Feature Graphs and Abstract Data Types: A Unifying Approach

Christoph BEIERLE and Udo PLETAT
IBM Deutschland GmbH
Science and Technology - LILOG
P.O. Box 80 08 80
7000 Stuttgart 80, West Germany
(electronic mail on EARN/BITNET:
BEIERLE at DSφLILOG, PLETAT at DSφLILOG)

Abstract:

Feature graphs appearing in unification-based grammar formalisms and algebraic specifications of abstract data types (ADTs) are both used for defining a collection of objects together with functions between these object sets. Starting from this observation we define an algebraic semantics for feature graphs by assigning an algebraic specification to each feature graph. This opens the rich world of semantical foundations for abstract data types to the area of feature graphs and thus to unification grammar formalisms. Using results from ADT theory we define a simple and fast syntactic decision procedure testing the usual consistency conditions on feature graphs like constant consistency, constant/complex consistency and acyclicity on the algebraic specification assigned to a feature graph. With this machinery at hand feature graph unification becomes union of feature graph specifications followed by the consistency test.

1. Introduction

Unification-based grammar formalisms have become a popular field of research. The subject has attracted the interest not only of (computer) linguists but also of computer scientists, especially in the area of logic programming, see e.g. [Pe 87]. Due to the formality of grammars we also observe activities developing foundations for the formal semantics of various approaches on unification grammars, e.g. [PS 84], [KR 86], [RK 86], [Pe 87], [Jo 87].

In this paper we investigate the relationship between feature graphs on the one hand and algebraic specifications of abstract data types (ADTs) on the other hand. There is a natural correlation between both these areas since a feature graph as well as an abstract data type defines a collection of objects together with functions relating the objects. We present a formal semantics for feature graphs by assigning to each feature graph G an equational ADT specification $\tau(G)$, called fg-specification. This opens the rich world of mathematical foundations of ADT specifications (e.g [GTW 78], [EM 85]) in order to obtain a better - not only - formal understanding of the nature of feature graphs.

In particular, we provide a model-theoretic characterization of various consistency conditions for feature graph specifications reflecting the consistency concepts usually imposed on feature graphs such as clash-freeness and acyclicity. These model-theoretic characterizations have proof-theoretic counterparts in terms of syntactic conditions on the deductive closure of the set of equations of $\tau(G)$.

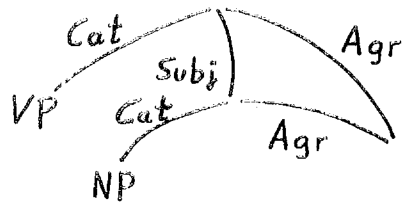
Although the proof-theoretic consistency characterizations are of syntactic nature, a test of their validity requires to examine the deductive closure of the set of equations of a fg-specification. Our objective is to restrict consistency checks to equations explicitly mentioned in a fg-specification. In the ADT-world there is a well-known tool for such tasks: the Knuth-Bendix algorithm ([KB 70]). We present a Knuth-Bendix like completion procedure transforming any fg-specification into a reduced normal form. We show that the model-theoretic consistency characterizations for G are equivalent to the presence resp. absence of certain types of equations in this reduced normal form.

These results are used for defining the semantics of the unification of two feature graphs G_1 and G_2 as the (set-theoretic componentwise) union of $\tau(G_1)$ and $\tau(G_2)$ followed by the normalization process using the completion algorithm and the consistency check on the resulting set of equations.

2. Feature graphs

A **feature graph** is a directed graph with a distinguished root node. The edges of the graph are called **features**. An **atomic** graph is just a symbol; it contains no features. A **complex** graph is a set of feature-value pairs where each value is either an atomic graph or again a complex one. Two paths starting at the root and ending at the same node are said to co-refer. **Feature graph unification** is a binary operation taking two graphs and returning a graph containing exactly the information of both graphs if they are unifiable, and fails otherwise. An atomic graph is **unifiable** only with itself and the empty graph. A complex graph G_1 is unifiable with the empty graph, and G_1 is unifiable with a complex graph G_2 if for all features in both G_1 and G_2 the respective values are unifiable.

Several notations for feature graphs have been suggested. The graphical representation of



could also be represented in matrix form:

$$\left[\begin{array}{ll} \text{Cat :} & \text{VP} \\ \text{Subj:} & \left[\begin{array}{ll} \text{Cat : NP} \\ \text{Agr : <1>} \end{array}\right] \\ \text{Agr : <1>} & \end{array}\right]$$

In the matrix notation coreference is indicated by numbers enclosed in brackets. Another notation, which for instance is employed in PATR, uses special equations indicating coreference of paths and atomic values at the end of of paths, respectively:

< Cat > = VP
< Subj Cat > = NP
< Subj Agr > = < Agr >

In the following we will discuss an equational representation of feature graphs in greater detail. This representation will be oriented towards the equational specification of abstract data types, thus making available the powerful machinery developed for such specifications.

Above we did not make a clear distinction between the syntax for describing feature graphs and feature graphs as semantical objects. In fact, such a distinction has been omitted to a large extent in the literature. The situation changed with approaches that formalize the concepts of feature graphs since such a distinction is essential for a formal treatment (see e.g. [KR 86], [Jo 87]). In the area of ADT specifications the strict separation of the syntactic and the semantic level has always been a central aspect. Our ADT-based approach to feature graphs adopts this two-level view in a natural way: Feature graph specifications are the syntactical means to describe feature graphs which are the models (or structures) of such specifications.

3. Equational Specifications

3.1 ADT Specifications: Syntax and Semantics

We introduce the basic notions of abstract data type specifications. More detailed information can be gathered in [GTW 78] or [EM 85].

A signature is a pair Σ = <S, O> where S is a set of sorts and O = $<O_{v,s}>_{v \in S^*, s \in S}$ is a family of sets of operators w.r.t. S. We write op: $s_1 \ldots s_n$ -> s for an operator whose i-th argument is of sort s_i and which delivers a result of sort s. The well-formed terms w.r.t. Σ and S-sorted variables V form an S-indexed family $T_\Sigma(V)$, and an equation over Σ and V is of the form l = r where l and r are terms of the same sort. An algebraic specification is a pair SP = <Σ, E> where Σ is a signature and E is a set of equations over Σ and some family of variables V.

Besides these syntactical concepts of ADT specifications we provide the basic semantical concepts of heterogenous algebras:

Given a signature Σ = <S, O>, a Σ-algebra A consists of a family of sets A = $< A_s >_{s \in S}$ and for each operator op ε $O_{v,s}$ there is a function op_A : A_v ---> A_s. A satisfies an equation l = r if for each assignment of values from A to the variables of l and r the evaluation of l and r in A yields the same element of A. A is a <Σ,E>-algebra if A satisfies every e ε E.

We say that E semantically implies an equation e if every <Σ,E>-algebra satisfies e. It is well-known that this model-theoretic notion of satisfaction coincides with the proof-theoretic notion of deduction (Birkhoff theorem) where e can be proved from E iff e can be deduced from E using the rules of the equational calculus (e.g. [EM 85]). We let E* denote the deductive closure of E.

The following theorem is one of the central results of ADT theory and forms the basis for defining the semantics of a specification.

Theorem:
For each algebraic specification SP = <Σ, E> there is an **initial** algebra T_{SP} satisfying the equations in E.

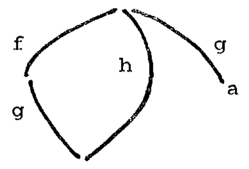
T_{SP} is the so-called quotient term algebra consisting of congruence classes - obtained by factorization according to the equations in E - of constant terms over Σ. The initial algebra T_{SP} is the ADT specified by SP. It can be characterized informally by two conditions: all its elements are denoted by some ground term ("no junk"), and it satisfies a ground equation e iff every other <Σ,E>-algebra also satisfies e ("no confusion").

3.2 Feature Graph Specifications

Feature graphs can be seen as particular algebraic specifications (see also [Pe 87]):

There are only constants (representing atomic values) and unary functions (representing the features) in the signature. We assume that ATOMS is the set of all atomic values and FEATURES is set of all features occuring in the feature graphs; both sets may be infinite in general. An equation s = t is given for paths having the same final node, or for paths ending at an atomic value.

For instance, consider the feature graph



Following e.g. Pereira and Shieber ([PS 84]) this feature graph can be described by the equations

$$< f\ g > = < h >$$
$$< g > \ = a$$

leaving the root symbol of the feature graph implicit and using the order for the attributes as in the examples of Section 2. However, in equational ADT specifications it is essential to state explicitly to which objects an equation can be applied.

For simplicitly, we first assume that we have only one sort which we call 'universe', and that we have a variable x of sort 'universe'. By using a functional notation for the attributes and thus the reverse order of the attributes as compared to e.g. [PS 84], we would arrive at the specification

```
   sorts        universe
   functions    a: -> universe
                f,g,h: universe -> universe
   equations    g(f(x)) = h(x)
                g(x) = a
```

However, by simply introducing a universally quantified variable x of sort 'universe' we run into problems: From

1. $g(f(x)) = h(x)$
2. $g(x) = a$

we can deduce
$h(x) = a$
by using the usual rules of the equational calculus and substituting $f(x)$ for x in equation (2). The problem is that x should be quantified only over all objects described by the original feature graph. But $f(x)$ is not neccessarily in this set, so we must find a way of avoiding such a substitution. A simple way of achieving this is to switch to another signature with an additional sort, say 'soi', denoting the 'sort of interest' and comprising all objects described by a feature graph.

The sort 'soi' is then a subsort of sort 'universe'. This could be expressed by extending the algebraic specifications to include also subsort relationships, thus moving from many-sorted specifications (as described in Section 3.1) to so-called order-sorted specifications (e.g. [GM 87]). Here, we want to stick to the simpler case of many-sorted specifications.

A subsort relationship in a many-sorted setting is expressed by an inclusion function which we will denote by

i: soi -> universe

in our case. In order to avoid problems with empty sorts ([GM 87], [EM 85]) we assume that there is a constant 'const' of sort 'soi'. For the rest of this paper we also assume that x is a variable of sort 'soi'. Thus, the feature graph above gives rise to the feature graph specification

```
   sorts        soi, universe
   functions    a: -> universe
                f,g,h: universe -> universe
                const: -> soi
                i: soi -> universe
   equations    g(f(i(x))) = h(i(x))
                g(i(x)) = a
```

motivating the following definition:

A **feature graph signature** (fg-signature) $\Sigma = \langle S, OP \rangle$ is a signature with

$$S = \{soi, universe\}$$
$$OP = Atoms(\Sigma) \cup Features(\Sigma) \cup \{const: \ \to soi\} \cup \{i: soi \to universe\}$$

where:

$$Atoms(\Sigma) \subseteq \{a: \ \to universe \mid a \in ATOMS\}$$
$$Features(\Sigma) \subseteq \{f: universe \to universe \mid f \in FEATURES\}$$

A **feature graph specification** (fg-specification) $SP = \langle \Sigma, E \rangle$ has a fg-signature Σ and a set of equations over Σ and $\{x\}$.

With the definitions above it shoud be obvious how to transform any feature graph G into a fg-specification $\tau(G)$: The signature of $\tau(G)$ contains all atoms and features occuring in G as constants resp. unary functions, and for any co-referring paths or any path ending with an atomic value there is an equation in $\tau(G)$. Thus, we have a well-defined function

$$\tau: \text{Feature graphs} \to \text{fg-specifications}$$

making available the machinery that has been developed for algebraic specifications.

4. Consistency

In [PS 84] the only inconsistency considered is an equation of the form a = b where a and b are distinct constants. Such a situation is called constant clash in [Pe 87] where additionally a constant / complex clash is considered. Such a clash is "any set of two equations e_1 and e_2 in which e_1 equates some path p to some other path or atomic value and e_2 equates a strict prefix of p to a constant" [Pe 87].

Whereas [PS 84] also consider cyclic feature graphs, Kasper and Rounds ([KR 86]), some PATR-II implementations, and also the STUF formalism described in [Us 86], only allow for acyclic feature graphs. We will show that the absence of cycles in a feature graph can also be expressed as a consistency condition on the corresponding fg-specification.

Below we use the machinery of abstract data types in order to define several notions of semantical consistency for a feature graph.

First we introduce a notation:

A term

$f_n(\ldots(f_1(t))\ldots) \in T_\Sigma(\{x\})$

with

$t \in T_\Sigma(\{x\})_{universe}$, $f_i \in Features(\Sigma)$, and $n \geq 0$ will be written as

$\rho(t)$

where $p = f_n \ldots f_1 \in Features(\Sigma)^*$.

Note that the notation $f_n(\ldots(f_1(t))\ldots)$ reflects the usual mathematical notation for function composition, whereas the path notation employed for feature graphs as sketched in Section 2 uses the reverse order $f_1 \ldots f_n$.

Let SP = <Σ, E> be a fg-specification.
A <Σ, E>-algebra A is

- constant consistent iff
  for all $a, b \in Atoms(\Sigma)$ with $a \neq b$ we have:
  $a_A \neq b_A$

- constant/complex consistent iff
  for all $a, b \in Atoms(\Sigma)$ and all $p1, p2, q \in Features(\Sigma)^+$ there exists an element $o \in A_{soi}$ with:
  $p1_A(i_A(o)) = a_A$
  $\Rightarrow$
  $p2_A(p1_A(i_A(o))) \neq q_A(i_A(o))$
  &
  $p2_A(p1_A(i_A(o))) \neq b_A$

- acyclic iff
  for all $p, q \in Features(\Sigma)^+$ there exists an element $o \in A_{soi}$ with:
  $p_A(q_A(i_A(o))) \neq q_A(i_A(o))$

SP is constant consistent, constant/complex consistent, or acyclic iff there is at least one model of SP having the respective property.

The above definition of consistency of a fg-specification SP suggests that one has to search through the entire class of models of SP in order to determine whether SP is consistent or not. The following theorem shows the power of initial models in the sense of shrinking the search space: only the initial model has to be considered.

Theorem:
The fg-specification SP is constant consistent, constant/complex consistent, or acyclic iff the initial algebra T_{SP} has the respective property.

The above properties can be proven for the initial model by using the deductive closure E* of a set of equations E.

Theorem:
The initial algebra T_{SP} of the fg-specification SP is

- constant consistent iff
  for all $a, b \in Atoms(\Sigma)$ with $a \neq b$
  we have: $a = b \notin E^*$

- constant/complex consistent iff
  for all $a, b \in Atoms(\Sigma)$ and all $p1, p2, q \in Features(\Sigma)^+$
  we have:
  $p1(i(const)) = a \in E^*$
  implies
  $p2(p1(i(const))) = q(i(const)) \notin E^*$
  and
  $p2(p1(i(const))) = b \notin E^*$

- acyclic iff
  for all $p, q \in Features(\Sigma)^+$
  we have:
  $p(q(i(const))) = q(i(const)) \notin E^*$

The equivalences established by these two theorems show us that the consistency of a fg-specification <Σ, E> can be tested by inspecting the deductive closure E* for the absence of certain equations, depending on which consistency aspects one is interested in. Since E* may be too large for performing these tests efficiently it would be desirable to be able to perform the consistency tests on E only. In the next section we develop a completion procedure for the set of equations E which transforms E into a normalized set E′ such that it is sufficient to check E′. The completion procedure thus provides a simple and fast decision procedure for our consistency constraints.

5. The completion procedure

Our completion procedure is a variant of the Knuth-Bendix algorithm ([KB 70]) which is a well-known method for testing properties of rewrite rule systems and for transforming equations into a set of rewrite rules, which then constitute a decision procedure for the equality. In general, there are some problems with the Knuth-Bendix algorithm: It may not terminate, or it may terminate with failure. However, we can show that due to the restricted form of equations these problems do not occur when dealing with fg-specifications.

We first define an order relation $\leq_T$ on the set $T_\Sigma(\{x\})$ of terms over an fg-signature Σ. We assume that ATOMS (resp. FEATURES) is linearly ordered by $\leq_{ATOMS}$ (resp. $\leq_{FEATURES}$). Then we order $T_\Sigma(\{x\})$ using the lexicographic ordering induced by $\leq_{ATOMS}$ and $\leq_{FEATURES}$:

Let $a, b \in ATOMS$, $f_i, g_i \in FEATURES$, and $t \in T_\Sigma(\{x\})$.

- $a <_T b$ if $a <_{ATOMS} b$
- $a <_T f_1(t)$
- $t <_T f_1(t)$
- $f_m(\ldots(f_1(t))\ldots) <_T g_n(\ldots(g_1(t))\ldots)$
  if $f_1 <_{FEATURES} g_1$

$\leq_T$ is the reflexive and transitive closure of $<_T$.

Let SP = <Σ,E> be a fg-specification. We assume that E does not contain any trivial equations of the form t = t (otherwise we can just eliminate such equations from E).

Lemma:
For all $l = r \in E$ we have either $l <_T r$ or $r <_T l$.

Thus, without loss of generality, we assume that $r <_T l$ for all $l = r \in E$ (otherwise we can just exchange the lefthand and the righthand side of the equation). We call E a **directed set of equations** and we may write
$l \to r$ instead of $l = r$.

Completion algorithm CP

Transform E into a directed set of equations and apply successively any of the two following rules until none is applicable any more:

- LHS reduction:

```
If there are two different equations
     (1)  p(l)  ->  r
     (2)     l  ->  r´
in E then:
     - Delete equation (1) from E
     - If r <_T p(r´)
          then add p(r´) -> r to E
     - If p(r´) <_T r
          then add r -> p(r´) to E
[Note: Nothing is added if p(r´) and r
       are identical!]
```

- RHS reduction:

```
If there are two different equations
     (1)  l   ->  p(l´)
     (2)  l´  ->  r
in E then:
     - Delete equation (1) from E
     - If l <_T p(r)
          then add p(r) -> l to E
     - If p(r) <_T l
          then add l -> p(r) to E
[Note: Nothing is added if p(r) and l
       are identical!]
```

where: $r, r´, l, l´ \varepsilon T_\Sigma(\{x\})$, and
$p \varepsilon$ Features$(\Sigma)^*$

Theorem:
For every fg-specification SP the completion procedure CP

- terminates on input SP = $<\Sigma, E>$
- delivers as output a fg-specification SP´ = $<\Sigma, E´>$
- and SP and SP´are equivalent in the sense of
  $E^* = (E´)^*$,
  i. e. E and E´ have the same deductive closure.

This theorem assures that the completion procedure performs only syntactical modifications on the fg-specifications, but does not change their meaning. So we can use SP´ in order to test the consistency constraints of SP. The next theorem shows that stepping from SP to SP´ simplifies this task: instead of inspecting the deductive closure E* it suffices to inspect the set of equations E´.

Theorem:
Let SP = $<\Sigma, E>$ be a fg-specification and SP´ = $<\Sigma, E´>$ be the result of running the completion procedure CP on SP. SP is

- constant consistent iff
  E´ does not contain an equation whose lefthand side is an atom $a \varepsilon$ Atoms(Σ)
- constant/complex consistent iff
  E´ does not contain an equation in which a term f(a) occurs where
  $f \varepsilon$ Features(Σ) and $a \varepsilon$ Atoms(Σ)
- acyclic iff
  E´ does not contain an equation
  p(t) -> t where $p \varepsilon$ Features$(\Sigma)^*$

The proof of this theorem is based on the fact that E´ is a confluent and terminating set of rewrite rules. Since the atoms are smaller than any non-atomic term with respect to the term order $<_T$, for any equation a -> t in E´ with an atom a, t must also be an atom. Therefore, any equation holding between atoms must be contained directly in E´, implying the constant consistency property of the theorem. The other parts of the theorem follow from similar observations.

6. Unification

In Section 2 we presented three different notations for feature graphs, and in 3.2. we introduced a translation τ from feature graphs to fg-specifications. On the other hand, it is straightforward to transform a fg-specification SP into a feature graph G: The atoms and features of G are those occuring in the signature of SP and the equations of E reflect the coreferring paths resp. paths ending with an atomic value in G. We denote this transformation by
   τ^{-1}: fg-specifications -> feature graphs

Although τ^{-1}(SP) and τ^{-1}(CP(SP)) may be syntactically different since CP(SP)) contains equations in a reduced normal form, the two graphs are equivalent in the sense of feature graph unification: they are unifiable with exactly the same feature graphs. Besides giving a basis for a simple consistency test and providing a normal form presentation for fg-specifications the completion procedure CP also provides the basis for a precise mathematical definition of feature graph unification. This is true regardless which consistency concept for feature graphs one wants to apply, e.g. if one wants to allow cyclic graphs or only acyclic ones. Thus, let **X-consistent** be either "constant consistent", "constant/complex consistent", "acyclic", or any combination thereof. Let G_1 and G_2 be feature graphs.

```
graph-unify(G_1 ,G_2 ) =
  let (Σ_1 ,E_1 ) = τ(G_1 ) in
  let (Σ,E) = CP(Σ_1 u Σ_2 , E_1 u E_2 ) in
    τ^-1 (Σ,E)    if (Σ,E) is X-consistent
    fail          if (Σ,E) is not X-consistent
```

7. Conclusions

We have presented a mathematical semantics of feature graphs and feature graph unification in terms of ADT specifications. It supports various consistency concepts used for feature graphs. The important notion of partiality ([Pe 87]) in the sense that arbitrary new features may be unified into a feature graph is supported since any feature graph specification can be extended by arbitrary features, atoms, and equations; there exists no ´largest´ feature graph specification (unless of course, one adds an artificial ´largest´ element, e.g. as the F-specification in the STUF formalism as described in [BPU 88]).

Another approach bringing together initial ADT specifications and feature graphs is given in [SA 87]. It uses an order-sorted approach, where the set of atoms and features must be fixed in advance, and where every element of a supersort must be in one of its subsorts. Compared to the order-sorted approach of [SA 87] a drawback of the work presented here is the asymetric treatment of the root of a feature graph (giving rise to the 'soi' sort) and the other nodes (being mapped to the 'universe' sort). We are currently extending out work in order to overcome this disadvantage ([BP 88]). Other areas of future work are the treatment of disjunctions and of functional uncertainty ([KR 86], [Jo 86]).

References

[Ai 84] Ait-Kaci, H.: A Lattice Theoretic Approach to Computation based on a Calculus of Partially Ordered Type Structures. PhD thesis, University of Pennsylvania, 1984

[BP 88] Beierle, C., Pletat, U.: The Algebra of Feature Graph Specifications. (in preparation)

[BPU 88] Beierle, C., Pletat, U., Uszkoreit, H.: An Algebraic Characterization of STUF. Proc. Symposium "Computerlinguistik und ihre theoretischen Grundlagen", Saarbrücken 1988.

[EM 85] Ehrig, H. and Mahr, B.: Foundations of Algebraic Specification 1. Springer Verlag, Berlin 1985.

[GM 87] Goguen, J. G. and Meseguer, J.: Order-Sorted Algebra I: Partial and Overloaded Operators, Errors and Inheritance. Computer Science Lab., SRI International, 1987.

[GTW 78] Goguen, J. A. and Thatcher, J. W. and Wagner, E.: An Initial Algebra Approach to The Specification, Correctness and Implementation of Abstract Data Types. In: Current Trends in Programming Methodology, R. T. Yeh, (ed), Prentice-Hall, 1978.

[Jo 86] Johnson, M.: Computing with regular path formulas. Draft, 1986.

[Jo 87] Johnson, M.: Attribute-Value Logic and Theory of Grammar. PhD Thesis, Stanford University, 1987.

[KB 70] Knuth, D.E., Bendix, P.B.: Simple Word Problems in Universal Algebra. In: J. Leech (Ed.): Computational problems in Universal Algebra. Pergamon Press, 1970.

[KR 86] Kasper, R.T., Rounds, W.C.: A logical semantics for feature structures. Proc. 24th Annual Meeting, ACL, 1986.

[Pe 87] Pereira, F.: Grammars and Logics of Partial Information. Proc. 4th Int. Conf on Logic Programming, May 1987.

[PS 84] Pereira, F., Shieber, S.M.: The semantics of grammar formalisms seen as computer languages. Proc. COLING-84, ACL, 1984.

[RK 86] Rounds, W.C., Kasper, R.: A complete logical calculus for record structures representing linguistic information. IEEE Symposium on Logic in Computer Science, 1986.

[SA 87] Smolka, G., Ait-Kaci, H.: Inheritance Hierarchies: Semantics and Unification. MCC Technical Report AI-057-87, 1987.

[Us 86] Uszkoreit, H.: Syntaktische and semantische Generalisierungen im strukturierten Lexikon. Proc. GWAI-86, (eds. C.R. Rollinger, W. Horn), Springer Verlag 1986.