Incremental Construction of Millstream Configurations Using Graph Transformation

Suna Bensch
Department of Computing Science
Umeå University (Sweden)
suna@cs.umu.se

Frank Drewes
Department of Computing Science
Umeå University (Sweden)
drewes@cs.umu.se

Helmut Jürgensen
Department of Computer Science
The University of Western Ontario (Canada)
hjj@csd.uwo.ca

Brink van der Merwe
Department of Computing Science
Stellenbosch University (South Africa)
abvdm@cs.sun.ac.za

Abstract

Millstream systems are a non-hierarchical model of natural language. We describe an incremental method for building Millstream configurations while reading a sentence. This method is based on a lexicon associating words and graph transformation rules.

1 Introduction

Language processing is an incremental procedure. This is supported by various psycholinguistic and cognitive neuroscience-based studies (see e.g. (Taraban and McClelland, 1988)). We do not postpone the analysis of an utterance or sentence until it is complete, but rather start to process immediately hearing the first words (or word parts). We present ongoing work regarding the incremental syntactic and semantic analysis of natural language sentences. We base this work on Millstream systems (Bensch and Drewes, 2010), (Bensch et al., 2010), 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 formalise the relation between them by interfaces. Millstream systems are motivated by contemporary linguistic theories (see e.g. (Jackendoff, 2002)). A Millstream system consists of a finite number of *modules* each of which describes a linguistic aspect and an *interface* which describes the dependencies 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*. For simplicity, we just consider Millstream systems containing only two modules, for syntax and semantics. With this simplifying assumption, a configuration of the Millstream system consists of two trees with links between them and represents the analysis of the sentence that is the yield of the syntax tree. An obvious question is how such a configuration can be constructed from a given sentence. Such a procedure would be a step towards automatic language understanding based on Millstream systems. We propose to use graph transformations for that purpose. By expressing language processing in terms of graph transformation we can employ a wealth of theoretical results relating graph transformations and monadic second-order logic. We mimic the incremental way in which humans process language, thus constructing a Millstream configuration by a step-by-step procedure while reading the words of a sentence from left to right. 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 the purpose of which is to construct an appropriate configuration. For a sentence like *Mary likes Peter*, for example, we first apply a lexicon entry corresponding to *Mary*, which results in a partial configuration that represents the syntactic, semantic and interface structure of *Mary*. We continue by applying the lexicon entry for *loves*, which yields a partial configuration representing *Mary loves*. Finally, a lexicon entry representing *Peter* is applied, resulting in the overall Millstream configuration for the entire sentence.

2 Millstream Configurations as Graphs

A configuration in a Millstream system is a tuple of ranked and ordered trees (in our restricted case, a pair consisting of the syntactic and the semantic representation of a sentence) with links between them. The (labelled) links indicate relations between the nodes. A typical link establishes a relation between two nodes belonging to different trees. In this paper, we want to 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 work with hypergraphs in which the hyperedges (but not the nodes) are labelled. For simplicity, we call hypergraphs graphs and their hyperedges edges. Edge labels are taken from a doubly ranked alphabet Σ, meaning that Σ is a finite set of symbols in which every symbol a has *source* and *target ranks* $rank_{src}(a), rank_{tar}(a) \in \mathbb{N}$ determining the number of sources and targets, respectively, that an edge label's a is required to have.

Definition 1 *Let Σ be a doubly ranked alphabet. A Σ-graph is a quadruple (V, E, src, tar, lab) consisting of finite sets V and E of* nodes *and* edges, source *and* target functions $src, tar\colon E \to V^*$*, and an* edge labelling function $lab\colon E \to \Sigma$ *such that $rank_{src}(lab(e)) = |src(e)|$ and $rank_{tar}(lab(e)) = |tar(e)|$ for all $e \in E$. The components of a graph G will also be referred to as $V_G, E_G, src_G, tar_G, lab_G$. By $\mathcal{G}_\Sigma$ we denote the class of all Σ-graphs.*

A *Millstream alphabet* is a doubly ranked alphabet Σ in which the target rank of each symbol is either 1 or 0. Symbols of target rank 1 are *tree symbols*; edges labelled with these symbols are *tree edges*. Symbols of target rank 0 are *link symbols*; edges labelled with link symbols are *links*. A tree or link symbol a may be denoted by $a_{(k)}$ to indicate that $rank_{src}(a) = k$. In the following, the term *tree* refers to an acyclic graph in which all edges are tree edges, each node is the target of exactly one edge, and there is exactly one node (the root) that is not a source of any edge. A Σ-*configuration* is a graph $G \in \mathcal{G}_\Sigma$ such that the deletion of all links from G results in a disjoint union of trees.

Figure 1 depicts a tree, built using the tree symbols $\mathsf{S}_{(2)}, \mathsf{VP}_{(2)}, \mathsf{NP}_{(1)}, \mathsf{V}_{(1)}, Mary_{(0)}, loves_{(0)}, Peter_{(0)}$. To save space we use the drawing style shown in Figure 2 instead. The links pointing to tree

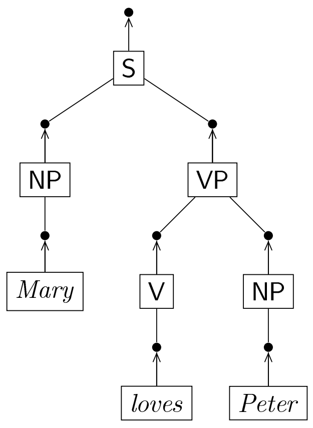

Figure 1: A tree in its (hyper)graph representation

symbols point to the target nodes of the tree edge representing that symbol.

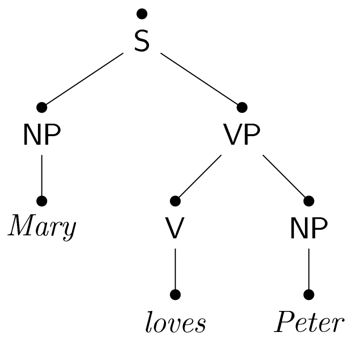

Figure 2: A more condensed representation of the tree in Figure 1

A k-ary link establishes a relation between k nodes by arranging them in a tuple. In this paper there is only one link symbol, this link symbol is of source rank 2, and connects nodes across the two trees every configuration consists of. These links are drawn as unlabelled dashed lines. With these conventions, a complete configuration looks as shown in Figure 3. This configuration consists of two trees, representing the (extremely simplified) syntactic and semantic structures of the sentence *Mary loves Peter*. The symbols in the semantic tree are interpreted as functions from a many-sorted algebra. The sorts of the algebra are the semantic domains of interest, and the evaluation of a (sub-) tree 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 (i.e., constants) returning elements of the sort *name*. The

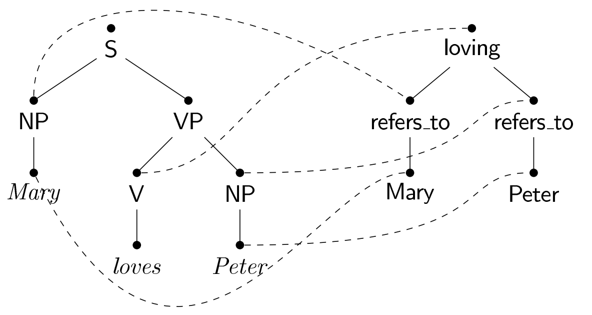

Figure 3: A sample configuration that relates a syntactic and a semantic tree

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 that, e.g., the verb of the sentence corresponds to the function loving, whose two arguments correspond to the two noun phrases of the sentence. In realistic settings, one would of course use more elaborate trees. However, since we primarily want to convey the idea behind our proposed approach, we use this simple type of configuration as our running example.

3 Incremental Construction of Configurations

In a Millstream system, we are given k *modules* for each of the k trees in a configuration. These modules are tree grammars or any other kind of device generating trees. Furthermore, we are given a logical *interface* that describes 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 take a more pragmatic point of view and investigate how configurations can be built up "from scratch" along a sentence using an approach based on implementing a lexicon by graph transformation.

We use graph transformation in the sense of the so-called double pushout (DPO) approach (Ehrig et al., 2010) (with injective morphisms). A rule r is a span $r = (L \supseteq K \subseteq R)$ of graphs L, K, R. The rule applies to a graph G if

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

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

In this case, applying r means to remove all nodes and edges from G that are in L but not in K, and to add all nodes and edges that are in R but not in K. Thus, the so-called glueing graph K 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, i.e., edges with an undefined attachment. If the result of the application of r to G is G', this may be denoted by $G \underset{r}{\Rightarrow} G'$. Moreover, if R is a set of graph transformation rules, and $G \underset{r}{\Rightarrow} G'$ for some $r \in R$, we denote this fact by $G \underset{R}{\Rightarrow} G'$.

Compared to general DPO rules, our lexicon rules are quite restricted as they never delete anything. In other words, we always have $L = K$, and hence the rules only glue new subgraphs to the existing (partial) configuration. We call rules of this kind *incremental* and denote the set of all incremental rules over a Millstream alphabet Σ by $\mathcal{R}_\Sigma$. In addition to the conditions 1 and 2 above, we restrict the applicability of rules further, by introducing a third condition:

3. $tar_G(e) \neq tar_R(e')$ for all tree edges $e \in E_G$ and $e' \in E_R \setminus E_K$.

This condition merely avoids useless nondeterminism leading into dead ends.

Derivations start with a common start graph. Since our example is extremely simple, it suffices to choose the graph that consists of the edge labelled with the root symbol $\mathsf{S}_{(2)}$ of the syntactic tree (together with the three attached nodes). The fact that all our rules satisfy $L = K$ means that we can depict a rule as just one graph, namely R, where the nodes and edges in L are drawn in blue. Graph transformation rules of this type are called lexicon entries. Figures 4, 5 and 6 show sample lexicon entries for the words *Mary*, *loves*, and *Peter*, respectively. Starting with the start graph (the blue subgraph in Figure 4) and applying the three rules in the order in which the words appear in the sentence takes us to the configuration in Figure 3. Note that the complete lexicon should contain another entry similar to the one in Figure 4, but with $Mary$ and Mary being replaced by $Peter$ and Peter, respectively. Similarly, there should be a variant of Figure 6 for the name Mary. This would make it possible to read the sentence *Peter loves Mary*. The reader should also note that, when reading the third word of the sentence, *Peter*, the corresponding variant of Figure 4 cannot be applied, because the first child of S is already present.

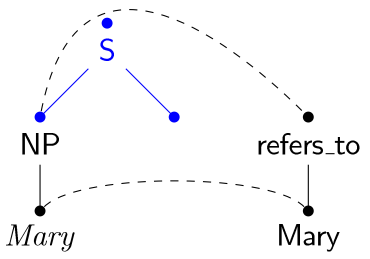

Figure 4: Lexicon entry for *Mary*

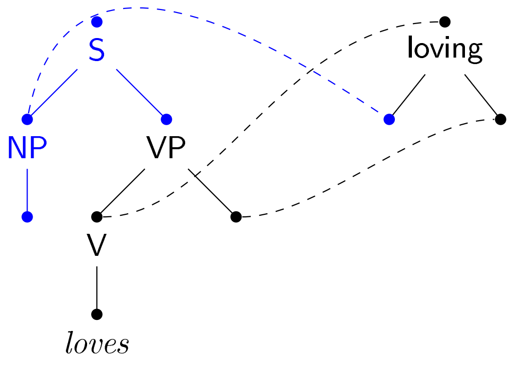

Figure 5: Lexicon entry for *loves*

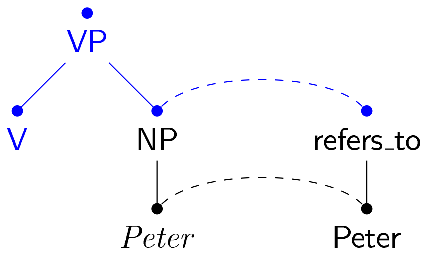

Figure 6: Lexicon entry for *Peter*

Definition 2 *A* reader *is a quadruple* $R = (\Sigma, W, \Lambda, S)$ *consisting of a finite set* W *of words (the input words), a Millstream alphabet* Σ*, a mapping* Λ *called the* lexicon*, and a start graph* $S \in \mathcal{G}_\Sigma$*. The lexicon assigns to every* $w \in W$ *a finite set* $\Lambda(w) \subseteq \mathcal{R}_\Sigma$ *of rules, the* lexicon entries.

A reading *of an input sentence* $w_1 \cdots w_n$ *by* R *is a derivation*

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

such that G_n *is a* Σ*-configuration. The set of all* Σ*-configurations that result from readings of* $w = w_1 \cdots w_n$ *is denoted by* $R(w)$*, and the language (of* Σ*-configurations) generated by* R *is* $L(R) = \bigcup_{w \in W^*} R(w)$*.*

Future work will have to develop methods for proving the correctness of readers with respect to a given Millstream system. This notion of correctness is given in the next definition.

Definition 3 *Let* MS *be a Millstream system having a distinguished syntactic module* M *(i.e., every configuration of* MS *contains a syntactic tree.) The set of all configurations of* MS *is denoted* $L(MS)$*. A reader* $R = (\Sigma, W, \Lambda, S)$ *is* correct with respect to MS *if* $L(R) = L(MS)$ *and, for every* $w \in W^*$ *and every* $G \in R(w)$*, the yield of the syntactic tree of* G *is equal to* w*.*

4 Future Work

More research will be necessary to find out whether the type of lexicon entries proposed is most appropriate. Bigger lexica to treat a greater variety and complexity of sentences need to be considered, and an implementation is required. An extension to render the readers of Section 3 more powerful might introduce nonterminal (hyper-)edges to act as indicators of “construction sites”. These nonterminals would be consumed when a lexicon entry is applied. An important question for future research is how to build lexica in a systematic way, possibly distinguishing lexica with different strategies, to accommodate different behaviours of readers. Future research will also have to study efficient algorithms for constructing lexica and readings. In particular, it should be possible to “learn”. For large lexica, efficient pattern matching algorithms are needed and optimisation algorithms would need to be examined.

References

Suna Bensch and Frank Drewes. 2010. Millstream systems – a formal model for linking language modules by interfaces. In F. Drewes and M. Kuhlmann, editors, *Proc. ACL 2010 Workshop on Appl. of Tree Automata in Natural Lang. Proc. (ATANLP 2010)*, 28–36. The Association for Computer Linguistics.

Suna Bensch, Frank Drewes, and Henrik Björklund. 2010. Algorithmic properties of Millstream systems. In Y. Gao, H. Lu, S. Seki, and S. Yu, editors, *Proc. 14th Intl. Conf. on Developments in Language Theory (DLT 2010)*, volume 6224 of *LNCS*, pages 54–65. Springer.

Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. 2006. *Fundamentals of Algebraic Graph Transformation*. Monogr. in Theor. Comp. Sci. An EATCS Series. Springer.

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

Roman Taraban and James McClelland. 1988. Constituent attachment and thematic role assignment in sentence processing: Influences of content-based expectations. *Journal of Memory and Language*, 27:597–632.