hyperedge-replacement graph languages. From this, we conclude that it is decidable for such readers whether all resulting configurations constructed are valid configurations of a given regular MSO Millstream system.

Let us remark here that nonterminal-boundedness is, of course, undecidable for graph transformation systems in general, since graph transformation systems can simulate Turing machines. It does not seem to be obvious whether a similar undecidability result holds in the special case of readers. However, the examples in this paper, except the one discussed in Section 5.3, are indeed nonterminal-bounded. Roughly speaking, this is because nonterminal-boundedness can only be violated if there are cycles that allow one to create a nonterminal e with a certain label from other nonterminals of which at least one carries the same label as e. Such a cycle occurs in the lexicon rules in Figure 11, which create a nonterminal labelled S from another such nonterminal. However, this cycle does not increase the number of nonterminals in the graph, which means that the reader is nonterminal-bounded.

More generally, the number of nonterminals in the configurations of a reader can be bounded from above by modelling the set of Parikh vectors of derived graphs by a vector addition system; here the Parikh vectors are obtained by counting the numbers of occurrences of nonterminal labels. Since boundedness of vector addition systems is decidable [33], this yields a sufficient condition for nonterminal-boundedness of readers. It does not yield a decision procedure, because reader rules may be disabled if the nonterminals do not appear in the right context.

Next, we recall the definition of hyperedge replacement grammars and languages. For this, given a label A of rank k, let $A^\bullet = (\{1,\dots,k\},\{e\},\mathsf{att},\mathsf{lab})$, where $\mathsf{att}(e) = 1\cdots k$ and $\mathsf{lab}(e) = A$.

Definition 6 (HR grammar and language, cf. [20])**.** *A* hyperedge-replacement grammar, *HR grammar for short, is a tuple* $\Gamma = (N,\Sigma,P,S)$*, where*

- N *and* Σ *are ranked alphabets of* nonterminal *and* terminal labels*, respectively,*
- P *is a finite set of* HR rules *(see below), and*
- S *is a finite set of start graphs.*

An HR rule is a lexicon rule of the form $A^\bullet ::= R$*, where* $A \in N$ *and* R *is straight.*

The hyperedge-replacement language *(HR language) generated by* Γ *is*

$$L(\Gamma) = \{G \in \mathcal{G}_\Sigma \mid G_0 \Rightarrow_P^* G \textit{ for some } G_0 \in S\}.$$

Let $\mathcal{R}$ and $\mathcal{R}'$ be readers. We say that $\mathcal{R}$ and $\mathcal{R}'$ are *equivalent* if $\mathcal{R}(u) = \mathcal{R}'(u)$ for every input sentence u. With respect to a reader $\mathcal{R} = (\Sigma,N,W,\Lambda,S)$, we say that a graph G is *linear* if it contains at most one nonterminal.

A lexicon rule is left-linear (right-linear) if its left-hand side (right-hand side) is linear, and it is linear if it is both left-linear and right-linear. The reader $\mathcal{R}$ is linear if each graph in S as well as all lexicon entries in Λ are linear.

An easy but important observation is that a left-linear lexicon rule $r = (L ::= R)$ is an HR rule with an additional context condition: provided that r can be applied to a graph G, its effect is the same as the effect of the HR rule obtained by making r context-free. To make this precise, let e be the unique nonterminal in L. Then $\lfloor r \rfloor$ denotes the HR rule obtained from r by removing all nodes in $V_L \setminus [\mathsf{att}_L(e)]$ and all edges in $E_L \setminus \{e\}$ from both L and R.

Observation 1. *For a left-linear lexicon rule* $r = (L ::= R)$ *and a linear graph* G*, there is at most one graph* H *such that* $G \Rightarrow_r H$*. If this graph* H *exists, then* $G \Rightarrow_{\lfloor r \rfloor} H$*.*

We now show that nonterminal-bounded readers are equivalent to linear ones.

Lemma 2. *For every nonterminal-bounded reader $\mathcal{R}$ there exists an equivalent linear reader $\mathcal{R}'$. If $\mathcal{R}$ is effectively nonterminal-bounded then $\mathcal{R}'$ can be constructed effectively.*

Proof. Let $\mathcal{R} = (\Sigma, N, W, \Lambda, S)$ be nonterminal-bounded with bound l. For a graph $G \in \mathcal{G}_{\Sigma \cup N}$, let $\#(G) = |\{e \in E_G \mid \mathsf{lab}_G(e) \in N\}|$ be the number of nonterminals occurring in G. Thus $\#(G) \leq l$ for all graphs G derived by $\mathcal{R}$.

Choose any total order $<_N$ on N. Assuming that we know l, we show how to construct an equivalent linear reader $\mathcal{R}' = (\Sigma, N', W, \Lambda', S')$.

For every sequence $A_1 <_N A_2 <_N \cdots <_N A_n$ of nonterminal labels with $n \leq l$ let N' contain the label $A_1 \cdots A_n$ of rank $\sum_{i=1}^{n} \mathsf{rk}(A_i)$. A graph $G \in \mathcal{G}_{\Sigma \cup N}$ containing n nonterminals $e_1, \ldots, e_n$ with $n > 0$, where $\mathsf{lab}_G(e_i) = A_i$, can be turned into a graph $\widetilde{G} \in \mathcal{G}_{\Sigma \cup N'}$ by replacing $e_1, \ldots, e_n$ with a single nonterminal e such that $\mathsf{lab}_{\widetilde{G}}(e) = A_1 \cdots A_n$ and $\mathsf{att}_{\widetilde{G}}(e) = \mathsf{att}_G(e_1) \cdots \mathsf{att}_G(e_n)$. For a graph $G \in \mathcal{G}_\Sigma$ we let $\widetilde{G} = G$.

Let $S' = \{\widetilde{G_0} \mid G_0 \in S\}$. For all $w \in W$ one modifies the lexicon entries $r = (L ::= R)$ in $\Lambda(w)$ as follows: Since an application of r does not necessarily replace all nonterminals in the current partial configuration, first, our construction has to extend r, turning it into several rules that cover the different cases. For this, let $\Delta = l - \max(\#(L), \#(R))$. Since we may assume, without loss of generality, that $\#(L) \leq l$ and $\#(R) \leq l$, Δ is non-negative. Now, construct all rules $r_+ = (L_+ ::= R_+)$ that can be obtained from r by adding at most Δ pairwise distinct new nonterminals to L. Each attached node of these new nonterminals is either a node in V_L or a new node not in $V_L \cup V_R$, which is then also added to L. The right-hand side R_+ is obtained from R by adding exactly the same new nonterminals and nodes to R that were added to L. Thus, in effect, r_+ just matches the additional nonterminals that may be present in the graph to which it is applied, but keeps them unchanged. Now, $\Lambda'(w)$ consists of all rules $\widetilde{r}_+$, for every r_+ constructed in this way. For $G_0 \in S$ one proves, by induction on the length of u, that $G_0 \Rightarrow^*_{\Lambda(u)} G$ if and only if $\widetilde{G_0} \Rightarrow^*_{\Lambda'(u)} \widetilde{G}$ for all $G \in \mathcal{G}_{\Sigma \cup N}$ with $\#(G) \leq l$. This completes the proof.

Thanks to Lemma 2, it suffices to consider linear readers $\mathcal{R} = (\Sigma, N, W, \Lambda, S)$ throughout the rest of this section. Accordingly, we are mainly interested in straight graphs $G \in \mathcal{G}_{\Sigma \cup N}$ containing exactly one nonterminal e. Such graphs are called *unary*. To express that a graph G is unary and that its unique nonterminal carries the label A, we denote G by $G_{[A]}$. A lexicon rule is called unary if it is of the form $L_{[A]} ::= R_{[B]}$ for nonterminal labels A and B. Since an HR rule r with the left-hand side $A^\bullet$ necessarily applies to $G_{[A]}$, there is exactly one graph H such that $G \Rightarrow_r H$ (see Observation 1). In the following, we denote this graph by $r(G)$. If r is linear (unary) then $r(G)$ is obviously linear (unary) as well.

Let the *size* $|G|$ of a graph G be the sum of the number of edges and the number of isolated nodes[7] of G. Given a number $k \in \mathbb{N}$ and a graph G, a *k-subgraph of G* is a maximal subgraph of G of size at most k containing a nonterminal of G. We denote by $sub_k(G)$ the set of all k-subgraphs of G.[8] Note that this set is finite for every G and k because, for finite Σ and N, the set of graphs of size at most k is finite. We only use $sub_k(G)$ for graphs G that are either unary or terminal. In the first case, each graph in $sub_k(G)$ contains the nonterminal of G, and in the second case $sub_k(G)$ is empty.

Lemma 3. *Let $r = (L ::= R)$ be a unary lexicon rule, let G and H be unary graphs, and consider a step $G \Rightarrow_r H$. For every $k \in \mathbb{N}$*

$$sub_k(H) = \bigcup_{G' \in sub_k(G)} sub_k(\lfloor r \rfloor(G')).$$

Proof. Let $p = \lfloor r \rfloor$. By Observation 1, $G \Rightarrow_r H$ implies that $H = p(G)$. Hence, we have to prove that

$$sub_k(p(G)) = \bigcup_{G' \in sub_k(G)} sub_k(p(G')). \tag{1}$$

We consider the two inclusions separately.

Let J be a k-subgraph of $p(G)$ and assume, without loss of generality, that $|J| = k$. Since the nonterminal of $p(G)$ is in J, the intersection of J and G is of size at most $k-1$. Hence, adding e and its attached nodes to this intersection yields a subgraph of G of size at most k, which means that this graph is a subgraph of a graph $G' \in sub_k(G)$. Obviously, $J \in sub_k(p(G'))$.

Conversely, let $G' \in sub_k(G)$ and $J \in sub_k(p(G'))$. By the definition of k-subgraphs and of hyperedge replacement, $sub_k(p(G')) \subseteq sub_k(p(G))$ and thus $J \in sub_k(p(G))$, as required.

This proves Equation 1 and, therefore, the lemma.

Theorem 2. *For every effectively nonterminal-bounded reader $\mathcal{R} = (\Sigma, N, W, \Lambda, S)$, one can effectively construct an HR grammar Γ such that $L(\Gamma) = L(\mathcal{R})$.*

Proof. By Lemma 2, we can assume that $\mathcal{R}$ is linear. Let k be the size of the largest left-hand side L of lexicon entries $L ::= R$ in Λ. For every $A \in N$, let $\mathcal{S}_A = \{G_{[A]} \in \mathcal{G}_{\Sigma \cup N} \mid |G_{[A]}| \leq k\}$. We construct a new reader $\mathcal{R}' = (\Sigma, N', W, \Lambda', S')$ with $N' = \{\langle A, \mathcal{S} \rangle \mid \mathcal{S} \subseteq \mathcal{S}_A\}$. The idea behind the construction is to keep track of $sub_k(G)$ in every intermediate graph G. This is done by augmenting the label of the unique nonterminal in G with $sub_k(G)$. For this, we construct Λ' by taking copies of the lexicon rules in Λ and replacing the nonterminal labels in their left-hand and right-hand sides by labels in N' in

[7] A node is isolated if no edge is attached to it.

[8] Recall that we do not distinguish between isomorphic graphs.

accordance with Lemma 3. For notational convenience, given a linear graph G and a set $\mathcal{S} \subseteq \mathcal{S}_A$, where $A \in N$, let $\langle G, \mathcal{S} \rangle$ denote the graph obtained from G by replacing the label of its nonterminal e, if present, by $\langle A, \mathcal{S} \rangle$. Thus, $\langle G, \mathcal{S} \rangle = G$ if G does not contain a nonterminal. To complete the construction of the reader $\mathcal{R}'$ we define:

- $S' = \{\langle G_0, sub_k(G_0)\rangle \mid G_0 \in S\}$ and
- for all $w \in W$, for all lexicon rules $r = (L_{[A]} ::= R)$ in $\Lambda(w)$ and for all $\mathcal{S} \subseteq \mathcal{S}_A$, $\Lambda'(w)$ contains the rule $\langle L, \mathcal{S}\rangle ::= \langle R, \mathcal{S}'\rangle$ with $\mathcal{S}' = \bigcup_{G \in \mathcal{S}} sub_k(\lfloor r \rfloor(G))$.

By induction on the length of derivations one proves that the following statements are true for all $G_0 \in S$, all linear graphs $G \in \mathcal{G}_{\Sigma \cup N}$, and all $u \in W^*$:

1. If $G_0 \Rightarrow^*_{\Lambda(u)} G$ then $\langle G_0, sub_k(G_0)\rangle \Rightarrow^*_{\Lambda'(u)} \langle G, sub_k(G)\rangle$.
2. Conversely, if $\langle G_0, sub_k(G_0)\rangle \Rightarrow^*_{\Lambda'(u)} \langle G, \mathcal{S}\rangle$ for a set $\mathcal{S}$, then $G_0 \Rightarrow^*_{\Lambda(u)} G$ and $\mathcal{S} = sub_k(G)$.

In particular, $\mathcal{R}'(u) = \mathcal{R}(u)$ for all $u \in W^*$. However, we know more than this. By the second property and the choice of k, if $\langle G_0, sub_k(G_0)\rangle \Rightarrow^*_{\Lambda'(u)} \langle G, \mathcal{S}\rangle$ for a graph $G_{[A]}$, then a lexicon rule $\langle L, \mathcal{S}\rangle ::= R$ in Λ' is applicable to $\langle G, \mathcal{S}\rangle$ if and only if L is a subgraph of a graph in $\mathcal{S}$. Consequently, the reader $\mathcal{R}''$ obtained by removing all lexicon entries $\langle L, \mathcal{S}\rangle ::= R$ such that L is not a subgraph of any of the graphs in $\mathcal{S}$ satisfies $\mathcal{R}''(u) = \mathcal{R}'(u)$ for all $u \in W^*$. By the reasoning above, a lexicon rule r of $\mathcal{R}''$ is applicable to an intermediate derived graph if and only if the label of the nonterminal in the left-hand side of r coincides with the label of the nonterminal in that graph. In other words, r is applicable if and only if $\lfloor r \rfloor$ is applicable. Hence, if $\mathcal{R}'''$ is the reader obtained from $\mathcal{R}''$ by replacing every lexicon rule with the HR rule $\lfloor r \rfloor$, then $\mathcal{R}'''(u) = \mathcal{R}''(u)$ for all $u \in W^*$. Hence, $L(\mathcal{R}) = L(\mathcal{R}''')$.

Based on Theorem 2, one could, in principle, replace nonterminal-bounded readers by HR grammars altogether. However, this would be impractical due to the huge number of rules required. Moreover, readers allow a more direct modelling of linguistic phenomena since all linguistic information, that is, missing syntactic and semantic roles, syntactic and semantic structures developed so far, etc. is visually depicted and not just encoded in a single nonterminal symbol throughout the derivation and in the lexicon rules. Nevertheless, this raises an interesting open question for future research: What is the power of context-free, but not necessarily linear readers?

As a consequence of Theorem 2 it can effectively be checked whether all graphs in the language of a reader satisfy a given MSO formula.

Theorem 3. *The following problem is decidable: Given an effectively nonterminal-bounded reader $\mathcal{R}$ and an MSO formula Φ on graphs as input, is it true that all graphs in $L(\mathcal{R})$ satisfy Φ?*

Proof. It is known that, for an MSO formula Φ and an HR grammar Γ, it is decidable whether all graphs generated by Γ satisfy Φ (see, e.g., [12, Theorem 4.4(3)]). Together with Theorem 2, which shows that we can effectively construct an HR grammar Γ so that $L(\Gamma) = L(\mathcal{R})$, this proves the claim.

As an immediate consequence of Theorem 3 and the definition of MSO Millstream systems, we obtain the main result of this paper.

Corollary 1. *The following problem is decidable: Given an MSO Millstream system MS and an effectively nonterminal-bounded reader $\mathcal{R}$ as input, is it true that $L(\mathcal{R}) \subseteq L(MS)$?*

As discussed in Section 2, regular MSO Millstream systems are a special case of MSO Millstream systems since the regular tree languages are exactly those which can be defined by MSO logic and since the transition from a regular tree grammar to an equivalent MSO formula is effective. Hence, Corollary 1 applies in particular to this class of Millstream systems, which was studied in earlier papers.

The inclusion $L(\mathcal{R}) \subseteq L(MS)$ means correctness or soundness of the reader, while the converse inclusion $L(MS) \subseteq L(\mathcal{R})$ show its completeness. Ideally, one should like both properties to hold. For applications, correctness may be the more important one. Incorrect analyses would be unacceptable; on the other hand, in the case of incompleteness there may be sentences which cannot be analysed. In a dynamic environment this problem might be addressed by modifying the reader.

5. Linguistic Relevance

The main result of the previous section indicates that nonterminal-bounded readers have nice algorithmic properties. One may ask how realistic the assumption of nonterminal-boundedness is. In this section we outline how nonterminal-bounded readers model the processing of certain structures occurring in natural language such as unlimited embedding, *wh*-dependency, and anaphoric reference.

5.1. Unlimited Embedding

In natural language some syntactic constituents can be embedded in other syntactic constituents of the same type. A complement clause (abbreviated CP) is such a syntactic constituent. It consists of a complementiser (C) and its complement which is a sentence (S). Complementisers are words such as *that*, *whether*, *if* and *since*. Figure 10 shows the syntactic structure of the sentence *John claims that Sarah knows that Mary cheated* in which the complement clause *that Mary cheated* occurs within the complement clause *that Sarah knows*.

A deeper embedding of complement clauses occurs in the sentence *The board doubts that Sarah remembers whether John claimed that Mary cheated.* Theoretically, there can be unlimited embeddings in natural language. We model

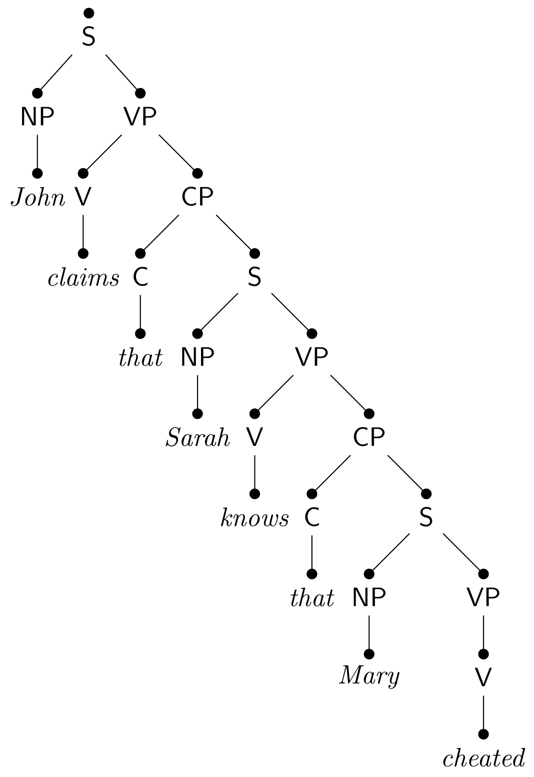
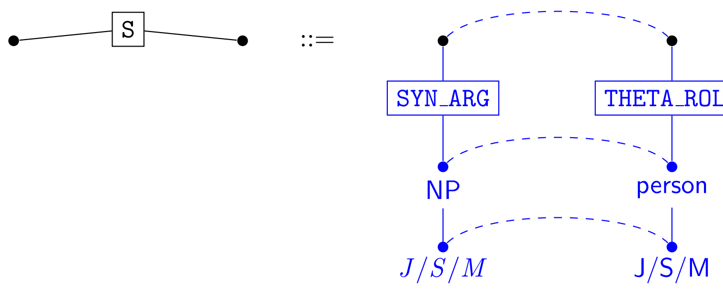


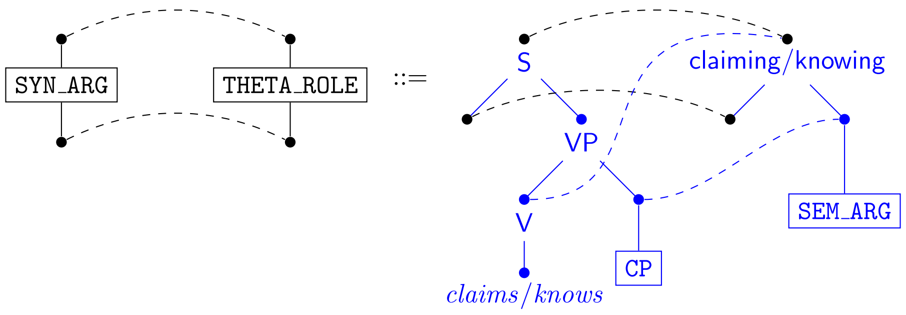
Figure 10: Complement clause *that Mary cheated* occurs within the complement clause *that Sarah knows*.

unlimited embedding with quite simple recursive rules as shown in the sample lexicon entries in Figure 11 for the reading of the sentence *John claims that Sarah knows that Mary cheated.*

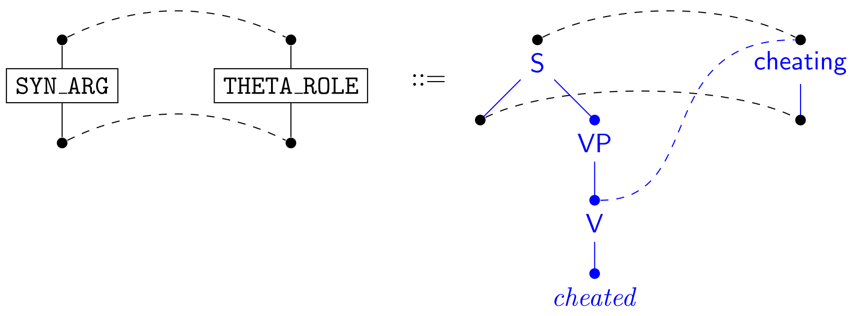
The right-hand side of Figure 11a is a sample lexicon entry for *John* or *Sarah* or *Mary* (similar to the lexicon entry in Figure 5a). The abbreviations $J/S/M$ and J/S/M, on the syntactic and semantic side, refer to *John*, *Sarah*, *Mary*, respectively. Figure 11b shows a sample lexicon entry for *claims* or *knows*. The nonterminal labels `CP` and `SEM_ARG` on the right-hand side of the lexicon entry show that a complement clause and a semantic argument are expected during further reading. Figure 11c shows the lexicon entry for the intransitive verb *cheated*. The right-hand side of this lexicon entry does not contain nonterminals that indicate missing syntactic and semantic categories. Figure 11d shows the lexicon entry for the complementiser *that*. The nonterminal labelled `S` on the right-hand side of the lexicon entry in Figure 11d indicates that an embedded sentence is expected during further reading. Starting with the start graph and applying the lexicon rules in Figure 11 in the order dictated by the sentence *John claims that Sarah knows that Mary cheated* leads to the Millstream



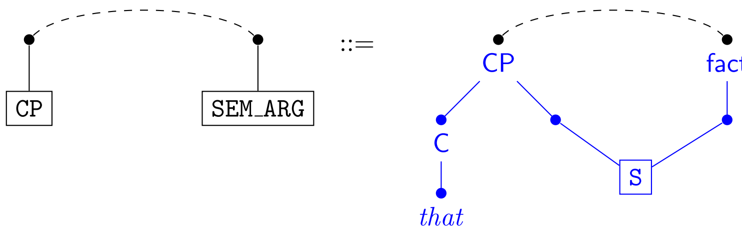
(a) Lexicon entries for *John*, *Sarah* and *Mary*, abbreviated by *J/S/M*.



(b) Lexicon entries for *claims* and *knows*.



(c) Lexicon entry for the intransitive verb *cheated*.



(d) Lexicon entry for the complementiser *that*.

Figure 11: Handling embedding in *John claims that Sarah knows that Mary cheated.*

configuration in Figure 12. In this way, unlimited embedding is modelled. Note that, although the rules in Figure 11 permit the creation of arbitrarily long readings with repeated occurrences of the nonterminal label S, the reader is still nonterminal-bounded since the cycle that turns one S into the next one consumes all intermediate nonterminals it creates.

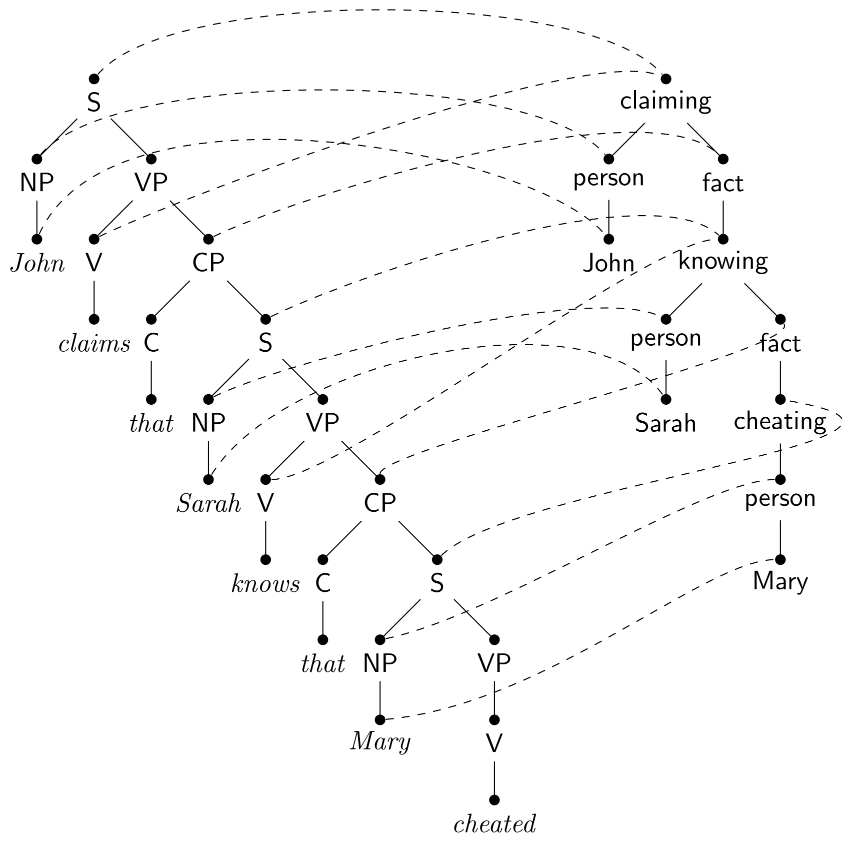


Figure 12: Millstream configuration including the syntactic and semantic structures of the sentence *John claims that Sarah knows that Mary cheated.*

5.2. Wh-*Dependencies*

In English sentences the so-called canonical position of a direct object is located immediately after the verb that assigns the direct object its thematic role. For example, in the sentence *Mary knows that John likes pizza*, the canonical position of the direct object *pizza* is located immediately after the verb *likes* that assigns the direct object its thematic role. In English the position of *wh*-phrases (such as *what*, *who*, *which*, etc.) at the beginning of a sentence indicates a direct question, like in the sentence *Whom does Mary love?* The notion of *wh*-dependencies (or long-distance dependencies) refers to dependencies between words or phrases that do not occur adjacent to each other in a sentence. *Wh*-dependencies arise from a syntactic mechanism called *wh*-movement that takes place in order to form a question. During *wh*-movement, the direct object of a sentence "moves" from its canonical position to the front of the main clause and leaves a "gap" behind which we illustrate by the symbol λ. The position of the fronted wh-phrase is referred to as filler position. For example, in the sentence *Mary loves Peter*, the direct object *Peter* moves to the front in order to form the question *Whom does Mary love?*

There is psycholinguistic evidence [19, 31] that in sentence comprehension the reading (or parsing) of a fronted *wh*-phrase leads to the prediction of a thematic role assigner (typically a verb) and thus to the creation of an incomplete dependency. Gibson [19] associates working memory cost (memory and storage cost) with the processing of *wh*-dependencies. Predictions must be maintained in the working memory and the cost for maintaining them increases as additional words are processed. In Millstream readers these predictions must be kept track of by nonterminals marking the corresponding "construction sites", that is, predicted syntactic or semantic categories to be integrated in the configuration at a later step. Thus, intuitively, the restriction of a Millstream reader $\mathcal{R}$ being nonterminal-bounded corresponds to the fact that humans can only memorize a bounded number of predictions during sentence comprehension.

Figures 13–16 show sample lexicon entries for handling the *wh*-dependency in the question *Whom does Mary love?* The lexicon entry in Figure 13 predicts a thematic role assigner, which is illustrated by the nonterminal label V and which is embedded into the subgraph with the root labeled VP. This subgraph also already marks the gap position with λ. The *wh*-phrase *whom* is linked with an indefinite entity on the semantic side which represents the indefinite entity asked for. The indefinite is connected to the semantic function interrogative as this indefinite entity is what is being asked for. Figure 14 shows a sample lexicon entry for *does*. Figure 15 is a sample lexicon entry for *Mary* occurring in a question. Note that the thematic role for *Mary* is known already (as agent) and that we are not waiting for further reading in order to determine the thematic role for *Mary* as in the lexicon entry in Figure 11a, for example.

Let us now analyse the reading of the question *Whom does Mary love?* Applying first the lexicon rule in Figure 13 and then the lexicon rule in 14, leads us to the partial Millstream configuration for *whom does* shown in Figure 17. In Figure 17 the nonterminal label S illustrates that a sentence is expected and the nonterminal label SEM_ARG illustrates that a semantic argument is expected. The expected thematic role assigner (indicated by the nonterminal labelled V) is within the subgraph with the root labeled S. Applying the lexicon rule in Figure 15 leads us to the partial configuration for *whom does Mary* illustrated in Figure 18. Finally, applying the lexicon rule in Figure 16 leads us to the Millstream configuration of *whom does Mary love* depicted in Figure 19. Figure 19 illustrates on the syntactic side that the direct object has been fronted and left a gap λ behind which is being asked for.

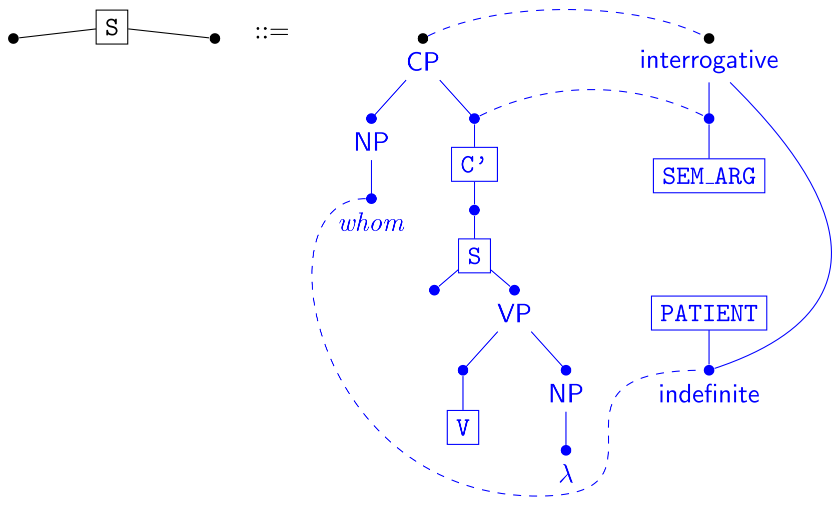


Figure 13: Lexicon entry for fronted *wh*-phrases. The nonterminal labelled `V` is the predicted thematic role assigner.

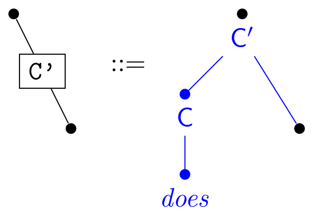


Figure 14: Lexicon entry for *does* as an auxiliar in a question.

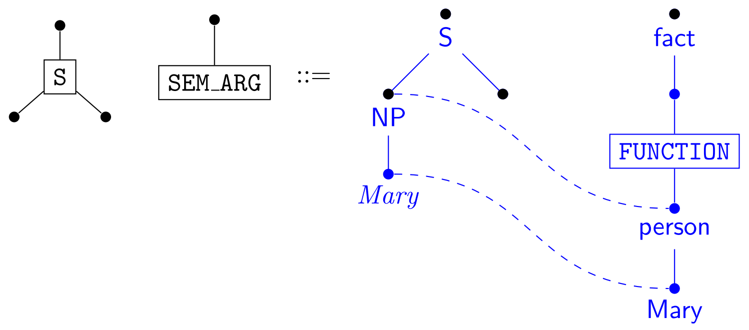


Figure 15: Lexicon entry for *Mary* occurring in a question, where the theta role of *Mary* is known.

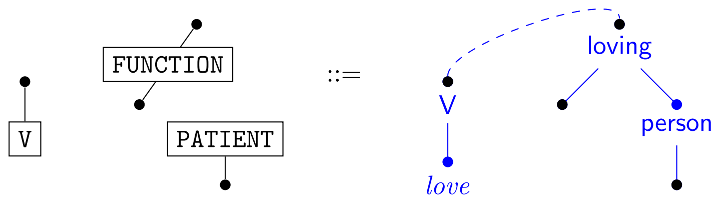


Figure 16: Lexicon entry for *love* as the predicted thematic role assigner for the indefinite entity.

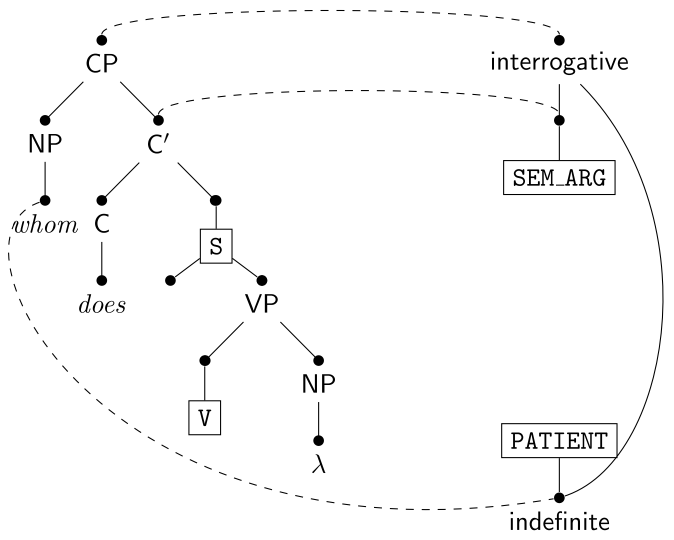


Figure 17: Partial Millstream configuration for *whom does*. The syntactic thematic role assigner and the thematic role of the indefinite entity, indicated by the nonterminals labelled `V` and `PATIENT`, respectively, need to be specified during further reading.

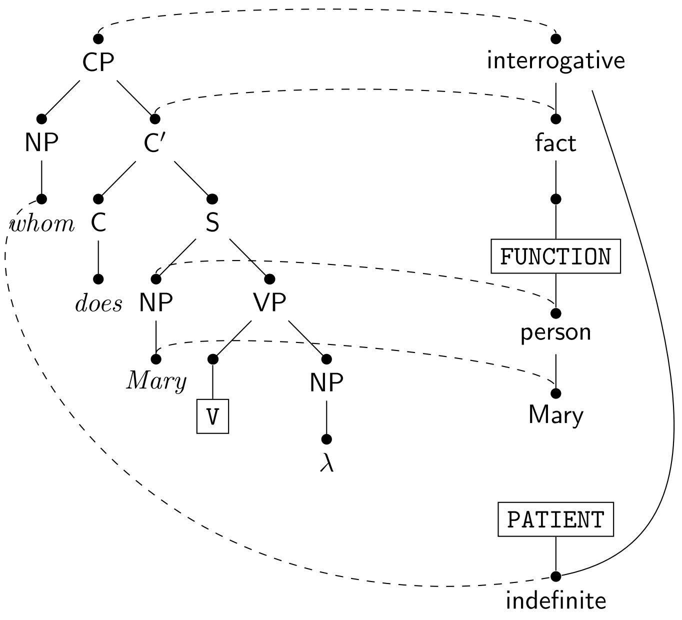


Figure 18: Partial Millstream configuration for *whom does Mary*. The syntactic thematic role assigner and the thematic role of the indefinite entity, indicated by the nonterminals labelled V and PATIENT, respectively, need to be specified during further reading.

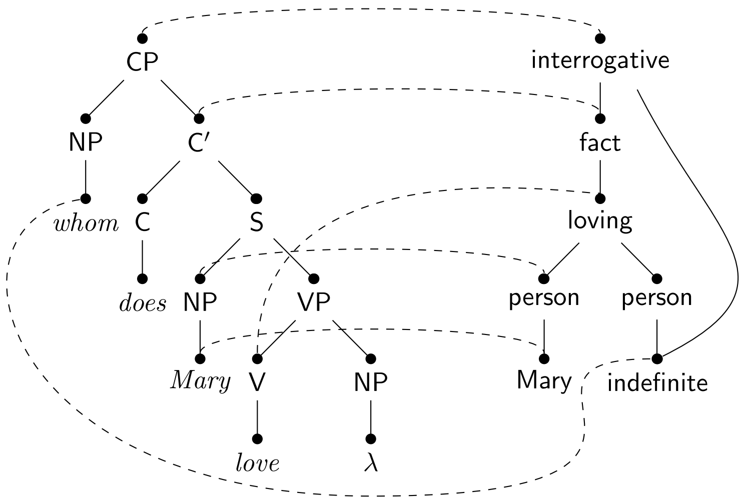
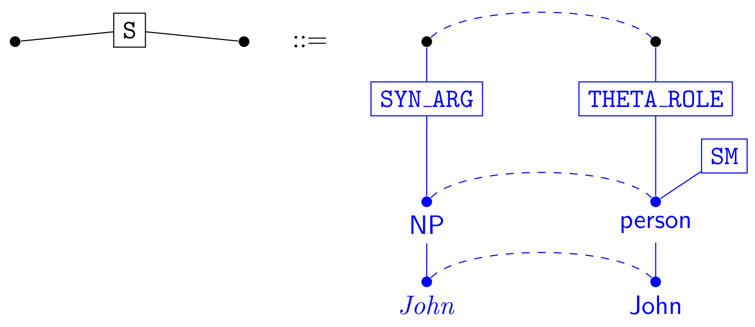
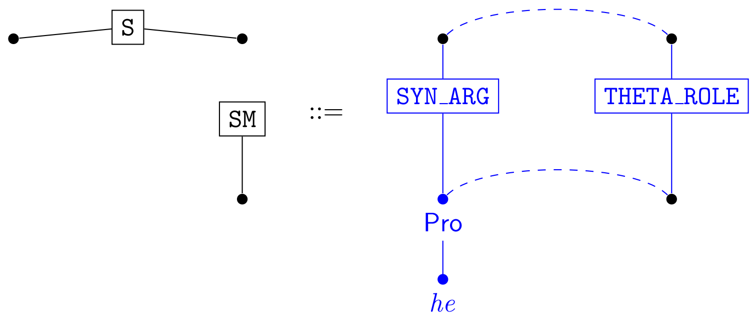


Figure 19: Millstream configuration for the question *Whom does Mary love?*



(a) Lexicon entry for *John*, where the semantic node representing the person John is marked as a possible (or expected) antecendent for a pronoun.



(b) Lexicon entry for the pronoun *he* referring to an antecendent.

Figure 20: Lexicon entries for handling anaphoric reference.

5.3. Anaphoric Reference

We finally discuss an example that, in its pure form, is not nonterminal-bounded.

Anaphoric reference means that a word (or anaphor) refers back to its antecedent, typically a noun or a noun phrase. For example, in the sentence *John thinks that he is smart*, the anaphor *he* refers back to its antecendent *John*. We want to illustrate how to model anaphoric reference by discussing the example sentence *John claims that he cheated*. Figure 20 shows sample lexicon entries for *John* and *he*, where *John* acts as an antecedent for the pronoun *he*. The nonterminal label SM in Figure 20a stands for *some male* and is a “placeholder” for an expected anaphor, *he* in this case.

Starting with the start graph and applying the lexicon rules in Figure 20a, Figure 11b, and Figure 11d, we obtain the partial configuration for *John claims that*, shown in Figure 21. The application of the lexicon entry for *he* shown in Figure 20b turns this into the partial configuration for *John claims that he* in Figure 22. Thus, the the nonterminal labelled SM is replaced by the nonterminal labelled THETA_ROLE and a link to the syntactic constituent Pro of the anaphor *he* is added. One could add a variant of this lexicon entry in

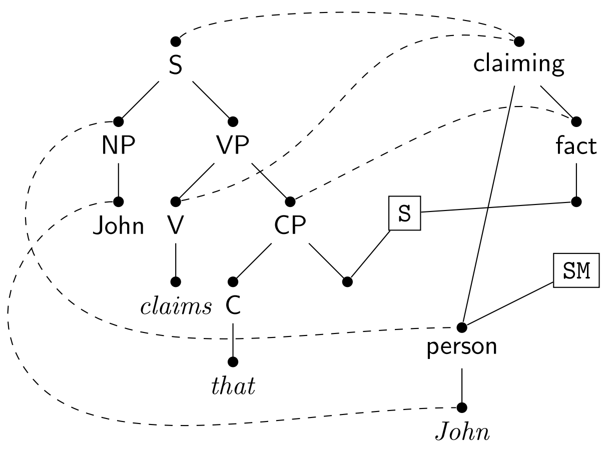


Figure 21: Partial Millstream configuration for *John claims that*, where *John* functions as an antecedent.

which the nonterminal labelled by `SM` is re-created on the right-hand side. This would make it possible to use the same antecedent for several occurrences of pronouns. Finally, Figure 23 shows the complete Millstream configuration for *John claims that he cheated,* which we get after applying the lexicon rule in Figure 11c to the partial configuration in Figure 22. In Figure 23 we see that the semantic argument of the function cheating is connected to the first semantic argument of the function claiming since *John* and *he* refer to the same person. This illustrates the usefulness of sharing.

The extension of our reader by the lexicon rules in Figure 20 gives rise to a nonterminal-unbounded reader. In the loop made possible by the rules in Figure 11 we may replace the repeated applications of the lexicon entry in Figure 11a by the one in Figure 20a, thereby creating an unbounded number of nonterminals labelled `SM`. On the one hand, it seems intuitively clear that this cannot be avoided if we really want to be able to deal with an unbounded number of occurrences of antecedents of anticipated pronouns. On the other hand, this is only needed for sentences in which a large number of pronouns refer to a large number of different antecedents in a non-local manner.[9] From a psycholinguistic point of view such sentences may be disregarded, because no human being could make sense of them as they are too ambiguous even in cases in which the numbers are rather small. For example, consider the sentence

> *John claimed that Bob thinks that Peter suggested that he had told Harry that he had deceived his brother.*

[9]Note that, for instance, sentences of the form $\langle phrase_1\rangle$ *and* $\langle phrase_2\rangle$ *and* $\langle phrase_3\rangle$ ... with local references within the individual phrases do not require nonterminal-unboundedness unless the individual phrases do.

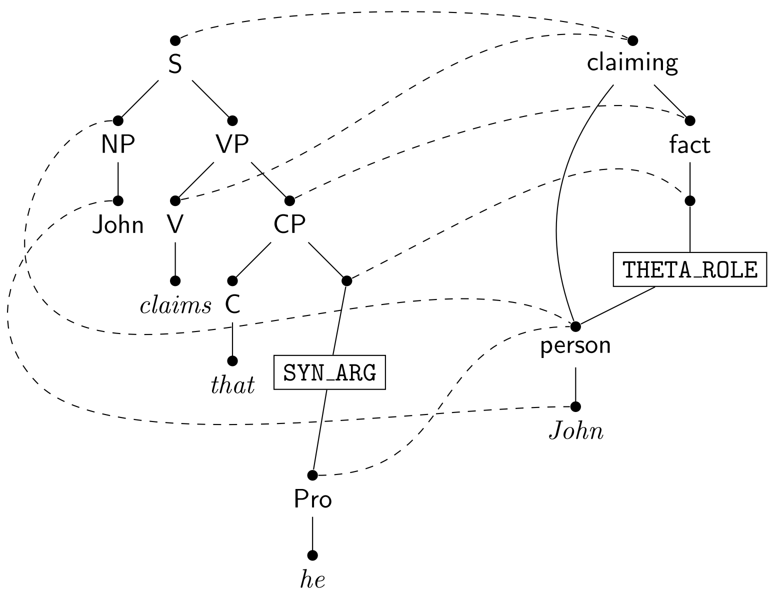


Figure 22: Partial Millstream configuration for *John claims that he*, where *John* functions as an antecedent for the pronoun *he*.

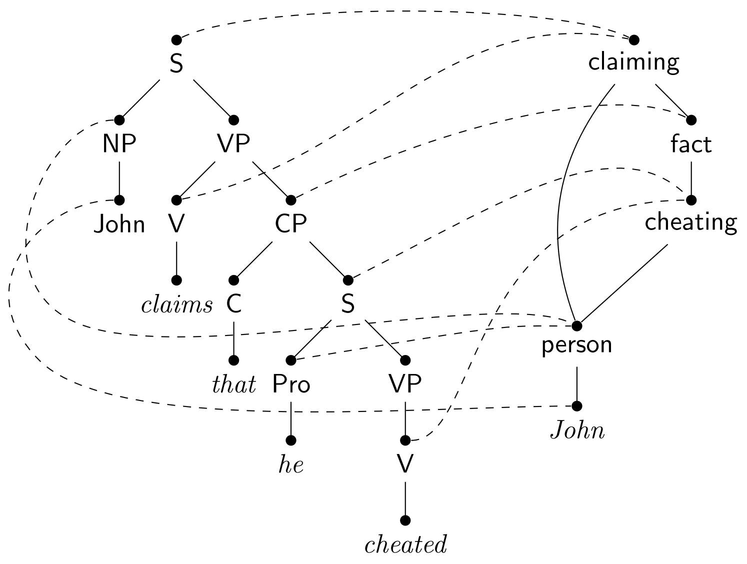


Figure 23: Millstream configuration for *John claims that he cheated*, where *John* functions as an antecedent for the pronoun *he*.

It is probably safe to assume that this sentence, though syntactically correct, would not occur in any real conversation because it is entirely unclear which of the pronouns refer to whom. In fact, even in cases where there is a unique correspondence between pronouns and the persons they refer to, the number of possible antecedents to be kept in mind must be small enough for the human brain to be able to handle them. Hence, it makes sense to think about how a reader $\mathcal{R} = (\Sigma, N, W, \Lambda, S)$ can be restricted so that it becomes nonterminal bounded.

Suppose we want to make sure that at most k copies of `SM` exist in the intermediate graphs at every stage during a reading. Using a similar idea as in the proof of Theorem 1, we may introduce new nonterminal labels '0', ..., 'k' of rank 0. For every start graph $G \in S$, we add another start graph to S, namely G with an added edge labelled by 0. This new edge will act as a counter in those readings which make use of `SM`. For this purpose, the rule in Figure 20a is replaced by k rules, each of which is obtained by adding an edge labelled with $i - 1$ to the left-hand side and one labelled with i to the right-hand side, for every $i \in \{1, \ldots, k\}$. In other words, the counter is increased by one each time another copy of `SM` is added. Similarly, the lexicon entry in Figure 20b is replaced by k entries that decrease the counter from $i \in \{1, \ldots, k\}$ to $i - 1$. Another copy of that rule allows one to remove the counter in the case of $i = 1$. The latter corresponds to the nondeterministic guess that no more copies of `SM` will occur in the remainder of the reading.

There are, of course, other ways in which counting can be implemented, but this one has the advantage that only rules which affect the counted quantity need to be copied and modified, or replaced by rule schemata.

Another method to force nonterminal-boundedness can be obtained directly from the construction leading to Theorem 2. Recall that the construction in the proof of Lemma 2 relies on knowing a nonterminal bound l of the reader. However, the construction works just as well if we simply choose the desired bound. Of course, the resulting linear reader will not be equivalent with the original reader any more. Instead, it will implement exactly the l-nonterminal-bounded readings of the original reader. In other words, the desired nonterminal bound can be passed as a parameter to the construction and Corollary 1 enables us to check whether all l-nonterminal-bounded readings yield correct configurations with respect to the given MSO Millstream system.

6. Conclusions and Future Work

In this paper, we have presented *readers* – graph transformation systems that incrementally perform a linguistic analysis of a sentence while reading that sentence from left to right. The result of this analysis is a Millstream configuration, that is, a graph of the kind provided by Millstream systems [4, 5], possibly containing shared substructures. The graph transformation rules of a reader are organized in a so-called lexicon. It associates a set of graph transformation rules with each possible input word. Reading a sentence means to apply lexicon entries, that is, graph transformation rules associated with the words in the sentence in the order in which these words appear. Intuitively, such a step integrates the information provided by the respective word into the partial configuration, thus building up the final configuration in a stepwise manner. Our examples show that nontrivial linguistic phenomena can be covered in this way, but more research will be required to find out exactly how the lexicon entries should be formed to be able to design readers that work well even for examples of realistic size. We sketch a few of the most important aspects to be addressed by future work.

Building Lexica in a Consistent Manner. The lexicon entries presented in this paper have been designed by hand in an ad-hoc manner. In particular, nonterminals have been invented where they seemed to be required, and they were used in a way that intuitively seemed to make sense. However, the development of lexica with tens of thousands of entries must follow well-defined design patterns, and the meaning of a given nonterminal in such a lexicon has to be specified in a reasonably precise way to make lexica extensible and enable the engineering of lexica, possibly even the automatic generation of lexica.

Degree of Nondeterminism. Reading a sentence may involve nondeterminism. On the one hand, the left-hand side of a given lexicon rule may match several places in the current partial configuration. On the other hand, two or more lexicon entries for the current word may apply. However, it is not unreasonable to hope that the degree of nondeterminism will usually be rather small in well-designed readers. The reason for this is that, although there will usually be several lexicon entries for a given word, the situations in which they apply are typically not the same. Since nondeterminism and, therefore, backtracking is a possible source of inefficiency, this question should be studied further.

Certainly, the way in which lexica are designed will affect the degree of nondeterminism. For example, one may distinguish between "lazy" and "eager" strategies. Imagine a situation where a word can give rise to two interpretations, so that more information coming later in the sentence is needed to find out which interpretation is the right one, thus giving rise to two different substructures in the configuration. An eager strategy would nondeterministically predict any of the two structures, that is, there would be two lexicon entries taking care of one interpretation each. In contrast, a lazy strategy would only create the common part of both structures, remembering the remaining information in the nonterminals until later, when the ambiguity can be resolved. Clearly, both strategies have their advantages and disadvantages. The eager strategy increases the amount of nondeterminism and the number of lexicon entries, but is also expected to lead to fewer nonterminals and smaller and less complex rules.

Rule Schemata. In implementations of practically relevant size, it obviously makes no sense to establish independent lexicon entries for all words of the language. This becomes most obvious when proper names such as *Mary* and *Peter* are concerned, but applies to many other word classes as well. For example, transitive verbs can be handled alike in many ways. However, the details might be more difficult even in supposedly easy cases: in the sentence *Mary likes wine, but she hates beer* one cannot substitute *Peter* for *Mary* because Mary's gender is reflected by the personal pronoun *she*. Thus, future work should study rule schemata that allow to specify lexicon rules which can be instantiated in flexible ways to create concrete lexicon entries.

Probabilities and Training. It seems that humans use a mixture of the lazy and eager strategies mentioned above, combined with probabilities: On the one hand, if one interpretation is (much) more probable, we tend to process a sentence according to this interpretation until it turns out that a re-interpretation is necessary. On the other hand, if several interpretations are equally reasonable, we tend to keep things undecided until the ambiguity is resolved.

This indicates that it may be a good idea to introduce lexicon rules with probabilities, and to use an eager strategy for alternatives with considerably different probabilities, but use a lazy strategy for the remaining cases. Probabilities immediately make training algorithms become a necessity, because it is obviously too difficult and cumbersome a task to determine the right probabilities by hand. Instead, one should develop training algorithms that derive appropriate probabilities for a given set of lexicon entries from corpora.

Computability and Complexity. Clearly, the construction leading to Corollary 1 is of too high complexity to be of immediate practical use. In fact, this is even the case if one considers Theorem 2 alone, bedause of the enormous size of the grammar constructed in its proof. Nevertheless, the result shows that reasonably general readers[10] have favourable algorithmic properties. Therefore, it makes sense to study further computability and complexity issues in connection with readers and Millstream systems. In particular, one should study special cases such as readers containing only lexicon rules with connected left-hand sides or even context-free readers, that is, readers the lexicon rules of which are context-free.

Corollary 1 leaves an interesting question open, namely under which conditions it is also possible to decide whether $L(MS) \subseteq L(\mathcal{R})$. A positive result regarding this question would enable us to check total correctness of nonterminal-bounded readers. It seems likely that total correctness is undecidable even for nonterminal-bounded readers, but there does not seem to be an obvious proof.

Future research will also have to study efficient algorithms for the construction of lexica and, once a lexicon is given, of readings. Language processing in general, regardless of whether natural languages or artifical languages are concerned, needs to be performed in near-realtime. Upon reading an input sentence, with a very short delay, possibly independently of the length of the sentence, an understanding of the input should be available. For natural language, the output may be the translation into another language or, for example, a description of the semantics of the input sentence. In this case, near-realtime processing is a

[10]Recall from Section 2 that nonterminal-boundedness, the only assumption used in Theorem 2, is not an unrealistically strong assumption.

must, because otherwise no reasonable communication is possible. The importance of near-realtime processing goes hand in hand with the fact that human language processing is an incremental process. Thus, future research will have to study the complexity of constructing readings from input sentences.

In a uniform setting, that is, when the reader is not considered to be fixed, finding and applying a lexicon rule requires solving the subgraph isomorphism problem, which is NP-complete. However, in practical cases the problem should nevertheless be solvable in a rather efficient manner, because typically only a reasonably small number of lexicon rules are associated with a given input word, and partial configurations as well as the left-hand sides of rules tend to consist of connected subgraphs of the k term graphs of a configuration, with links in between. The exact impact of these facts on the complexity of rule application remains to be studied, however. Even optimisation algorithms that modify lexica according to certain criteria could be of interest. These could, for instance, make lexica more compact, increase laziness or eagerness, make them more suitable for fast pattern matching, etc.

Practical Implementation. An implementation of readers that allows us to implement and execute readers of realistic size must be made. Such an implementation should be based on a general graph transformation engine such as AGG, GP, and GrGen.NET [34, 32, 17]. Currently, one of our students works on a prototype implementation of readers.

Other Applications of Graph Transformation to Natural Language Processing. Clearly, there are many more tasks in natural language processing that graph transformation could be useful for. HR grammars may be used to describe, for instance, models of natural language that include sharing of substructures, a situation which occurs naturally as described in Section 5 or to handle trees with back references, giving rise to cycles. Graph transformation rules may be used to specify and implement model transformations that take one graph as input and yield another one as output. The computed transformation may, for example, be a translation between different languages, it may simplify a given linguistic analysis, augment it with additional information, or analyse it further. Moreover, since the union of two graphs is a graph, graph transformation is ideally suited as a formal way of studying, expressing and implementing procedures that take several graphs as input or yield several as output.

Acknowledgements

We are grateful to the referees whose comments helped us to improve this paper, and in particular to one referee who pointed out several errors and inaccuracies.

Research reported in this paper was in part supported by a grant from the Natural Sciences and Engineering Research Council of Canada and a grant from the Swedish Governmental Agency for Innovation Systems (VINNOVA).

References

[1] *ACL 2010. 48th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, 11–16 July 2010, Uppsala, Sweden.* Uppsala Universitet, 2010. Available at `http://www.aclweb.org/anthology/P/P10`.

[2] M. A. Arbib, Y. Give'on: Algebra automata I: Parallel programming as a prolegomena to the categorical approach. *Information and Control* **12** (1968), 331–345.

[3] M. Bauderon, B. Courcelle: Graph expressions and graph rewriting. *Mathematical Systems Theory* **20** (1987), 83–127.

[4] S. Bensch, F. Drewes: Millstream systems. Report UMINF 09.21, Department of Computing Science, Umeå University, Umeå, Sweden, 2009.

[5] S. Bensch, F. Drewes: Millstream systems – a formal model for linking language modules by interfaces. In F. Drewes, M. Kuhlmann (editors): *Proc. ACL 2010 Workshop on Applications of Tree Automata in Natural Language Processing (ATANLP 2010).* 28–36. Association for Computational Linguistics, 2010.

[6] S. Bensch, F. Drewes, H. Björklund: Algorithmic properties of Millstream systems. In Y. Gao, H. Lu, S. Seki, S. Yu (editors): *Proc. 14th Intl. Conf. on Developments in Language Theory (DLT 2010). Lecture Notes in Computer Science* **6224**, 54–65, Springer-Verlag, Berlin, 2010.

[7] S. Bensch, F. Drewes, H. Jürgensen, B. van der Merwe: Incremental construction of millstream configurations using graph transformation. In *Proc. 9th Intl. Workshop on Finite State Methods and Natural Language Processing (FSMNLP 2011).* 93–97. Association for Computational Linguistics, 2011.

[8] S. Bensch, F. Drewes, H. Jürgensen, B. van der Merwe: Correct readers for the incremental construction of Millstream configurations by graph transformation. Report UMINF 12.17, Umeå University, 2012.

[9] N. Beuck, A. Köhn, W. Menzel: Incremental parsing and the evaluation of partial dependency analyses. In K. Gerdes, E. Hajičová, L. Warner (editors): *depling 2011. Proceedings, International Conference on Dependency Linguistics, Depling 2011, Barcelona, September 5–7 2011. Exploring Dependency Grammar, Semantics and the Lexicon.* 290–299. depling, 2011. Available at `http://www.depling.org/proceedingsDepling2011/`.

[10] F. Costa, P. Frasconi, V. Lombardo, G. Soda: Towards incremental parsing of natural language using recursive neural networks. *Applied Intelligence* **19** (2003), 9–25.

[11] F. Costa, P. Frasconi, V. Lombardo, P. Sturt, G. Soda: Ambiguity resolution analysis in incremental parsing of natural language. *IEEE Trans. Neural Networks* **16** (2005), 959–971.

[12] B. Courcelle: Graph rewriting: An algebraic and logic approach. In J. van Leuwen (editor): *Handbook of Theoretical Computer Science*, B. 193–242. Elsevier, Amsterdam, 1990.

[13] F. Drewes, A. Habel, H.-J. Kreowski: Hyperedge replacement graph grammars. In G. Rozenberg (editor): *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. 1: Foundations.* Ch. 2, 95–162. World Scientific, Singapore, 1997.

[14] F. Drewes, B. Hoffmann, M. Minas: Contextual hyperedge replacement. In A. Schürr, D. Varró, G. Varró (editors): *Applications of Graph Transformations with Industrial Relevance - 4th International Symposium, AGTIVE 2011, Budapest, Hungary, October 4-7, 2011, Revised Selected and Invited Papers. Lecture Notes in Computer Science* **7233**, 182–197, Springer-Verlag, Berlin, 2012.

[15] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer: *Fundamentals of Algebraic Graph Transformation. Monographs in Theoretical Computer Science. An EATCS Series.* Springer-Verlag, Berlin, 2006.

[16] J. Engelfriet: Context-free graph grammars. In G. Rozenberg, A. Salomaa (editors): *Handbook of Formal Languages.* Vol. 3: *Beyond Words.* Ch. 3, 125–213. Springer-Verlag, Berlin, 1997.

[17] R. Geiß, G. V. Batz, D. Grund, S. Hack, A. Szalkowski: GrGen: A fast spo-based graph rewriting tool. In *Proc. 3rd Intl. Conf. on Graph Transformation (ICGT 2006). Lecture Notes in Computer Science* **4178**, 383–397, Springer-Verlag, Berlin, 2006.

[18] C. Ghezzi, D. Mandrioli: Incremental parsing. *IEEE Trans. Progr. Languages and Systems* **1** (1979), 58–70.

[19] E. Gibson: Dependency locality theory: A distance-based theory of sentence processing difficulty. In A. Marantz, Y. Miyashita, W. O'Neil (editors): *Image, Language, Brain: Papers from the First Mind Articulation Project Symposium.* 95–126, MIT Press, Cambridge, MA, 2000.

[20] A. Habel: *Hyperedge Replacement: Grammars and Languages. Lecture Notes in Computer Science* **643**. Springer-Verlag, Berlin, 1992.

[21] A. Habel, H.-J. Kreowski: May we introduce to you: Hyperedge replacement. In *Proceedings of the Third Intl. Workshop on Graph Grammars and Their Application to Computer Science. Lecture Notes in Computer Science* **291**, 15–26, Springer-Verlag, Berlin, 1987.

[22] A. Habel, H.-J. Kreowski, D. Plump: Jungle evaluation. *Fund. Inform.* **15** (1991), 37–60.

[23] H. Hassan, K. Sima'an, A. Way: A syntactic language model based on incremental CCG parsing. In *2008 IEEE Workshop on Spoken Language Technology SLT 2008, Proceedings, December 15–18, 2008, Goa, India.* 205–208. IEEE Press, 2008.

[24] C. F. Hockett: Grammar for the hearer. In R. Jakobson (editor): *Structure of Language and Its Mathematical Aspects. Proceedings of the Twelfth Symposium in Applied Mathematics, Held in New York City, April 14–15, 1960. Proceedings of Symposia in Applied Mathematics* **XII**, 220–236, American Mathematical Society, Providence, Rhode Island, 1961.

[25] L. Huang, K. Sagae: Dynamic programming for linear-time incremental parsing. In ACL2010 [1], 1077–1086. Available at `http://www.aclweb.org/anthology//P/P10/P10-1110.pdf`.

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

[27] B. Jones, J. Andreas, D. Bauer, K. M. Hermann, K. Knight: Semantics-based machine translation with hyperedge replacement grammars. In M. Kay, C. Boitet (editors): *Proc. 24th Intl. Conf. on Computational Linguistics (COLING 2012): Technical Papers.* 1359–1376, 2012.

[28] J. Kunze: *Abhängigkeitsgrammatik. studia grammatica* **XII**. Akademie-Verlag, Berlin, 1975, 504 pp.

[29] P. C. R. Lane, J. B. Henderson: Incremental syntactic parsing of natural language corpora with simple synchrony networks. *IEEE Trans. Knowledge and Data Engineering* **13**(2) (2001), 219–231.

[30] J. Nivre: Algorithms for deterministic incremental dependency parsing. *Computational Linguistics* **34** (2008), 513–553.

[31] C. Phillips, N. Kazanina, S. H. Abada: ERP effects of the processing of syntactic long-distance dependencies. *Cognitive Brain Research* **22** (2005), 407–428.

[32] D. Plump: The graph programming language GP. In *Proc. 3rd Intl. Conf. on Algebraic Informatics (CAI 2009). Lecture Notes in Computer Science* **5725**, 99–122, Springer-Verlag, Berlin, 2009.

[33] C. Rackoff: The covering and boundedness problems for vector addition systems. *Theoretical Computer Science* **6** (1977), 223–231.

[34] G. Taentzer: AGG: A graph transformation environment for modeling and validation of software. In *Proc. 2nd Intl. Workshop on Applications of Graph Transformations with Industrial Relevance (AGTIVE 2003). Lecture Notes in Computer Science* **3062**, 446–453, Springer-Verlag, Berlin, 2004.

[35] R. Taraban, J. L. McClelland: Constituent attachment and thematic role assignment in sentence processing: Influences of content-based expectations. *Journal of Memory and Language* **27** (1988), 597–632.

[36] T. A. Wagner, S. L. Graham: Efficient and flexible incremental parsing. *IEEE Trans. Progr. Languages and Systems* **20** (1998), 980–1013.

[37] S. Wu, A. Bachrach, C. Cardenas, W. Schuler: Complexity metrics in an incremental right-corner parser. In ACL2010 [1], 1189–1198. Available at `http://www.aclweb.org/anthology//P/P10/P10-1121.pdf`.