Compiling HPSG constraint grammars into logic programs

Thilo Götz

Abstract

We show how to *automatically* translate HPSG grammars rendered as theories of a feature logic into constraint logic programs. The number of choice points in the resulting programs is reduced to a minimum. We prove soundness and a restricted version of completeness for this translation, using results of (Höhfeld and Smolka 1988).

This general compilation enables us to compute with HPSG grammars without having to hand-code each individual grammar as a logic program. There already exist appropriate CLP languages, e.g. CUF (Dörre and Dorna 1993), and the resulting programs can be further optimised using known logic programming techniques.

1 Introduction

HPSG (Pollard and Sag 1994) grammars are expressed as sets of implications in an attribute-value language. These implications are interpreted as universal constraints that apply everywhere. The more or less informal notions of HPSG were given a logical foundation in, among others, (King 1994). Yet although the logical notions are clear enough, there is no operational semantics to go with the logic. Thus, in NLP practice, HPSG grammars are implemented as phrase structure grammars or logic programs. HPSG principles are then encoded as relations (e.g., ID-schemata) or added as feature logic constraints to relations wherever appropriate (e.g., the Head-Feature Principle). (Meurers 1994) pointed out that a more principled connection between HPSG grammars and their encoding in a relational language can and should be established. The purpose of this paper is to show that this sort of encoding can in fact be done automatically. In other words, we provide an operational semantics for HPSG grammars.

To do this, we define a feature logic that serves a double purpose: it is used to encode HPSG grammars, and as a constraint language for the CLP scheme of (Höhfeld and

Acknowledgments: the research reported here was carried out in the context of SFB 340, project B4, funded by the Deutsche Forschungsgemeinschaft. I would like to thank Dale Gerdemann, Paul John King, Detmar Meurers, Bill Rounds and three anonymous referees for helpful comments on an earlier version of this paper. Author's address: SFB 340, University of Tübingen, Germany. Email: tg@sfs.nphil.uni-tuebingen.de

Smolka 1988). Since the definite clause programs that result from the translation use the same constraint language as the HPSG grammars, it is not very hard to prove properties of the translation.

Since the focus of this paper is more on logical issues, we will not address any efficiency questions. It should be mentioned, though, that two optimised versions of the translation described in this paper have already been implemented. One version compiles HPSG grammars directly into PROLOG programs, encoding typed feature structures as PROLOG terms using ideas presented in (Mellish 1992). A compiler into CUF is in preparation.

2 A Feature Constraint Language: $\mathcal{FL}$

We describe a typed feature logic that can be used as a constraint language in the sense of (Höhfeld and Smolka 1988). The logic is based on those of (King 1994), (Smolka 1992) and (Dörre 1994).

2.1 **Definition**

A signature is a quintuple $\langle \mathsf{VAR}, \mathcal{T}, \preceq, \mathcal{F}, \mathit{approp} \rangle$ *s.t.*

- VAR *is a countably infinite set of variables*
- $\mathcal{T}$ *is a finite set of types, and* $\langle \mathcal{T}, \preceq \rangle$ *is a join semi-lattice*
- $\mathcal{V} = \{t \in \mathcal{T} \mid \text{if } t' \preceq t \text{ then } t' = t\}$ *is the set of minimal types*
- $\mathcal{F}$ *is a finite set of feature names*
- $\mathit{approp} : \mathcal{V} \times \mathcal{F} \rightharpoonup \mathcal{T}$ *is a partial function from pairs of minimal types and features to types*

2.2 **Definition (feature terms)**

- X *is a term* — *if* $X \in \mathsf{VAR}$
- t *is a term* — *if* $t \in \mathcal{T}$
- $f : \phi$ *is a term* — *if* $f \in \mathcal{F}$ *and* ϕ *is a term.*
- $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\neg\phi_1$ *are terms* — *if* ϕ_1 *and* ϕ_2 *are terms.*[1]

Write Term for the set of terms.

2.3 **Definition (formulae)**

- $X = \phi$ *is a formula if* X *is a variable and* ϕ *is a term.*
- $F_1 \& F_2$ *is a formula if* F_1 *and* F_2 *are formulae.*

2.4 **Definition (interpretation)**

An interpretation is a quadruple $\mathcal{I} = \langle \mathcal{U}, \mathcal{S}, \mathcal{A}, [\![\cdot]\!]^{\mathcal{I}} \rangle$ *s.t.*

[1] Implication is defined as usual: $\psi \Rightarrow \phi \equiv \neg\psi \vee \phi$

- *$\mathcal{U}$ is a set of objects, the domain of $\mathcal{I}$*
- $\mathsf{ASS} = \mathcal{U}^{\mathsf{VAR}}$ *is the set of variable assignments in $\mathcal{I}$*
- *$\mathcal{S} : \mathcal{U} \to \mathcal{V}$ is a total function from the set of objects to the set of minimal types*
- *$\mathcal{A} : \mathcal{F} \to \mathcal{U}^{\mathcal{U}}$ is an attribute interpretation function s.t.*
  - *for each $u \in \mathcal{U}$, for each $f \in \mathcal{F}$, if approp$(\mathcal{S}(u), f)$ is defined and approp$(\mathcal{S}(u), f) = t$, then $(\mathcal{A}(f))(u)$ is defined and $\mathcal{S}((\mathcal{A}(f))(u)) \preceq t$*
  - *for each $u \in \mathcal{U}$, for each $f \in \mathcal{F}$, if $(\mathcal{A}(f))(u)$ is defined, then approp$(\mathcal{S}(u), f)$ is defined and $\mathcal{S}((\mathcal{A}(f))(u)) \preceq$ approp$(\mathcal{S}(u), f)$*
- *Term interpretation is defined with respect to a variable assignment α:*
  - $[\![X]\!]^{\mathcal{I}}_{\alpha} = \{\alpha(X)\}$ *if $X \in \mathsf{VAR}$*
  - $[\![t]\!]^{\mathcal{I}}_{\alpha} = \{u \in \mathcal{U} \mid \mathcal{S}(u) \preceq t\}$ *if $t \in \mathcal{T}$*
  - $[\![f : \phi]\!]^{\mathcal{I}}_{\alpha} = \{u \in \mathcal{U} \mid (\mathcal{A}(f))(u) \in [\![\phi]\!]^{\mathcal{I}}_{\alpha}\}$ *if $f \in \mathcal{F}$ and ϕ is a term*
  - $[\![\phi_1 \wedge \phi_2]\!]^{\mathcal{I}}_{\alpha} = [\![\phi_1]\!]^{\mathcal{I}}_{\alpha} \cap [\![\phi_2]\!]^{\mathcal{I}}_{\alpha}$, $[\![\phi_1 \vee \phi_2]\!]^{\mathcal{I}}_{\alpha} = [\![\phi_1]\!]^{\mathcal{I}}_{\alpha} \cup [\![\phi_2]\!]^{\mathcal{I}}_{\alpha}$, $[\![\neg\phi]\!]^{\mathcal{I}}_{\alpha} = \mathcal{U} \backslash [\![\phi]\!]^{\mathcal{I}}_{\alpha}$
- *Formulae are assigned sets of variable assignments as denotations:*
  - $[\![X = \phi]\!]^{\mathcal{I}} = \{\alpha \in \mathsf{ASS} \mid \alpha(X) \in [\![\phi]\!]^{\mathcal{I}}_{\alpha}\}$ *if $X \in \mathsf{VAR}$ and ϕ is a term*
  - $[\![F_1 \& F_2]\!]^{\mathcal{I}} = [\![F_1]\!]^{\mathcal{I}} \cap [\![F_2]\!]^{\mathcal{I}}$ *if F_1 and F_2 are formulae*

There are several points to note about this logic

1. every object is of exactly one minimal type
2. if an object is of some minimal type, then all and only the features appropriate for the type are defined on the object
3. feature terms denote sets of objects, formulae denote sets of variable assignments

2.1 Adding relation symbols: $\mathcal{R}(\mathcal{FL})$

Our feature logic fulfills all the requirements of (Höhfeld and Smolka 1988) for a constraint language. We can therefore simply apply their schema to extend our language with relation symbols and obtain a sound and complete operational semantics for definite clause programs.

2.5 **Definition** ($\mathcal{R}(\mathcal{FL})$)

1. *$\mathcal{R}$ is a decidable set of relation symbols*
2. *the variables of $\mathcal{R}(\mathcal{FL})$ are the variables of $\mathcal{FL}$*

3. *the constraints of $\mathcal{R}(\mathcal{FL})$ are defined inductively as follows*

   (a) *every $\mathcal{FL}$ constraint is an $\mathcal{R}(\mathcal{FL})$ constraint*

   (b) *if $r \in \mathcal{R}$ is a relation symbol with arity n and $\vec{x}$ is an n-tuple of pairwise distinct variables, then $r(\vec{x})$ is an $\mathcal{R}(\mathcal{FL})$ constraint called an atom*

   (c) *if F and G are $\mathcal{R}(\mathcal{FL})$ constraints, then so are*

      - $\emptyset$ *(empty conjunction)*
      - $F \;\&\; G$ *(conjunction)*
      - $F \rightarrow G$ *(implication)*

4. *a definite clause is an $\mathcal{R}(\mathcal{FL})$ constraint of the form $A \leftarrow \phi \;\&\; B_1 \;\&\; \ldots \;\&\; B_n$, where $A, B_1, \ldots, B_n$ are atoms and ϕ is an $\mathcal{R}(\mathcal{FL})$ constraint*

5. *an interpretation $\mathcal{A}$ of $\mathcal{R}(\mathcal{FL})$ is obtained from an $\mathcal{FL}$-interpretation $\mathcal{I}$ with domain $\mathcal{U}$ by choosing for every $r \in \mathcal{R}$ a relation $r^{\mathcal{A}}$ on $\mathcal{U}$ with the same arity, and by defining:*

   (a) *the domain of $\mathcal{A}$ is $\mathcal{U}$*

   (b) $[\![\phi]\!]^{\mathcal{A}} = [\![\phi]\!]^{\mathcal{I}}$ *if ϕ is an $\mathcal{FL}$ formula*

   (c) $[\![r(\vec{x})]\!]^{\mathcal{A}} = \{\alpha \in \mathsf{ASS} \mid \alpha(\vec{x}) \in r^{\mathcal{A}}\}$

   (d) $[\![\emptyset]\!]^{\mathcal{A}} = \mathsf{ASS}, \qquad [\![F \;\&\; G]\!]^{\mathcal{A}} = [\![F]\!]^{\mathcal{A}} \cap [\![G]\!]^{\mathcal{A}}$

   (e) $[\![F \rightarrow G]\!]^{\mathcal{A}} = (\mathsf{ASS} - [\![F]\!]^{\mathcal{A}}) \cup [\![G]\!]^{\mathcal{A}}$

A goal in this setup is a conjunction of atoms and $\mathcal{FL}$ constraints. If $\mathcal{P}$ is a set of definite clauses and G is a goal, then a $\mathcal{FL}$ constraint ϕ is a $\mathcal{P}$-answer of G just in case ϕ is satisfiable and $\phi \rightarrow G$ is valid in every model of $\mathcal{P}$. (Höhfeld and Smolka 1988) prove two propositions showing that the minimal model properties of conventional logic programs extend to their schema for arbitrary constraint languages. We will make use of the $\mathcal{FL}$-instances of these propositions in our soundness proof later on.

2.6 **Proposition (Höhfeld and Smolka 1988, theorem 4.4, p. 12)**

Let $\mathcal{P}$ be a set of definite clauses in $\mathcal{R}(\mathcal{FL})$ and let $\mathcal{I}$ be an $\mathcal{FL}$ interpretation. Then the equations

$$r^{\mathcal{A}_0} := \emptyset, \qquad r^{\mathcal{A}_{i+1}} := \{\alpha(\vec{x}) \mid (r(\vec{x}) \leftarrow G) \in \mathcal{P} \wedge \alpha \in [\![G]\!]^{\mathcal{A}_i}\}$$

define a chain $\mathcal{A}_0 \subseteq \mathcal{A}_1 \subseteq \ldots$ of $\mathcal{R}(\mathcal{FL})$ interpretations extending $\mathcal{I}$. Moreover, the union $\bigcup_{i \geq 0} \mathcal{A}_i$ is the least model of $\mathcal{P}$ extending $\mathcal{I}$.

2.7 **Proposition (Höhfeld and Smolka 1988, proposition 4.5, p. 12)**

Let $\mathcal{P}$ be a set of definite clauses in $\mathcal{R}(\mathcal{FL})$, let G be a goal and ϕ an $\mathcal{FL}$ constraint. Then $\phi \rightarrow G$ is valid in every model of $\mathcal{P}$ iff it is valid in every minimal model of $\mathcal{P}$.

3 HPSG grammars and grammaticality

We've set up a feature logic with terms and formulae, and we've shown how to apply the CLP scheme of (Höhfeld and Smolka 1988) to extend our language with relation symbols and obtain an operational semantics for definite clause programs in that extended language. What we haven't discussed yet is what we think HPSG grammars are in this setup, and what notion of grammaticality we should employ.

A HPSG grammar simply is a set of feature terms. It is important to note that we use the *terms* for HPSG, and not the *formulae* – the formulae serve the sole purpose to turn our feature logic into a variable constraint language. A model of a set of feature terms is an interpretation s.t. every object in the domain is in the denotation of every term in the set.

3.1 **Definition (model)**

A model of a set $\mathcal{G}$ of feature terms with respect to a signature Σ is an interpretation $\mathcal{I} = \langle \mathcal{U}, \mathcal{S}, \mathcal{A}, [\![\cdot]\!]^{\mathcal{I}} \rangle$ s.t. for every $u \in \mathcal{U}$ and for every $\phi \in \mathcal{G}$ there is an $\alpha \in \mathsf{ASS}$ s.t. $u \in [\![T]\!]^{\mathcal{I}}_{\alpha}$.

We consider grammaticality to be a relation between grammars and terms. Since grammaticality is sometimes considered to be a relation between grammars and objects, we'll use the neutral term admission instead. A grammar admits some term just in case the grammar has at least one model where the term has a non-empty denotation.

3.2 **Definition (admission)**

$\mathcal{G}$ admits a term ϕ iff there is a model $\mathcal{I}$ of $\mathcal{G}$ and a variable assignment α s.t. $[\![\phi]\!]^{\mathcal{I}}_{\alpha} \neq \emptyset$.

In general, a HPSG grammar can be an arbitrary set of terms. For the purposes of our translation, however, we require those terms to be of a specific format: $t \Rightarrow \phi$, where t is a type symbol, and ϕ is an arbitrary term. Logically, this is no restriction, since any term ϕ that is not in the required format can be made to be so by substituting the equivalent $\top \Rightarrow \phi$. Computationally, this is not so attractive and there are in fact more intelligent methods for (automatically) bringing larger classes of grammars into the required format, but for the purposes of this paper, we will simply ignore the issue and focus on this restricted form of grammars.

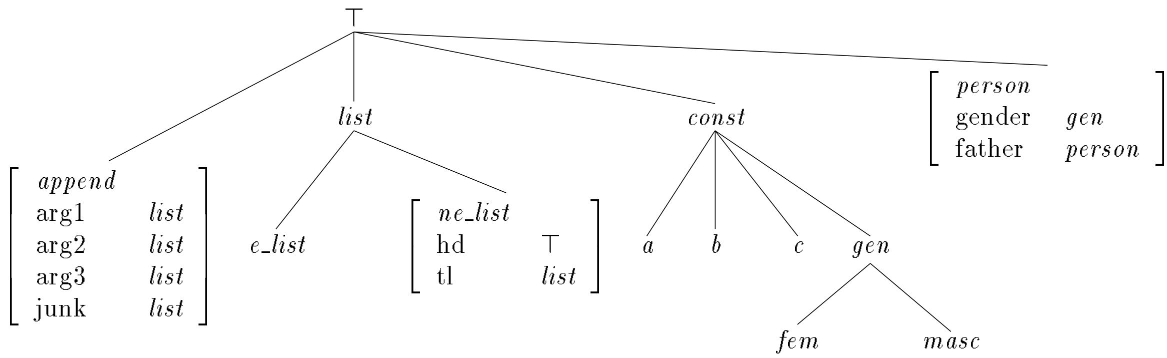
3.3 **Definition (grammar)**

A grammar is a pair $\langle \Sigma, R \rangle$ s.t. $\Sigma = \langle \mathsf{VAR}, \mathcal{T}, \preceq, \mathcal{F}, approp \rangle$ is a signature, R is finite and $R \subseteq \{t \Rightarrow \phi \mid t \in \mathcal{T} \text{ and } \phi \text{ is a term }\}$

3.4 **Example**

This simple grammar shows a junk slot encoding of the append relation[2]. The type hierarchy with the appropriateness conditions is depicted as a graph in the usual manner.

[2] See (Aït-Kaci 1984)



$$\begin{aligned} \textit{append} \Rightarrow (\ & \text{arg1: } \textit{e\_list} \wedge \text{arg2: } L \wedge \text{arg3: } L \wedge \text{junk: } \textit{e\_list} \\ \vee\ & \text{arg1: (hd: } H \wedge \text{tl: } T_1) \wedge \text{arg2: } L \wedge \text{arg3: (hd: } H \wedge \text{tl: } T_2) \wedge \\ & \text{junk: (hd: } (\textit{append} \wedge \text{arg1: } T_1 \wedge \text{arg2: } L \wedge \text{arg3: } T_2) \wedge \text{tl: } \textit{e\_list}) \end{aligned}$$

We will use this grammar later to explain the compilation process.

4 From grammars to logic programs

We now have a formal language for HPSG with a notion of grammaticality, and we have $\mathcal{R}(\mathcal{FL})$ with an operational semantics. We just somehow need to connect the two. The basic idea is very simple: for each grammar $\mathcal{G}$ we generate an $\mathcal{R}(\mathcal{FL})$-program $\mathcal{P}$ that defines the unary relation *gram* encoding grammaticality. We can then compute if ϕ is admitted by $\mathcal{G}$ by asking if the goal $gram(X)$ & $X = \phi$ has a $\mathcal{P}$-answer.

4.1 **Definition (correct translation)**

Let $\mathcal{P}$ be a logic program that defines the relation gram and $\mathcal{G}$ a grammar. $\mathcal{P}$ is a correct translation of $\mathcal{G}$ iff

$\mathcal{G}$ admits term ϕ iff the goal $gram(X)$ & $X = \phi$ has a $\mathcal{P}$-answer.

We first define an inheritance function *IC* that will later simplify our translation.

4.2 **Definition** (*IC*)

$IC : \mathcal{T} \rightarrow$ Term is a total function from the set of types to the set of terms s.t. $IC(t) = \bigwedge_{t \leq t', t' \Rightarrow \phi \in R} \phi$, where the empty conjunction is $\top$. Let $\mathcal{G} = \langle \Sigma, R \rangle$ be a grammar s.t. the sets of free variables of the terms in R are pairwise disjoint. Then $\mathcal{G}_{IC} = \langle \Sigma, \{t \Rightarrow IC(t) \mid t \in \mathcal{V} \wedge IC(t) \neq \top\} \rangle$

Clearly, $\mathcal{G}_{IC}$ is a grammar equivalent to $\mathcal{G}$.

We now partition the set of types $\mathcal{T}$ into three subsets: constrained, hiding and simple types. The motivation for the distinctions is computational and in some sense already an optimisation. In short, it obviates the need for some rather complex partial evaluation later on and reduces the number of choice points.

Here's the intuitive meaning of the three sets:

- **constrained types** are exactly those types that unify with some antecedent of the grammar, and whose structure is thus constrained by the conditions stated in the grammar.
- **hiding types** are types that are not constrained themselves but "hide" constrained types somewhere in their structure (loosely speaking).
- **simple types** are neither constrained themselves nor can they hide constrained types anywhere. In other words, the grammar doesn't say anything about them.

4.3 **Definition (type interaction)**

We say that two types t and t' interact if they have a common subtype, i.e., $\exists t'' \in \mathcal{T} .\ t'' \preceq t \wedge t'' \preceq t'$.

4.4 **Definition (directly constrained type)**

A directly constrained type is a type that serves as antecedent of an implicational constraint in the grammar, i.e., the set of directly constrained types is $\{t \mid t \Rightarrow \phi \in R\}$.

4.5 **Definition (constrained type)**

A constrained type is a type that interacts with a directly constrained type. Write C for the set of constrained types.

4.6 **Definition (hiding type)**

The set of hiding types is the smallest set $H \subseteq \mathcal{T}$ s.t.

1. *if t is not a constrained type and t interacts with a t' which is not a constrained type s.t. $approp(t', f)$ is defined and $approp(t', f)$ is a constrained type, then $t \in H$*
2. *if t is not a constrained type and t interacts with a t' which is not a constrained type s.t. $approp(t', f)$ is defined and $approp(t', f) \in H$, then $t \in H$*

4.6 is a recursive definition the base case of which is a type that has a constrained type appropriate for at least one of its features.

4.7 **Definition (simple type)**

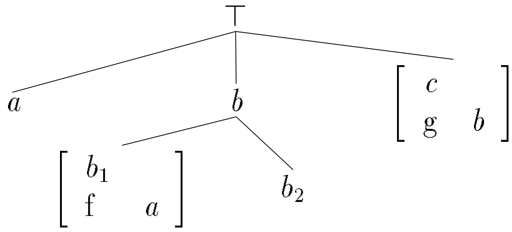
A simple type is a type that is neither a constrained nor a hiding type.

4.8 **Definition (hiding feature)**

If t is a constrained or hiding type, then f is a hiding feature on t iff $approp(t, f)\downarrow$ and $approp(t, f)$ is a constrained or hiding type.

The hiding features are the computationally interesting features on a type, as we will see later on.

4.9 **Example**



Suppose a is the only directly constrained type. Then the set of constrained types is $\{a,\top\}$. By clause 1. of def. 4.6, we know that b_1 is a hiding type, since it interacts with itself and has a constrained type (a) appropriate for feature f. By the same clause, b is a hiding type, since it interacts with b_1. $\top$ also interacts with b_1, but it's a constrained type, and thus cannot be a hiding type. Finally, by clause 2. of def. 4.6, we know that c is a hiding type, since it embeds a hiding type under feature g. So the only simple type in this hierarchy is b_2.

In ex. 3.4, the set of constrained types is $\{append,\top\}$, and the set of hiding types is $\{ne\_list, list\}$. All other types are simple types.

We will define three different sets of definite clauses defining the *gram*-relation, one for simple types, one for hiding types and one for constrained types. Notice that we only define clauses for minimal types since under our definition of interpretation, every object is of some minimal type.

4.10 **Definition** ($\mathcal{P}_s(\mathcal{G})$)

Let $\mathcal{G}$ be a grammar and S the set of simple types in $\mathcal{G}_{IC}$. Define

$\mathcal{P}_s(\mathcal{G}) = \{gram(X) \ \leftarrow X = t \mid t \in S \wedge t \in \mathcal{V}\}$

All the clauses for simple types represent base cases of the *gram* relation. From our definition of simple types, it is intuitively clear that any term of a simple type is admissible (if it's satisfiable). A proof of this fact is given in prop. 4.15. For ex. 3.4, one of the clauses in $\mathcal{P}_s(\mathcal{G})$ will be

$$gram(X) \leftarrow X = person$$

So despite the cyclic appropriateness conditions of the person type, we are assured that any object of type person is in the gram relation.

4.11 **Definition** ($\mathcal{P}_h(\mathcal{G})$)

Let $\mathcal{G}$ be a grammar and H the set of hiding types in $\mathcal{G}_{IC}$. For each type $t \in H$, define $Clause(t) := gram(X) \ \leftarrow X = (t \ \wedge \ f_1\colon Y_1 \ \wedge \ldots \wedge \ f_n\colon Y_n) \ \& \ gram(Y_1) \ \& \ldots \& \ gram(Y_n)$, where $f_1 \ldots f_n$ are the hiding features of t and $X, Y_1, \ldots, Y_n$ are pairwise distinct variables. Then let

$\mathcal{P}_h(\mathcal{G}) = \{Clause(t) \mid t \in H \wedge t \in \mathcal{V}\}$

This definition introduces a program clause for each minimal hiding type by saying that an object of a hiding type is grammatical just in case the objects under the type's hiding features are also grammatical. The non-hiding features play no role, since they only map to objects of simple types, which are grammatical in any case. For ex. 3.4, one of the clauses in $\mathcal{P}_h(\mathcal{G})$ will be

$$gram(X) \leftarrow X = (ne\_list \wedge \mathrm{hd} : Y_1 \wedge \mathrm{tl} : Y_2) \;\&\; gram(Y_1) \;\&\; gram(Y_2)$$

For constrained types, the definition is exactly parallel, except that additionally we need to add the information from the implicational constraints.

4.12 **Definition** ($\mathcal{P}_c(\mathcal{G})$)

Let $\mathcal{G}$ be a grammar and C the set of constrained types in $\mathcal{G}_{IC}$. If $t \in C$ then $Clause(t)$:= $gram(X) \leftarrow X = (t \wedge IC(t) \wedge f_1\colon Y_1 \wedge \ldots \wedge f_n\colon Y_n) \;\&\; gram(Y_1) \;\&\ldots\&\; gram(Y_n)$, where $f_1 \ldots f_n$ are the hiding features of t and $X, Y_1, \ldots, Y_n$ are pairwise distinct variables not occuring in $IC(t)$. Then let

$$\mathcal{P}_c(\mathcal{G}) = \{Clause(t) \mid t \in C \wedge t \in \mathcal{V}\}$$

For ex. 3.4, the only clause in $\mathcal{P}_c(\mathcal{G})$ is

$$\begin{aligned} gram(X) \leftarrow X = (\,(\; & \mathrm{arg1}\colon e\_list \wedge \mathrm{arg2}\colon L \wedge \mathrm{arg3}\colon L \wedge \mathrm{junk}\colon e\_list \\ \vee\; & \mathrm{arg1}\colon (\mathrm{hd}\colon H \wedge \mathrm{tl}\colon T_1) \wedge \mathrm{arg2}\colon L \wedge \mathrm{arg3}\colon (\mathrm{hd}\colon H \wedge \mathrm{tl}\colon T_2) \wedge \\ & \quad \mathrm{junk}\colon (\mathrm{hd}\colon (append \wedge \mathrm{arg1}\colon T_1 \wedge \mathrm{arg2}\colon L \wedge \mathrm{arg3}\colon T_2) \wedge \mathrm{tl}\colon e\_list)) \wedge \\ & \mathrm{arg1}\colon Y_1 \wedge \mathrm{arg2}\colon Y_2 \wedge \mathrm{arg3}\colon Y_3 \wedge \mathrm{junk}\colon Y_4) \;\& \\ & gram(Y_1) \;\&\; gram(Y_2) \;\&\; gram(Y_3) \;\&\; gram(Y_4) \end{aligned}$$

The complete program is simply the union of the three sets we've just defined.

4.13 **Definition** ($\mathcal{P}(\mathcal{G})$)

$\mathcal{P}(\mathcal{G}) = \mathcal{P}_s(\mathcal{G}) \cup \mathcal{P}_h(\mathcal{G}) \cup \mathcal{P}_c(\mathcal{G})$

4.1 Soundness

Having defined the translation, we can now investigate it's correctness. We start by proving a strong soundness result showing that the minimal model construction for the definite clause programs can be used to build models for the corresponding grammar. The reader not interested in the technical details may wish to go directly to prop. 4.19.

4.14 **Definition**

Let $\mathcal{I} = \langle \mathcal{U}, \mathcal{S}, \mathcal{A}, [\![\cdot]\!]^{\mathcal{I}} \rangle$ be an interpretation.
If $\mathcal{U}' \subseteq \mathcal{U}$, then $\mathcal{I}_{|\mathcal{U}'} = \langle \mathcal{U}', \mathcal{S}_{|\mathcal{U}'}, \mathcal{A}_{|\mathcal{U}'}, [\![\cdot]\!]^{\mathcal{I}_{|\mathcal{U}'}} \rangle$ where for each $f \in \mathcal{F}$, for each $u \in \mathcal{U}'$, $\mathcal{A}_{|\mathcal{U}'}(f)(u)\downarrow$ iff $\mathcal{A}(f)(u)\downarrow$ and $\mathcal{A}(f)(u) \in \mathcal{U}'$, and if $\mathcal{A}_{|\mathcal{U}'}(f)(u)\downarrow$, then $\mathcal{A}_{|\mathcal{U}'}(f)(u) = \mathcal{A}(f)(u)$.

This definition takes an interpretation and "makes it smaller". Note that $\mathcal{I}_{|\mathcal{U}'}$ is not always an interpretation.

4.15 **Proposition**

Let $\mathcal{G} = \langle \Sigma, R \rangle$ be a grammar, $S_{\mathcal{G}}$ be the set of simple types in $\mathcal{G}$, $\mathcal{I} = \langle \mathcal{U}, \mathcal{S}, \mathcal{A}, [\![\cdot]\!]^{\mathcal{I}} \rangle$ be an interpretation and $\mathcal{W} \subseteq \mathcal{U}$ s.t. $\mathcal{W} = \{u \in \mathcal{U} \mid \mathcal{V}(u) \in S_{\mathcal{G}}\}$. Then $\mathcal{J} = \langle \mathcal{W}, \mathcal{S}_{|\mathcal{W}}, \mathcal{A}_{|\mathcal{W}}, [\![\cdot]\!]^{\mathcal{J}} \rangle$ is a model of $\mathcal{G}$.

Proof

To show that $\mathcal{J}$ is a model of $\mathcal{G}$, we first need to show that $\mathcal{J}$ is an interpretation. For this we need to show in particular that $\mathcal{A}_{|\mathcal{W}}$ meets definition 2.4:

for each $u \in \mathcal{U}$, for each $f \in \mathcal{F}$, if $approp(\mathcal{S}(u), f)$ is defined and $approp(\mathcal{S}(u), f) = t$, then $(\mathcal{A}(f))(u)$ is defined and $\mathcal{S}((\mathcal{A}(f))(u)) \preceq t$

$\Longrightarrow$ for each $u \in \mathcal{W}$ and $f \in \mathcal{F}$, if $approp(\mathcal{S}_{|\mathcal{W}}(u), f)$ is defined and $approp(\mathcal{S}_{|\mathcal{W}}(u), f) = t$, then $(\mathcal{A}_{|\mathcal{W}}(f))(u)$ is defined and $\mathcal{S}_{|\mathcal{W}}((\mathcal{A}_{|\mathcal{W}}(f))(u)) \preceq t$, since t must be a simple type

and

for each $u \in \mathcal{U}$, for each $f \in \mathcal{F}$, if $(\mathcal{A}(f))(u)$ is defined, then $approp(\mathcal{S}(u), f)$ is defined and $\mathcal{S}((\mathcal{A}(f))(u)) \preceq approp(\mathcal{S}(u), f)$

$\Longrightarrow$ for each $u \in \mathcal{W}$, for each $f \in \mathcal{F}$, if $(\mathcal{A}_{|\mathcal{W}}(f))(u)$ is defined, then $approp(\mathcal{S}_{|\mathcal{W}}(u), f)$ is defined and $\mathcal{S}_{|\mathcal{W}}((\mathcal{A}_{|\mathcal{W}}(f))(u)) \preceq approp(\mathcal{S}_{|\mathcal{W}}(u), f)$

Thus, $\mathcal{J}$ is an interpretation.

Furthermore,

for each $u \in \mathcal{W}$, $\mathcal{S}_{|\mathcal{W}}(u)$ is a simple type

$\Longrightarrow$ for each $u \in \mathcal{W}$, for each $(t \Rightarrow T) \in R$, for each $\alpha \in \mathsf{ASS}$, $u \in [\![t \Rightarrow T]\!]^{\mathcal{J}}_{\alpha}$ (since $[\![t]\!]^{\mathcal{J}}_{\alpha} \cap [\![\mathcal{S}_{|\mathcal{W}}(u)]\!]^{\mathcal{J}}_{\alpha} = \emptyset$)

$\Longrightarrow$ for each $u \in \mathcal{W}$, for each $(t \Rightarrow T) \in R$, for some $\alpha \in \mathsf{ASS}$, $u \in [\![t \Rightarrow T]\!]^{\mathcal{J}}_{\alpha}$

$\Longrightarrow$ $\mathcal{J}$ is a model of $\mathcal{G}$

$\square$

This result says that as far as simple types are concerned, satisfiability and admissibility are equivalent. In other words, if there's a term that we know to be of a simple type with respect to some grammar, then that grammar admits that term iff the term is at all satisfiable.

The next proposition makes a related claim about hiding types. Intuitively, it says that whether an object of a hiding type is part of a model of a certain grammar only depends on whether the objects it's mapped to are part of a model. Note that this is not generally true of all objects. For constrained types, this property also depends on what the mapped to objects "look like". This will be made explicit in proposition 4.17.

4.16 **Proposition**

Let $\mathcal{G} = \langle \Sigma, R \rangle$ be a grammar, $\mathcal{I} = \langle \mathcal{U}, \mathcal{S}, \mathcal{A}, [\![\cdot]\!]^{\mathcal{I}} \rangle$ be an interpretation and $\mathcal{W} \subseteq \mathcal{U}$ s.t.

$\mathcal{I}_{|\mathcal{W}}$ *is a model of* $\mathcal{G}$*. If* t *is a minimal hiding type and* $u \in \mathcal{U}$ *s.t.* $\mathcal{S}(u) = t$ *and for each feature* f*, if* $\mathcal{A}(f)(u)\downarrow$ *then* $\mathcal{A}(f)(u) \in \mathcal{W}$*, then* $\mathcal{I}_{|\mathcal{W}\cup\{u\}}$ *is a model of* $\mathcal{G}$*.*

Proof

Clearly, $\mathcal{I}_{|\mathcal{W}\cup\{u\}}$ is an interpretation. Furthermore,

t is a hiding type

$\Longrightarrow$ for each $t' \Rightarrow T \in R$, for each $\alpha \in \mathsf{ASS}$, $[\![t]\!]_\alpha^{\mathcal{I}_{|\mathcal{W}\cup\{u\}}} \cap [\![t']\!]_\alpha^{\mathcal{I}_{|\mathcal{W}\cup\{u\}}} = \emptyset$

$\Longrightarrow$ for each $t' \Rightarrow T \in R$, for each $\alpha \in \mathsf{ASS}$, $u \in [\![t' \Rightarrow T]\!]_\alpha^{\mathcal{I}_{|\mathcal{W}\cup\{u\}}}$

$\Longrightarrow$ $\mathcal{I}_{|\mathcal{W}\cup\{u\}}$ is a model of $\mathcal{G}$

$\square$

4.17 **Proposition**

Let $\mathcal{G} = \langle \Sigma, R\rangle$ *be a grammar,* $\mathcal{I} = \langle \mathcal{U}, \mathcal{S}, \mathcal{A}, [\![\cdot]\!]^{\mathcal{I}}\rangle$ *be an interpretation and* $\mathcal{W} \subseteq \mathcal{U}$ *s.t.* $\mathcal{I}_{|\mathcal{W}}$ *is a model of* $\mathcal{G}$*. If* t *is a minimal constrained type and* $u \in \mathcal{U}$ *s.t.* $\mathcal{S}(u) = t$*, for each feature* f*, if* $\mathcal{A}(f)(u)\downarrow$ *then* $\mathcal{A}(f)(u) \in \mathcal{W}$ *and if* $t \Rightarrow T \in IC(R)$ *then for some* $\alpha \in \mathsf{ASS}$*,* $u \in [\![T]\!]_\alpha^{\mathcal{I}_{|\mathcal{W}\cup\{u\}}}$*, then* $\mathcal{I}_{|\mathcal{W}\cup\{u\}}$ *is a model of* $\mathcal{G}$*.*

Proof

Let $\mathcal{J} = \mathcal{I}_{|\mathcal{W}\cup\{u\}}$. Clearly, $\mathcal{J}$ is an interpretation. Furthermore,

t is a minimal constrained type and if $t \Rightarrow T \in IC(R)$, then for some α, $u \in [\![T]\!]^{\mathcal{J}}$

$\Longrightarrow$ for each t', for some α, if $t' \Rightarrow T' \in IC(R)$ then $u \in [\![t' \Rightarrow T']\!]^{\mathcal{J}}$

$\Longrightarrow$ $\mathcal{J}$ is a model of $\mathcal{G}$

$\square$

4.18 **Proposition**

Let $\mathcal{G}$ *be a grammar,* $\mathcal{I} = \langle \mathcal{U}, \mathcal{S}, \mathcal{A}, [\![\cdot]\!]^{\mathcal{I}}\rangle$*,* $X \subseteq \wp(\mathcal{U})$ *s.t. for each* $\mathcal{U}' \in X$*,* $\mathcal{I}_{|\mathcal{U}'}$ *is a model of* $\mathcal{G}$*, and let* $\mathcal{W} = \bigcup X$*. Then* $\mathcal{I}_{|\mathcal{W}}$ *is a model of* $\mathcal{G}$*.*

4.19 **Proposition**

If $\mathcal{G}$ *is a grammar,* $\mathcal{I} = \langle \mathcal{U}, \mathcal{S}, \mathcal{A}, [\![\cdot]\!]^{\mathcal{I}}\rangle$ *a* $\mathcal{FL}$*-interpretation and* $\mathcal{M}$ *the minimal model of* $\mathcal{P}(\mathcal{G})$ *extending* $\mathcal{I}$*, then* $\mathcal{I}_{|\{u\in\mathcal{U} | \langle u\rangle \in [\![gram]\!]^{\mathcal{M}}\}}$ *is a model of* $\mathcal{G}$*.*

Proof

Firstly, since for each simple type t, $(gram(X) \leftarrow X = t) \in \mathcal{P}(\mathcal{G})$, we know that for each $i \geq 1$, $\mathcal{U}_s = \{u \in \mathcal{U} \mid \mathcal{S}(u) \text{ is a simple type}\} \subseteq \{u \in \mathcal{U} \mid \langle u\rangle \in [\![gram]\!]^{\mathcal{M}_i}\}$ . . . (1)

We can now do an induction on the construction of $\mathcal{M}$.

Clearly, $\mathcal{I}_{|\{u\in\mathcal{U}|\langle u\rangle\in[\![gram]\!]^{\mathcal{M}_0}\}} = \mathcal{I}_{|\emptyset}$ is a model of $\mathcal{G}$.

Now suppose $i \geq 1$. Let $\mathcal{W} = \{u \in \mathcal{U} \mid \langle u\rangle \in [\![gram]\!]^{\mathcal{M}_{i-1}}\}$ and $\mathcal{W}' = \{u \in \mathcal{U} \mid \langle u\rangle \in [\![gram]\!]^{\mathcal{M}_i}\}$.

For each $u \in \mathcal{U}$,

$u \in \mathcal{W}'$

$\Longrightarrow \mathcal{S}(u)$ is a simple type

$\quad\Longrightarrow u \in \mathcal{U}_s, \mathcal{U}_s \subseteq \mathcal{W}'$ and $\mathcal{I}_{|\mathcal{U}_s}$ is a model of $\mathcal{G}$, and by (1) and prop. 4.15

$\quad\mathcal{S}(u)$ is a hiding type

$\quad\Longrightarrow \exists(gram(X) \leftarrow X = (\mathcal{S}(u) \ \wedge \ f_1 : Y_1 \ \wedge \ \dots) \ \& \ gram(Y_1) \ \& \ \dots) \in \mathcal{P}(\mathcal{G})$. $\exists\alpha \in$ ASS. $\alpha \in [\![X = (\mathcal{S}(u) \ \wedge \ f_1 : Y_1 \ \wedge \ \dots) \ \& \ gram(Y_1) \ \& \ \dots]\!]^{\mathcal{M}_{i-1}}$ and $\alpha(X) = u$

$\quad\Longrightarrow \mathcal{I}_{|\{u\}\cup\mathcal{W}\cup\mathcal{U}_s}$ is a model of $\mathcal{G}$, and by (1) and propositions 4.15, 4.16 and 4.18

$\quad\mathcal{S}(u)$ is a constrained type

$\quad\Longrightarrow \exists(gram(X) \leftarrow X = (\mathcal{S}(u) \ \wedge \ IC(\mathcal{S}(u)) \ \wedge \ f_1 : Y_1 \ \wedge \ \dots) \ \& \ gram(Y_1) \ \& \ \dots) \in$ $\mathcal{P}(\mathcal{G})$. $\exists\alpha \in \mathsf{ASS}$. $\alpha \in$ $[\![X = (\mathcal{S}(u) \ \wedge \ IC(\mathcal{S}(u)) \ \wedge \ f_1 : Y_1 \ \wedge \ \dots) \ \& \ gram(Y_1) \ \& \ \dots]\!]^{\mathcal{M}_{i-1}}$ and $\alpha(X) = u$

$\quad\Longrightarrow \mathcal{I}_{|\{u\}\cup\mathcal{W}\cup\mathcal{U}_s}$ is a model of $\mathcal{G}$ by (1) and propositions 4.15, 4.17 and 4.18

$\Longrightarrow \mathcal{I}_{|\mathcal{W}'}$ is a model of $\mathcal{G}$

Thus, by induction and prop. 4.18, $\mathcal{I}_{|\{u\in\mathcal{U}|\langle u\rangle\in[\![gram]\!]^{\mathcal{M}}\}}$ is a model of $\mathcal{G}$.

□

4.20 **Proposition (soundness)**

Let $\mathcal{G}$ be a grammar and ϕ a term. If $gram(X) \ \& \ X = \phi$ has a $\mathcal{P}(\mathcal{G})$-answer then $\mathcal{G}$ admits ϕ.

Proof

By props. 2.6, 2.7 and 4.19. □

This completes our soundness result: every answer computed by a program we automatically compile from a grammar is a correct answer. We will now investigate the opposite direction, i.e., if some term is admitted by the grammar, does the program also provide an answer for it?

4.2 Completeness

Our method is not complete. For some queries in some grammars there can be no finite proofs (or refutations). E.g., adding the (reasonable) constraint

$$person \Rightarrow \text{father: gender: } male$$

to our example 3.4, this would be translated to

$$gram(X) \leftarrow X = (person \wedge \text{father: gender: } male \wedge \text{father: } Y_1) \ \& \ gram(Y_1)$$

which goes into an infinite loop on the query

$$gram(X) \ \& \ X = person$$

This particular example could be handled using lazy evaluation as proposed in (Aït-Kaci et al. 1993), but there are really pathological examples. (Aït-Kaci et al. 1993) show that their system is complete for finding refutations. However, we conjecture that this is due to the fact that their system does not handle disjunction.

What we can show is that we can find a proof if the query is satisfiable in a model of the grammar that is finite in a certain sense. For the linguist, this means that every individual linguistic structure has to be finite, which seems to be a reasonable restriction.

4.21 **Definition** ($Path_{\mathcal{I}}$)

If $\mathcal{I} = \langle \mathcal{U}, \mathcal{S}, \mathcal{A}, [\![\cdot]\!]^{\mathcal{I}} \rangle$ *and* $u \in \mathcal{U}$*, then* $Path_{\mathcal{I}}(u) = \{\pi \mid \mathcal{A}^*(\pi)(u)\downarrow\}$[3]

4.22 **Definition (semi-finite interpretation)**

An interpretation $\mathcal{I} = \langle \mathcal{U}, \mathcal{S}, \mathcal{A}, [\![\cdot]\!]^{\mathcal{I}} \rangle$ *is called semi-finite iff for each* $u \in \mathcal{U}$*,* $Path_{\mathcal{I}}(u)$ *is finite.*

4.23 **Proposition**

Let $\mathcal{I} = \langle \mathcal{U}, \mathcal{S}, \mathcal{A}, [\![\cdot]\!]^{\mathcal{I}} \rangle$ *be an interpretation,* $\mathcal{G}$ *a grammar,* $\mathcal{P}(\mathcal{G})$ *the program associated with* $\mathcal{G}$ *and* $\mathcal{M}$ *the minimal model of* $\mathcal{P}(\mathcal{G})$ *extending* $\mathcal{I}$*.*

If $\mathcal{W} \subseteq \mathcal{U}$ s.t. $\mathcal{I}_{|\mathcal{W}}$ *is a semi-finite model of* $\mathcal{G}$ *and* $u \in \mathcal{W}$*, then* $u \in \{v \mid \langle v \rangle \in [\![gram]\!]^{\mathcal{M}}\}$*.*

Proof

The proof runs by induction on $|Path_{\mathcal{I}}(u)|$.

$|Path_{\mathcal{I}}(u)| = 1$

$\Longrightarrow Path_{\mathcal{I}}(u) = \{\varepsilon\}$ and

1) $\mathcal{S}(u)$ is a simple type
   $\Longrightarrow gram(X) \leftarrow X = \mathcal{S}(u) \in \mathcal{P}(\mathcal{G})$
   $\Longrightarrow u \in \{v \mid \langle v \rangle \in [\![gram]\!]^{\mathcal{M}}\}$
2) $\mathcal{S}(u)$ is not a hiding type, since hiding types have at least one feature defined on them
3) $\mathcal{S}(u)$ is a constrained type
   $\Longrightarrow gram(X) \leftarrow X = (\mathcal{S}(u) \ \& \ IC(\mathcal{S}(u))) \in \mathcal{P}(\mathcal{G})$
   $\Longrightarrow u \in \{v \mid \langle v \rangle \in [\![gram]\!]^{\mathcal{M}}\}$

Furthermore,

$i \geq 1$, for each $v \in \mathcal{W}$ s.t. $|Path_{\mathcal{I}}(v)| \leq i$, $\langle v \rangle \in [\![gram]\!]^{\mathcal{M}}$ and $|Path_{\mathcal{I}}(u)| = i + 1$

$\Longrightarrow$

1) $\mathcal{S}(u)$ is a simple type
   $\Longrightarrow gram(X) \leftarrow X = \mathcal{S}(u) \in \mathcal{P}(\mathcal{G})$

[3]Where $\mathcal{A}^*$ is the reflexive transitive closure of $\mathcal{A}$

$\Longrightarrow u \in \{v \mid \langle v \rangle \in [\![gram]\!]^{\mathcal{M}}\}$

2) $\mathcal{S}(u)$ is a hiding type

$\Longrightarrow gram(X) \leftarrow X = (\mathcal{S}(u)\ \&\ f_1 \colon Y_1\ \&\ldots)\ \&\ gram(Y_1)\ldots \in \mathcal{P}(\mathcal{G})$

$\Longrightarrow \exists\alpha \in \mathsf{ASS}.\ \alpha \in [\![X = (\mathcal{S}(u)\ \&\ f_1 \colon Y_1\ \&\ldots)\ \&\ gram(Y_1)\ldots]\!]^{\mathcal{M}}$, $\alpha(X) = u$ and for each j, $\alpha(Y_j) = \mathcal{A}(f_j)(u)$, by the inductive hypothesis

$\Longrightarrow \exists\alpha.\ \alpha(X) = u$ and $\langle\alpha(X)\rangle \in [\![gram]\!]^{\mathcal{M}}$

$\Longrightarrow u \in \{v \mid \langle v \rangle \in [\![gram]\!]^{\mathcal{M}}\}$

3) $\mathcal{S}(u)$ is a constrained type

$\Longrightarrow gram(X) \leftarrow X = (\mathcal{S}(u)\ \&\ IC(\mathcal{S}(u))\ \&\ f_1 \colon Y_1\ \&\ldots)\ \&\ gram(Y_1)\ldots \in \mathcal{P}(\mathcal{G})$

$\Longrightarrow \exists\alpha \in \mathsf{ASS}.\ \alpha \in [\![X = (\mathcal{S}(u)\ \&\ IC(\mathcal{S}(u))\ \&\ f_1 : Y_1\ \&\ldots)\ \&\ gram(Y_1)\ldots]\!]^{\mathcal{M}}$, $\alpha(X) = u$ and for each j, $\alpha(Y_j) = \mathcal{A}(f_j)(u)$, by the inductive hypothesis

$\Longrightarrow \exists\alpha.\ \alpha(X) = u$ and $\langle\alpha(X)\rangle \in [\![gram]\!]^{\mathcal{M}}$

$\Longrightarrow u \in \{v \mid \langle v \rangle \in [\![gram]\!]^{\mathcal{M}}\}$

$\square$

5 Conclusion and outlook

We have presented a compilation scheme that translates HPSG constraint grammars into efficient logic programs by using a classification of types. The compilation is sound and semi-finitely complete.

It should be pointed out that (Aït-Kaci 1984) and particularly (Aït-Kaci et al. 1993) address a problem very similar to the one presented here. They provide a proof system which has the very nice property of lazy sort-unfolding, i.e., applying constraints only when necessary. However, for this to work, the theory has to obey certain well-formedness conditions that cannot be directly restated in our system. We are currently investigating semantic constraints on grammars that would allow a notion of laziness rather close to the one of (Aït-Kaci et al. 1993).

There's a small drawback to introducing a stronger notion of laziness: we lose the strong soundness property. Prop. 4.19 tells us that every satisfiable term subsumed by a computed answer is also an answer. This is no longer true when we use lazy sort-unfolding, but probably well worth losing for the computational advantages that more laziness offers. Of course, our approach is already lazy to some degree in that we only unfold under hiding features.

Another point to note is that the compilation process is completely insensitive to the form of the constraints (e.g., whether they contain negation), it simply doesn't care. What the constraints can look like is the problem of the constraint solver, and if we use CUF as our underlying constraint solver, we can use full negation. If our constraint solver is less powerful, we need to be more restrictive, but this does not affect the basic compilation process.

We feel that the greatest advantage of our approach as far as NLP is concerned is that the result of our compilation process is a logic program. It puts constraint-based grammar processing into the well-understood realm of logic programming and allows application of all the well-known techniques of that area. We can use off-line program transformation as well as dynamic reordering strategies, tabulation techniques (Dörre 1993) etc. An important consequence of compiling into logic programs is that we can try to find suitably weak constraints on grammars to be able to extract phrase structure rules from the definite clause programs. This would enable us to further optimise program execution, using the usual string position indexing techniques.

References

Aït-Kaci, H. (1984). *A lattice theoretic approach to computation based on a calculus of partially ordered type structures.* Ph. D. thesis, University of Pennsylvania.

Aït-Kaci, H., A. Podelski, and S. C. Goldstein (1993). Order-Sorted Theory Unification. Technical Report 32, Digital Equipment Corporation.

Dörre, J. (1993, August). Generalizing Earley deduction for constraint-based grammars. In J. Dörre (Ed.), *Computational aspects of constraint based linguistic descriptions I*, pp. 25–41. Universität Stuttgart: DYANA-2 Deliverable R1.2.A.

Dörre, J. (1994). Feature-Logik und Semiunifikation. Arbeitspapiere des SFB 340 Nr. 48, Universität Stuttgart.

Dörre, J. and M. Dorna (1993, August). CUF - a formalism for linguistic knowledge representation. In J. Dörre (Ed.), *Computational aspects of constraint based linguistic descriptions I*, pp. 1–22. Universität Stuttgart: DYANA-2 Deliverable R1.2.A.

Höhfeld, M. and G. Smolka (1988). Definite Relations over Constraint Languages. Technical Report 53, LILOG, IBM Deutschland.

King, P. J. (1994). An expanded logical formalism for head-driven phrase structure grammar. Arbeitspapiere des SFB 340 Nr. 59, Universität Tübingen.

Mellish, C. (1992). Term-Encodable Description Spaces. In D. R. Brough (Ed.), *Logic Programming: New Frontiers.* Kluwer Academic Publishers.

Meurers, W. D. (1994). On Implementing an HPSG theory – Aspects of the logical architecture, the formalization, and the implementation of head-driven phrase structure grammars. In: Erhard W. Hinrichs, W. Detmar Meurers, and Tsuneko Nakazawa: *Partial-VP and Split-NP Topicalization in German – An HPSG Analysis and its Implementation.* Arbeitspapiere des SFB 340 Nr. 58, Universität Tübingen.

Pollard, C. and I. A. Sag (1994). *Head-Driven Phrase Structure Grammar.* Chicago: University of Chicago Press.

Smolka, G. (1992). Feature-Constraint Logics for Unification Grammars. *Journal of Logic Programming 12*, 51–87.