Robust Parsing as a Constraint Optimization Problem within a Finite State Approach (Draft)

Jörg Didakowski
University of Hamburg, Department of Informatics
Joerg.Didakowski@studium.uni-hamburg.de

Abstract

Dependency parsing of natural language as a *real-life scenario* can be treated as an *optimization problem* which can be solved using *constraint processing techniques*. In this article it is shown that *finite state techniques* are adequate to represent and solve such problems in a natural way. This is facilitated by the concept of the *Semiring-based Constraint Satisfaction Problem (SCSP)* which can be represented and solved with *Weighted Finite State Transducers (WFST).* The approach benefits from *regular approximation* and *parametrization* and includes the notion of *locality*, *decomposition* and *simultaneousness* in order to achieve efficient processing.

1 Introduction

Due to the increasing interest in dependency-based representations in *natural language processing* in the recent years several methods realizing dependency parsing have been developed. The focus of this article is on robust dependency parsing as a constraint-based approach. Several approaches to *constraint-based dependency parsing* exist. Some are based on *constraint processing techniques* used in artificial intelligence, others are based on *Finite State Machines (FSM)*[1] studied in *automata theory.* This article relates both directions to each other and tries to take advantage of both sides.

Maruyama (1990) formulates dependency parsing as a *Constraint Satisfaction Problem (CSP)* where a CSP is to find a consistent assignment of values to variables. He proposes the formalism *Constraint Dependency Grammar (CDG)* as a non-generative approach. In his approach parsing is treated as a disambiguation problem over initially ambiguous dependency relations where it is desired to eliminate all structural ambiguities. Conditions on natural language are formulated by means of first-order predicate calculus formula which are implicitly assumed to be universally quantified. CDG can allow non-projective dependencies simply by not forbidding them. Solving a CSP is NP complete in

[1]A FSM is a generalization of the more familiar *finite-state automation (FSA)*, *finite-state transducer (FST)* and their weighted counterparts *weighted finite-state automaton (WFSA)* and *weighted finite-state Transducer (WFST).*

general. However, Maruyama realizes CDG with a running time $\mathcal{O}(n^4)$ with the help of consistency based methods which narrow down the structural ambiguities. He shows that CDG has a weak generative capacity beyond context-free grammars. Harbusch (1997) goes further and shows that CDG is more powerful than the *Tree-Adjoining Grammar (TAG)* and that each TAG can be translated into an equivalent CDG. Helzerman & Harper (1996) extended CDG by the processing of lexically ambiguous sentences using the *MUltiply SEgmented CSP (MUSE CSP)* which allows to represent similar CSPs compactly. But CDG lacks robustness if no analysis exists satisfying all constraints. In contrast, the approach has the problem of disambiguation if more than one analysis is left.

An extension of CDG which is elementary for this article is presented by Heinecke, Kunze, Menzel, & Schröder (1998). They reformulate CDG as a *Constraint Optimization Problem (COP)* by including weights which can be assigned to constraints in order to make them defeasible or soft. This extension is implemented within the *Weighted Constraint Dependency Grammar (WCDG)* framework which allows to model gradation which is not restricted to the notion of competence and performance but including "many more aspects where weighted information proves helpful" (Schröder (2002)). This perspective allows to include degrees of grammaticality, structural preferences and uncertain information and to make WCDG error tolerant and robust. But solving a COP is NP complete in general, too, and consistency methods are not very applicable in order to solve them. That is, a consistency algorithm may remove only a few ambiguities or none at worst. Foth, Menzel, & Schröder (2000) propose a transformation-based approach in order to overcome this problem. The basic idea is to start with an arbitrary dependency analysis for a sentence and to repair it step by step guided by the weights of the constraints. But this method can not guarantee to find the optimal solution in all cases. The running time is clearly better than a complete search in practice, but still remains exponential in the worst case.[2]

In contrast to the above mentioned approaches Koskenniemi (1990) presents an eliminative approach to dependency parsing which is implemented by means of *Finite State Automata (FSA).* He proposes the *Finite State Intersection Grammar (FSIG)* which is based on the methodological paradigm of *Constraint Grammar (CG)* (Karlsson (1990)). Rules are written via regular expressions which make use of the so-called *context restriction operation.* FSIG combines different levels of disambiguation, namely part-of-speech disambiguation, clause boundary recognition and syntactic disambiguation. A fundamental difference between CDG and FSIG is that the weak generative capacity of the latter corresponds to regular grammars and that the latter produces underspecified dependency structures represented as syntactic tags. However, Yli-Jyrä (2004) proposes an extension to FSIG where it is possible to represent complete non-projective dependency structures by colored bracketing. In such an encoding scheme dependencies are indicated implicitly via matching pairs of symbols. An integration of soft constraints is proposed in Didakowski (2008b) via *Weighted Finite State Automata (WFSA).* This approach enables to formulate "linguistic criteria" which are ranked by preference in order to model e.g. degrees of grammaticality and structural preferences separately. Parsing with FSIG can

[2]The algorithm stores intermediary results. If the algorithm is interrupted early a current optimal result can be returned. That is, the algorithm has the so-called anytime property (Foth, Menzel, & Schröder (2000)).

lead to finite state machines that are not viable any more. This is due to enormous internal results during parsing even if the space complexity is linear in the worst case (cf. Tapanainen (1997)). In Didakowski (2008a) an approach based on *Weighted Finite-State Transducers (WFST)* is presented which minimizes internal results during parsing. Constraints are written for partial dependency trees separately and they are applied in such a fashion that bottom-up parsing is performed. But the approach is restricted to projective dependency structures and the writing of the constraints is more a spelling out of weighted chunk patterns.

1.1 Motivation

Yli-Jyrä (2001) shows that FSIG can be framed to a CSP and solved by a constraint solver instead. Vempaty (1992) on the other hand introduces the idea of representing and solving a CSP with the help of acyclic FSAs which recognize finite sets of constraint tuples whose scope is over all constraint variables. He shows that some problems can be solved by FSAs much faster in comparison to other approaches. Amilhastre, Fargier, & Marquis (2002) extend Vampaty's approach by a degree of importance which is associated with each constraint. This valuation is modeled via the weights of an WFSA and allows to order solutions by preference. But a general representation and solving of COPs with the help of FSMs is not addressed in the literature yet. The weight structure of an WFSA or WFST can be exchanged via a semiring structure. How could this relate to different CSP schemes like the weighted CSP, the probabilistic CSP, the fuzzy CSP which allow to model COPs? Is it possible to represent quantified constraints with the help of cyclic FSMs in a more compact manner and furthermore, is it possible to represent a constraint relation separated from its scope?

A basic argument for using weights, costs, rankings, etc. as an extension of CSPs is to represent real-life scenarios where the knowledge is neither completely available nor crisp. In such scenarios the ability of stating whether an instantiation of variables is allowed or not is insufficient or sometimes even impossible. Robust parsing of natural language has to cope with gradation as an intrinsic property of linguistic data (see Aarts (2007)). Despite this computational complication there also exist some properties of natural language which may reduce the complexity of the parsing problem. Karlsson (2010) shows that an absolute limit on center-embeddings exists in written and spoken language.[3] Yli-Jyrä (2003) shows for Danish that the nested crossing depth of dependencies is in general pretty low.[4] By ignoring some mathematical beauty the question is whether a regular approximation is sufficient for a real-life scenario like natural language parsing (see Mohri & Sproat (2006)). However, a regular approximation may have advantages for practical systems. The complexity of dependency trees could be parametrized (cf. Yli-Jyrä (2004)) e.g. the maximum depth of a dependency tree, the maximal crossings of dependency links etc. Limits

[3]Karlsson (2010) studies different types of recursion and iteration in written and spoken language empirically and examines empirical determinable constraints on the number of recursive and iterative cycles.

[4]Yli-Jyrä (2003) studies a collection of constraints imposed on the non-projectivity of dependency structures in natural language on the basis of the Danish Dependency Treebank (DDT) empirically.

on structural complexity could resolve some ambiguity and by this means the number of non-typical analysis could be reduced. Problems could be simplified so that the solving complexity is lowered. It may even be possible to transform an intractable problem into a tractable approximated one. Furthermore, a problem could be split up into its regular parts and covered within an extended finite-state approach.

In my opinion the possibilities of FSMs have not been fully exploited not only in the area of constraint processing but also in robust dependency parsing. I think that approaches in the area of constraint processing can help to make the parsing with FSMs efficient. That means, in this article constraint processing techniques and finite-state techniques are brought together in order to realize efficient dependency parsing. The approach represented in this article is based on the *Semiring-based CSP (SCSP)*, a general framework which is proposed by Bistarelli, Montanari, & Rossi (1997). Different instances of the framework may correspond to known or new CSP schemes via specific choices of the *semiring*. It is shown that SCSPs can be represented and solved with FSMs or rather WFSTs in a natural way, that the approach is well capable of solving different SCSPs simultaneously and that *tree decomposition* is the key for efficient processing. This article is not about how exactly to model linguistic information or rules, or how to find an adequate cost-structure but a general chart is given how to implement dependency parsing as COP within the FSM framework.

According to this motivation the article is organized as follows: section 2 gives basic definitions and notations. The following section 3 shows how a SCSP can be represented by WFSTs. Problem solving over such a representation is demonstrated in section 4. Finally, section 5 shows how finite state techniques can be used to realize dependency parsing as an optimization problem.

2 Definitions and notations

Costs, probabilities, rankings, etc. are implemented via a semiring structure in order to be independent of a concrete instantiation. Let $S \neq \emptyset$ be a set and let $\oplus$ (called addition) and $\otimes$ (called multiplication) be binary operations on S, then $(S, \oplus, \otimes, \bar{0}, \bar{1})$ is called a semiring if $(S, \oplus, \bar{0})$ is a commutative monoid, $(S, \otimes, \bar{1})$ is a monoid and $\otimes$ distributes over $\oplus$. Furthermore, a semiring is called *c-semiring* if the operation $\oplus$ induces a *partial order* over S. The partial order is defined as follows: $a \leq_S b$ iff $a \oplus b = b$. This order is used to compare elements in S where $a \leq_S b$ intuitively means that b is "better" than a (cf. Bistarelli, Montanari, & Rossi (1997)).

In this article the idea of representing and solving COPs using WFSTs is explored. A WFST $T = (\Sigma_1, \Sigma_2, Q, q_0, F, E, \lambda, \rho)$ over a semiring S is an 8-tuple such that Σ_1 is the finite input alphabet, Σ_2 is the finite output alphabet, Q is the finite set of states, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of final states, $E \subseteq Q \times (\Sigma_1 \cup \epsilon) \times (\Sigma_2 \cup \epsilon) \times S \times Q$ is the set of transitions, λ is the initial weight and $\rho : F \to S$ is the final weight function mapping final states to elements in S. A WFST is capable of recognizing a rational transduction of S-rational series (cf. Kuich & Salomaa (1985)). The concepts of regular languages and rational series are used extensively in order to demonstrate the approach.

In the context of regular languages and rational series, Σ and Δ with $\Delta \subset \Sigma$ denote alphabets, and the standard extended regular expression operations are used:[5] complement ($\overline{A}$), Kleene closure (A^*), Kleene plus (A^+), iterated concatenation (A^n), inversion (A^{-1}), concatenation (AB), union ($A \cup B$), intersection ($A \cap B$), cross product ($A \times B$), composition ($A \circ B$), domain ($Dom(A)$) and range ($Range(A)$). Additionally, the optional rewriting operator $A \updownarrow {}^{\alpha}{}_{\beta}$ is defined by $A \circ ((\Sigma^*(\alpha \times \beta))^*\Sigma^*)$ and the corresponding variant $A \downarrow {}^{\alpha}{}_{\beta}$ is defined by $Range(A \updownarrow {}^{\alpha}{}_{\beta})$. Furthermore, the mandatory rewriting operator $A \Updownarrow {}^{\alpha}{}_{\beta}$ is defined by $A \circ ((\overline{\Sigma^*\alpha\Sigma^*}(\alpha \times \beta))^*\overline{\Sigma^*\alpha\Sigma^*})$ and the corresponding variant $A \Downarrow {}^{\alpha}{}_{\beta}$ is defined by $Range(A \Updownarrow {}^{\alpha}{}_{\beta})$. The precedence of the operators corresponds to the order in which they are listed. The empty string is denoted by ϵ and a polynomial $\omega\epsilon$ with $\omega \in S$ is denoted by $\langle\omega\rangle$. The distinction between a regular language A and the identity relation which maps every string of A onto itself and the distinction between a rational series of which all coefficients are $\bar{1}$ and their support is ignored. The representation of $\mathcal{X}$ concerning formal languages and formal series is denoted by $L(\mathcal{X})$ and the automata-theoretic representation is denoted by $A(\mathcal{X})$.

3 Representation of SCSPs

The *Semiring-based CSP (SCSP)* is a unifying framework for a variety of extensions of the CSP formalism where finite domains are presumed. It is based on a semiring structure which allows to represent for example crisp, weighted, fuzzy, probabilistic or set-based CSPs depending on the instantiation of this structure. The main idea of Bistarelli, Montanari, & Rossi (1997) behind this is the following:

> "[...] a semiring (that is, a domain plus two operations satisfying certain properties) is all that is needed to describe many constraint satisfaction schemes. In fact, the domain of the semiring provides the levels of consistency (which can be interpreted as cost, or degree of preference, or probabilities, or others), and the two operations define a way to combine constraints together. More precisely, we define the notion of constraint solving over any semiring. Specific choices of the semiring will then give rise to different instances of the framework, which may correspond to known or new constraint solving schemes."

Formally a SCSP consists in a *constraint system*, *constraints* and a *constraint problem*. In this section these notions which are parametric with respect to the notion of the c-semiring are defined and mapped to regular languages and rational series. The definitions of SCSPs given in Bistarelli, Montanari, & Rossi (1997) are used and it is tried to be as close as possible to these definitions in order to exemplify our approach.

3.1 Constraint system

A constraint system specifies the c-semiring to be used along with the set of all variables and their domain. Formally, a constraint system $CS = (S, D, V)$ is

[5] The operations are mainly named with regard to the operations on regular languages and relations but the mapping to rational series should be obvious (see Kuich & Salomaa (1985)).

a 3-tuple such that S is a c-semiring, V is an ordered set of variables and the finite set D is their domain (cf. Bistarelli, Montanari, & Rossi (1997)).

A value of the domain can be represented as a symbol in the alphabet of a regular language since the domain is finite. But, concerning the automata-theoretic representation it is advisable to keep the size of the alphabet small for practical reasons. Therefore, a value (numerical, symbolic etc.) can be represented in a special coding as a string instead. Given the domain $D = \{d_1, d_2, \dots, d_m\}$ and a mapping $\varphi : D \rightarrow \Delta^+$ defined by $\varphi(d_r) = d_r{}'$ with $d_r{}' \in \Delta^+$ for every $r \leq m$ then a value d_r is represented by the finite regular language (singleton):

$$L(d_r) = \{\varphi(d_r)\} \tag{1}$$

and the domain D is represented by the union of the individual values:

$$L(D) = L(d_1) \cup L(d_2) \cup \dots \cup L(d_m) \tag{2}$$

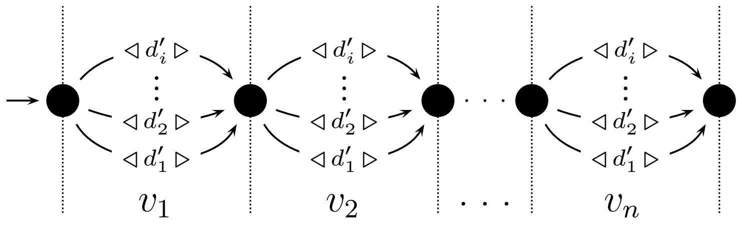
Then a variable $v \in V$ can be represented as a finite regular language containing all possible values of the domain. Thus, a variable is represented in an extensional way:

$$L(v) = L(D) \tag{3}$$

The set of variables V can be represented by a finite regular language accepting a sequence of such variables. If V is totally ordered via the ordering $\succ$ and $v_1 \succ v_2 \succ \dots \succ v_n$ then V is represented by the concatenation of all variables in the defined order where each variable is marked by the special symbols $\triangleleft, \triangleright \in \Sigma - \Delta$ giving a start and an end point of their values:

$$L(V) = \triangleleft\, L(v_1)\, \triangleright \triangleleft\, L(v_2)\, \triangleright \cdots \triangleleft\, L(v_n)\, \triangleright = (\triangleleft\, L(v)\, \triangleright)^{|V|} \tag{4}$$

The WFST $A(V)$ representing V is sketched in figure 1 (the weights of the identity WFST are left out in the figure).



$\triangleleft\, d'_i\, \triangleright$
$\triangleleft\, d'_2\, \triangleright$
$\triangleleft\, d'_1\, \triangleright$
v_1
v_2
v_n

Figure 1: WFST representing the set V

In order to represent a subset of V each variable which is not in the subset is marked by the symbols $\blacktriangleleft, \blacktriangleright \in \Sigma - \Delta$ instead of the symbols $\triangleleft$ and $\triangleright$. Then, the power set $\mathcal{P}(V)$ can be represented as a finite regular language:

$$L(\mathcal{P}(V)) = \begin{pmatrix} \triangleleft\, L(v_1)\, \triangleright \\ \cup \\ \blacktriangleleft\, L(v_1)\, \blacktriangleright \end{pmatrix} \begin{pmatrix} \triangleleft\, L(v_2)\, \triangleright \\ \cup \\ \blacktriangleleft\, L(v_2)\, \blacktriangleright \end{pmatrix} \cdots \begin{pmatrix} \triangleleft\, L(v_n)\, \triangleright \\ \cup \\ \blacktriangleleft\, L(v_n)\, \blacktriangleright \end{pmatrix} = \begin{pmatrix} \triangleleft\, L(v)\, \triangleright \\ \cup \\ \blacktriangleleft\, L(v)\, \blacktriangleright \end{pmatrix}^{|V|} \tag{5}$$

The WFST $A(\mathcal{P}(V))$ representing the power set of V is sketched in figure 2 (the weights of the identity WFST are left out in the figure). According to that, the

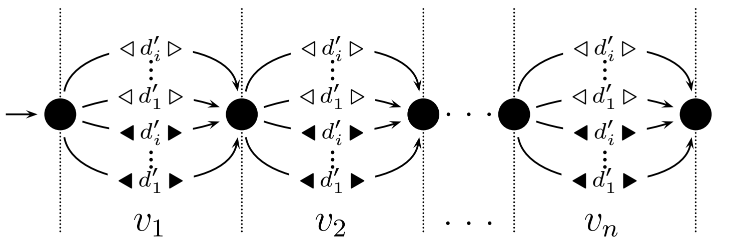

$\triangleleft\, d'_i \,\triangleright$
$\triangleleft\, d'_1 \,\triangleright$
$\blacktriangleleft\, d'_i \,\blacktriangleright$
$\blacktriangleleft\, d'_1 \,\blacktriangleright$
v_1
v_2
$\cdots$
v_n

Figure 2: WFST representing the power set of V

empty set $\varnothing$ is represented by the regular language:

$$L(\varnothing) = \blacktriangleleft L(v_1) \blacktriangleright \blacktriangleleft L(v_2) \blacktriangleright \cdots \blacktriangleleft L(v_n) \blacktriangleright = (\blacktriangleleft L(v) \blacktriangleright)^{|V|} \tag{6}$$

The union of two subsets of V is denoted by $L(A_1) \sqcup L(A_2)$ with $L(A_1), L(A_2) \subseteq L(P(V))$. This operation can be defined via the extended regular expression operations:

$$\begin{gathered} L(A_1 \cup A_2) = L(A_1) \sqcup L(A_2) = \\ \left(\begin{array}{c} L(A_1) \Downarrow^{\triangleleft}{}_{\triangleleft_1} \Downarrow^{\triangleright}{}_{\triangleright_1} \downarrow^{\blacktriangleleft}{}_{\triangleleft_2} \downarrow^{\blacktriangleright}{}_{\triangleright_2} \downarrow^{\triangleleft_1}{}_{\triangleleft_2} \downarrow^{\triangleright_1}{}_{\triangleright_2} \\ \cap \\ L(A_2) \Downarrow^{\triangleleft}{}_{\triangleleft_2} \Downarrow^{\triangleright}{}_{\triangleright_2} \downarrow^{\blacktriangleleft}{}_{\triangleleft_1} \downarrow^{\blacktriangleright}{}_{\triangleright_1} \end{array} \right) \Downarrow^{\triangleleft_2}{}_{\triangleleft} \Downarrow^{\triangleright_2}{}_{\triangleright} \Downarrow^{\triangleleft_1}{}_{\triangleleft} \Downarrow^{\triangleright_1}{}_{\triangleright} \end{gathered} \tag{7}$$

The markers $\triangleleft$ and $\triangleright$ are renamed with $\triangleleft_1$ and $\triangleright_1$ in $L(A_1)$ via the rewriting $\Downarrow^{\triangleleft}{}_{\triangleleft_1} \Downarrow^{\triangleright}{}_{\triangleright_1}$ and the markers $\triangleleft$ and $\triangleright$ are renamed with $\triangleleft_2$ and $\triangleright_2$ in $L(A_2)$ via the rewriting $\Downarrow^{\triangleleft}{}_{\triangleleft_2} \Downarrow^{\triangleright}{}_{\triangleright_2}$. With the help of this renaming it is possible to expand A_1 with A_2 by accepting variables of A_2 beyond A_1 via the optional rewriting $\downarrow^{\blacktriangleleft}{}_{\triangleleft_2} \downarrow^{\blacktriangleright}{}_{\triangleright_2}$ and it is possible to expand A_2 with A_1 by accepting variable values of A_1 beyond A_2 via the optional rewriting $\downarrow^{\blacktriangleleft}{}_{\triangleleft_1} \downarrow^{\blacktriangleright}{}_{\triangleright_1}$ at the same time. Common variables of the two sets are handled in connection with $L(A_1)$ via the optional rewriting $\downarrow^{\triangleleft_1}{}_{\triangleleft_2} \downarrow^{\triangleright_1}{}_{\triangleright_2}$. In the end the calculation of the union is performed via the intersection operation. Afterwards the renaming is reversed.

3.2 Constraint

A *constraint* over a given constraint system specifies the involved variables and the allowed tuples of values for those variables and assigns an element in S to each tuple. This element can be interpreted as the tuple weight or cost or level of confidence or any other measurable feature. Formally, a constraint is a pair (def, con) where $con \subseteq V$ is called the *type* of a constraint and $def : D^{|con|} \to S$ is called the *value* of a constraint. That is, the value of a constraint is a mapping of tuples of values to elements in S and the type of a constraint represents the scope of the tuples (cf. Bistarelli, Montanari, & Rossi (1997)).

From the point of view of regular languages and rational series a constraint can be represented as a pair $(L(def), L(con))$ consisting of a S-rational series and a regular language representing its value and type.[6] The type of a constraint is a subset of V and therefore can be represented as a subset of

[6]It is possible to represent a constraint as a rational series of a regular language defining a

$L(\mathcal{P}(V))$ denoting *con*. Given $\vartheta : \mathcal{P}(V) \to L(\mathcal{P}(V))$ mapping every subset of V to a corresponding subset of $L(\mathcal{P}(V))$ denoting the same variables, then $L(con)$ is defined by:

$$L(con) = \vartheta(con). \tag{8}$$

The allowed tuples of values for the variables in *con* can be extensionally represented as a subset of $(\triangleleft L(D) \triangleright)^{|con|}$. Therefore, $L(def)$ is a S-rational series with finite support defining a function mapping tuples of values for *con* to elements in S:

$$L(def) : (\triangleleft L(D) \triangleright)^{|con|} \to S \tag{9}$$

Quantified constraints with existentially or universally quantified variables can be represented in a more compact and less extensional manner. This can be achieved by the following approach: It is assumed that the tuples of values of quantified constraints have no fixed arity and therefore the value of a quantified constraint is defined by the mapping $def : D^+ \to S$ where the infinite set of tuples over D is denoted by D^+. The value of such a constraint can be represented as a rational series with infinite support defining a function:

$$L(def) : (\triangleleft L(D) \triangleright)^+ \to S \tag{10}$$

In order to illustrate this approach a WFST recognizing the value of a constraint including one existentially quantified variable v_k which is constrained to the variable values $d_{k_1} d_{k_2} \dots d_{k_m}$ is sketched in figure 3 (the weights of the identity WFST are left out in the figure). Via this underspecified representation it is

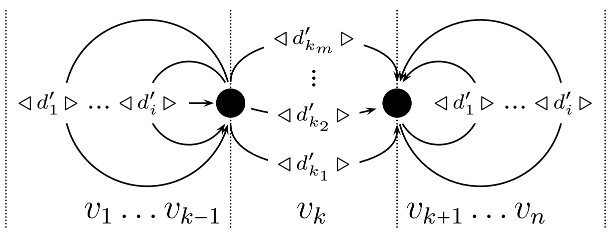

$\triangleleft d'_1 \triangleright \dots \triangleleft d'_i \triangleright$
$\triangleleft d'_{k_m} \triangleright$
$\vdots$
$\triangleleft d'_{k_2} \triangleright$
$\triangleleft d'_{k_1} \triangleright$
$\triangleleft d'_1 \triangleright \dots \triangleleft d'_i \triangleright$
$v_1 \dots v_{k-1}$
v_k
$v_{k+1} \dots v_n$

Figure 3: WFST representing one existentially quantified variable

possible to define the value of a quantified constraint without knowing its type. In the following a quantified constraint which has an unknown type is denoted by $(def, \star)$ and $(L(def), \star)$ respectively and a set of such constraints is denoted by $C^\star$.

Predicate logic formula where propositions apply to substrings (cf. Hulden (2008) and Yli-Jyrä & Koskenniemi (2004)) can be used to realize such quantified constraints where the techniques have to be expanded using weights or costs etc. (cf. Didakowski (2008b)). Note that unbound crossing or embedding dependencies are not covered by such constraints since they are regular.

3.3 Constraint problem

A constraint problem P over a constraint system CS is a pair $P = (C, con)$ where C is a set of constraints and $con \subseteq V$, the type of the constraint problem,

function from $L(P(V))$ to S, too. In this representation the value and type remain accessible. However, in order to alleviate the issue we want to be as close as possible to the approach of Bistarelli, Montanari, & Rossi (1997).

is a set of variables of which it is desired to know the possible assignments satisfying all constraints. Here, in contrast to the work of Bistarelli, Montanari, & Rossi (1997) it is not assumed that $(\mathit{def}_1, \mathit{con}') \in C$ and $(\mathit{def}_2, \mathit{con}') \in C$ implies $\mathit{def}_1 = \mathit{def}_2$.

The type of a constraint system can be represented as the finite regular language $L(\mathit{con}) \subseteq L(\mathcal{P}(V))$ and a constraint in C as a pair consisting of a rational series and a regular language as defined in section 3.2.

4 Solving of SCSPs

This section deals with the solving of SCSPs with WFSTs. In the SCSP scheme, the semiring values assigned to the tuples of each constraint are used to compute corresponding semiring values for the tuples of values assigned to the variables of the constraint problem via the semiring operations $\oplus$ and $\otimes$. This is facilitated by the two operations over constraints *combination* ($\otimes$) and *projection* ($\Downarrow$) which are based on the tuple projection ($\downarrow$). By means of these operations the solution of a SCSP can be defined.

4.1 Tuple projection

Given two sets of variables I and I' with $I' \subseteq I \subseteq V$, then $t \downarrow {}^{I}{}_{I'}$ denotes the projection of any tuple of values t from the set of variables I to the set of variables I'. The tuple projection can be realized by a regular relation defining a mapping from $(\triangleleft\, L(D)\, \triangleright)^{|I|}$ to $(\triangleleft\, L(D)\, \triangleright)^{|I'|}$ which is denoted by $L(t \downarrow {}^{I}{}_{I'})$. Such a relation can be constructed easily via the variable sets $L(I)$ and $L(I')$ by the extended regular expression operations:

$$L(t \downarrow {}^{I}{}_{I'}) = (((L(I) \updownarrow {}_{\blacktriangleleft}{}^{\triangleleft} \updownarrow {}_{\blacktriangleright}{}^{\triangleright} \circ L(I')) \Updownarrow {}^{\blacktriangleleft \Delta^{+} \blacktriangleright}{}_{\epsilon})^{-1} \Updownarrow {}^{\blacktriangleleft \Delta^{+} \blacktriangleright}{}_{\epsilon})^{-1} \qquad (11)$$

The set of variables $L(I)$ is mapped to the set of variables $L(I')$ via the optional rewriting $\updownarrow {}_{\blacktriangleleft}{}^{\triangleleft} \updownarrow {}_{\blacktriangleright}{}^{\triangleright}$ and via the composition operation. Then the tuples of values for I and I' are isolated via the rewriting $\Updownarrow {}^{\blacktriangleleft \Delta^{+} \blacktriangleright}{}_{\epsilon}$ where the inversion operation helps to apply the rewriting to both sets of variables.

4.2 Combination

Combining two constraints means building a new constraint involving all the variables in the types of the original ones and computing a tuple projection for both constraints. The semiring values of the tuples of the resulting constraint are the multiplication ($\otimes$) of the semiring values of the appropriate sub-tuples associated to the original constraints (cf. Bistarelli, Montanari, & Rossi (1997)).

The combination of two constraints $(L(\mathit{def}_1), L(\mathit{con}_1)) \otimes (L(\mathit{def}_2), L(\mathit{con}_2))$ is the constraint $(L(\mathit{def}), L(\mathit{con}))$. The type can be calculated via the union operation over subsets of V (see subsection 3.1):

$$L(\mathit{con}) = L(\mathit{con}_1) \sqcup L(\mathit{con}_2) \qquad (12)$$

The value can be calculated by composing $L(\mathit{def}_1)$ and $L(\mathit{def}_2)$ respectively with the tuple projection $L(t \downarrow {}^{\mathit{con}}{}_{\mathit{con}_1})$ and $L(t \downarrow {}^{\mathit{con}}{}_{\mathit{con}_2})$ respectively and by taking the intersection of the domains of the results:

$$L(def) = \left(\begin{array}{c} Dom(L(t \downarrow^{con}{}_{con_1}) \circ L(def_1)) \\ \cap \\ Dom(L(t \downarrow^{con}{}_{con_2}) \circ L(def_2)) \end{array} \right) \tag{13}$$

If it is assured that the constraints $(L(def_1), L(con))$ and $(L(def_1), L(con))$ have the same type the combination is the constraint $(L(def_1) \cap L(def_2), L(con))$ and if both types of the constraints are unknown in case of quantified constraints the combination is the constraint $(L(def_1) \cap L(def_2), \star)$.

4.3 Projection

With help of the projection operation a constraint can be projected over a set of variables in order to obtain a new constraint (cf. Bistarelli, Montanari, & Rossi (1997)). Given the constraint type *con* and the set of variables I with $I \subseteq con \subseteq V$, then the projection denoted by $(L(def), L(con)) \Downarrow_I$ is the constraint $(L(def'), L(con'))$. The type of the constraint is the regular language:

$$L(con') = L(I) \tag{14}$$

The value is calculated by applying the tuple projection $L(t \downarrow^{con}{}_I)$ to $L(def)$:

$$\begin{array}{c} L(def') = \\ Range(L(def) \circ L(t \downarrow^{con}{}_I)) \end{array} \tag{15}$$

4.4 Solution

Given a constraint problem $P = (C, con)$ the solution of a SCSP is defined as the constraint induced on the variables in *con* by the whole constraint problem. For each tuple of values for the variables in *con* such a constraint provides an associated semiring value. First, all constraints in C are combined via the combination operation, then the resulting constraint is projected over *con* (cf. Bistarelli, Montanari, & Rossi (1997)).

Using the definitions mentioned above the solution of a CSP is the constraint:

$$(\bigotimes_{(L(def), L(con')) \in C} (L(def), L(con'))) \Downarrow_{con} \tag{16}$$

Note that the intersection operation as well as the composition operation presume that the semiring operation $\otimes$ is commutative (cf. Kuich & Salomaa (1985)). However, given a c-semiring which is commutative the solving of a SCSP with WFSTs works fine.

The time and space complexity of solving a SCSP using WFSTs is exponential in respect to the number of constraints in the worst case. That is due to the composition operation for which the intersection is the special case where both operands are WFSTs with identical input and output labels for each transition. The time and space complexity for this operation is $\mathcal{O}(|N| \times |M|)$ where $|N|$ and $|M|$ denote the number of states of the WFST N and M respectively. In order to solve a SCSP this composition operation is applied repeatedly bound by the number of constraints (cf. Vempaty (1992)). Note, that the order in which the constraints are combined may have an enormous effect on the size of the WFSTs during problem solving. But keep in mind that the finite-state techniques are well capable of combining constraints in an off-line preprocessing step.

4.5 Best level of consistency

The *best level of consistency* gives an idea about how much the constraints of a given problem can be satisfied (cf. Bistarelli, Montanari, & Rossi (1997)).

Given a constraint problem $CP = (C, con)$, the best level of consistency of a CP is defined as the constraint $blevel(CP) = (\bigotimes_{(def,con)\in C}(L(def), L(con))) \Downarrow_{\varnothing}$ whose value is a S-rational series mapping the empty string to the corresponding level of consistency and whose type is a subset of $L(P(V))$ denoting the empty set $(L(\varnothing))$.

A more interesting notion of solution provides only the tuples that have an associated semiring value which coincides with the value of $blevel(CP)$. Here $\leq_S$ has to define a total order. Otherwise it could happen that none of the tuples has an associated semiring value equal to the value of $blevel(CP)$. Given such a total order a *single source shortest path algorithm* can be used in order to obtain the desired tuples out of a WFST. The best path search can be calculated in $\mathcal{O}(|Q| + |E|)$ in the acyclic case if Q is the set of states and E is the set of transitions (cf. Mohri (2002)). If S contains multiobjectives (multicriteria) and $\leq_S$ does not define a total order a *multiobjective shortest path algorithm* can be used (see Tarapata (2007)).

4.6 Infinite domains

In this section it is discussed how to handle infinite domains. The basic idea is to represent an infinite domain as an infinite regular language. If for example the domain consists of all possible nonempty strings over the alphabet Δ the values can be represented by the infinite regular language Δ^+. Thus, an infinite domain can be covered in a constraint system since it is possible to represent the infinite domain as an infinite regular language. But the formulation of constraints over infinite domains is slightly limited. It is for example impossible to formulate that two variables equal in an extensional way.[7] That is, a constraint can not cover the copy language as well as the mirror language or the palindrome language. However, constraints over infinite domains can be formulated if they are regular. Given the above mentioned domain it is for example possible to define an upper or lower bound for the string length, to express equality presuming a bound string length, to forbid some characters or simply to assign concrete values.

4.7 Information assigned to variables

Sometimes information is associated with the variables in V and therefore with the variable positions in $L(V)$. If quantified constraints as defined in subsection 3.2 are used this information is inaccessible in the formulation of such a constraint because the variable positions can not be identified directly. To overcome this problem, the information associated with the variables can be coded on string level and added to the variables in $L(V)$. Given the set of variables $V = \{v_1, v_2, \dots, v_n\}$, the information associated with a variable is given by the mapping $\phi : V \to \Delta^*$ defined by $\phi(v_k) = i_k$ with $i_k \in \Delta^*$ for every $k \leq n$. The marker $\diamond \in \Delta$ is used to separate the information from the variable values. Then

[7]Equality could only be formulated by means of special symbols marking the corresponding variables where the equality is denoted in an intensional way.

a variable v_k together with their associated information can be represented as the regular language:

$$L(v_k) = \{\phi(v_k)\} \diamond L(D) \tag{17}$$

The variable information can be used in the formulation of constraints in order to identify sets of variables via the projection operation.

4.8 Simultaneous solving

If quantified constraints are used (see subsection 3.2) and if information is added to the variables in $L(V)$ (see subsection 4.7) it is possible to represent different constraint systems compactly in one representation provided that they are defined over the same semiring and that they have the same domain. Via this representation constraint problems can be solved simultaneously on different constraint systems.

In order to define the simultaneous solving the simultaneous projection has to be defined first. In this connection the simultaneousness of α and β is written as α/β. Given the constraint systems $(S, D, V_1), (S, D, V_2), \ldots, (S, D, V_n)$ and given the constraint types $con_1, con_2, \ldots, con_n$ and the sets of variables $I_1, I_2, \ldots, I_n$ with $I_1 \subseteq con_1 \subseteq V_1, I_2 \subseteq con_2 \subseteq V_2, \ldots, I_n \subseteq con_n \subseteq V_n$, then the simultaneous projection of quantified constraints which is denoted by $(L(def), L(con_1)/L(con_2)/\ldots/L(con_n)) \Downarrow_{I_1/I_1/\ldots/I_n}$ is the constraint $(L(def_1/def_2/\ldots/def_n), L(con'_1/co$
The type of the constraint is the regular language:

$$L(con'_1/con'_2/\ldots/con'_n) = L(I_1) \cup L(I_2) \cup \cdots \cup L(I_n) \tag{18}$$

and the value is the S-rational series:

$$\begin{gathered} L(def_1/def_2/\ldots/def_n) = \\ Range(L(def) \circ (L(t \downarrow^{con_1}{}_{I_1}) \cup L(t \downarrow^{con_2}{}_{I_2}) \cup \cdots \cup L(t \downarrow^{con_n}{}_{I_n}))) \end{gathered} \tag{19}$$

With help of the simultaneous projection the simultaneous solution can be defined. Given the constraint systems $(S, D, V_1), (S, D, V_2), \ldots, (S, D, V_n)$ and given the constraint types $con_1, con_2, \ldots, con_n$ and $con'_1, con'_2, \ldots, con'_n$ with $con'_1 \subseteq con_1 \subseteq V_1, con'_2 \subseteq con_2 \subseteq V_2, \ldots, con'_n \subseteq con_n \subseteq V_n$ and given the constraint problem over different variable sets $(C^\star, L(con_1)/L(con_2)/\ldots/L(con_n))$, then the simultaneous problem solving is defined as follows. First the quantified constraints with unknown type are combined:

$$(def, \star) = (\bigotimes\nolimits_{(L(def'),\star) \in C^\star} (L(def'), \star)) \tag{20}$$

The type $L(con'_1)/L(con'_2)/\ldots/L(con'_n)$ is assigned to the resulting constraint. Then the constraint is projected over the (simultaneous) type $con_1/con_2/\ldots/con_n$:[8]

$$(def, L(con'_1)/L(con'_2)/\ldots/L(con'_n)) \Downarrow_{con_1/con_2/\ldots/con_n} \tag{21}$$

Here it is not necessary that the constraint systems differ from each other. That is the definition not only includes the possibility to solve different constraint problems on different constraint systems simultaneously but also the possibility to solve different constraint problems on the same constraint system. Note that in this approach the best level of consistency is calculated over all involved SCSPs.

[8]This approach has some similarities to the MUSE CSP proposed by Helzerman & Harper (1996). Via the MUSE CSP it is possible to represent several CSPs compactly provided that they have some common variables which have the same domains and constraints. By this, the work required to apply constraints can be reduced.

5 Dependency parsing

In this section, it is shown how to implement dependency parsing as a constraint optimization problem within a finite state approach. After a short introduction into WCDG it is shown how to implement its basic features within the finite state framework including some new perspectives. Then, a problem decomposition approach is presented making efficient non-projective dependency parsing with WFSTs possible.

5.1 WCDG – a short overview

In the WCDG formalism natural language parsing is treated as a disambiguation problem over initially totally ambiguous dependency relations where it is desired to find the most preferable dependency structures by means of quantitative preferences. This optimization problem is realized by an instance of the *Valued Constraint Satisfaction Problem (VCSP)* which is similar to the additive VCSP but it uses the multiplication operation for combining penalties for computational reasons (Schröder (2002)). The penalty of a constraint can take a factor between 0 to 1 where 0 denotes a crisp constraint and where a constraint with the penalty 1 has no effect. This instance of the VCSP allows to express that one dependency structure is more preferable than another via negative preferences.

Dependency parsing of natural language can be mapped to a constraint optimization problem if one manages to specify what the constraint variables are, how value assignments represent dependency structures and how constraints can be used to find these appropriate value assignments. The constraint variables are specified as follows. Each word in a sentence is represented as an individual variable. The values for these variables are pairs consisting of the information about the kind of dependency relation the word is involved as a dependent and about the position of the dominating head. In the following the kind of dependency relation is called dependency label and the position of the dominating head is called head position. A final assignment of variables for the syntactic dependency tree in figure 4 is given by: v_1 = (DET, 2), v_2 = (SUBJ, 3), v_3 = (S, NIL), v_4 = (AUX, 3), v_5 = (PP, 4), v_6 = (DET, 7), v_7 = (PN, 5). The numbering of the variables corresponds to the word positions. The root node is indicated via the head position NIL.

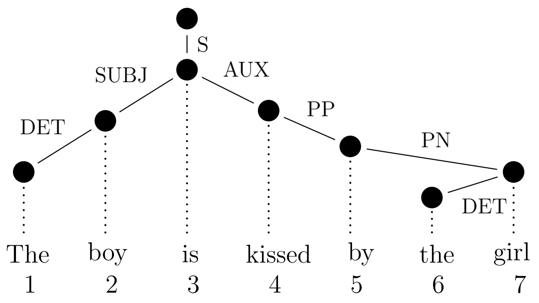


Figure 4: syntactic dependency tree

In order to include the handling of lexical ambiguities and the handling of several representational levels the definition of a constraint variable is expanded as follows: For each reading of a word in a sentence and each representational level an individual variable is defined. In this connection the head position covered by a variable is extended by the information about the reading of a word. Furthermore, it is possible to identify sets of variables by their representational level.

Conditions which can be of different strength are formulated as unary or binary constraints via first-order predicate calculus formulas which are implicitly assumed to be universally quantified. All subtrees must satisfy these formulas which allow to model linguistic phenomena like degrees of grammaticality and structural preferences. In WCDG it is possible to formulate constraints which receive a fixed penalty. An example formula concerning the lexical categories which are involved in the object relation is given:

$$\begin{array}{l}\{\text{X:SYN}\} : \text{'object category'} : 0.1 : \\ \text{X.label} = \text{OBJ} \text{ -} > \text{X@cat} = \text{NN X\^{}cat} = \text{VVFIN}\end{array} \tag{22}$$

The formula states that it is strongly preferred (a penalty of 0.1) that all dependency edges (*X:SYN*) with the label OBJ (*X.label = OBJ*) have a noun as modifier (*X@cat = NN*) and a finite verb as governor (*X^cat = VVFIN*). It is also possible to formulate so-called *dynamic constraints* which do not have a fixed penalty. They receive a penalty depending on the context in which they are evaluated. Such constraints can be used for example to favor short distances of words. The longer the distance, the higher the penalty. They can also be used to give a penalty to specific words in a lexicon or to give a penalty for missing arguments specified by a lexicon. An example formula concerning the preference of short edges is given:

$$\begin{array}{l}\{\text{X!SYN}\} : \text{'prefer short edges'} : [\ \exp([\ 1 \text{ - abs(X@to - X\^{}to) }]/10)\ ] : \\ \text{abs(X@to - X\^{}to)} \ < \ 2\end{array} \tag{23}$$

The formula states that all dependency edges except those that have NIL as governor (*X!SYN*) are preferred to have a short distance between modifier and governor (*abs(X@to - X^to)*). In McCrae, Foth, & Menzel (2008) an extension of these 'local' constraints to global phenomena via the additional predicates *is* and *has* is presented. The first predicate expresses conditions on the dependents of a given word and the latter expresses conditions on the dependency edge above a given word. The conditions which increase the expressivity of the WCDG constraints are formulated by means of *ancillary constraints.* Via cascaded and recursive invocations of these constraints additional conditions are not limited in applicability to neighbouring edges.

5.2 Naive implementation of WCDG

WCDG is implemented within the VCSP framework as mentioned in the previous section. Bistarelli, Montanari, Rossi, Schiex, Verfaillie et al. (1999) show that it is possible to pass from any VCSP problem to an equivalent SCSP problem and vice-versa provided the semiring of the SCSP is totally ordered. Furthermore, in section 3 it is shown that a SCSP can be represented and solved with WFSTs in a natural way. Thus, a VCSP can be represented and solved with WFSTs indirectly via a SCSP. Therefore, it should be possible to implement WCDG with WFSTs within the SCSP framework. In the following it is tried to realize the basic features of WCDG with the help of WFSTs within the SCSP framework.

5.2.1 Constraint system

In order to implement WCDG a weighted CSP can be modeled as a SCSP over the tropical semiring $(\mathbb{R}^+ \cup \{+\infty\}, min, +, +\infty, 0)$. This SCSP instance implements negative preferences since preference combination by + returns a lower or equal preference (cf. Bistarelli, Montanari, & Rossi (1997)). In WCDG penalties assigned to a constraint have a penalty factor between 0 to 1. However, using the negative logarithm the costs can be transfered into the tropical semiring.

For further definitions, the representational levels are omitted in the beginning. Each word of a sentence is represented by an individual variable where the total order of the variables corresponds to the total order of the words in the sentence. The variable values are represented on string level in a special string encoding including the information about the dependency label and about head position. Properties of a word like the lexical category or the lemma form are added as information to the variables as mentioned in subsection 4.7. In order to illustrate a possible encoding of a variable an instantiation together with its associated information is given by the following example:

$$\text{Mann [NN Case=nom Number=sg Gender=masc] } \mathbf{1} \diamond \text{[SUBJ] } \mathbf{5} \tag{24}$$

The variable information consists in the lemma form *Mann* and its part-of-speech with some morphological features ([NN Case=nom Number=sg Gender=masc]) and it consists in the position of the word in the sentence (*1*).[9] The variable value consists in the label SUBJ and the head position *5* which corresponds to the position of the head in the sentence.

5.2.2 Constraints

The quantified constraints of WCDG which receive a fixed weight can be implemented via predicate logic formula where propositions apply to substrings (Hulden (2008)) and they can be compiled in an off-line preprocessing step. Note that one is not restricted to universal quantification and that one is not restricted to at most two quantified variables in this approach. It is possible to make use of universally and existentially quantified formulas over any number of variables. That is, the approach is not restricted to unary and binary constraints. Given the regular language α which is defined via a predicate logic formula and a penalty ω, then the value of a quantified constraint $(def, \star)$ can be defined by the following S-rational series:[10]

$$L(def) = \alpha \cup \overline{\alpha}\langle\omega\rangle \tag{25}$$

If the penalty is ∞ which corresponds to $-log(0)$ then the constraint is crisp and if the weight is 0 which corresponds to $-log(1)$ then the constraint has no effect.

[9]The features of a complex category have a fixed set of possible values. They are defined with respect to an inheritance hierarchy and they are represented as transition labels. Underspecification is realized as the disjunction of all maximal subtypes of a super type.

[10]In Didakowski (2008b) it is proposed to restrict universally quantified variables optionally. Through this a WFST representing a constraint can be smaller in size hence no complementation operation is used. However, such a strategy presumes positive preferences in the constraints (see section 5.2.4).

The implementation of dynamic constraints is more complicated because complementation is not defined for S-rational series in general (the problem arises if S has more than two elements) and the complementation operation is exhaustively used in the compilation of such predicate logic formulas. In the following a chart is given avoiding this problem. The basic idea is to represent the penalty on string level and to assign it to the value of a constraint variable. An example of a variable with the associated penalty 0.7 is the following:

$$\text{Mann [NN] } \mathbf{1} \diamond \text{[SUBJ] } \mathbf{5}\ \mathbf{0.7} \tag{26}$$

Given the regular language α which is constructed via a predicate logic formula including this penalty and given a S-rational series β mapping a finite grading of penalty into an element in S and given a regular relation γ which removes the penalties on string level from the constraint variable values, then the value of a dynamic constraint $(def, \star)$ can be defined by the following S-rational series:

$$L(def) = Range((\alpha \cap \beta) \circ \gamma) \tag{27}$$

If the penalty of a variable is not constrained, the variable receives the penalty 0 which is the best among all possible weights. Via the assignment of penalties to individual variables, for example shortest and longest distance can be implemented (cf. Didakowski (2008b) in the context of longest match).

The extended local constraints are implemented in another way as mentioned in section 5.1. Complex labels with finite feature structures are used in order to 'transport' syntactic and lexical information instead of using ancillary constraints which check some information in a cascaded or recursive manner. That is, the information is still available if needed. The size of the domain grows rapidly using complex labels. But using an adequate string encoding this does not matter in the finite-state approach, because the domain can still be represented compactly. By this it is for example possible to handle subcategorisation frames and to represent basic properties of a subtree, e.g. to determine whether it has a determiner or a verb particle or a passive marker etc. Note that the finite set of feature structures may correspond to the finite set of ancillary constraints in some way where the individual features take boolean feature values. In this view the feature structures are only a way to store the results of the checks of the ancillary constraints. An example for a variable value with a complex label including information about whether the subtree has a determiner and whether the dominating head is involved in a dependency relation as modifier with the label S is given:

$$\text{Mann [NN] } \mathbf{1} \diamond \text{[SUBJ is\_S=true has\_DET=true] } \mathbf{5} \tag{28}$$

The use of complex labels opens new possibilities in the formulation of constraints.

Several representational levels can be represented by one constraint variable where the separation of the levels is transferred into the domain of the SCSP. In order to realize this it is assumed that the different levels are totally ordered. If n representational levels are used a variable value consists of n pairs consisting of a dependency label and a head position written one after another. The belonging of a pair to a level is geared to the total order of the levels and corresponds to the position at which it is listed. A variable value which includes a level for the syntactic structure l_1 and which includes a level for the thematic structure l_2 can be coded on string level as follows where $l_1 > l_2$:

$$\text{Mann [NN] } \mathbf{1} \diamond \text{[SUBJ] } \mathbf{5}\text{[AGENS] } \mathbf{7} \tag{29}$$

In the example the label SUBJ together with the head position **5** refer to the syntactic structure and the label AGENS together with the head position **7** refer to the thematic structure of a sentence. If a word is not involved in some of the representational levels a special label for 'undefined' may exist.

5.2.3 Explicit dependency structures

A lot of work deals with the representation of dependency structures on string level. Oflazer (2003) uses special symbols to represent "channels" which make it possible to represent projective dependency trees. Yli-Jyrä (2005) on the other hand uses a bracketing scheme in order to represent projective dependency trees. Furthermore, Yli-Jyrä (2004) presents a colored bracketing scheme which allows to represent non-projective dependency trees with some restrictions. In Maruyama (1990) a more explicit representation of dependency trees is proposed as shown above. Some good reasons give motivation for realizing such an approach with finite state techniques. First, the sentence length seems to be very limited in general (see Mohri & Sproat (2006)). Second, checking the agreement of the head position shown by the dependent and the actual position of the head can be indicated via special symbols in an implicit manner. Later on, this agreement can be made explicit. In this connection a constraint consists of two parts, one that checks the elementary constraint and the other that makes the agreement of the position information explicit. Additionally, the problem of checking the agreement can be decomposed in checking individual digits or digit complexes. Furthermore, the direction of checking can be changed by reversing a WFST in order to decrease the amount of hypotheses. This approach is sketched in figure 5.

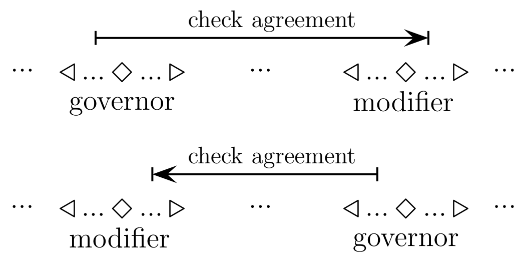


Figure 5: direction of agreement checking of the position information

5.2.4 Semiring instantiations

With a weighted CSP as mentioned in subsection 5.2.1 it is unnatural to formulate that one grammatically correct structure is more preferable than another. Here the problem of natural language parsing may have a bipolar character including negative and positive preferences where the first models for example degrees of grammaticality and the latter models structural preferences. In a bipolar problem an indifference element expresses neither positive nor negative preferences and is the best among the negative preferences and the worst among the positive preferences. The *max-semiring* $(\mathbb{R} \cup \{-\infty, +\infty\}, max, +, -\infty, 0)$ can be used to implement such bipolar problems where 0 is the indifferent element. A general framework for bipolar structures is given in Bistarelli, Pini, Rossi, &

Venable (2005) which allows to have a richer structure for negative preferences with respect to the positive ones or vice versa.

In Didakowski (2008b) an approach is presented where several linguistic criteria which are represented as a semiring can be ranked by preference. This is facilitated by means of a semiring composition which defines a lexical order via the semiring operation $\oplus$. There, the ranking is used in order to achieve a correct implementation of *longest match* based on semiring-weights, and in order to outrank a longest match strategy by other preferences.

After parsing it is sometimes requested to know the conflicting constraints which caused penalties. The information about conflicting constraints can be handled by the semiring $(2^{P(M)}, \cup, \bigcup, \varnothing, \{\varnothing\})$ where M is a set of constraint identifiers, $\cup$ is the classical set union and $\bigcup$ is the operation of pairwise union of sets and is defined by $\bigcup(A, B) = \{f | a \in A \wedge b \in B \wedge f = a \cup b\}$. It is easy to see that $\varnothing$ is the identity of $\cup$, that $\{\varnothing\}$ is the identity of $\bigcup$, that $\varnothing$ is the annihilator of $\bigcup$, that $\cup$ and $\bigcup$ are commutative and that $\bigcup$ distributes over $\cup$. Assuming that the constraint identifiers are inserted by the corresponding constraints and that the semiring is conveniently combined with the semiring which handles the preferences, the constraint identifiers can be collected after problem solving.

5.2.5 Solution

If a sentence is lexically or morphologically ambiguous the several readings can be handled by different sets of variables. But instead of solving a SCSP for each reading independently (this would be combinatorial explosive) the several SCSPs for each reading can compactly be represented in one representation and solved simultaneous as presented in subsection 4.8.[11] If a sentence has n lexical or morphological readings there exist n constraint systems $(S, D, V_1), (S, D, V_2), \dots, (S, D, V_n)$ for the different variable sets. Assumed that all variables of the individual constraint systems are of interest actually n concrete SCSPs can be solved simultaneously as follows. First, all quantified constraints are combined: $(def, \star) = (\bigotimes_{(L(def'), \star) \in C^\star}(L(def'), \star))$. A corresponding (simultaneous) type is assigned to the resulting constraint. Then the constraint is simultaneously projected over the variables of the several constraint systems:

$$(L(def), L(V_1)/L(V_2)/\dots/L(V_n)) \Downarrow_{V_1/V_2/\dots/V_n} \tag{30}$$

There is an essential difference to the FSIG interpretation as a CSP in Yli-Jyrä (2001) where lexical and morphological properties are treated as variables such that the disambiguation of these properties together with the detection of the dependency relations are covered in one CSP. It would also be possible to represent the different readings of a word by different variables within one SCSP. But such an approach complicates the formulation of constraints because one has to keep track of the information about which readings belong to the same word. Furthermore, the number of readings would increase the number of variables of a constraint system and problem.

[11] This approach is similar to the work of Helzerman & Harper (1996) where the MUSE CSP is used to represent the CSPs for each lexical reading of an ambiguous sentence compactly. Furthermore, they use the MUSE CSP in order to represent multiple sentence hypotheses of a speech recognizer. Such an approach is of interest for the finite state approach, too.

Computing a solution with WFSTs via the method mentioned above can be impracticable. An example for this are FSIGs where the parsing can lead to finite state machines that are not viable any more (cf. Tapanainen (1997)). This is due to the enormous internal results which are caused by the intersection operation. Provided that the number of quantified constraints in the grammar is fixed one could believe that all quantified constraints can be combined to one constraint represented by one monolithic WFST. But in practice it is impossible to combine all constraints of an adequate grammar. Instead of combining all quantified constraints before projecting it is possible to combine portions of the constraints and to projected them individually over the corresponding variables. Afterwards the resulting constraints can be combined. On the side of FSIG a lot of work deals with techniques in order to restrain the size of internal results (see Koskenniemi, Tapanainen, & Voutilainen (1992)).

However, the uncoupled agreement checking of the position information complicates the combination of constraints in an off-line preprocessing step (see section 5.2.3). Furthermore, a constraint does not cover unbound crossing and embedding dependencies. In section 5.3 tree composition is proposed in order to overcome these problems.

5.3 Tree decomposition

To overcome the problems mentioned in section 5.2.5 *tree decomposition* can be implemented. The notion of tree decomposition originates from graph theory. In constraint processing it refers to the decomposition of a problem in subproblems (clusters) organized in an acyclic graph where the values of variables that relate adjacent subproblems (variables whose removal disconnect the subproblems) are obtained by combining the solutions of the subproblems. The main idea behind this is to solve the whole problem tree-like and to detect inconsistency at a local level where local inconsistency implies global inconsistency (cf. Bistarelli, Montanari, & Rossi (1997)). In our approach tree decomposition is restricted to one representational level. Different representational levels may correspond to different tree decompositions.

5.3.1 The general subproblem

In order to realize tree decomposition the general subproblem has to be specified. In our approach the general subproblem consists in determining partial dependency trees. One word in a dependency tree is independent representing the root node. The same holds for partial dependency trees. The root node of a partial dependency tree is called the *ceiling* of a partial dependency tree. Via the notion of ceiling a partial dependency tree can be defined:

> A partial dependency tree is the subgraph of a ceiling c
>
> i. which includes the word defining c
>
> ii. which does not contain any other ceiling of a partial dependency tree, and
>
> iii. which includes the direct dependents of c
>
> iv. where the words dominated by c have to be connected (these words form together with the ceiling a continuous substring of the sentence)

The ceiling of a partial dependency structure can be the dependent in another dependency relation. Through this a partial dependency tree can be incorporated within another partial dependency tree via the ceiling building a more complex projective dependency structure. More precisely, the subproblem consists in determining which direct dependents a word with special properties can take, it consists in specifying in which sequence the involved words should occur and it consists in specifying which subsets of the involved words are convex.[12] A word which takes no dependents forms the simplest subproblem (a partial dependency tree including only the ceiling). That is, conditions are formulated over partial dependency trees of depth one unless a word takes no dependents. This approach has some similarities to the work of Duchier (1999) where *role constraints* are used to express grammatical conditions between head and dependent and where *word-order constraints* are used to constrain sequentiality and convexity. Figure 6 sketches this formulation of the general subproblem. The

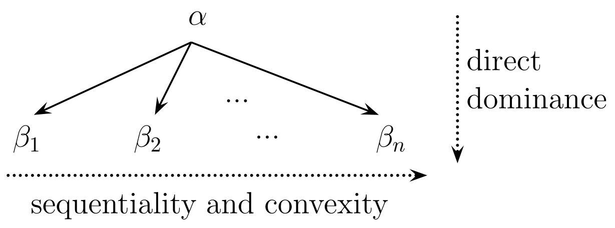


Figure 6: determining partial dependency trees

approach enables compact pre-compilation because you only need to combine constraints which are formulated over a smaller problem.

5.3.2 Local consistency rules

In order to choose and solve subproblems in a tree-like fashion and to detect local inconsistency the notion of *local consistency rule* is used. Via local consistency rules it is possible to realize a tree-like solving of a problem in a bottom-up way (cf. Bistarelli, Montanari, & Rossi (1997)). It is assumed that quantified constraints are formulated for the general subproblem as mentioned above and that all of them are combined: $(\bigotimes_{(L(def),\star)\in C^\star}(L(def),\star))$. In the following the ceiling of a subproblem refers to the variable representing the ceiling of the corresponding partial dependency trees.

Only the ceiling of a subproblem is of interest for a connected subproblem which is chosen afterwards; the ceiling relates both subproblems. Here a ceiling can be a dependent within a related subproblem. Therefore, the tuples of the constraint value $L(def)$ are projected over the ceiling. The incorporation of this tuple projection is realized by a rational transduction $L(def)'$ defining a function:

$$L(def)' : (\lhd_d L(D) \rhd_d)^* \lhd L(D) \rhd (\lhd_d L(D) \rhd_d)^* \times \lhd_d L(D) \rhd_d \to S \quad (31)$$

In this definition the special variable markers $\lhd_d$ and $\rhd_d$ are used in order to indicate variables which represent dependents. This information is used for choosing individual subproblems.

[12]A totally ordered set S is said to be convex with respect to a total order, if for any $x \notin S$, x either precedes all elements of S or follows all elements of S (cf. Duchier (1999)).

For further definitions some ancillary regular relations are defined where the special symbol $\top \notin \Sigma$ is used. The following regular relation is used to mark variables representing dependents by the special symbol $\top$ optionally:

$$mark = (\lhd_d \Delta^+ \rhd_d ((\epsilon \times \top) \cup \epsilon))^+ \tag{32}$$

The following regular relation is used to state that a subproblem is connected to a subproblem that was chosen and solved in the preceding step by checking the existence of the special symbol $\top$ (where the symbol $\top$ is eliminated):

$$contain = \begin{pmatrix} \lhd \Delta^+ \rhd \\ \cup \\ \lhd_d \Delta^+ \rhd_d \end{pmatrix}^* \lhd_d \Delta^+ \rhd_d (\top \times \epsilon) \begin{pmatrix} \lhd \Delta^+ \rhd \\ \cup \\ \lhd_d \Delta^+ \rhd_d \end{pmatrix}^* \tag{33}$$

With help of these regular relations two rational transductions representing two local consistency rules can be defined which choose and solve subproblems simultaneously by assuming them at each position optionally.

The local consistency rule for subproblems with a type *con* with $|con| = 1$ is defined as follows:

$$\begin{gathered} rule_1 = \\ ((\lhd \Delta^+ \rhd)^* ((\lhd \Delta^+ \rhd) \circ L(def)' \circ mark))^* (\lhd \Delta^+ \rhd)^* \end{gathered} \tag{34}$$

In this definition the subexpression $(\lhd \Delta^+ \rhd) \circ L(def)' \circ mark$ implements the statement $|con| = 1$ and it implements the marking of the subproblems as 'currently solved' by the marker $\top$. The subexpression $((\lhd \Delta^+ \rhd)^* \dots)^* (\lhd \Delta^+ \rhd)^*$ implements that a subproblem is assumed to be at each position optionally. Variables involved in a subproblems of size one can act as dependent in a connected subproblem of size greater than one. Variables which are not involved in a subproblem of size one can act as ceiling in a subproblem of size greater than one.

A local consistency rule for subproblems with a type *con* with $|con| > 1$ is defined as follows:

$$\begin{gathered} rule_2 = \\ (\begin{pmatrix} \lhd \Delta^+ \rhd \\ \cup \\ \lhd_d \Delta^+ \rhd_d \end{pmatrix}^* (contain \circ L(def)' \circ mark))^* \begin{pmatrix} \lhd \Delta^+ \rhd \\ \cup \\ \lhd_d \Delta^+ \rhd_d \end{pmatrix}^* \end{gathered} \tag{35}$$

In this definition the subexpression $contain \circ L(def)' \circ mark$ implements the statement $|con| > 1$ and it implements that a subproblem is connected to a subproblem that was chosen and solved in the preceding step. Note that the special marker $\top$ is not in Σ. By this, the marker occurs exactly at those positions where a subproblem was chosen and solved in the preceding step. The subexpression $((\lhd \Delta^+ \rhd \cup \lhd_d \Delta^+ \rhd_d)^* \dots)^* (\lhd \Delta^+ \rhd \cup \lhd_d \Delta^+ \rhd_d)^*$ implements that the subproblem is assumed to be at each position optionally. If the rule solves a subproblem it 'eliminates' at least one variable from the search space (these 'eliminated' variables do not occur in the connected subproblems chosen afterwards).

5.3.3 Solution

At each step of a bottom-up walk subproblems are chosen and solved simultaneously (cf. subsection 4.8) and by this means the problem is decomposed into its regular parts. In this connection the solving of subproblems and the combining of the connected subproblems is a simultaneous process.

Given the following constraint systems $(S, D, V_1), (S, D, V_2), \ldots, (S, D, V_n)$ for the several readings of an ambiguous sentence where all variables are of interest, and given the regular language $L(V_1/V_2/\ldots/V_n) = L(V_1) \cup L(V_2) \cup \cdots \cup L(V_n)$ representing the several variable sets compactly, then the solving of a constraint problem over these constraint systems via a bottom-up walk is defined via a sequence of local consistency rules as mentioned above if i denotes the maximum iteration and $i > 1$:

$$\begin{aligned} problem^i &= problem^{i-1} \circ rule_2 \\ problem^1 &= L(V_1/V_2/\ldots/V_n) \circ rule_1 \end{aligned} \tag{36}$$

Here the depth of the decomposition tree is parametric in respect of the iteration of the local consistency rule $rule_2$. The solving of a problem is successful, if the decomposition is single rooted and if it forms a tree. This statement is implemented via the regular language:

$$root = \triangleleft_d \Delta^+ \triangleright_d \tag{37}$$

If one is interested in determining the values for a subset of variables of the constraint problem that satisfy a subset of the constraints, the decomposition can be multi-rooted forming a forest. This implements *partial constraint satisfaction* (cf. Bistarelli, Freuder, & O'Sullivan (2004)). This is realized by punishing the roots of the decomposition structure in order to get an exhaustive solution. The following S-rational series is used to give a penalty ω to the corresponding variables:

$$root = (\triangleleft_d \Delta^+ \triangleright_d \langle \omega \rangle)^+ \tag{38}$$

then the value of the solution of a problem can be defined by the following S-rational series:

$$Dom(problem^i \circ root) \tag{39}$$

The domain is taken because it contains all the chosen, solved and combined subproblems. If several representational levels are used (surface syntactic, thematic, etc.), a tree decomposition has to be performed for each representational level separately. The solution of the first level may be calculated in a bottom-up fashion as mentioned above. Based on the result the solution of the second level may be calculated in a bottom-up fashion, too, and so on. The result is the combination of the solutions of the several representational levels. If some variables are not of interest for a specific representational level these variables can be 'eliminated' via projection before calculating the corresponding solution.

In the following an algorithm is presented which performs the problem solving in a bottom-up way as mentioned above and which detects if the problem is stable and which extracts the best and the most exhaustive solutions of the optimization problem. To simplify matters only one representational level is assumed:[13]

```
**Input**: $A(root), A(V_1/V_2/\dots/V_n), A(rule_1), A(rule_2)$
$analysis_1 = Compose(A(V_1/V_2/\dots/V_n), A(rule_1))$
while($|analysis_1| \neq |analysis_2 = Compose(analysis_1, A(rule_2))|$)        (40)
    $analysis_1 = analysis_2$
**Output**: $Bestpath(Project1(Compose(analysis_1, A(punish))))$
```

First $A(V)$ is composed with $A(rule_1)$ (by the function *Compose*) in order to solve subproblems over one variable. Then the result is iteratively composed with $A(rule_2)$ performing a bottom-up walk by solving subproblems tree-like until the problem is stable. Note that via the use of the marker $\top$ it is assured that each subproblem is chosen and solved only once. The point at which the problem is stable can be detected by the break condition ($|analysis_1| \neq |analysis_2 = compose(analysis_1, A(rule_2))|$). This condition checks whether new subproblems were solved in a previous step by keeping track of the size of the resulting WFSTs (this could also be done by checking of existence of the symbol $\top$). Afterwards either a single rooted decomposition is demanded by $A(punish)$ or the roots of a decomposition forest are punished strongly by a bad weight (by $A(punish)$). Then, the first projection is taken (by the function *Project*1) in order to extract the solution. Finally, the best solution is calculated via a best path search (by the function *Bestpath*). The output of the algorithm is the best and most exhaustive solution of the optimization problem.

5.3.4 Parsing structure

After parsing the decomposition structure itself displays the desired parsing structure. Thus, using the tree decomposition method the handling of the parsing structure (via position information) in the constraints becomes obsolete. By this, the subproblems can be additionally simplified. In order to realize this the decomposition structure has to be indicated via brackets or channels etc. on string level. For this purpose several techniques can be used in order to represent projective parsing structures (see section 5.2). In the following two extensions concerning tail recursion and non-projectivity are presented where further details of the implementations are omitted but a general chart is given in order to realize this extensions.

After parsing the depth of the resulting dependency tree and forest respectively is bound by the depth of the decomposition tree and forest respectively. In the following it is shown that this is not necessarily the case. The idea behind this is that a subproblem can remain regular even if the depth of the covered partial dependency trees is unbound. This is possible in the case of left or right recursion of connected subproblems. This tail recursion can be replaced by iteration (cf. Koskenniemi (1990)). An example for a dependency tree including right embedding which can be covered by iteration is shown in figure 7. In order to replace the tail recursion by iteration the corresponding subproblems have to be merged in an off-line preprocessing step. In this connection one just has to keep track of the decomposition structure of the merged subproblems. The

[13]Here $|T|$ denotes the size of a WFST T in respect to the number of states and transitions.

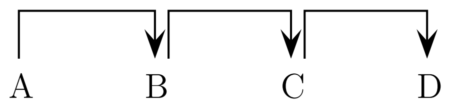


Figure 7: dependency tree with right embedding

result of the merging can be used to choose and solve subproblems including this iteration via local consistency rules. But in this connection the rules have to keep track of the tail recursion. For the subproblems for which tail recursion is replaced by iteration the recursion has to be disabled. This can be realized via the use of additional special marker symbols which provide the necessary information.

In the following it is shown that an extension to non-projective dependency structures is possible. The idea behind this is to compile possible crossings in an off-line preprocessing step in order to avoid an expensive on-line calculation. An example for a non-projective dependency forest is given in figure 8. The dependency forest is 2-planar, that is the forest can be separated into two planes (into two planar graphs) (cf. Yli-Jyrä (2003)).[14] This partial non-projective forest can be covered by one subproblem which emerges from the merging of several interwoven subproblems. In order to enable this interleaving the definition of a partial dependency tree (see section 5.3.1) is softened by allowing that the words dominated by a ceiling are unconnected. According to that the general subproblem additionally consists in determining which direct dependents of another ceiling can occur between the direct dependents of the ceiling of the actual partial dependency tree (including sequentiality and convexity). This extension of the general subproblem is sketched in figure 9. The result of the merging can be used by local consistency rules in order to choose and solve subproblems

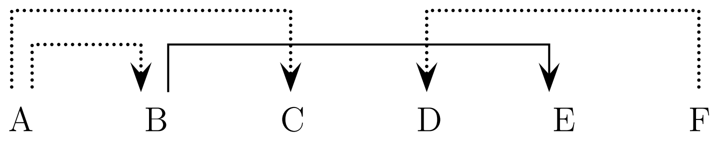


Figure 8: non-projective dependency forest

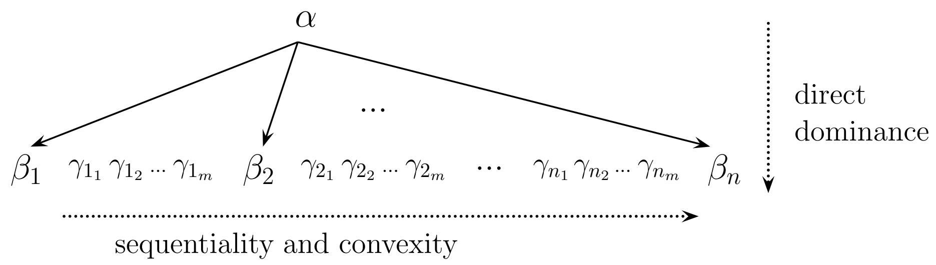

α
...
direct dominance
β1 γ1₁ γ1₂ ··· γ1ₘ β2 γ2₁ γ2₂ ··· γ2ₘ ··· γn₁ γn₂ ··· γnₘ βn
sequentiality and convexity

Figure 9: determining partial dependency trees

[14] "Planarity of a dependency tree is the requirement that the links do not cross when drawn above the sentence" (Yli-Jyrä (2003)).

including non-projectivity. In this connection a chosen and solved subproblem can cover more than one ceiling and it can be connected to more than one subproblem chosen afterwards. However, the algorithm presented in section 5.3.3 still works fine with this extension. Here the multiplanarity is parametric with respect to the number of planes but also other criteria could be used in order to parameterise the crossings. In order to keep track of the different planes (the sub-subproblems) which are involved in the subproblem a colored bracketing scheme as presented in Yli-Jyrä (2003) can be used where the number of used colors corresponds to the number of planes.

6 Conclusion

An approach implementing dependency parsing as a *Constraint Satisfaction Problem (CSP)* by means of *Weighted Finite State Transducers (WFST)* was presented. The approach is based on the *Semiring-based Constraint Satisfaction Problem (SCSP)* which enables an implementation of the basic features of the *Weighted Constraint Dependency Grammar (WCDG).*

After it was shown how to realize SCSPs with WFSTs in a natural way it was presented how the several features of WCDG can be mapped into a finite state approach where some of the features are handled in a new way. This includes the representation of the general parsing problem, the implementation of constraints with fixed penalty, dynamic constraints and extended local constraints and this includes the handling of lexically or morphologically ambiguous sentences and the handling of several representational levels. Furthermore, it was shown that via the notion of locality, decomposition and simultaneousness an efficient bottom-up solving of the parsing problem via tree decomposition is possible. This is realized via an extended finite-state approach which decomposes the parsing problem in its regular parts and which allows to process non-projective dependency structures.

The theoretical approach opens new possibilities in constraint-based dependency parsing and also in constraint processing in general. This article shows that finite state techniques have a potential which is still not utilized in theoretical approaches as well as in practical applications of natural language processing.

References

Aarts, B. (2007). *Syntactic gradience: the nature of grammatical indeterminacy.* Oxford University Press.

Amilhastre, J., H. Fargier, & P. Marquis (2002). Consistency restoration and explanations in dynamic CSPs – application to configuration. *Artificial Intelligence* 135. 199–234.

Bistarelli, S., E. C. Freuder, & B. O'Sullivan (2004). Encoding Partial Constraint Satisfaction in the Semiring-Based Framework for Soft Constraints. In: *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI).* 240–245.

Bistarelli, S., U. Montanari, & F. Rossi (1997). Semiring-based constraint satisfaction and optimization. *Journal of the ACM* 44. 201–236.

Bistarelli, S., U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, & H. Fargier (1999). Semiring-based CSPs and valued CSPs: Frameworks, properties and comparison. *Constraints* 4. 199–240.

Bistarelli, S., M. S. Pini, F. Rossi, & K. B. Venable (2005). Positive and Negative Preferences. In: *Workshop on preferences and soft constraints (Soft 2005).*

Didakowski, J. (2008a). Local Syntactic Tagging of Large Corpora using Weighted Finite State Transducers. In: *Text Resources and Lexical Knowledge – Selected Papers from the 9th Conference on Natural Language Processing (KONVENS 2008).* 65–78.

Didakowski, J. (2008b). SynCoP - Combining Syntactic Tagging with Chunking Using Weighted Finite State Transducers. In: *Proceedings of Finite-State Methods and Natural Language Processing (FSMNLP 2007).* 107–118.

Duchier, D. (1999). Axiomatizing Dependency Parsing Using Set Constraints. In: *Sixth Meeting on Mathematics of Language.* 115–126.

Foth, K. A., W. Menzel, & I. Schröder (2000). A Transformation-based Parsing Technique with Anytime Properties. In: *4th Int. Workshop on Parsing Technologies, IWPT-2000.* 89–100.

Harbusch, K. (1997). The Relation Between Tree-Adjoining Grammars and Constraint Dependency Grammars. In: *In Proceedings of MOL5 - Fifth Meeting on the Mathematics of Language.* 38–45.

Heinecke, J., J. Kunze, W. Menzel, & I. Schröder (1998). Eliminative parsing with graded constraints. In: *Proceedings of the 17th International Conference on Computational Linguistics, 36th Annual Meeting of the ACL, Coling-ACL '98.* 526–530.

Helzerman, R. A. & M. P. Harper (1996). MUSE CSP: An extension to the constraint satisfaction problem. *Journal of Artificial Intelligence Research* 5. 5–239.

Hulden, M. (2008). Regular expressions and predicate logic in finite-state language processing. In: *Proceedings of Finite-State Methods and Natural Language Processing (FSMNLP 2008).* 67–77.

Karlsson, F. (1990). Constraint grammar as a framework for parsing running text. In: *Proceedings of COLING-90.* (vol. 3). 168–173.

Karlsson, F. (2010). Syntactic recursion and iteration. In: H. van der Hulst (ed.) *Recursion and Human Language.* Berlin/New York: Mouton de Gruyter. 43–67.

Koskenniemi, K. (1990). Finite-state parsing and disambiguation. In: *Proceedings of COLING-90.* (vol. 2). 229–232.

Koskenniemi, K., P. Tapanainen, & A. Voutilainen (1992). Compiling and using finite-state syntactic rules. In: *Proceedings of the 14th conference on Computational linguistics.* Morristown, NJ, USA: Association for Computational Linguistics. 156–162.

Kuich, W. & A. Salomaa (1985). *Semirings, Automata and Languages.* Springer-Verlag New York, Inc.

Maruyama, H. (1990). Structural disambiguation with constraint propagation. In: *Proceedings of the 28th annual meeting on Association for Computational Linguistics.* 31–38.

McCrae, P., K. Foth, & W. Menzel (2008). Modelling Global Phenomena with Extended Local Constraints. In: *In Proceedings of the 5th International Workshop on Constraints and Language Processing (CSLP 2008).* 48–60.

Mohri, M. (2002). Semiring Frameworks and Algorithms for Shortest-Distance Problems. *Languages and Combinatorics* 7. 321–350.

Mohri, M. & R. Sproat (2006). On a Common Fallacy in Computational Linguistics. *A Man of Measure: Festschrift in Honour of Fred Karlsson on his 60th Birthday* 19. 432–439.

Oflazer, K. (2003). Dependency Parsing with an Extended Finite-State Approach. *Computational Linguistics* 29. 515–544.

Schröder, I. (2002). *Natural Language Parsing with Graded Constraints.* Ph.D. thesis. University of Hamburg, Department of Computer Science.

Tapanainen, P. (1997). Applying a finite-state intersection grammar. In: *Finite-State language processing.* MIT Press. 311–327.

Tarapata, Z. (2007). Selected multicriteria shortest path problems: An analysis of complexity, models and adaptation of standard algorithms. *International Journal of Applied Mathematics and Computer Science* 17. 269–287.

Vempaty, N. R. (1992). Solving Constraint Problems Using Finite State Automata. In: *Proceedings of the 10th National Conference on AI.* 453–458.

Yli-Jyrä, A. (2001). Structural correspondence between finite-state intersection grammar and constraint satisfaction problem. In: *Finite-State Methods in Natural Language Processing (FSMNLP 2001), ESSLLI Workshop.* 1–4.

Yli-Jyrä, A. (2003). Multiplanarity - a model for dependency structures in treebanks. In: *Second Workshop on Treebanks and Linguistic Theories.* University Press. 189–200.

Yli-Jyrä, A. (2004). Axiomatization of restricted non-projective dependency trees through finite-state constraints that analyse crossing bracketings. In: *Proceedings of the Workshop of Recent Advances in Dependency Grammar, COLING'04 Workshop.* 33–40.

Yli-Jyrä, A. (2005). Approximating Dependency Grammars Through Intersection of Regular Languages. In: M. Domaratzki, A. Okhotin, K. Salomaa, & S. Yu (eds.) *Implementation and Application of Automata, 9th International Conference, CIAA 2004, Kingston, Canada, July 22-24, 2004, Revised Selected Papers.* (Lecture Notes in Computer Science, vol. 3317). Springer. 281–292.

Yli-Jyrä, A. & K. Koskenniemi (2004). Compiling contextual restrictions on strings into finite-state automata. In: *Proceedings of the Eindhoven FASTAR Days 2004*.