Correct Readers for the Incremental Construction of Millstream Configurations by Graph Transformation*

Suna Bensch[1], Frank Drewes[1], Helmut Jürgensen[2], and Brink van der Merwe[3]

[1] Dept. of Computing Science, Umeå University (Sweden)
{suna, drewes}@cs.umu.se
[2] Dept. of Computer Science, The University of Western Ontario, London (Canada)
hjj@csd.uwo.ca
[3] Dept. of Computer Science, Stellenbosch University (South Africa)
abvdm@cs.sun.ac.za

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 rules for the graph transformations.

1 Introduction

Millstream systems model different aspects of language structure in a parallel and co-ordinated way [2, 3]. A Millstream configuration of a sentence represents the analysis of that sentence with respect to different criteria with 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. The main 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 independent of the number and types of linguistic aspects considered, we illustrate them by a rather simple example that considers only the issues of syntax and semantics – and even these in a very much simplified setting. Space limitations do not permit full-blown examples. However, we believe that the readers will be able to extrapolate from our presentation.

Language processing in general, regardless of whether natural languages or artifical languages are concerned, is most useful if it occurs in quasi-realtime. Upon reading an input sentence, with a very short delay, possibly independent

* This article is a revised and extended version of [4]

of the length of the sentence, an understanding of the input should be available. If the language is a programming languge, the output may be a compiled program. If the language is a 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, quasi-realtime processing is a must, because otherwise no reasonable communication is possible. As a goal one looks for some representation of the input sentence, covering those aspects that are needed for the intended application. There are two important points: (1) One wants to construct the target representation in quasi-realtime; (2) one does not want to neglect important aspects.

Human language processing is an incremental process. This view is supported by various psycholinguistic and cognitive neuroscience-based studies (see for example [21]). One does not postpone the analysis of an utterance or sentence until it is complete; rather, one starts processing the sentence immediately when hearing or reading 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 [13] and by Wagner and Graham [22]; studies of incremental parsing for natural languages using various grammar models and various computing paradigms by Beuck et al. [5], Costa et al. [7, 6], Hassan et al. [16], Huang and Sagae [17], Lane and Henderson [19], Nivre [20] and Wu et al. [23]. 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 them. The structural representation is intended to reflect all relevant aspects as described by a single formal grammar.

We believe 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. A Millstream system consists of not just one, but several formal modules, possibly of quite different mathematical nature, which are connected by interface conditions. With each of the modules representing one of the linguistic levels, these aspects can be considered in parallel with links establishing the required connections and expressing relevant constraints.

Millstream systems have been introduced in [2, 3] as a generic mathematical framework for the description of natural language. These systems describe linguistic aspects such as syntax and semantics in parallel and provide the possibility to express the relation between them by so-called interfaces. Roughly speaking, a Millstream system consists of a finite number of *modules* each of which describes a linguistic aspect and an *interface* which describes the depen-

dencies among these aspects. The interface establishes links between the trees given by the modules, thus turning unrelated trees into a meaningful whole called a *configuration*.

Consider – for simplicity – a Millstream system containing only two modules, a syntactic and a semantic one, which models the syntax and semantics of a natural language. A configuration of the Millstream system consisting of two trees with links between them represents the 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. In this paper, we propose 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[4]. 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 likes Peter*, for example, we first apply the 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 the 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. 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 appear in human language processing as well – they are 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 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 Theorem 13 which states that, under certain conditions, this question is decidable for so-called regular MSO Millstream systems. In other words, given a regular MSO Millstream system MS and a reader satisfying the mentioned conditions, it can effectively be checked whether all readings yield correct configurations of MS.

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

Our graph transformation rules use a special case of the well-known double-pushout (DPO) approach [11]. Since general DPO rules give rise to Turing-complete graph transformation systems, they are too powerful for our purposes. 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 [1] and Habel and Kreowski [15]; see also [8, 14, 12, 9]. We base our readers on an extension of hyperedge replacement that allows us to replace several hyperedges at once, and may depend on context. This type of rules was inspired by *contextual hyperedge replacement* as proposed in [10] for applications in software refactoring. However, in that paper only single hyperedges are replaced and the context consists of isolated nodes.

Our paper is structured as follows. In Section 2 we introduce the required formal background about Millstream systems and the representation of Millstream configurations as graphs. We then explain the incremental construction of Millstream configurations by graph transformation in Section 3 using the formal concept of *reader*. In Section 4 we consider the correctness of such readers and prove the main result of the paper. We discuss the results and the next steps of this ongoing research in Section 5.

2 Millstream Configurations as Graphs

Throughout the paper, we let $\mathbb{N}$ denote the set of non-negative integers. For a set S, S^* denotes the set of all finite sequences (or strings) over 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, the reflexive and transitive closure of a binary relation $\Rightarrow$ is denoted by $\Rightarrow^*$.

A configuration in a Millstream system is a tuple of ranked and ordered trees – in our case, it is a pair of such trees consisting of the syntactic and the semantic representations of a sentence – with links between them. The links indicate relations between the nodes and may belong to different categories, that is, they may carry labels. A typical link establishes a relation between two nodes, where one belongs to one tree and the other one belongs another tree. However, in general, links may have an arbitrary arity, and may also link two or more nodes belonging to the same tree.

In this paper, we represent configurations in a way which is suitable for graph transformation. For this, 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, respextively, 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 following, we simply call a ranked alphabet an alphabet. Moreover, we refer to hypergraphs just as graphs and to their hyperedges as edges. The precise definition follows.

Definition 1 (Graph). *Let Σ be an alphabet. A* Σ-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* $e \in E$*.*

The components of a graph G are also 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.

Example 2 (Graph). Figure 1 shows a graph G consisting of nodes $v_1, \ldots, v_5$

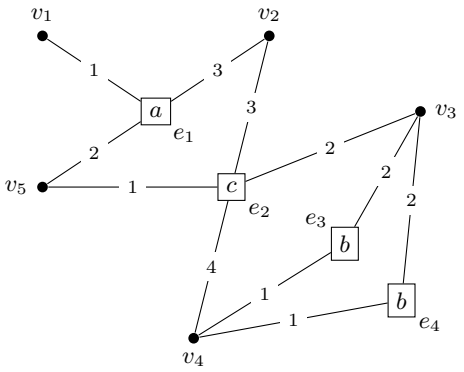
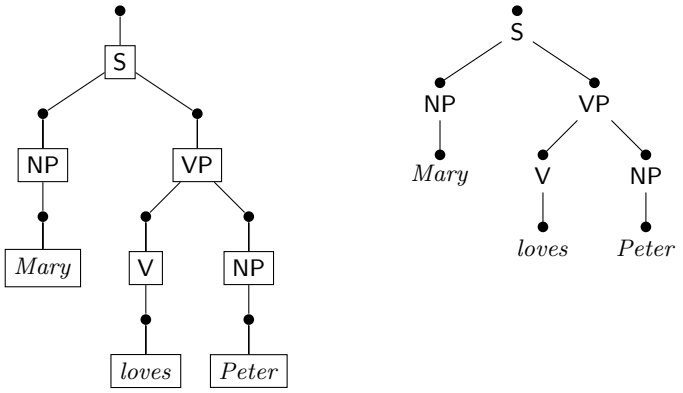


Fig. 1: A graph

and edges $e_1, \ldots, e_4$. The edges are drawn as rectangles with their labels inside. For e_2, for instance, we have $\mathsf{att}_G(e_2) = v_5 v_3 v_2 v_4$ and $\mathsf{lab}_G(e_2) = c$. 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.

Configurations of Millstream systems consist of trees with additional links between their nodes. Trees in this sense are formal expressions built from a set of symbols. Each symbol a has an arity k which determines the number of subtrees it requires. The arity k of a is also indicated by denoting a as $a_{(k)}$. To represent trees as graphs in the sense of the preceding definition, each $a_{(k)}$ is turned into a symbol of rank $k+1$. The first k attached nodes of a hyperedge labelled with e correspond to the roots of the k subtrees, whereas the last attached node corresponds to the root of the tree. However, when building trees using graph transformation, it turns out to be convenient to generalize them in order to allow for shared subtrees. To make this notion of trees precise, consider an alphabet Σ_{t} such that $\mathsf{rk}(a) \geq 1$ for all $a \in \Sigma_{\mathrm{t}}$. A *path of length* n in a graph $G \in \mathcal{G}_{\Sigma_{\mathrm{t}}}$ is an alternating sequence $v_0 e_1 \cdots e_n v_n$ of nodes $v_0, \ldots, v_n \in V_G$ and edges $e_1, \ldots, e_n \in E_G$ such that, for every $i \in \{1, \ldots, n\}$ with $\mathsf{att}_G(e_i) = u_1 \cdots u_{k+1}$, one has $v_{i-1} \in \{u_1, \ldots, u_k\}$ and $v_i = u_{k+1}$. Such a path is also called a (v_0, v_n)-path. By definition, G is a tree, if there is a node $v \in V_G$, its root, such that

- for every node $u \in V_G$, there is a (u, v)-path in G,
- there does not exist any node $u \in V_G$ such that there is a (u, u)-path of length > 0 in G.



(a) A tree in its (hyper)graph representation (where the last attached node of each edge is the one on top)

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

Fig. 2: A tree represented as a (hyper)graph

Figure 2(a) depicts a tree (without sharing) over the alphabet $\mathsf{S}_{(3)}, \mathsf{VP}_{(3)}$, $\mathsf{NP}_{(2)}, \mathsf{V}_{(2)}, \mathit{Mary}_{(1)}, \mathit{loves}_{(1)}, \mathit{Peter}_{(1)}$. We do not specify the order of the nodes in $\mathsf{att}(e)$ explicitly, but use the convention that they should be read from left to right. To save space and make drawings of trees more comprehensible, we use

the more usual and less redundant drawing style shown in Figure 2(b) in the following.

The alphabets used to build Millstream configurations contain two types of symbols: *tree symbols* and *link symbols*. Edges labelled with these symbols are called tree edges and links, respectively. Tree edges are those edges of which the trees consist, and links are used to establish connections between nodes in the trees.

Definition 3 (Millstream alphabet and configuration). *A* Millstream alphabet *is a ranked alphabet Σ that is partitioned into disjoint alphabets $\Sigma_\mathrm{t}, \Sigma_\mathrm{l}$ of tree symbols and link symbols, respectively. A* Millstream configuration *(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 trees in the sense explained above.*

A link of arity k, that is, a link labelled with a link symbol of rank k, simply establishes a relation between k nodes by arranging them in a tuple. In the examples of this paper, there will be only one link symbol: this link symbol is of rank 2, and it connects nodes across the two trees of which every configuration consists, namely a syntactic and a semantic tree. Therefore, we draw links as unlabelled lines connecting the nodes in question. Furthermore, to distinguish these lines from those in the trees, they are drawn in a dashed style.

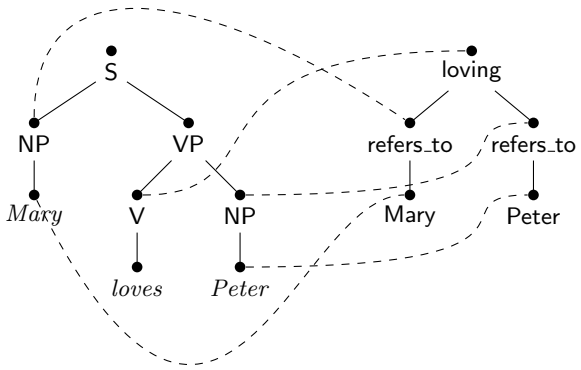


Fig. 3: A sample configuration that relates a syntactic and a semantic tree

With these conventions, a complete configuration looks as shown in Figure 3. It consists of two trees, representing the (extremely simplified) syntactic and semantic structures of the sentence *Mary loves Peter*. The symbols in the semantic tree should be interpreted as functions from a many-sorted algebra. The sorts of the algebra are the semantic domains of interest, and the evaluation of a tree (or a subtree thereof) yields an element of one of these sorts. 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 refers_to 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 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 (of course) use more elaborate trees on both the syntactic and the semantic sides. For example, one would 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 add the temporal dimension. That node would then be linked with the inflection that gives rise to it (see, e.g., [18]). 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.

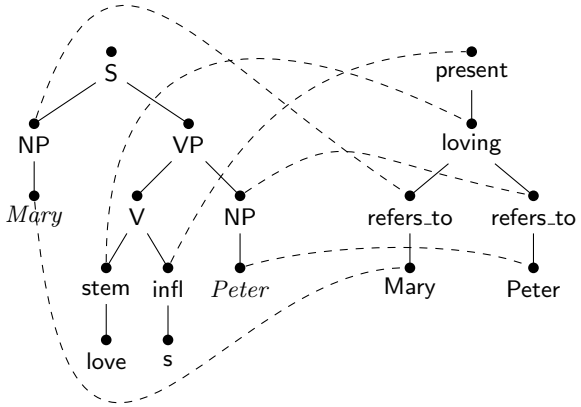


Fig. 4: A slightly less simplified configuration

We finish this section by explaining informally what a Millstream system in the sense of [3] is. We are not going to make use of the formal definition of Millstream systems here. Let $k \geq 1$, and consider a Millstream alphabet Σ. A Millstream system consists of k tree automata (the *modules*) and a logical *interface* Φ. The tree automata are any formal devices that specify sets of trees

over Σ_t (called tree languages). The interface is a logical formula expressing a property of configurations over Σ, to which end a configuration is viewed as a logical structure in some standard way. A configuration is a valid configuration of the Millstream system, if its k trees are elements of the tree languages specified by the modules and the configuration as a whole satisfies Φ.

A particular type of Millstream system which will be of interest in Section 4 is the regular MSO Millstream system. This is the case of a Millstream system in which the modules are finite-state tree automata and the interface is a formula in monadic second-order logic[5].

3 Building Configurations Incrementally by Graph Transformation

Suppose we are given a sentence such as *Mary loves Peter*, that we want to "understand". More precisely, we want to find a way to turn this sentence into a correct Millstream configuration, if possible. First of all, we have to define what is a correct Millstream configuration corresponding to this sentence. As described above, a Millstream system contains k modules, one for each of the k trees in a configuration. These modules are tree automata or any other kind of device specifying tree languages. Furthermore, we are given a logical *interface* describing which configurations – consisting of k trees generated by the modules and a set of links between them – are considered to be correct. In the current paper we investigate how configurations can be built "from scratch" along a sentence using a graph transformations. We use a lexicon which associates words with graph transformation rules. Each transformation is associated with the word the lexicon entry refers to. Building a configuration for the sentence *Mary loves Peter* means to apply three rules in succession, the first being associated with *Mary*, the second with *loves*, and the third with *Peter*. In case of ambiguities, this process is nondeterministic, but a lot of nondeterminism disappears because, typically, only one of the graph transformation rules for a given word applies in a concrete situation.

We use graph transformation in the sense of the so-called double pushout (DPO) approach [11] 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: $r = (L \supseteq K \subseteq R)$. 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 best described as a two step process. We first obtain a graph D from G 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

[5] In this case, the modules are actually negligible, because the power of the interface suffices to specify the tree languages defined by the modules.

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 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 with 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 satisfying the following requirements:*

1. *K is the graph obtained from L by deleting all nonterminals.*
2. *For every edge $e \in E_R \setminus E_K$, the inclusion*

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

holds.

Intuitively, the requirements mean that the left-hand side except the nonterminals is nothing but context around the nonterminals the presence of which 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 except for those nodes to which nonterminals were attached.

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 following.

Figure 5 shows sample lexicon rules for *Mary*, *loves*, and *Peter*. Lexicon rules or graph transformation rules of this kind are referred to as lexicon entries in the following. The nonterminals are depicted in typewriter font surrounded by boxes, the tree edges and links are drawn as before. The nonterminals 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, tree edges and links drawn blue 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. The nonterminals `SYN ARG` and `OBJ` indicate that

a syntactic argument and a syntactic object is missing, the nonterminal `THETA ROLE` indicates that a thematic role in the semantic structure is missing. 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 nonterminals `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 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 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*. Before we discuss such a reading of a sentence and a slightly more complex example we define first a *reader*.

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 with* 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.*

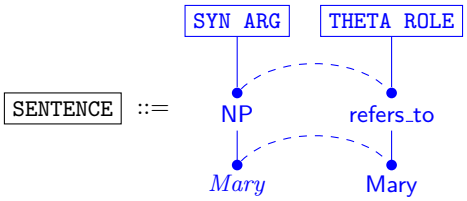
A reading *of an input sentence* $w_1 \cdots w_n \in W^*$ *by* $\mathcal{R}$ *is a derivation*

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

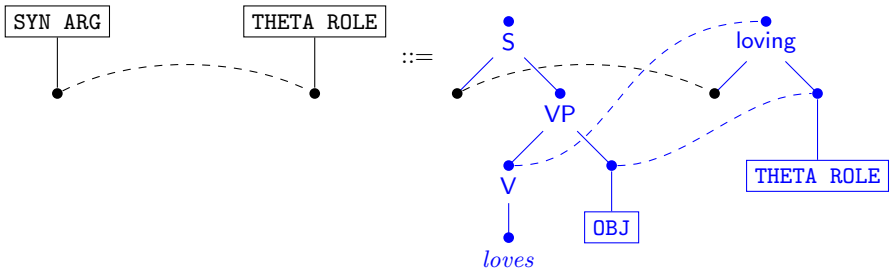
such that $G_0 \in S$ *and* G_n *is a* Σ*-graph. The set of all* Σ*-graphs that result from readings of* $u = w_1 \cdots w_n$ *is denoted by* $\mathcal{R}(u)$*, and the language of all such graphs is* $L(\mathcal{R}) = \bigcup_{u \in W^*} \mathcal{R}(u)$*.*

In the context of a reader as in the definition, a rule in $\Lambda(w)$ is also called 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 $G_0 \Rightarrow_{\Lambda(u)} G_n$.

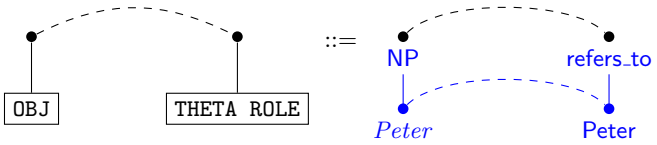
Now let us discuss a reading of a sentence and a slightly more complicated example than the one above. We consider English sentences involving the verbs *to seem* and *to try*; they are raising and control verbs, respectively. In sentences like *Mary seems to sleep*, the noun *Mary* is the syntactic 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 second configuration, the fact that *Mary*



(a) Lexicon entry for *Mary*



(b) Lexicon entry for *loves*

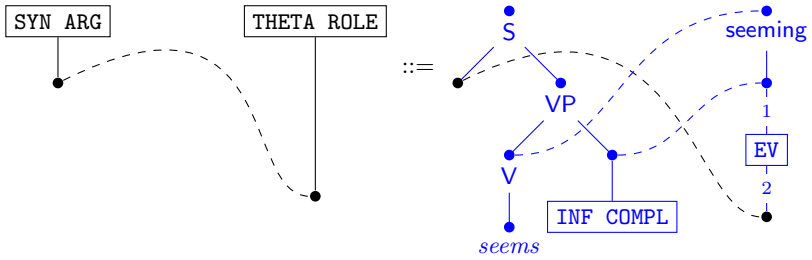


(c) Lexicon entry for *Peter*

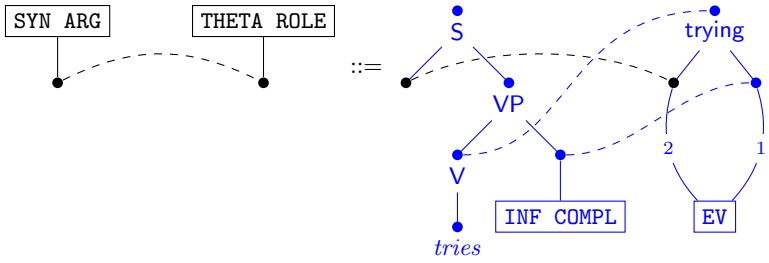
Fig. 5: Lexicon entries for *Mary*, *loves*, and *Peter*; the left-hand side of the lexicon entry for *Mary* is the (unique) start graph

is the semantic argument of both *tries* and *to sleep* is represented by sharing rather than duplicating the shared subtree. 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 nonterminals `INF COMPL` and `EV` are abbreviations for *infinitival complement* and *eventual event*. Let us consider the reading of the sentence *Mary tries to sleep* and the stepwise application of the lexicon rules needed. We start 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.

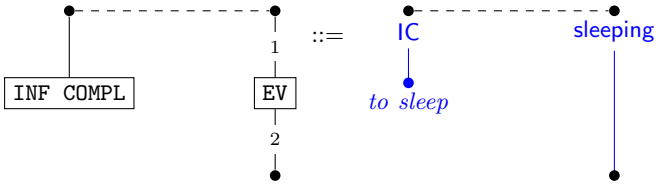
The nonterminals `SYN ARG` and `THETA ROLE` in Figure 7 illustrate that the syntactic and semantic roles of *Mary* are not determined yet. Now we read *tries* and apply the lexicon rule in Figure 6b leading to the partial configuration shown in Figure 8.



(a) Lexicon entry for *seems*



(b) Lexicon entry for *tries*



(c) Lexicon entry for *to sleep*

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

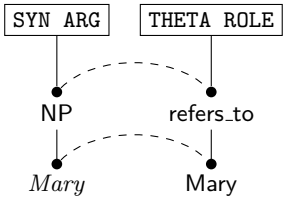


Fig. 7: Partial configuration after having read *Mary*

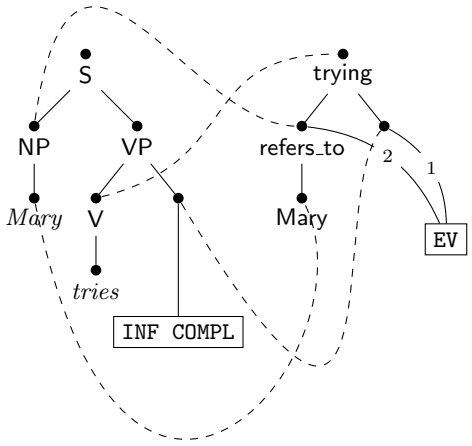


Fig. 8: Partial configuration after having read *Mary tries*

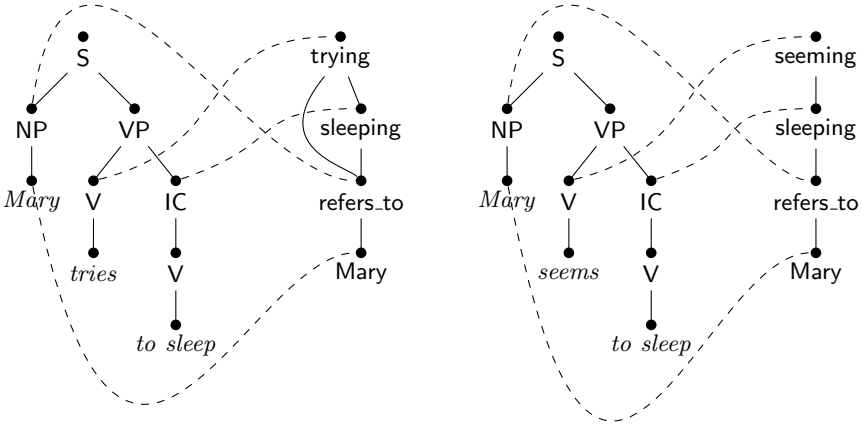


Fig. 9: Configurations representing the sentences *Mary seems to sleep* and *Mary tries to sleep*

Figure 8 shows that the syntactic and semantic roles of *Mary* have been specified as syntactic subject and first semantic argument (agent). Furthermore, the linked nonterminals `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*, replaces the nonterminals `INF COMPL` and `EV` in the partial configuration in Figure 8 and results in the final Millstream configuration depicted in the left-hand side 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 right-hand side of Figure 9.

When designing readers, it is useful to be able to assume that the words in the input sentence come in a certain order, that is, that the period is always the last input "word". The following result makes this possible, as it states that input sentences can be restricted by regular languages.

Theorem 6. *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 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). To construct $\mathcal{R}'$, we let $N' = N \cup Q$, where the rank of each $q \in Q$ is zero, and 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 every lexicon entry $r \in \Lambda(w)$ ($w \in W$), Λ'_0 contains $|Q|$ lexicon entries r^q ($q \in Q$) obtained by adding single edges labelled q and $\delta(q, w)$ to the left- and right-hand side, respectively, of r. If, furthermore, $\delta(q, w) \in F$, then we let Λ'_1 contain the lexicon entry $r^{q,\mathrm{f}}$ obtained by only adding q to the left-hand side of r, but keeping the right-hand side 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. It follows by a straightforward induction on $|u|$ 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 $\delta(u)$ to it. Using this, and the fact that a reading terminates as soon as (and only if) a rule $r^{q,\mathrm{f}} \in \Lambda'_1$ is applied (which means that the state reached in A is a final one), the statement of the theorem follows. □

An important question is how the correctness of a reader with respect to a given Millstream system can be proved. In other words, one would like to make sure 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

In this section, we show that, for readers $\mathcal{R}$ satisfying two conditions, $L(\mathcal{R})$ is a context-free graph language – more precisely, a hyperedge-replacement graph language. From this, we conclude that it is decidable for such readers whether all configurations constructed by $\mathcal{R}$ are valid configurations of a given regular MSO Millstream system.

Definition 7 (HR grammar and language [14]). *For a graph G and an edge $e \in E_G$, let $G|_e$ denote the subgraph of G consisting of e and its attached nodes, and let $G - e$ denote the graph obtained from G by removing the edge e. A* hyperedge-replacement graph 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 DPO rule $r = (L \supseteq K \subseteq R)$ such that $L = L|_e$ for a nonterminal e, and $K = L - e$. Thus, also HR rules will be denoted as $L ::= R$ in the following.

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

$$L(\Gamma) = \{G \in \mathcal{G}_\Sigma \mid G_0 \overset{*}{\underset{P}{\Rightarrow}} G \text{ for some } G_0 \in S\}.$$

Throughout the rest of this section, we consider the following conditions, for a reader $\mathcal{R} = (\Sigma, N, W, \Lambda, S)$:

(1) The language $L(\mathcal{R})$ is of bounded degree, that is, there is a bound d on the number of edges that can share an attached node.
(2) $\mathcal{R}$ is nonterminal bounded, that is, there is a bound l such that no graph G with $G_0 \Rightarrow_{\Lambda(u)} G$ ($G_0 \in S$, $u \in W^*$) contains more than l nonterminals.

We say that two readers $\mathcal{R}$ and $\mathcal{R}'$ are *equivalent* if $\mathcal{R}(u) = \mathcal{R}'(u)$ for every input sentence u.

Lemma 8. *If $L(\mathcal{R})$ is of bounded degree, then an equivalent reader $\mathcal{R}'$ can effectively be constructed such that every graph derived by $\mathcal{R}'$, including those containing nonterminals, is of bounded degree.*

Proof. The proof proceeds in two steps. First, we augment each nonterminal with nondeterministically guessed information about how many edges it will eventually attach to each of the nodes to which it is attached. In the second step, we remove, from every nonterminal, the attached nodes for which this number is zero.

Let d be the maximum degree of nodes in the configurations generated by $\mathcal{R}$. Employing a guess-and-verify strategy, we can "guess", for every nonterminal e and every node v to which e is attached, how many edges will be attached to

v in derivation steps e gives rise to. This number $d' \leq d$ is remembered in the label of e, and the right-hand sides of rules are modified to make sure that d' is divided between the nonterminals, tree edges and link edges attached to v in a step that replaces e.

Formally, for every $A \in N$ of rank k and all $i_1, \ldots, i_k \leq d$, we introduce a new nonterminal label $A_{i_1,\ldots,i_k}$ of rank k. Let N_0 be the alphabet of nonterminals obtained in this way. For a graph $G \in \mathcal{G}_{\Sigma \cup N_0}$, we let $\underline{G}$ denote the graph in $\mathcal{G}_{\Sigma \cup N}$ that is obtained by replacing every label $A_{i_1,\ldots,i_k} \in N_0$ by A, that is, by forgetting the added information. Furthermore, for a node $v \in V_G$, let $\deg_G(v) = \sum_{e \in E_G} \deg_G(v, e)$, where

$$\deg_G(v,e) = \begin{cases} 0 & \text{if } \mathsf{lab}_G(e) \in \Sigma \text{ and } v \notin [\mathsf{att}_G(e)] \\ 1 & \text{if } \mathsf{lab}_G(e) \in \Sigma \text{ and } v \in [\mathsf{att}_G(e)] \\ \sum_{j \in J} i_j & \text{if } \mathsf{lab}_G(e) = A_{i_1,\ldots,i_k} \in N_0 \text{ and } J = \{j \mid \mathsf{att}_G(e)_j = v\}. \end{cases}$$

We first turn S into S_0 by replacing labels in N by labels in N_0 in all meaningful ways, where meaningful means that the (guessed) degree of every node is at most d. In other words, S_0 is the set of all graphs G such that $\underline{G} \in S$ and $\deg_G(v) \leq d$ for all $v \in V_G$.

Next, we define a new set Λ_0 of lexicon entries in a similar way, making sure that the information added to the start graphs is propagated. We let Λ_0 consist of all lexicon entries $r = (L ::= R)$ such that the following hold:

1. The lexicon entry $\underline{L} ::= \underline{R}$ is in Λ.
2. For all $v \in V_R$, $\deg_R(v) \leq d$ and, if $v \in L$, then $\deg_L(v) = \deg_R(v)$.

By virtue of this construction, the reader $\mathcal{R}_0 = (\Sigma, N_0, W, \Lambda_0, S_0)$ satisfies the following: For all $u \in W^*$ and all graphs $G_0 \in S_0$, $G \in \mathcal{G}_{\Sigma \cup N_0}$ and $H \in \mathcal{G}_\Sigma$,

(a) if $G_0 \Rightarrow_{\Lambda_0(u)} G$, then $\deg_G(v) \leq d$ for all $v \in V_G$,
(b) if $G \Rightarrow_{\Lambda_0(u)} H$, then $\deg_G(v) = \deg_H(v)$ for all $v \in V_G$, and
(c) $\underline{G} \Rightarrow_{\Lambda(u)} H$ if and only if $G \Rightarrow_{\Lambda_0(u)} H$.

It follows that $L(\mathcal{R}_0) = L(\mathcal{R})$. Moreover, in every graph G occurring in S_0 or as the left- or right-hand side of a lexicon entry, we can replace all nonterminals e carrying a label $\mathsf{lab}_G(e) = A_{i_1,\ldots,i_k}$ by a nonterminal of rank $l \leq k$ which is obtained by removing all those nodes v_j from $\mathsf{att}_G(e) = v_1 \cdots v_k$ for which $i_j = 0$. Thus, in the new alphabet N' of nonterminal labels, $A_{i_1,\ldots,i_k}$ has the rank $|\{j \mid 1 \leq j \leq k \wedge i_j > 0\}|$. The resulting reader $\mathcal{R}' = (\Sigma, N', W, \Lambda', S')$ still generates $L(\mathcal{R})$ and satisfies (a)–(c). In particular, since it satisfies (a), the degree of every node in a graph that is derivable from a graph in S' is at most d. This completes the proof. □

Lemma 9. *If $\mathcal{R} = (\Sigma, N, W, \Lambda, S)$ is a nonterminal-bounded reader, then one can effectively construct an equivalent reader $\mathcal{R}' = (\Sigma, N', W, \Lambda', S')$ that is linear in the following sense: for all $G_0 \in S$, every graph G such that $G_0 \Rightarrow_{\Lambda(u)} G$ for some $u \in W^*$ contains at most one nonterminal.*

Proof. 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 it. Choose any total order on N, and assume that $\#(G) \leq l$ for all graphs G derived by $\mathcal{R}$. For every sorted sequence $A_1 \cdots A_n$ of nonterminal labels such that $n \leq l$, we let N' contain the label $A_1 \cdots A_n$ of rank $\sum_{i=1}^{n} \mathsf{rk}(A_i)$. A graph $G \in \mathcal{G}_{\Sigma\cup N}$ that contains n nonterminals $e_1, \ldots, e_n$, 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 $\mathsf{att}_{\widetilde{G}}(e) = \mathsf{att}_G(e_1) \cdots \mathsf{att}_G(e_n)$.

Let $S' = \{\widetilde{G_0} \mid G_0 \in S\}$. To modify the lexicon entries in Λ in an appropriate way, consider one of them, say $r = (L ::= R)$. Let $\Delta = l - \max(\#(L), \#(R))$ (which we may assume to be non-negative). First, construct every rule $r_+ = (L_+ ::= R_+)$ that can be obtained from r by adding at most Δ nonterminals to L (possibly attached to new nodes that are also added to L) and modifying R in the same way. 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, Λ' consists of all rules $\widetilde{r}_+$, for every r_+ constructed in this way. For $G_0 \in S$, it should be clear 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 $\#(G) \leq l$. One proves this fact by induction on $|u|$. This completes the proof. □

We note here that, in the proof above, every node has the same degree in $\widetilde{G}$ as it has in G. Hence, the construction in the proof does not destroy the property ensured by applying Lemma 8. This will be used in the proof of Theorem 12 below.

For $k \in \mathbb{N}$, let us say that the *k-neighbourhood* of an edge e in a graph G is $G|_e$ if $k = 0$. Otherwise, it is the union of $G|_e$ and the $(k-1)$-neighbourhoods of all edges e' such that $[\mathsf{att}_G(e)] \cap [\mathsf{att}_G(e')] \neq \emptyset$. In other words, the k-neighbourhood of e in G consists of all edges and their attached nodes which can be reached on a path of length at most k.

Lemma 10. *Let Σ be a finite ranked alphabet and $d, k \in \mathbb{N}$. The set of all k-neighbourhoods of Σ-graphs of degree at most d is finite (up to isomorphism).*

Proof. Obvious. □

Lemma 11. *Let r be a lexicon entry of a linear reader $\mathcal{R}$ and consider a step $G \Rightarrow_r H$, where G and H contain the nonterminals e and f, respectively. For every k, the k-neighbourhood of f in H is uniquely determined by r and the k-neighbourhood of e in G (up to isomorphism).*

Proof. By Property 3 in Definition 4, every edge $e' \in E_H \setminus E_G$ satisfies $[\mathsf{att}_H(e')] \cap V_G \subseteq [\mathsf{att}_G(e)]$. In particular, this holds for $e' = f$. Hence, all nodes and edges in the k-neighbourhood of f in H are either in the k-neighbourhood of e in G or added to G by r. □

Theorem 12. *If a reader $\mathcal{R} = (\Sigma, N, W, \Lambda, S)$ satisfies conditions (1) and (2), then $L(\mathcal{R})$ is an HR language.*

Proof. By first applying Lemma 8 and then Lemma 9, we can make sure that $\mathcal{R}$ is linear, and that the set of all graphs that can be derived from the start graphs is of bounded degree (see the remark following the proof of Lemma 9). Now, let d be the maximum degree a node in a graph derived can have, and let k be the maximum distance of edges in L from the unique nonterminal in L, over all lexicon entries $L ::= R$ in $\mathcal{R}$. Given a graph $G \in \mathcal{G}_{\Sigma \cup N}$ and an edge $e \in E_G$, denote the k-neighbourhood[6] of e in G by G^e_k.

Now, we modify S and Λ as follows. For every graph $G \in S$, if e is its unique nonterminal (provided that G contains a nonterminal), we replace its label by $\langle \mathsf{lab}_G(e), G^e_k \rangle$. This yields a new set S' of start graphs. For every lexicon entry $r = (L ::= R)$, let Λ' contain all lexicon entries $L' ::= R'$ that can be constructed as follows. If e is the nonterminal edge in L, L' is obtained by relabelling e with $\langle \mathsf{lab}_L(e), G' \rangle$, where $G' = H^e_k$ for some supergraph H of G that is of degree at most d. Note that this yields a finite number of left-hand sides L', by Lemma 10. The right-hand side R' is obtained from R by replacing the label of its nonterminal e' (if it exists) by $\langle \mathsf{lab}_R(e'), G'' \rangle$, where $G'' = (H')^{e'}_k$ and H' is determined by the condition $H \Rightarrow_r H'$. By Lemma 11, G'' is uniquely determined by G' and r.

Let $\mathcal{R}' = (\Sigma, N', W, \Lambda', S')$, where N' is the set of nonterminal labels occurring in S' and Λ'. For a graph $G \in \mathcal{G}_{\Sigma \cup N'}$, let $\underline{G}$ denote the graph in $\mathcal{G}_{\Sigma \cup N}$ by replacing each label $\langle A, G \rangle$ by A. Note that $S = \{\underline{G} \mid G \in S'\}$. By induction on the length of derivations, the following statements can be proved for $u \in W^*$ and $G \in S'$:

(a) If $\underline{G} \Rightarrow_{\Lambda(u)} H$, then $G \Rightarrow_{\Lambda'(u)} H'$, where $\underline{H'} = H$.
(b) If $G \Rightarrow_{\Lambda'(u)} H$, then $\underline{G} \Rightarrow_{\Lambda(u)} \underline{H}$.
(c) If $G \Rightarrow_{\Lambda'(u)} H$ and H contains a nonterminal edge e, then $\mathsf{lab}_H(e) = \underline{H}^e_k$.

In particular, (a) and (b) imply that $L(\mathcal{R}') = L(\mathcal{R})$. Moreover, by (c), it should be clear that the context can be removed from all rules in Λ' without affecting $L(\mathcal{R}')$. More precisely, for every rule $L ::= R$ in Λ', if e is the nonterminal edge in L, let Λ'' contain the rule $L|_e ::= R'$, where R' is obtained from R by removing all nodes and edges that are in L but not in $L|_e$. Then the language generated remains unaffected and all lexicon entries have become HR rules, showing that the language generated is a HR language. □

As mentioned in the beginning of this section, we can now use the previous theorem to check (one direction of) the correctness of a reader with respect to a given regular MSO Millstream system.

Theorem 13. *For regular MSO Millstream systems MS and readers $\mathcal{R}$ that satisfy (1) and (2), it is decidable whether $L(\mathcal{R}) \subseteq L(MS)$.*

Proof. For a regular MSO Millstream system MS, the language $L(MS)$ is the set of all graphs that satisfy a given MSO formula Φ. Moreover, it is known that,

[6] We do not distinguish between isomorphic neighbourhoods, but assume that they are silently replaced by unique representatives of their respective equivalence classes.

for an MSO formula Φ, it is decidable whether a given HR grammar Γ generates only graphs satisfying Φ (see, e.g., [8]). □

5 Future Work

As this paper is about ongoing work, more research will be necessary for being able to find out whether the type of lexicon entries proposed here is the most appropriate one. For this, bigger examples being able to treat a much greater variety and complexity of sentences need to be considered, and an implementation should be made.

An important question for future research is how to build lexica in a consistent manner. For example, one may distinguish between different strategies. Assume that the currently read word can give rise to two slightly different structures, corresponding to different interpretations, and that one of the coming words will determine which of them is the right one. A lexicon could then use a "lazy" strategy and add only the common part of the two structures, leaving it to later reading steps to choose between the differing parts. Alternatively, an "eager" approach could nondeterministically choose between the two structures, making backtracking necessary if it later turns our that the path taken was the wrong one. It seems that humans use a mixture of these strategies, combined with probabilities: On the one hand, if one interpretation is 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. When it comes to readers, different strategies may result in different complexities of lexica (for instance in terms of size or the complexity of pattern matching), lead to more or less nondeterminism and, thus, backtracking.

Theorem 13 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 readers.

Future research will also have to study efficient algorithms for the construction of lexica and, once a lexicon is given, of readings. In particular, it should be possible to "learn" by adding new entries to a lexicon, which should affect existing entries as little as possible. For large lexica, efficient pattern matching algorithms are needed that, given a partial configuration and an input word, retrieve all applicable entries. Finally, we would like to mention optimisation algorithms that modify lexica according to certain criteria. For instance, these could make lexica more compact, increase laziness or eagerness, make them more suitable for fast pattern matching, etc.

References

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

2. Bensch, S., Drewes, F.: Millstream systems. Report UMINF 09.21, Department of Computing Science, Umeå University, Umeå, Sweden (2009)
3. Bensch, S., Drewes, F.: Millstream systems – a formal model for linking language modules by interfaces. In: Drewes, F., Kuhlmann, M. (eds.) Proc. ACL 2010 Workshop on Applications of Tree Automata in Natural Language Processing (ATANLP 2010). The Association for Computer Linguistics (2010)
4. Bensch, S., Drewes, F., Jürgensen, H., van der Merwe, B.: Incremental construction of millstream configurations using graph transformation. In: Proc. 9th Intl. Workshop on Finite State Methods and Natural Language Processing (FSMNLP 2011). pp. 93–97. Association for Computational Linguistics (2011)
5. Beuck, N., Köhn, A., Menzel, W.: Incremental parsing and the evaluation of partial dependency analyses. In: Gerdes, K., Hajičová, E., Warner, L. (eds.) depling 2011. Proceedings, International Conference on Dependency Linguistics, Depling 2011, Barcelona, September 5–7 2011. Exploring Dependency Grammar, Semantics and the Lexicon. pp. 290–299. depling.org (2011)
6. Costa, F., Frasconi, P., Lombardo, V., Soda, G.: Towards incremental parsing of natural language using recursive neural networks. Applied Intelligence 19, 9–25 (2003)
7. Costa, F., Frasconi, P., Lombardo, V., Sturt, P., Soda, G.: Ambiguity resolution analysis in incremental parsing of natural language. IEEE Trans. Neural Networks 16, 959–971 (2005)
8. Courcelle, B.: Graph rewriting: an algebraic and logic approach. In: van Leuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 193–242. Elsevier, Amsterdam (1990)
9. Drewes, F., Habel, A., Kreowski, H.J.: Hyperedge replacement graph grammars. In: Rozenberg, G. (ed.) Handbook of Graph Grammars and Computing by Graph Transformation. Vol. 1: Foundations, chap. 2, pp. 95–162. World Scientific, Singapore (1997)
10. Drewes, F., Hoffmann, B., Minas, M.: Contextual hyperedge replacement. In: Proc. Applications of Graph Transformation With Industrial Relevance 2011 (AGTIVE 2011). Lecture Notes in Computer Science, Springer (2012), to appear
11. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Monographs in Theoretical Computer Science. An EATCS Series, Springer (2006)
12. Engelfriet, J.: Context-free graph grammars. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages. Vol. 3: Beyond Words, chap. 3, pp. 125–213. Springer (1997)
13. Ghezzi, C., Mandrioli, D.: Incremental parsing. IEEE Trans. Progr. Languages and Systems 1, 58–70 (1979)
14. Habel, A.: Hyperedge Replacement: Grammars and Languages, Lecture Notes in Computer Science, vol. 643. Springer (1992)
15. Habel, A., Kreowski, H.J.: 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, vol. 291, pp. 15–26. Springer (1987)
16. Hassan, H., Sima'an, K., Way, A.: 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. pp. 205–208. IEEE Press (2008)
17. Huang, L., Sagae, K.: Dynamic programming for linear-time incremental parsing. In: ACL 2010. 48th Annual Meeting of the Association for Computational

Linguistics, Proceedings of the Conference, 11–16 July 2010, Uppsala, Sweden. pp. 1077–1086. Uppsala Universitet (2010), available at http://www.aclweb.org/anthology//P/P10/P10-1110.pdf

18. Jackendoff, R.: Foundations of Language: Brain, Meaning, Grammar, Evolution. Oxford University Press (2002)
19. Lane, P.C.R., Henderson, J.B.: Incremental syntactic parsing of natural language corpora with simple synchrony networks. IEEE Trans. Knowledge and Data Engineering 13(2), 219–231 (2001)
20. Nivre, J.: Algorithms for deterministic incremental dependency parsing. Computational Linguistics 34, 513–553 (2008)
21. Taraban, R., McClelland, J.L.: Constituent attachment and thematic role assignment in sentence processing: Influences of content-based expectations. Journal of Memory and Language 27, 597–632 (1988)
22. Wagner, T.A., Graham, S.L.: Efficient and flexible incremental parsing. IEEE Trans. Progr. Languages and Systems 20, 980–1013 (1998)
23. Wu, S., Bachrach, A., Cardenas, C., Schuler, W.: Complexity metrics in an incremental right-corner parser. In: ACL 2010. 48th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, 11–16 July 2010, Uppsala, Sweden. pp. 1189–1198. Uppsala Universitet (2010), available at http://www.aclweb.org/anthology//P/P10/P10-1121.pdf