Weak Subsumption Constraints for Type Diagnosis: An Incremental Algorithm*

(Extended Abstract)

Martin Müller Joachim Niehren

German Research Center for Artificial Intelligence (DFKI)

Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany

{mmueller,niehren}@dfki.uni-sb.de

Abstract

We introduce constraints necessary for type checking a higher-order concurrent constraint language, and solve them with an incremental algorithm. Our constraint system extends rational unification by constraints $x \subseteq y$ saying that "x has at least the structure of y", modelled by a weak instance relation between trees. This notion of instance has been carefully chosen to be weaker than the usual one which renders semi-unification undecidable. Semi-unification has more than once served to link unification problems arising from type inference and those considered in computational linguistics. Just as polymorphic recursion corresponds to subsumption through the semi-unification problem, our type constraint problem corresponds to weak subsumption of feature graphs in linguistics. The decidability problem for weak subsumption for feature graphs has been settled by Dörre [Dör94]. In contrast to Dörre's, our algorithm is fully incremental and does not refer to finite state automata. Our algorithm also is a lot more flexible. It allows a number of extensions (records, sorts, disjunctive types, type declarations, and others) which make it suitable for type inference of a full-fledged programming language.

Keywords: type inference, weak subsumption, unification, constraints, constraint programming

*In: Joint COMPULOGNET/ELSNET/EAGLES Workshop on Computational Logic for Natural Language Processing, April 3-5, Edinburgh, Scotland, 1995.

1 Introduction

We give an algorithm which is at the heart of a type diagnosis system for a higher-order concurrent constraint language, viz. the γ calculus [Smo94] which is the underlying operational model of the programming language Oz [ST94]. The algorithm decides satisfiability of constraints containing equations $x{=}y$ and $x{=}f(\overline{y})$, and weak subsumption constraints $x{\subseteq}y$ over infinite constructor trees with free variables. The algorithm is given fully in terms of constraint simplification. One the one hand, this gives credit to the close relationship between type inference and constraint solving (e.g., [Wan87, AW93, KPS94] and many others). On the other hand it establishes yet another correspondence between unification problems arising from polymorphic type inference and unification based grammar formalisms: The most prominent one is the equivalence of type checking polymorphic recursion [Myc84, Hen88] with semi-unification [KTU93, DR90] both of which are undecidable in general. To avoid this undecidability, we chose a weaker instance relation to give semantics to $x{\subseteq}y$. For example, we allow $f(a\ b)$ as an instance of $f(x\ x)$ even if $a \neq b$. On the type side, this type of constraints maintains some of the polymorphic flavour, but abandons full parametric polymorphism [MN95].

We start out from the set of infinite constructor trees with holes (free variables). We give a semantics which interprets the tree assigned to a variable dually: As itself and the set of its "weak" instances. Our algorithm terminates, and can be shown to be correct and complete under this semantics. The decidability problem for our constraints turned out to be equivalent to weak subsumption over feature graphs solved by Dörre [Dör94] for feature graphs with feature (but no arity) constraints.

However, only half of Dörre's two-step solution is a constraint solving algorithm. The second step relies on the equivalence of non-deterministic and deterministic finite state automata. In contrast, our algorithm decides satisfiability in a completely incremental manner and is thus amenable to be integrated in an concurrent constraint language like Oz [ST94] or AKL [JH91].

The extension of our algorithm towards feature trees is easily possible (see [MN95]). This allows to do type diagnosis for records [ST92] and objects. An entirely set-based semantics allows to naturally extend the algorithm to a full-fledged type diagnosis system, covering – among other aspects – sorts, disjunctive types, and recursive data type declarations [NPT93].

Type Diagnosis. As an illustrating example for the form of type diagnosis we have in mind, consider the following γ program:

$$\exists x \exists y \exists z \exists p \;\; p{:}u\,v/v{=}cons(x\;u) \wedge p\,y\,y \wedge x{=}f(y\;z)$$

This program declares four variables x, y, z, and p. It defines a relational abstraction p, which states that its two arguments u and v are related through the equation $v = cons(x u)$.[1] Furthermore, it states the equality $x{=}f(y\;z)$ and applies p to yy. This application $p\,y\,y$ reduces to a copy of the abstraction p with the actual arguments yy replaced for the formal ones uv:

$$\begin{array}{rl} & \exists x \exists y \exists z \exists p \;\; p{:}u\,v/v{=}cons(x\;u) \wedge p\,y\,y \wedge x{=}f(y\;z) \\ \rightarrow & \exists x \exists y \exists z \exists p \;\; p{:}u\,v/v{=}cons(x\;u) \wedge y{=}cons(x\;y) \wedge x{=}f(y\;z) \end{array}$$

Observe how the abstraction p is defined by reference to the global variable x, while the value of x is defined through an application of p: $p\,y\,y \wedge x{=}f(y\,z)$. Such a cycle is specific to the γ calculus since no other language offers explicit declaration of logic variables global to an abstraction (be it logic, functional, or concurrent languages, e.g., Prolog, ML [HMM86], or Pict [PT95]).

The types of the variables involved are described by the following constraint.[2] For ease of reading, we slightly abuse notation and pick the type variables identical to the corresponding object variables:

$$p{=}\langle u\;v\rangle \wedge v{=}cons(x\;u) \wedge y{\subseteq}u \wedge y{\subseteq}v \wedge x{=}f(y\;z)$$

$\langle u\;v\rangle$ is the relational type of p, and the application gives rise to the constraint $y{\subseteq}u \wedge y{\subseteq}v$, which says that y is constrained by both formal arguments of the procedure p. The subconstraint $x{=}f(y\;z) \wedge y{\subseteq}v \wedge v{=}cons(x\;u)$ reflects the cyclic dependency between x and p. It says that y be in the set of instances of v which depends through $v{=}cons(x\;u)$ on x, and at the same time that x should be exactly $f(y\;z)$.

Type diagnosis along this line is discussed in depth in [MN95].

Related Work. Apart from the already mentioned work, related work includes investigations about membership constraints (e.g., [NPT93]), type analysis for untyped languages (Soft Typing) [AW93, CF91, WC93], constraint-based program analysis [KPS94] and the

[1] Note that $p{:}u\,v/v{=}cons(x\;u)$ is different from a named λ abstraction $p = \lambda u.cons(x\;u)$ because it is relational rather than functional, and also different to the Prolog program $p(u, v) :- v = cons(x\,u).$, because Prolog does not allow variables to be global wrt. a predicate but rather existentially quantifies x.

[2] The formal account of the derivation of type constraints from programs will be given in [Mül96].

derivation of recursive sets from programs [FSVY91]. For proofs and a detailed discussion of related work see [MN95].

Plan of the Paper. This paper is structured as follows. In the Section 2 below we present our constraints along with their semantics and give necessary notation. Section 3 gives a simple algorithm which is correct but non-terminating. Section 4 gives the rules of the full algorithm. Section 5 concludes and gives a brief outlook.

2 Constraints and Semantics

We assume a signature Σ of function symbols with at least two elements ranged over by f, g, h, a, b, c and an infinite set of *base variables* $\mathcal{BV}$ ranged over by χ. If V is a further set of variables then $\mathcal{IT}(V)$ stands for the set of all finite or infinite *trees* over signature Σ and variables V. Trees of $\mathcal{IT}(V)$ are always ranged over by s and t. The set of variables occurring in a tree t is denoted by $\mathcal{V}(t)$. Sequences of variables are written as $\overline{x}$, or $\overline{\chi}$.

We build constraints over a set of *constraint variables* ranged over by x, y, z, u, v, w. Constraint variables must contain at least base variables. The syntax of our *constraints* ϕ, ψ is as follows:

$$x, y \ ::= \ \chi \qquad \text{and} \qquad \phi, \psi \ ::= \ x{=}y \mid x{=}f(\overline{y}) \mid x{\subseteq}y \mid \phi \wedge \psi$$

As *atomic constraints* we consider equations $x{=}y$ or $x{=}f(\overline{y})$ and weak subsumption constraints $x{\subseteq}y$. Constraints are atomic constraints closed under conjunction. *First-order formulae* build over constraints ϕ are denoted by Φ. We define $\equiv$ to be the least binary relation on ϕ such that $\wedge$ is associative and commutative. For convenience, we shall use the following notation:

$$\phi \text{ in } \psi \qquad \text{iff} \quad \text{exists } \phi' \text{ with } \phi \wedge \phi' \equiv \psi$$

As semantic structures we pick *tree-structures* which we also call $\mathcal{IT}(V)$ for some set V. The domain of a tree-structure $\mathcal{IT}(V)$ is the set of trees $\mathcal{IT}(V)$. Its interpretation is defined by $f^{\mathcal{IT}(V)}(\overline{t}) = f(\overline{t})$. We define the application $f(\overline{T})$ of f to a sequences of sets of trees $\overline{T}$ elementwise, $f(\overline{T}) = \{f(\overline{t}) \mid \overline{t} \in \overline{T}\}$. Given a tree $s \in \mathcal{IT}(V)$, the set $\mathsf{Inst}_V(s)$ of *weak instances of* s is defined as the greatest fixed point of:

$$\mathsf{Inst}_V(s) = \begin{cases} \mathcal{IT}(V) & \text{if } t = x \text{ for some } x \\ f(\overline{\mathsf{Inst}_V(s)}) & \text{if } t = f(\overline{s}) \text{ for some } \overline{s} \end{cases}$$

Notice that this definition implies $f(a\ b) \in \mathsf{Inst}_V(f(x\ x))$, even if $a \neq b$. Let V_1, V_2 be two sets whose elements we call variables. A V_1-V_2-*substitution* σ is a mapping from V_1 to $\mathcal{IT}(V_2)$. By homomorphic extension, every substitution can be extended to a mapping from $\mathcal{IT}(V_1)$ to $\mathcal{IT}(V_2)$. The set of *strong instances of* s is defined by $\mathsf{Inst'}_V(s) = \{\sigma(s) \mid \sigma \text{ is a } \mathcal{V}(s)\text{-}V\text{-substitution}\}$. Note that $\mathsf{Inst'}_V(s) \subseteq \mathsf{Inst}_V(s)$, and that $f(a\,b) \notin \mathsf{Inst'}_V(f(x\ x))$ if $a \neq b$. Using $\mathsf{Inst'}_V(s)$ instead of $\mathsf{Inst}_V(s)$ would make satisfiability of our constraints equivalent to semi-unification and undecidable [KTU90, DR90].

Let σ be a V_1-V_2-substitution, $\{x, y, \overline{z}\} \subset V_1$, and ϕ, ψ constraints such that $\mathcal{V}(\phi) \subseteq V_1$, $\mathcal{V}(\psi) \subseteq V_1$. Then we define:

$$\begin{array}{lll@{\qquad}lll}
\models_\sigma x{=}y & \text{iff} & \sigma(x){=}\sigma(y) & \models_\sigma x{=}f(\overline{z}) & \text{iff} & \sigma(x){=}f^{\mathcal{IT}(V_2)}(\overline{\sigma(y)}) \\
\models_\sigma x{\subseteq}y & \text{iff} & \mathsf{Inst}_{V_2}(\sigma(x)) \subseteq \mathsf{Inst}_{V_2}(\sigma(y)) & \models_\sigma \phi \wedge \psi & \text{iff} & \models_\sigma \phi \text{ and } \models_\sigma \psi
\end{array}$$

A V_1-V_2-*solution* of ϕ is a V_1-V_2-substitution satisfying $\models_\sigma \phi$. A constraint ϕ is called *satisfiable*, if there exists a V_1-V_2-solution for ϕ. The notion of $\models_\sigma$ extends to arbitrary first-order formulae Φ in the usual way. We say that a formula Φ is *valid*, if $\models_\sigma \Phi$ holds for all V_1-V_2-substitutions σ with $\mathcal{V}(\Phi) \subseteq V_1$. In symbols, $\models \Phi$.

Our setting is a conservative extension of the usual rational unification problem. This means that free variables in the semantic domain do not affect equality constraints. A constraint ϕ is *satisfiable in the tree-model* $\mathcal{IT}(V)$, if there exists a $\mathcal{BV}$-V-solution of ϕ. The trees of $\mathcal{IT}(\emptyset)$ are called *ground trees*.

Proposition 2.1 *Suppose ϕ not to contain weak subsumption constraints. Then ϕ is satisfiable if and only if it is satisfiable in the model of ground trees.*

The statement would be wrong for ϕ's containing weak subsumption constraints. For instance, consider the following ϕ with $a \neq b$:

$$\phi \;\equiv\; x{\subseteq}z \wedge y{\subseteq}z \wedge x{=}a \wedge y{=}b$$

This ϕ is not satisfiable in the model of ground trees, since the set $\mathsf{Inst}_\emptyset(t)$ is a singleton for all ground trees t, whereas any V_1-V_2-solution σ of ϕ has to satisfy $\{a, b\} \subseteq \mathsf{Inst}_{V_2}(\sigma(z))$. However, there exists a $\{x, y, z\}$-$\{v\}$-solution σ of ϕ, where $\{v\}$ is an singleton: $\sigma(x) = a\,, \sigma(y) = b\,, \sigma(z) = v\,.$

Proposition 2.2 *For all x, y, z, u, v the following statements hold:*

1) $\models x{=}y \rightarrow x{\subseteq}y\,,$ 2) $\models x{\subseteq}y \wedge y{\subseteq}z \rightarrow x{\subseteq}z\,,$ 3) $\models x{=}f(\overline{y}) \rightarrow x{\subseteq}f(\overline{y})\,,$
4) $\not\models x{\subseteq}y \wedge y{\subseteq}x \rightarrow x{=}y$ 5) $\not\models x{=}f(u\ v) \wedge x{\subseteq}y \wedge y{=}f(z\ z) \rightarrow u{=}v\,.$

Weak Subsumption vs. Sets of Weak Instances. In the remainder of this section we compare our sets of weak instance with Dörre's notion of weak subsumption. Let us consider constructor trees as special feature trees with integer-valued features, a distinguished feature label (e.g., [NP93, Bac94]), and a distinguished feature arity. Given feature constraints $x[f]y$ saying that x has direct subtree y at feature f, the equation $x{=}f(y_1 \ldots y_n)$ can be considered equivalent to:[3]

$$x[\mathsf{arity}]n \wedge x[\mathsf{label}]f \wedge x[1]y_1 \wedge \ldots \wedge x[n]y_n.$$

Let us write $s[f]{\downarrow}$ to say that the tree s has some direct subtree at f. A *simulation* between $\mathcal{IT}(V_1)$ and $\mathcal{IT}(V_2)$ is a relation $\Delta \subseteq \mathcal{IT}(V_1) \times \mathcal{IT}(V_2)$ satisfying: If $(t, s) \in \Delta$ then

(Arity Simulation)	If $t[\mathsf{label}]{\downarrow}$ and there is an n such that $t[\mathsf{arity}]n$, then $s[\mathsf{arity}]n$.
(Feature Simulation)	If $t[f]{\downarrow}$ and there is a tree t' such that $t[f]t'$, then $s[f]{\downarrow}$, $s[f]s'$, and $(t', s') \in \Delta$.

Now, the weak subsumption preorder $\precsim^V$ is defined by:

$$t \precsim^V s \qquad \text{iff} \qquad \text{there is a simulation } \Delta \subseteq V \times V \text{ such that } (s, t) \in \Delta$$

We have the following lemma:

Lemma 2.3 *For all constructor trees s, t it holds that: $\mathsf{Inst}_V(s) \subseteq \mathsf{Inst}_V(t)$ iff $s \precsim^V t$.*

A similar statement can be derived for the set of strong instances and a strong subsumption preorder following [Dör94]. The difference between $\precsim^V$ and Dörre's notion of weak subsumption is that he does not require *(Arity Simulation)*, while we naturally do since we start from constructor trees. For type checking, constructor trees seem more natural: For illustration note that the arity of a procedure is essential type information.

[3] This simpler encoding of constructor trees not using arity constraints has been suggested by one of the referees.

3 A Non-terminating Solution

$(Decom)$	$\dfrac{x{=}f(\overline{u}) \wedge \phi}{\overline{u}{=}\overline{v} \wedge \phi}$	$x{=}f(\overline{v})$ in ϕ.
$(Clash)$	$\dfrac{x{=}f(\overline{u}) \wedge \phi}{\perp}$	$x{=}g(\overline{v})$ in ϕ, *and* $f \neq g$.
$(Elim)$	$\dfrac{x{=}y \wedge \phi}{x{=}y \wedge \phi[y/x]}$	$x \in \mathcal{V}(\phi)$, and $x \neq y$.
$(Descend)$	$\dfrac{x{\subseteq}y \wedge \phi}{x{=}f(\overline{u}) \wedge \overline{u}{\subseteq}\overline{z} \wedge \phi}$	$\overline{u}$ fresh, $y{=}f(\overline{z})$ in ϕ.

Figure 1: A Non-Terminating Algorithm

In order to solve our constraints one could come up with the system given in Figure 1. Besides the three usual unification rules for rational trees, the only additional rule is (Descend). This algorithm is correct and very likely to be complete in that for an unsatisfiable constraint ϕ there is a derivation from ϕ to $\perp$. However, this intuitive algorithm loops due to the introduction of new variables.

$$\dfrac{\dfrac{x{\subseteq}y \wedge y{=}f(x)}{x{=}f(x_1) \wedge x_1{\subseteq}x \wedge y{=}f(x)}}{\dots} \quad \begin{matrix}\text{Descend}\\ \text{Descend}\end{matrix} \quad \dfrac{\dfrac{x{\subseteq}y \wedge y{=}f(y)}{x{=}f(x_1) \wedge x_1{\subseteq}y \wedge y{=}f(y)}}{\dots}$$

Note that some form of descending is necessary in order to derive the clash from the inconsistent constraint $y{=}f(u) \wedge u{=}a \wedge z{=}f(x) \wedge x{\subseteq}y \wedge x{\subseteq}z \wedge \phi$

4 Algorithm

To consider trees with free variables as set of instances means that we need to compute intersections of such sets and to decide their emptiness. When we simplify $x{\subseteq}y \wedge x{\subseteq}z$ in a context ϕ, we have to compute the intersection of the sets of instances of y and z. In order to avoid the introduction of new variables we add a new class of variables to represent such intersections, and one new constraint. *Intersection variables* are defined as nonempty finite subsets of base variables. In order capture the intended semantics, we write $\chi_1{\cap}\dots{\cap}\chi_n$

instead of $\{\chi_1\} \cup \ldots \cup \{\chi_n\}$. The equality $\equiv$ on intersection variables is the equality on powersets, which satisfies:

$$x \cap y \equiv y \cap x, \quad (x \cap y) \cap z \equiv x \cap (y \cap z), \quad x \cap x \equiv x.$$

We call an x a *component* of y, if $y \equiv x \cap z$ for some z. The set of components of a variable x is denoted by $\mathcal{C}(x)$. Note that $x \cap y \in \mathcal{V}(\phi)$ implies $x \in \mathcal{C}(\mathcal{V}(\phi))$ but in general not $x \in \mathcal{V}(\phi)$. As additional constraint we introduce $x \subseteq f(\overline{y})$, with the semantics:

$$\models \ x \subseteq f(\overline{y}) \leftrightarrow \exists u (x \subseteq u \wedge u = f(\overline{y}))\,.$$

Complete semantics has to take care of intersection variables such as $y \cap z$. Constraint solving will propagate intersection variables into most constraint positions. That is, our algorithm actually operates on the following constraints:

$$x, y \ ::= \ \chi \mid x \cap y \qquad \text{and} \qquad \phi, \psi \ ::= \ x = y \mid x = f(\overline{y}) \mid x \subseteq y \mid x \subseteq f(\overline{y}) \mid \phi \wedge \psi$$

However, if started with a constraint containing only base variables, our algorithm maintains this invariant for the equational constraints.

Let us call a variable x *immediately determined by f*, in ϕ, written $x \circ f(\overline{y})$, if one of $x = f(\overline{y})$ or $x \subseteq f(\overline{y})$ is in ϕ for some $f(\overline{y})$. We say that x is *immediately determined* in ϕ if it is immediately determined by some f in ϕ. Call x *determined*, written $x \leq_\phi f(\overline{u})$ if x is immediately determined in ϕ, or $x \subseteq y \cap z$ and $y \circ f(\overline{u})$ are in ϕ. Obviously, if $x \leq_\phi f(\overline{y})$, then the top-level constructor of x must be f.

We define the application of an operator $[y/x]$ to intersection variables, only if x is a base variable. If $z \equiv (\chi_1 \cap \ldots \cap \chi_n)$, then $z[y/x]$ we define:

$$z[y/x] \equiv \chi_1[y/x] \cap \ldots \cap \chi_n[y/x]\,.$$

We say that $[y/x]$ applied to intersection variables performs *deep substitution*. The following property holds for deep substitution:

$$\mathcal{C}(\mathcal{V}(x = y \wedge \phi)) \ = \ \mathcal{C}(\mathcal{V}(x = y \wedge \phi[y/x]))\,.$$

Note however that $\mathcal{V}(x = y \wedge \phi) \neq \mathcal{V}(x = y \wedge \phi[y/x])$ if $\phi \equiv z \subseteq x \cap y$. The variable $x \cap y$ is contained in the first but not in the second set. We can now specify our algorithm for constraint simplification. It is given by the rules in Figure 2 and Figure 3.

$(Decom)$	$\dfrac{x{=}f(\overline{u}) \wedge \phi}{\overline{u}{=}\overline{v} \wedge \phi}$	$x{=}f(\overline{v})$ in ϕ.
$(Clash)$	$\dfrac{\phi}{\perp}$	$x{\leq_\phi}f(\overline{u})$, $x{\cap}y{\leq_\phi}g(\overline{v})$, and $f \neq g$.
$(Elim)$	$\dfrac{x{=}y \wedge \phi}{x{=}y \wedge \phi[y/x]}$	$x \in \mathcal{C}(\mathcal{V}(\phi)) \cap \mathcal{BV}$, and $x \not\equiv y$.

Figure 2: Rational Tree Unification

The Rule *(Decom)* is known from usual unification for rational trees. Up to the application condition $x \in \mathcal{C}(\mathcal{V}(\phi)) \cap \mathcal{BV}$, this also applies to rule *(Elim)*. This side condition accounts for deep substitution. The *(Clash)* rule contains as special cases:

$$\frac{x{=}f(\overline{y}) \wedge x{=}g(\overline{z}) \wedge \phi}{\perp}\ f \neq g \qquad \text{and} \qquad \frac{x{\subseteq}f(\overline{y}) \wedge x{\subseteq}g(\overline{z}) \wedge \phi}{\perp}\ f \neq g\ .$$

Its full power comes in interaction with the rules in Figure 3. Then it allows to derive a clash if for a variable x a constructor is known, and for some variable $x{\cap}y$ a distinct constructor is derivable.

$(Propagate1)$	$\dfrac{x{\cap}y{\subseteq}z \wedge \phi}{x{\cap}y{\subseteq}z{\cap}u \wedge \phi}$	$x{\subseteq}u$ in ϕ, $z{\cap}u \not\equiv z$.
$(Propagate2)$	$\dfrac{x{\cap}y{\subseteq}f(\overline{u}) \wedge \phi}{x{\cap}y{\subseteq}f(\overline{u}{\cap}\overline{v}) \wedge \phi}$	$x{\leq_\phi}f(\overline{v})$, $\overline{u}{\cap}\overline{v} \not\equiv \overline{u}$.
$(Collapse)$	$\dfrac{x{\subseteq}y{\cap}u \wedge \phi}{x{\subseteq}y{\cap}z{\cap}u \wedge \phi}$	$y{\subseteq}z$ in ϕ, and $y{\cap}z{\cap}u \not\equiv y{\cap}z$.
$(Descend1)$	$\dfrac{x{=}f(\overline{u}) \wedge \phi}{x{=}f(\overline{u}) \wedge \overline{u}{\subseteq}\overline{v} \wedge \phi}$	$x{\leq_\phi}f(\overline{v})$, $\overline{u}{\subseteq}\overline{v}{\cap}\overline{w}$ not in ϕ
$(Descend2)$	$\dfrac{\phi}{x{\cap}y{\subseteq}f(\overline{u}) \wedge \phi}$	$x{\cap}y \in \mathcal{V}(\phi)$, $x{\leq_\phi}f(\overline{u})$, and not $x{\cap}y{\circ}g(\overline{v})$ in ϕ.

Figure 3: Simplifying Membership Constraints

Rules *(Propagate1)* and *(Propagate2)* propagate intersection variables into the right hand side of weak subsumption contraints. The *(Collapse)* rule collapses chains of variables related via weak subsumption constraints. In other words, these rules propagate lower bounds with respect to the weak subsumption relation.

The rules (*Descend1*) and (*Descend2*) replace (*Descend*) from the non-terminating algorithm in Figure 1. The Descend rules are the only rules introducing new weak subsumption constraints. The rule (*Descend2*) introduces a constructor for intersection-variables $x{\cap}y$ by adding a constraint of the form $x{\cap}y{\subseteq}f(\overline{u})$. If the rule is applied, then the intersection of x and y is forced to be nonempty. Nonemptiness is implied by ϕ, if $x{\cap}y$ occurs in ϕ ($x{\cap}y \in \mathcal{V}(\phi)$).

Note that (*Descend1*) and (*Descend2*) are carefully equipped with side conditions for termination. For example, the following derivations are **not** possible:

$$\frac{x{=}f(u)}{x{=}f(u) \wedge x{\subseteq}f(u)} \qquad \frac{x{\subseteq}y \wedge x{=}f(x) \wedge x{\subseteq}f(y)}{x{\subseteq}y \wedge x{\subseteq}y \wedge x{=}f(x) \wedge x{\subseteq}f(y)} \qquad \frac{x{=}f(y)}{y{\subseteq}y \wedge x{=}f(y)} \ .$$

We can prove that our algorithm performs equivalence transformations with respect to substitutions σ which meet the intended semantics of intersection variables, i.e., *intersection-correct* substitutions:

Definition 4.1 (Intersection Correct) *We say that a substitution σ is* intersection-correct for x and y, *if it satisfies:*

$$\sigma(x{\cap}y) = \sigma(x) \cap \sigma(y) \ .$$

We say that a substitution σ is intersection-correct, *if the following properties holds for all intersection variables x, y and z:*

If x, y, $x{\cap}y \in \mathsf{dom}(\sigma)$, then σ is intersection-correct for x and y.
If x, $x{\cap}y \in \mathsf{dom}(\sigma)$, then σ is intersection-correct for $x{\cap}y$ and y.

Note that σ is intersection-correct for x and $x{\cap}y$, iff $\sigma(x{\cap}y) \ \subseteq \ \sigma(x)$. We call a constraint ϕ *intersection-satisfiable*, if ϕ has an intersection-correct solution.

Proposition 4.2 *Let ϕ be a constraint only containing base variables only. Then ϕ is satisfiable, if and only if it is intersection satisfiable.*

We denote the set of all intersection-correct solutions of ϕ with $\mathrm{Sol}^I(\phi)$. Assume σ to be a substitution. A V*-extension* of σ is a substitution $\tilde{\sigma}$ such that $\mathsf{dom}(\tilde{\sigma}) = \mathsf{dom}(\sigma) \cup V$ such that σ and $\tilde{\sigma}$ coincide on $\mathsf{dom}(\sigma)$. We denote the set of all intersection-correct V-extensions of σ with $\mathrm{Ext}^I_V(\sigma)$. Let ϕ and ψ be constraints. We say that ϕ *intersection-implies* ψ, written $\phi \models^I \psi$, if

$$\mathrm{Ext}^I_{\mathcal{V}(\psi)}(\mathrm{Sol}^I(\phi)) \;\subseteq\; \mathrm{Sol}^I(\psi) \quad \text{and} \quad \mathrm{Sol}^I(\phi) = \emptyset \text{ iff } \mathrm{Ext}^I_{\mathcal{V}(\psi)}(\mathrm{Sol}^I(\phi)) = \emptyset$$

We call ϕ and ψ *intersection-equivalent* if $\phi \models^I \psi$ and $\psi \models^I \phi$, and write $\phi \models\!\!\!\!\!\dashv^I \psi$. Both conditions ensure the following Lemma:

Lemma 4.3 *If ϕ is not intersection satisfiable, then $\phi \models^I \psi$ holds vacuously for all ψ. Furthermore, if $\phi \models\!\!\!\!\!\dashv^I \psi$, then ϕ is intersection satisfiable if and only if ψ is.*

Given the above notions, the following two theorems are our main results. For the proofs the reader is referred to [MN95].

Theorem 4.4 (Termination) *The rule system given in Figures 2 and 3 terminates.*

Theorem 4.5 (Correctness and Completeness) *Let ϕ be a constraint containing base variables only. Then the following statements are equivalent:*

1. *ϕ is intersection-satisfiable.*
2. *There exists an irreducible $\psi \neq \bot$ derivable from ϕ.*
3. *There exists a irreducible $\psi \neq \bot$ that is intersection-equivalent to ϕ.*
4. *$\bot$ cannot be derived from ϕ.*

Acknowledgements. We would like to thank Ralf Treinen for pointing us to Dörre's paper and the anonymous referees for useful remarks. The research reported in this paper has been supported by the Bundesminister für Bildung, Wissenschaft, Forschung und Technologie (FTZ-ITW-9105), the Esprit Project ACCLAIM (PE 7195), the Esprit Working Group CCL (EP 6028), and a fellowship of the Graduiertenkolleg 'Kognition' at the Universität des Saarlandes of the first author.

5 Outlook

We have presented an algorithm for deciding satisfiability of weak subsumption constraints over infinite constructor trees with holes. Our motivation to solve such constraints grew out of a type inference problem. Formally, the problem is equivalent to type checking a weak form of polymorphic recursion. Type checking polymorphic recursion is equivalent to semi-unification and to subsumption of feature graphs. All three are undecidable [Hen88, KTU93, DR90]. We establish a similar correspondence between a type inference problem and weak subsumption of feature graphs: The latter has been investigated by Dörre looking for a logical treatment of coordination phenomena in unification based grammar formalisms [Dör94]. Our starting point from the constraint language Oz however lead us to an incremental algorithm, in contrast to the automata based solution of Dörre.

References

[AW93] A. Aiken and E. Wimmers. Type Inclusion Constraints and Type Inference. In *ACM Conference on Functional Programming and Computer Architecture*, pages 31–41, Copenhagen, Denmark, June 1993.

[Bac94] Rolf Backofen. *Expressivity and Decidability of First-order Languages over Feature Trees.* Doctoral Dissertation. Universität des Saarlandes, Technische Fakultät, 66041 Saarbrücken, Germany, 1994.

[CF91] R. Cartwright and M. Fagan. Soft Typing. In *ACM Conference on Programming Language Design and Implementation*, pages 278–292, June 1991.

[Dör94] Jochen Dörre. Feature-Logic with Weak Subsumption Constraints. In Rupp et al. [RRJ94], chapter 7, pages 187–203.

[DR90] Jochen Dörre and William C. Rounds. On Subsumption and Semiunification in Feature Algebras. In *IEEE Symposium on Logic in Computer Science*, pages 300–310, 1990.

[FSVY91] T. Frühwirth, E. Shapiro, M. Y. Vardi, and E. Yardeni. Logic Programs as Types for Logic Programs. In *IEEE Symposium on Logic in Computer Science*, pages 300–309, 1991.

[Hen88] F. Henglein. Type Inference and Semi-Unification. In *ACM Conference on LISP and Functional Programming*, pages 184–197, January 1988.

[HMM86] Robert Harper, Dave MacQueen, and Robin Milner. Standard ML. Technical Report ECS-LFCS-86-2, Department of Computer Science, University of Edinburgh, 1986.

[JH91] Sverker Janson and Seif Haridi. Programming Paradigms of the Andorra Kernel Language. In *International Logic Programming Symposium*, pages 167–186, 1991.

[KPS94] Dexter Kozen, Jens Palsberg, and Michael I. Schwartzbach. Efficient Inference of Partial Types. *Journal of Computer and System Sciences*, 49(2):306–324, 1994. has also appared in Proc. FOCS, pp.363-371.

[KTU90] A. Kfoury, J. Tiuryn, and P. Urzyczyn. The Undecidability of the Semi-Unification Problem. In *ACM Symposium on Theory of Computation*, pages 468–476, May 1990.

[KTU93] A. J. Kfoury, J. Tiuryn, and Urzycyn. Type Recursion in the Presence of Polymorphic Recursion. *ACM Transactions on Programming Languages and Systems*, pages 290–311, 1993.

[MN95] Martin Müller and Joachim Niehren. A Type is a Type is a Type. Research report, DFKI, Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1995. In Preparation.

[Mül96] Martin Müller. *Type Diagnosis for a Higher-Order Concurrent Constraint Language.* Doctoral Dissertation. Universität des Saarlandes, Technische Fakultät, 66041 Saarbrücken, Germany, 1996. In Preparation.

[Myc84] Alan Mycroft. Polymorphic Type Schemes and Recursive Definitions. In *International Symposium on Programming*, number 167 in Lecture Notes in Computer Science, 1984.

[NP93] Joachim Niehren and Andreas Podelski. Feature Automata and Recognizable Sets of Feature Trees. In *Tapsoft*, pages 356–375, April 1993.

[NPT93] Joachim Niehren, Andreas Podelski, and Ralf Treinen. Equational and Membership Constraints for Infinite Trees. In Claude Kirchner, editor, *Proceedings of the RTA '93*, pages 106–120, 1993.

[PT95] Benjamin C. Pierce and David N. Turner. PICT User Manual. Technical Report to appear, LFCS, Edinburgh, 1995. Version 3.5g.

[RRJ94] C. J. Rupp, M. A. Rosner, and R. L. Johnson, editors. *Constraints, Languages, and Computation.* Academic Press, 1994.

[Smo94] Gert Smolka. A Foundation for Concurrent Constraint Programming. In Jean-Pierre Jouannaud, editor, *Constraints in Computational Logics*, volume 845 of *Lecture Notes in Computer Science*, pages 50–72, München, Germany, 7–9 September 1994.

[ST92] Gert Smolka and Ralf Treinen. Records for Logic Programming. In *International Conference and Symposium on Logic Programming*, pages 240–254, November 1992. Has appeared in the Journal of Logic Programming, April 1994.

[ST94] Gert Smolka and Ralf Treinen, editors. *DFKI Oz Documentation Series*. Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1994. Documentation and System available through anonymous ftp from ps-ftp.dfki.uni-sb.de or through WWW from http://ps-www.dfki.uni-sb.de.

[Wan87] Mitchell Wand. A Simple Algorithm and Proof for Type Inference. *Fundamenta Informaticae*, 10:115–122, 1987.

[WC93] Andrew K. Wright and Robert Cartwright. A Practical Soft Type System for Scheme. Technical Report 93-218, Rice University, December 1993.