Algorithms for Type Inference with Coercions

Andreas Weber*
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen
72076 Tübingen, Germany
E-mail: weber@informatik.uni-tuebingen.de

Abstract

This paper presents algorithms that perform a type inference for a type system occurring in the context of computer algebra. The type system permits various classes of coercions between types and the algorithms are complete for the precisely defined system, which can be seen as a formal description of an important subset of the type system supported by the computer algebra program AXIOM.

Previously only algorithms for much more restricted cases of coercions have been described or the frameworks used have been so general that the corresponding type inference problems were known to be undecidable.

1 Introduction

Type systems are now widely recognized to be of central importance for the design of a symbolic computation system (see e. g. [16], [2], [3], [13], [17], [6], [21], [22] to mention only some of the more recent papers on the topic). Especially the type system of AXIOM [7] is of growing influence (some recent systems whose type system is strongly influenced by it are GAUSS [15] and WEYL [24]).

In order to make an AXIOM like type system feasible for a user it is necessary that a system performs an automatic type inference which resolves the overloading of the operators and automatically inserts coercion functions between appropriate types.

Ideally a user should be able to write down an expression like

$$\mathtt{t} - \begin{pmatrix} 1 & 0 \\ 3 & \frac{1}{2} \end{pmatrix}$$

which — as a mathematician would conclude — denotes a 2 × 2-matrix over $\mathbb{Q}[\mathtt{t}]$ where the t in the expression is the usual shorthand for t times the identity matrix. Then the system should perform a type inference for the expression which resolves the overloaded operators and automatically inserts coercion functions where necessary.

In an AXIOM like type system, this expression involves the following types and type constructors: The integral domain I of integers, the unary type constructor QF which forms the quotient field of an integral domain, the binary type constructor UP which forms the ring of univariate polynomials over some ring in a specified indeterminate, and the type constructor $\mathrm{M}_{2,2}$ building the 2 × 2-matrices over a commutative ring.

In order to type this expression correctly several coercion functions have to be used, cf. [22].

Some heuristics for type inference with coercions are implemented in AXIOM [20, 4], but these heuristics do not correctly handle such a complex example as the one given above[1] and they are "incomplete", i. e. the system does not find all of the possible coercions — if it finds any at all!

Some formal systems of coercions between types in a computer algebra context are described in [17, 4, 2]. However, the systems considered in these papers are either much more restrictive than the one we will consider [17], or they are so general that the type inference problem becomes undecidable [4, 2].

We will present some algorithms for type inference which are complete for a class of coercions covering many cases of coercions occurring in the context of computer algebra — e. g. they can handle the example given above. The classes of coercions we consider are semantically investigated in [22] which we refer to for more details and examples. In [22] we have also shown that our system of coercions is more extensive than the ones which were investigated in the context of functional programming [8, 14, 5, 10].

Although our system is more general in one respect than the one given in [2] it is more restrictive in another one (cf. Sec. 4.2). Thus the presented algorithms are also a solution for the open problem stated in [2] namely finding restrictions on the coercions which yield a decidable type inference problem.

2 Preliminaries

An *order-sorted signature* is a triple $(S, \leq, \Sigma)$, where S is a set of sorts, $\leq$ a partial order on S, and Σ a family $\{\Sigma_{\omega,\sigma} \mid \omega \in S^*, \sigma \in S\}$ of not necessarily disjoint sets of operator symbols.

For notational convenience, we will often write $f : (\omega)\sigma$ instead of $f \in \Sigma_{\omega,\sigma}$; $(\omega)\sigma$ is called an *arity* and $f : (\omega)\sigma$ a *declaration*. The signature $(S, \leq, \Sigma)$ is often identified with Σ. If $|\omega| = n$ then f is called a n-ary operator symbol. 0-ary operator symbols are *constant symbols*.

As in [19] we will assume in the following that for any f there is only a single $n \in \mathbb{N}$ such that f is a n-ary operator symbol.

A σ-sorted variable set is a family $V = \{V_\sigma \mid \sigma \in S\}$ of disjoint, nonempty sets. For $x \in V_\sigma$ we also write $x : \sigma$ or x_σ.

*Supported by the *Deutsche Forschungsgemeinschaft*, grant Lo 231/5-1.

[1] AXIOM attributes a type
`Polynomial SquareMatrix(2,Fraction Integer)`
to this expression instead of
`SquareMatrix(2, Polynomial Fraction Integer).`
As a consequence it cannot compute its determinant!

The set of *order-sorted terms* of sort σ freely generated by V, $T_\Sigma(V)_\sigma$, is the least set satisfying

- if $x \in V_{\sigma'}$ and $\sigma' \leq \sigma$, then $x \in T_\Sigma(V)_\sigma$
- if $f \in \Sigma_{\omega,\sigma'}$, $\omega = \sigma_1 \cdots \sigma_n$, $\sigma' \leq \sigma$ and $t_i \in T_\Sigma(V)_{\sigma_i}$ then $f(t_1, \ldots, t_n) \in T_\Sigma(V)_\sigma$.

The set of all order-sorted terms over Σ freely generated by V will be denoted by

$$T_\Sigma(V) := \bigcup_{\sigma \in S} T_\Sigma(V)_\sigma.$$

The set of all *ground terms* over Σ is $T_\Sigma := T_\Sigma(\{\})$. If $t \in T_\Sigma(V)_\sigma$ we will also write $t : \sigma$.

A signature is *regular*, if each term $t \in T_\Sigma(V)$ has a least sort.

The *complexity* of a term $t \in T_\Sigma(V)$, com(t) is inductively defined as follows:

- $\text{com}(t) = 1$, if $t \in V_\sigma$ or $t \in \Sigma_{\epsilon,\sigma}$ for some $\sigma \in S$,
- if $f \in \Sigma_{\omega,\sigma'}$, $\omega = \sigma_1 \cdots \sigma_n$, and $t_i \in T_\Sigma(V)_{\sigma_i}$ then

$$\text{com}(f(t_1, \ldots, t_n)) = \max(\text{com}(t_1), \ldots, \text{com}(t_n)) + 1.$$

3 The Language of Types

As in [22, 2], a *type* will just be an element of the set of all order-sorted terms over a regular signature $(S, \leq, \Sigma)$ freely generated by some family of infinite sets $V = \{V_\sigma \mid \sigma \in S\}$.

For more details and examples we refer to [22].

This formalism is well suited to express the subset of the type system of AXIOM [7], in which only non-parameterized categories[2] are considered and the properties correspond to the non-parameterized categories.

3.1 Coercions

The classes of coercions we will consider are generally the same as in [22]. We thus refer to [22] for an investigation of semantic properties of the system and for more examples.

We will write $t_1 \trianglelefteq t_2$ if there is a coercion $\phi : t_1 \longrightarrow t_2$.

3.2 General Conditions

If $\phi : t_1 \longrightarrow t_2$ and $\psi : t_2 \longrightarrow t_3$ are coercion functions then the composition $\psi \circ \phi$ will be a coercion. Thus the relation generated by $\trianglelefteq$ will be transitive. It will be convenient to define that the identity on a type is a coercion.

3.2.1 Coercions between Base Types

We will assume that the coercions between base types are effectively given, i. e. that we can decide for two base types t_1 and t_2 whether $t_1 \trianglelefteq t_2$ or not.

3.2.2 Structural Coercions

Definition 1. The n-ary type constructor ($n \geq 1$) f induces a *structural coercion*, if there are sets $\mathcal{A}_f \subseteq \{1, \ldots, n\}$ and $\mathcal{M}_f \subseteq \{1, \ldots, n\}$ such that the following condition is satisfied:

Whenever there are declarations $f : (\sigma_1 \cdots \sigma_n)\sigma$ and $f : (\sigma'_1 \cdots \sigma'_n)\sigma'$ and ground types $t_1 : \sigma_1, \ldots, t_n : \sigma_n$ and $t'_1 : \sigma'_1, \ldots, t'_n : \sigma'_n$ such that $t_i = t'_i$ if $i \notin \mathcal{A}_f \cup \mathcal{M}_f$ and there are coercions

$$\begin{aligned} &\phi_i : t_i \longrightarrow t'_i, && \text{if } i \in \mathcal{M}_f,\\ &\phi_i : t'_i \longrightarrow t_i, && \text{if } i \in \mathcal{A}_f,\\ &\phi_i = \text{id}_{t_i} = \text{id}_{t'_i}, && \text{if } i \notin \mathcal{A}_f \cup \mathcal{M}_f,\end{aligned}$$

then there is a *uniquely defined* coercion

$$\begin{aligned}&\mathcal{F}_f(t_1, \ldots, t_n, t'_1, \ldots, t'_n, \phi_1, \ldots, \phi_n) :\\ &\qquad f(t_1, \ldots, t_n) \longrightarrow f(t'_1, \ldots, t'_n).\end{aligned}$$

The type constructor f is *covariant in its i-th argument*, if $i \in \mathcal{M}_f$. It is *contravariant in its i-th argument*, if $i \in \mathcal{A}_f$.

Instead of the adjective "covariant" we will sometimes use the adjective "monotonic", and instead of "contravariant" we will sometimes use "antimonotonic", because both terminologies are used in the literature and reflect different intuitions which are useful in different contexts.

We refer to [22] for examples of type constructors which induce a structural coercion.

Although many important type constructors arising in computer algebra are covariant in all arguments it is not justified to assume that this property will always hold as was done in [2]. For instance the type constructor for building "function spaces" is contravariant in its first argument, cf. [22, 1]. Constructions like the fixpoint field of a certain algebraic extension of $\mathbb{Q}$ under a group of automorphisms in Galois theory — see e. g. [9] — would give other — more algebraic examples — of type constructors which are contravariant.[3]

Some examples of type constructors which are neither monotonic nor antimonotonic are given e. g. in [8] or in [23].

3.2.3 Direct Embeddings

Definition 2. Let $f : (\sigma_1 \cdots \sigma_n)\sigma$ be a n-ary type constructor. If for all ground types $t_1 : \sigma_1, \ldots, t_n : \sigma_n$ there is a coercion function $\Phi^i_{f,t_1,\ldots,t_n} : t_i \longrightarrow f(t_1, \ldots, t_n)$, then we say that *$f$ has a direct embedding at its i-th position.*

Moreover, let

$$\mathcal{D}_f = \{i \mid f \text{ has a direct embedding at its } i\text{-th position}\}$$

be the *set of direct embedding positions of f.*

Examples of type constructors having direct embeddings can be found in [22].

Remark. The definition for direct embeddings given above is slightly more restrictive than the one given in [22] by requiring an appropriate coercion for *all* possible ground types into the parameterized type and not only for *some* ground types. However, this additional requirement — which is in general necessary for an algorithmic type inference — is reasonable for all examples we know.

4 Algorithms for Type Inference with Coercions

In the following section we will restrict the types to the ones which can be expressed as terms of a finite order-sorted signature. As is shown in [23] we can also assume that the signature is regular.

Let op be a n-ary operation. We will assume that op is given a *profile* of the form

$$\text{op} : \xi_1 \times \cdots \times \xi_n \longrightarrow \xi_{n+1},$$

[2]Category in the sense of the AXIOM language, not in the sense of category theory!

[3]In the group theory program GAP [18] such constructs are implemented as functions and not as type constructors. For a discussion of this point we refer to [23, Chap. 3.6.1].

$\mathcal{S} \leftarrow \mathsf{CSGT}(t).$

[Sorts of types a type t is coercible to. $\mathcal{S}$ is the set of sorts of types in which t can be coerced to. Assumes that the signature is finite, only direct embeddings and structural coercions are present.]

```
(1) [t base type.] if $\mathrm{com}(t) = 0$ then $\{\, \mathcal{S} \leftarrow \mathsf{CSBT}(t)$; return$\}$.
(2) [Recurse.] Let $t = g(t_1, \ldots, t_m)$; for $i = 1, \ldots, m$ do $\mathcal{S}_i \leftarrow \mathsf{CSGT}(t_i)$; for $(\sigma_1, \ldots, \sigma_m) \in \mathcal{S}_1 \times \cdots \times \mathcal{S}_m$ do $\{$ if there is
    $g : (\sigma_1 \cdots \sigma_m)\overline{\sigma}$ such that $\overline{\sigma} \notin \mathcal{S}$ then $\{\, \mathcal{T} \leftarrow \mathcal{T} \cup \{g(v_{\sigma_1}, \ldots, v_{\sigma_m})\}; \mathcal{S} \leftarrow \mathcal{S} \cup \{\overline{\sigma}\}; \mathcal{S}' \leftarrow \mathcal{S}; \mathcal{T}' \leftarrow \mathcal{T} \,\}\,\}$.
(3) [Compute Direct Embeddings.] for $\overline{t} \in \mathcal{T}$ do $\{$ if there are $\overline{\sigma}, f : (\sigma_1 \cdots \sigma_n)\sigma', i \in \{1, \ldots, n\}$ such that $\overline{t} : \overline{\sigma}$ and $\sigma_i = \overline{\sigma}$ and
    $i \in \mathcal{D}_f$ and $\sigma' \notin \mathcal{S}$ then $\{\, \mathcal{S}' \leftarrow \mathcal{S}' \cup \{\sigma'\}; \mathcal{T}' \leftarrow \mathcal{T}' \cup \{f(v_{\sigma_1}, \ldots, v_{\sigma_n})\} \,\}\,\}$.
(4) [Iterate if something is added.] if $\mathcal{S}' \neq \mathcal{S}$ then $\{\, \mathcal{S} \leftarrow \mathcal{S}'; \mathcal{T} \leftarrow \mathcal{T}'$; goto (3) $\}$.
```

where

$\mathcal{S} \leftarrow \mathsf{CSBT}(t).$

[Sorts of types a base type t is coercible to. $\mathcal{S}$ is the set of sorts of types in which t can be coerced to. Assumes that the signature is finite, only direct embeddings and structural coercions are present.]

```
(1) [Initialize.] $\mathcal{T} \leftarrow \{t' \mid t \trianglelefteq t'$ and $t'$ is a base type$\}$; $\mathcal{S} \leftarrow \{\sigma' \mid t : \sigma'\}$; $\mathcal{S}' \leftarrow \mathcal{S}$; $\mathcal{T}' \leftarrow \mathcal{T}$.
(2) [Compute Direct Embeddings.] for $\overline{t} \in \mathcal{T}$ do $\{$ if there are $\overline{\sigma}, f : (\sigma_1 \cdots \sigma_n)\sigma', i \in \{1, \ldots, n\}$ such that $\overline{t} : \overline{\sigma}$ and $\sigma_i = \overline{\sigma}$ and
    $i \in \mathcal{D}_f$ and $\sigma' \notin \mathcal{S}$ then $\{\, \mathcal{S}' \leftarrow \mathcal{S}' \cup \{\sigma'\}; \mathcal{T}' \leftarrow \mathcal{T}' \cup \{f(v_{\sigma_1}, \ldots, v_{\sigma_n})\} \,\}\,\}$.
(3) [Iterate if something is added.] if $\mathcal{S}' \neq \mathcal{S}$ then $\{\, \mathcal{S} \leftarrow \mathcal{S}'; \mathcal{T} \leftarrow \mathcal{T}'$; goto (2) $\}$.
```

Figure 1: Algorithms computing sorts of types a given type can be coerced to

where ξ_i, $1 \leq i \leq n+1$, is either a type variable v_{τ_l}, $l \leq k$, or a ground type $\overline{t}_i$. Given objects $o_1, \ldots, o_n$ having types $t_1, \ldots, t_n$ respectively, the expression

$$\mathrm{op}(o_1, \ldots, o_n)$$

will be well typed having type ξ_{n+1} iff the following conditions are satisfied.

1. If $\xi = \overline{t}_i$ for some ground type $\overline{t}_i$ then $t_i \trianglelefteq \overline{t}_i$.
2. If $\xi_i = \xi_j = v_{\tau_l}$ for some $i \neq j$ then there is a type $t : \tau_l$ such that $t_i \trianglelefteq t$ and $t_j \trianglelefteq t$.
3. If $\xi_i = v_{\tau_k}$ then there is a type $t : \tau_k$ such that $t_i \trianglelefteq t$.

Notice that if we require that all objects have ground types — if they have a type at all — then algorithms solving the problems imposed by the above conditions can be used to solve the type inference problem using a bottom-up process.[4]

If we do not restrict the possible coercions then determining whether for given types t_1 and t_2 there is a type t such that $t_1 \trianglelefteq t$ and $t_2 \trianglelefteq t$ might be an undecidable problem, cf. [2].

In the following we will restrict the possible coercions to coercions between base types,[5] direct embeddings and structural coercions.

[4]Similar ideas can be found in [2, Sec. 4] and in [16].

[5]By the assumption of a finite signature there are only finitely many base types and we will assume that the finitely many coercions between base types are effectively given.

4.1 Computing Properties of Coercible-to Types

Proposition 3. *Assume that the types are terms of a finite, regular order-sorted signature and that there are only coercions between base types, direct embeddings and structural coercions. Then for any type t, the set*

$$\mathcal{S}_t = \{\sigma \mid \exists t' . t' : \sigma \text{ and } t \trianglelefteq t'\}$$

is effectively computable.

Proof. We claim that the set $\mathcal{S}_t$ will be computed by the algorithm $\mathsf{CSGT}(t)$ (see Fig. 1).

All computations which are used in CSGT and CSBT can be performed effectively. Since the signature is finite there are always only finitely many possibilities which have to be checked in the existential clauses of the algorithms and so algorithm CSBT will terminate and so will CSGT. Algorithm CSGT is correct (i. e. $\mathsf{CSGT}(t) \subseteq \mathcal{S}_t$), because only types and the sort of types t can be coerced to are computed. Its completeness (i. e. $\mathsf{CSGT}(t) \supseteq \mathcal{S}_t$) follows from the fact that structural coercions cannot add new sorts to $\mathcal{S}_t$. □

4.2 Common Upper Bounds

In the following we will rule out antimonotonic structural coercions, i. e. we will require that $\mathcal{A}_f = \emptyset$ for all type constructors f.

Notice that the restriction $\mathcal{A}_f = \emptyset$ does not exclude type constructors like the constructor FS building the space of functions from the framework.[6] Only the automatic insertion of a coercion

[6]See [22] for a discussion of the properties of FS with respect to coercions.

```
$\mathcal{U} \leftarrow \mathsf{CSMUBGT}(t_1, t_2)$

[$\mathcal{U}$ is a complete set of minimal upper bounds of two types $t_1$ and $t_2$. Requires that only direct embeddings and structural coercions are used, $|\mathcal{D}_f| \leq 1$ and $\mathcal{A}_f = \emptyset$ for any type constructor $f$. Assumes that algorithm CSMUBBT returns a finite set.]

(1) [$t_1$ and $t_2$ base types.] if $\mathrm{com}(t_1) = 1$ and $\mathrm{com}(t_2) = 1$ then $\{ \mathcal{U} \leftarrow \mathsf{CSMUBBT}(t_1, t_2)$; return $\}$.

(2) [Ensure that $\mathrm{com}(t_1) \leq \mathrm{com}(t_2)$.] if $\mathrm{com}(t_1) > \mathrm{com}(t_2)$ then $\{ h \leftarrow t_1; t_1 \leftarrow t_2; t_2 \leftarrow h \}$.

(3) [$t_1$ a base type.] if $\mathrm{com}(t_1) = 1$ then $\{$ let $t_2 = f(t_2^1, \ldots, t_2^n)$; if $|\mathcal{D}_f| = 0$ then $\{ \mathcal{U} \leftarrow \emptyset$; return$\}$; let $\mathcal{D}_f = \{i\}$;
    $\mathcal{U}' \leftarrow \mathsf{CSMUBGT}(t_1, t_2^i)$;

    (3.1) if $\mathcal{U}' = \emptyset$ then $\{ \mathcal{U} \leftarrow \emptyset$; return$\}$;

    (3.2) if $\mathcal{U}' \neq \emptyset$ then $\{$ if $i \in \mathcal{M}_f$ then $\{ \mathcal{U} \leftarrow \emptyset$; for $t' \in \mathcal{U}'$ do $\mathcal{U} \leftarrow \mathcal{U} \cup \{f(t_2^1, \ldots, t_2^{i-1}, t', t_2^{i+1}, \ldots, t_2^n)\} \}$; if $i \notin \mathcal{M}_f$
          then $\{$ if $t_2^i \in \mathcal{U}'$ then $\mathcal{U} \leftarrow \{t_2\}$ else $\mathcal{U} \leftarrow \emptyset\}$ return $\} \}$.

(4) [General case.] let $t_1 = g(t_1^1, \ldots, t_1^m)$; let $t_2 = f(t_2^1, \ldots, t_2^n)$; $\mathcal{U} \leftarrow \emptyset$.

(5) [Structural coercions.] if $f = g$ then $\{$ for $i \in \mathcal{M}_f$ do $\mathcal{U}_i \leftarrow \mathsf{CSMUBGT}(t_1^i, t_2^i)$; let $\mathcal{M}_f = \{j_1, \ldots, j_l\}$; if $t_1^k = t_2^k$
    for all $k \in \{1, \ldots, n\} - \mathcal{M}_f$ then $\{$ for $(t'_{j_1}, \ldots, t'_{j_l}) \in \mathcal{U}_{j_1} \times \cdots \times \mathcal{U}_{j_l}$ do $\{$ for $k \in \{1, \ldots, n\} - \mathcal{M}_f$ do $t'_k \leftarrow t_1^k$;
    $\mathcal{U} \leftarrow \mathcal{U} \cup \{f(t'_1, \ldots, t'_n)\} \} \} \}$.

(6) [Direct embeddings in $g$.] if $|\mathcal{D}_g| = 1$ then $\{$ let $\mathcal{D}_g = \{i\}$; $\mathcal{U}' \leftarrow \mathsf{CSMUBGT}(t_1^i, t_2)$; if $\mathcal{U}' \neq \emptyset$ then $\{$ if $i \in \mathcal{M}_g$ then $\{$
    for $t' \in \mathcal{U}'$ do $\mathcal{U} \leftarrow \mathcal{U} \cup \{g(t_2^1, \ldots, t_2^{i-1}, t', t_2^{i+1}, \ldots, t_2^m)\} \}$; if $i \notin \mathcal{M}_g$ and $t_1^i \in \mathcal{U}'$ then $\mathcal{U} \leftarrow \mathcal{U} \cup \{t_1\} \} \}$.

(7) [Direct embeddings in $f$.] if $|\mathcal{D}_f| = 1$ then $\{$ let $\mathcal{D}_f = \{i\}$; $\mathcal{U}' \leftarrow \mathsf{CSMUBGT}(t_1, t_2^i)$; if $\mathcal{U}' \neq \emptyset$ then $\{$ if $i \in \mathcal{M}_f$ then $\{$
    for $t' \in \mathcal{U}'$ do $\mathcal{U} \leftarrow \mathcal{U} \cup \{f(t_2^1, \ldots, t_2^{i-1}, t', t_2^{i+1}, \ldots, t_2^n)\} \}$; if $i \notin \mathcal{M}_f$ and $t_2^i \in \mathcal{U}'$ then $\mathcal{U} \leftarrow \mathcal{U} \cup \{t_2\} \} \}$.
```

Figure 2: An algorithm computing a complete set of minimal upper bounds

giving rise to the antimonotony is excluded. For instance, instead of having FS as a type constructor which is antimonotonic in its first argument and monotonic in its second, it is one which is only monotonic in its second argument. Such a restriction does not seem to cause a loss of too much expressiveness. This is an important difference to the system in [2], in which all type constructors have to be monotonic in all arguments. Type constructors which are antimonotonic in some argument have to be excluded from that system in general, because it is not possible that a type constructor being antimonotonic in some argument can be made monotonic in that argument without changing the intended meaning of the type constructor. Thus our framework is more general in this respect than the one in [2]. However, direct embeddings are a special form of the "rewrite relations" for coercion considered in that paper.

So the following can be seen as a solution of one of the open problems stated in [2], namely finding restrictions on the system of coercions which will yield a decidable type inference problem.

Definition 4. If for two types t_1 and t_2 there is a type t such that $t_1 \trianglelefteq t$ and $t_2 \trianglelefteq t$ then t is called a *common upper bound* of t_1 and t_2.

A *minimal upper bound* $\mathrm{mub}(t_1, t_2)$ of two types t_1 and t_2 is a type t satisfying the following conditions.

1. The type t is a common upper bound of t_1 and t_2.
2. If t' is a type which is a common upper bound of t_1 and t_2 such that $t' \trianglelefteq t$, then $t \trianglelefteq t'$.

A *complete set of minimal upper bounds* for two types t_1 and t_2 is a set $\mathrm{CSMUB}(t_1, t_2)$ such that

1. all $t \in \mathrm{CSMUB}(t_1, t_2)$ are a minimal common upper bound of t_1 and t_2, and
2. for every type t' which is a common upper bound of t_1 and t_2 there is a $t \in \mathrm{CSMUB}(t_1, t_2)$ such that $t \trianglelefteq t'$.

If two types t_1 and t_2 have no minimal upper bound then the complete sets of minimal upper bounds are all empty. In this case we will write $\mathrm{CSMUB}(t_1, t_2) = \emptyset$. We will write $|\mathrm{CSMUB}(t_1, t_2)|$ to denote the smallest cardinality of a complete set of minimal upper bounds of t_1 and t_2.

If the partial order induced by the relation $\trianglelefteq$ is a quasi-lattice then $|\mathrm{CSMUB}(t_1, t_2)| \leq 1$ for all types t_1 and t_2. However, in [23, Chap. 4.5] it is shown that this partial order will not be a quasi-lattice in general.

In the following we will assume that for any two *base types* t_1^{b} and t_2^{b} a *finite* complete set of minimal upper bounds can be computed effectively, say by $\mathsf{CSMUBBT}(t_1^{\mathrm{b}}, t_2^{\mathrm{b}})$. We will give an algorithm computing for any two types t_1 and t_2 a complete set of minimal upper bounds and will show that this set is finite.

Theorem 5. *Assume that all coercions are coercions between base types, direct embeddings and structural coercions. Moreover, assume that for all type constructors f there is at most one direct embedding position, i. e. $|\mathcal{D}_f| \leq 1$, and no antimonotonic coercions are present, i. e. $\mathcal{A}_f = \emptyset$, and for any base types t_1^{b} and t_2^{b} there is a finite complete set of minimal upper bounds with respect to the set of base types which can be effectively computed by a function $\mathsf{CSMUBBT}(t_1^{\mathrm{b}}, t_2^{\mathrm{b}})$.*

Then for any two types t_1 and t_2 there is a finite complete set of minimal upper bounds which can be effectively computed.

Proof. We claim that algorithm $\mathsf{CSMUBGT}$ (see Fig. 2) terminates for any input parameters t_1 and t_2 and computes a complete set of minimal upper bounds which is finite.

This claim can be proved by induction on the complexity of t_1 and t_2 along the steps of the algorithm. For the details of the proof we refer to [23, Chap. 4.7.1]. □

Remark. Since algorithm CSMUBGT uses the type constructors given by its arguments and does not have to perform a search on all type constructors, it is not necessary that the signature is finite. It is only necessary that there is an effective algorithm which computes for any type constructor f the sets $\mathcal{D}_f$ and $\mathcal{M}_f$, and that the conditions imposed on algorithm CSMUBBT are fulfilled.[7]

An example of an infinite signature with such properties is a finite signature extended with a type constructor $\mathrm{M}_{m,n}$ for any $m, n \in \mathbb{N}$ with the intended meaning of building the $m \times n$-matrices over commutative rings. It is natural to define $\mathcal{M}_{\mathrm{M}_{m,n}} = \{1\}$ for all $m, n \in \mathbb{N}$ and to have $\mathcal{D}_{\mathrm{M}_{m,n}} = \emptyset$ for $m \neq n$ and $\mathcal{D}_{\mathrm{M}_{n,n}} = \{1\}$ for any $n \in \mathbb{N}$.

4.3 Solving the Type Inference Problem

Combining algorithms CSGT (see Fig. 1) and CSMUBGT (see Fig. 2) we can solve the subproblems 1–3 stated at the beginning of Sec. 4.

So if the conditions stated in Prop. 3 and Theorem 5 on the coercions are fulfilled and all 0-ary operations of the object lang uage — i. e. the simple objects — have ground types then it is possible to decide for any expression if it is typeable with a ground type. Moreover, in the positive case it is possible to compute a finite complete set of minimal types for the expression.

4.4 Some remarks on the practical and theoretical complexity of the algorithms

The presented algorithms seem to be well suited for a practical implementation.

An inspection of the examples provided by AXIOM suggests that the iteration step (4) in algorithm CSGT (see Fig. 1) is seldom needed, which favourably affects the running time of the algorithm.

It is possible to construct examples of type constructors having structural coercions and direct embeddings at certain positions so that the cardinality of a complete set of minimal upper bounds of two types t_1 and t_2 is exponential in $\min(\mathrm{com}(t_1), \mathrm{com}(t_2))$.[8] Thus the space complexity[9] of algorithm CSMUBGT (see Fig. 2) can be exponential. However, such examples seem to be of little practical significance. Moreover, in a large practical system such as AXIOM there are usually many type constructors but nevertheless the complexity[10] of the occurring types is usually quite small. Since the number of occurring type constructors does not affect the running time of the presented algorithms[11] it seems to be possible to use these algorithms for type inference in a large system.

The possibility to extend the algorithms to handle infinite families of type constructors — as we have shown above — seems to be quite important with respect to their use in a practical system.

Acknowledgements. The present paper is part of the author's PhD-thesis [23] written under the supervision of Prof. R. Loos at the University of Tübingen. I would like to thank Prof. Loos for initiating and supervising my research.

I am grateful to the anonymous referees for their suggestions which helped to clarify the presentation and to M. Ehrengard for checking the style of the manuscript.

[7] If the signature is finite, these conditions will always be fulfilled if the coercions between the base types are effectively given.

[8] The key step for such a construction can be found in [23, Chap. 4.5].

[9] Here we use the term *complexity* with its complexity theoretic content.

[10] Recall Sec. 2 for the definiton of *complexity of a type*.

[11] Using a suitable representation for type constructors the access operations computing $\mathcal{M}_f$ and $\mathcal{D}_f$ for a given type constructor f take constant time.

Currently the presented algorithms are being implemented as part of the system for recognition of handwritten formulas [11] which is supported by the *Deutsche Forschungsgemeinschaft.*

References

[1] Cardelli, L. A semantics of multiple inheritance. *Information and Computation 76* (1988), 138–164.

[2] Comon, H., Lugiez, D., and Schnoebelen, P. A rewrite-based type discipline for a subset of computer algebra. *Journal of Symbolic Computation 11* (1991), 349–368.

[3] Dalmas, S. A polymorphic functional language applied to symbolic computation. In *Proc. Symposium on Symbolic and Algebraic Computation (ISSAC '92)* (Berkeley, CA, July 1992), P. S. Wang, Ed., Association for Computing Machinery, pp. 369–375.

[4] Fortenbacher, A. Efficient type inference and coercion in computer algebra. In *Design and Implementation of Symbolic Computation Systems (DISCO '90)* (Capri, Italy, Apr. 1990), A. Miola, Ed., vol. 429 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 56–60.

[5] Fuh, Y.-C., and Mishra, P. Type inference with subtypes. *Theoretical Computer Science 73* (1990), 155–175.

[6] Hearn, A. C., and Schrüfer, E. An order-sorted approach to algebraic computation. In Miola [12], pp. 134–144.

[7] Jenks, R. D., and Sutor, R. S. *AXIOM: The Scientific Computation System.* Springer-Verlag, New York, 1992.

[8] Kaes, S. Type inference in the presence of overloading, subtyping, and recursive types. In *Proceedings of the 1992 ACM Conference on Lisp and Functional Programming* (San Francisco, CA, June 1992), Association for Computing Machinery, pp. 193–205.

[9] Lang, S. *Algebra.* Addison-Wesley, Reading, MA, 1971.

[10] Lincoln, P., and Mitchel, J. C. Algorithmic aspects of type inference with subtypes. In *Conference Record of the Nineteenth Annual ACM Symposium on Principles of Programming Languages* (Albuquerque, New Mexico, Jan. 1992), Association for Computing Machinery, pp. 293–304.

[11] Marzinkewitsch, R. Operating computer algebra systems by handprinted input. In *Proc. Symposium on Symbolic and Algebraic Computation (ISSAC '91)* (Bonn, Germany, July 1991), S. M. Watt, Ed., Association for Computing Machinery, pp. 411–413.

[12] Miola, A., Ed. *Design and Implementation of Symbolic Computation Systems — International Symposium DISCO '93* (Gmunden, Austria, Sept. 1993), vol. 722 of *Lecture Notes in Computer Science*, Springer-Verlag.

[13] Missura, S. A. Extending AlgBench with a type system. In Miola [12], pp. 359–363.

[14] Mitchell, J. C. Type inference with simple subtypes. *Journal of Functional Programming 1*, 3 (July 1991), 245–285.

[15] Monagan, M. B. Gauss: a parameterized domain of computation system with support for signature functions. In Miola [12], pp. 81–94.

[16] RECTOR, D. L. Semantics in algebraic computation. In *Computers and Mathematics* (Massachusetts Institute of Technology, June 1989), E. Kaltofen and S. M. Watt, Eds., Springer-Verlag, pp. 299–307.

[17] SANTAS, P. S. A type system for computer algebra. In Miola [12], pp. 177–191.

[18] SCHÖNERT, M., BESCHE, H. U., BREUER, T., CELLER, F., MNICH, J., PFEIFFER, G., POLIS, U., AND NIEMEYER, A. *GAP — Groups Algorithm, and Programming*. Lehrstuhl D für Mathematik, RWTH Aachen, Apr. 1992. Available via anonymous ftp at `samson.math.rwth-aachen.de` in `pub/gap`.

[19] SMOLKA, G., NUTT, W., GOGUEN, J. A., AND MESEGUER, J. Order-sorted equational computation. In *Resolution of Equations in Algebraic Structures, Volume 2*, H. Aït-Kaci and M. Nivat, Eds. Academic Press, 1989, chapter 10, pp. 297–367.

[20] SUTOR, R. S., AND JENKS, R. D. The type inference and coercion facilities in the Scratchpad II interpreter. *ACM SIGPLAN Notices 22*, 7 (1987), 56–63. SIGPLAN '87 Symposium on Interpreters and Interpretive Techniques.

[21] WEBER, A. A type-coercion problem in computer algebra. In *Artifical Intelligence and Symbolic Mathematical Computation — International Conference AISMC-1* (Karlsruhe, Germany, Aug. 1992), J. Calmet and J. A. Campbell, Eds., vol. 737 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 188–194.

[22] WEBER, A. On coherence in computer algebra. In Miola [12], pp. 95–106.

[23] WEBER, A. *Type Systems for Computer Algebra*. Dissertation, Fakultät für Informatik, Universität Tübingen, July 1993.

[24] ZIPPEL, R. The Weyl computer algebra substrate. In Miola [12], pp. 303–318.