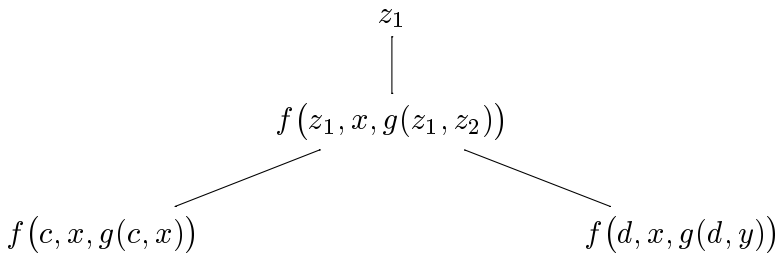
A functional reconstruction of anti-unification

Norwegian Computing Center
APPLIED RESEARCH AND DEVELOPMENT

NOTE



DART/04/04

Bjarte M. Østvold

2004-04-19

Norwegian Computing Center
APPLIED RESEARCH AND DEVELOPMENT

NR Note

Title: A functional reconstruction of anti-unification

Date: 2004-04-19
Year: 2004
Note no.: DART/04/04

Author: Bjarte M. Østvold

Abstract: In 1970 both Plotkin and Reynolds defined the anti-unification of a set of literal clauses, that is, atomic formulas or negated atomic formulas. They gave similar imperative anti-unification algorithms and proved the algorithms correct.

We formulate an alternative anti-unification algorithm using recursive equations and prove its correctness. This functional-style algorithm gives a modular view of anti-unification, where each equation captures a different computational aspect. Modularity makes the algorithm well-suited as a starting point for designing related algorithms. It can easily be converted into a program in a functional programming language.

Keywords: anti-unification, unification, term lattice

Target group: All

Availability: Open

Project:

Project no.:

Research field: Theoretical computer science

No. of pages: 20

Norwegian Computing Center
Gaustadalléen 23, P.O. Box 114 Blindern, NO-0314 Oslo, Norway
Telephone: +47 2285 2500, telefax: +47 2269 7660, http://www.nr.no

A functional reconstruction of anti-unification

Bjarte M. Østvold[1,2]

[1] Norwegian Computing Center*

[2] Department of Informatics, University of Oslo

Abstract

In 1970 both Plotkin and Reynolds defined the anti-unification of a set of literal clauses, that is, atomic formulas or negated atomic formulas. They gave similar imperative anti-unification algorithms and proved the algorithms correct.

We formulate an alternative anti-unification algorithm using recursive equations and prove its correctness. This functional-style and deterministic algorithm can easily be converted into a program in a functional programming language.

Contents

*Email: bjarte@nr.no. Postal address: Norwegian Computing Center, PO Box 114 Blindern, NO-0314 Oslo, Norway.

1 Introduction

Anti-unification, the dual operation of unification in the lattice of terms, is part of the core techniques used in program synthesis from incomplete information, for instance in inductive logic programming [5] and in synthesis of functional programs [8].

Back in 1970 Plotkin and Reynolds [6, 7] gave essentially the same definition of anti-unification and the same imperative algorithm. We pick up their trail and give an anti-unification algorithm formulated as recursive equations.

The present algorithm differs from previous ones in several respects. It is functional and deterministic, for easy conversion into a program in a functional programming language, and it is more detailed, especially in the treatment of substitution. The latter is important for making such conversion as simple as possible. Huet's anti-unification algorithm from 1976, treated in detail by Lassez *et al.* [2], is also functional, but non-deterministic. Also, Huet's algorithm does not lead directly to a functional implementation in the case of anti-unifying n terms; our algorithm, however, may be implemented directly in this case. We also believe such an implementation to be quite efficient, though at present there is no analysis of the algorithm's behaviour.

Plotkin calls a term being the anti-unification of a set of terms for a *least generalisation* and we do the same. Plotkin and Reynolds study literal clauses, that is, atomic formulas or negated atomic formulas. We only study terms, not literals, but the difference is not significant: the algorithms view the inputs simply as syntactic objects and do not assign any meaning to symbols. Therefore, a formula $P(\cdots)$, where P is a predicate symbol, can be interpreted as a term with a function symbol f_P. Similarly, the formula $\neg P(\cdots)$ can be interpreted as term with a function symbol $f_{\neg P}$. Plotkin also extended his algorithm to more general logical clauses, but we do not follow up this.

2 Preliminaries

Our notational convention is as follows: x, y, z for variables; f, g, h for function symbols; c, d for nullary function symbols, called constant symbols; s, t, u, v for terms; T for sets of terms; θ, σ, ρ for substitutions; and n, m, k, l for natural numbers. We also permit subscripts, etc. To get a more readable notation we drop the brackets after constant symbols, writing just c instead of $c()$.

2.1 Terms

The *arity*, also called *rank*, of a symbol f is a natural number, $\mathcal{A}(f)$. A *ranked alphabet* is a finite set of symbols, each with unique arity. If Σ is a ranked alphabet then Σ_n denotes the ranked alphabet of all n-ary symbols in Σ.

Def. 2.1 (Terms, $T_{(\Sigma,X)}$) *Let Σ be a ranked alphabet and let X be a set of symbols, called* variables*, such that $\Sigma \cap X = \emptyset$. The set of Σ, X-terms, denoted*

$T_{(\Sigma,X)}$, *is the smallest set such that:*

$$X \subseteq T_{(\Sigma,X)}$$
$$f \in \Sigma_n \wedge t_1 \in T_{(\Sigma,X)} \wedge \cdots \wedge t_n \in T_{(\Sigma,X)} \implies f(t_1, \ldots, t_n) \in T_{(\Sigma,X)} \quad (1)$$

In the case $n = 0$ equation (1) is to be interpreted as $\Sigma_0 \subseteq T_{(\Sigma,X)}$. When a term t is headed by a function symbol, $t = f(\ldots)$, we use $head(t)$ to denote that function symbol. Note that the following is *not* a valid term: $f(a, f(b))$ since f must have unique arity.

Def. 2.2 (Syntactic equivalence, $s \equiv t$) $\equiv$ *is the smallest relation on terms such that*

$$x \equiv x$$
$$t_1 \equiv s_1 \wedge \cdots \wedge t_n \equiv s_n \implies f(t_1, \ldots, t_n) \equiv f(s_1, \ldots, s_n) \qquad \text{if } \mathcal{A}(f) = n$$

Trivially, $\equiv$ is an equivalence relation.

The set of variables in a term is defined as follows.

Def. 2.3 (Term variables, $\mathcal{V}(t)$)

$$\mathcal{V}(x) = \{x\}$$
$$\mathcal{V}\big(f(t_1, \ldots, t_n)\big) = \mathcal{V}(t_1) \cup \cdots \cup \mathcal{V}(t_n)$$

Note in particular that $\mathcal{V}(c) = \emptyset$. We allow the shorthand $\mathcal{V}(t_1, \ldots, t_n) = \mathcal{V}(t_1) \cup \cdots \cup \mathcal{V}(t_n)$. If $\mathcal{V}(t) = \emptyset$ then t is called a *ground term*.

We define two syntactic measures on terms, for use in inductive proofs.

Def. 2.4 (Term height, $\mathcal{H}(t)$; term size, $\mathcal{S}(t)$)

$$\mathcal{H}(x) = 0 \qquad \mathcal{S}(x) = 0$$
$$\mathcal{H}\big(f(t_1, \ldots, t_n)\big) = 1 + \max\big\{\mathcal{H}(t_1), \ldots, \mathcal{H}(t_n)\big\}$$
$$\mathcal{S}\big(f(t_1, \ldots, t_n)\big) = 1 + \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_n)$$

Note specifically that $\mathcal{H}(c) = \mathcal{S}(c) = 1$.

2.2 Substitution

A *binding* is a pair of a term t and a variable x, written t/x. A *binding set*, denoted B, is a finite set of bindings, $\{t_1/x_1, \ldots, t_n/x_n\}$, such that variables $x_1, \ldots, x_n$ are distinct. In particular the empty set is a binding set, and it is written as $\{\}$.

A binding set can be interpreted as a function from the set of variables to the set of terms. By that interpretation we say that a binding set $\{t_1/x_1, \ldots, t_n/x_n\}$ is *injective* when $t_1, \ldots, t_n$ are distinct. The empty binding set $\{\}$ is injective by definition.

We define domain and range of binding sets, and some related notation.

Def. 2.5 (Domain, $\mathcal{D}(B)$; range, $\mathcal{R}(B)$)

$$\mathcal{D}\big(\{t_1/x_1, \ldots, t_n/x_n\}\big) = \{x_1, \ldots, x_n\}$$
$$\mathcal{R}\big(\{t_1/x_1, \ldots, t_n/x_n\}\big) = \mathcal{V}(t_1, \ldots, t_n)$$
$$(\_/x) \in B \quad \textit{if} \quad \exists t\big((t/x) \in B\big) \qquad (t/\_) \in B \quad \textit{if} \quad \exists x\big((t/x) \in B\big)$$

For convenience we sometimes write $(\_/x) \notin B$ and $(t/\_) \notin B$ with the obvious meaning. Note that, for any binding set B, if $(\_/x) \in B$ holds then x is unique. The same is true of $(t/\_) \in B$ only if B is injective.[1]

A binding set is called a *substitution*, denoted θ, if it satisfies the following, called the *well-formedness criterion for substitutions*:

$$\mathcal{D}(\theta) \cap \mathcal{R}(\theta) = \emptyset. \tag{2}$$

In particular the empty binding set $\{\}$ is a substitution, called the empty substitution. We also use σ, ρ to denote substitutions.

Def. 2.6 (Substitution, $t\theta$)

$$\begin{aligned} x\theta &= s && \textit{if } (s/x) \in \theta \\ x\theta &= x && \textit{if } (\_/x) \notin \theta \\ f(t_1, \ldots, t_n)\theta &= f(t_1\theta, \ldots, t_n\theta) \end{aligned}$$

As expected, the following properties hold for substitution:

$$\begin{aligned} t\{\} &= t \\ s\{t_1/x_1, \ldots, t_n/x_n\} &= \big(\cdots\big(s\{t_1/x_1\}\big)\cdots\big)\{t_n/x_n\} \end{aligned} \tag{3}$$

The first is trivial. Regarding the latter, the well-formedness criterion (2) ensures that an arbitrary ordering of substitutions on the right-hand side of (3) may be used.

Ex. 2.7 (Parallel substitution on binding sets) The definition for substitution works equally well for all binding sets, even though binding sets do not in general satisfy the well-formedness criterion (2). But equation (3)—that breaks substitution application into a sequence of applications—does not hold in this case. Instead, such an operation on binding sets can be seen as a parallel substitution, where variables in the range are replaced simultaneously:

$$f(x, y)\{y/x, x/y\} = f\big(x\{y/x, x/y\}, y\{y/x, x/y\}\big) = f(y, x) \qquad \diamond$$

In the following we only consider substitutions, never general binding sets.

The following example illustrates that some transformations of terms can only be achieved by applying a series of substitutions.

[1] Our definition of range is similar to that of Lassez *et al.* [2]; other definitions are also in use.

Ex. 2.8 Consider the terms $f(x, y)$ and $f(y, x)$. There is no way we can transform one into the other using a *single* substitution. In particular, $\{y/x, x/y\}$ is not well-formed by condition (2). By applying a series of substitutions, however, the transformation can be achieved:

$$\big(\big(f(x, y)\{z_1/x, z_2/y\}\big)\{y/z_1, x/z_2\}\big). \qquad \diamond$$

For our results on anti-unification we only need a weak notion of substitution composition: the composition of θ and σ, written $\theta\sigma$, is defined only if $\rho = \theta \cup \sigma$ is a substitution and then $\theta\sigma = \rho$. This form of composition is associative, and so we may write $\theta_1\theta_2 \cdots \theta_n$.

An injective substitution on the form $\{y_1/x_1, \ldots, y_n/x_n\}$ is called a *renaming substitution*, that is, $y_1, \ldots, y_n$ are distinct. We can now define a useful equivalence relation on terms. Informally, $s \equiv_\alpha t$ if s and t differ only in the choice of variable names.

Def. 2.9 (α-equivalence, $s \equiv_\alpha t$) *Let s, t be terms. Then $s \equiv_\alpha t$ if there exist renaming substitutions $\theta_1, \ldots, \theta_n, \theta'_1, \ldots, \theta'_{n'}$ such that*

$$s\theta_1 \ldots \theta_n \equiv t \wedge t\theta'_1 \ldots \theta'_{n'} \equiv s. \tag{4}$$

Lemma 2.10 *Relation $\equiv_\alpha$ is an equivalence relation.*

We leave out the elementary proof.

We now present, without proof, a result about applying series of substitutions.

Lemma 2.11 *For any term t and substitutions $\sigma_1, \ldots, \sigma_n$ there exists a substitution θ such that $t\sigma_1 \cdots \sigma_n \equiv_\alpha t\theta$.*

This lemma means that, as long as concrete variable names are immaterial, one may as well use single substitutions in the place of series of substitutions. We exploit this fact when designing the algorithm in Sec. 4.3.

2.3 Generalisation of terms

Intuitively a term s is more general than a term t if t may be obtained from s by applying a series of substitutions, that is, t is obtained by instantiating s. The next definition makes this precise.

Def. 2.12 (Term generality order: $\succeq, \succ$) *Relations $\succeq, \succ$ are the smallest relations on terms such that*

$$\begin{array}{ll} s \succeq t & \text{if } \exists\theta_1 \cdots \theta_n (s\theta_1 \cdots \theta_n \equiv t) \\ s \succ t & \text{if } s \succeq t \wedge t \not\succeq s \end{array}$$

Lemma 2.13 *If $s \succeq t$ then $t \not\succ s$.*

Proof. By induction on $n = \mathcal{H}(t)$ (Def. 2.4).
Basic step: $n = 0$, meaning $t = x$. Then the only possibility is $s = y$. Have $y \succeq x$, $x \not\succeq y$ as required.
Induction hypothesis: The lemma holds for $n \leq k$.
Induction step: Must prove the case $n = k + 1$. Consider $t = f(t_1, \ldots, t_m)$, $\mathcal{H}(t) = k + 1$. There are two cases for s.

(i) $s = x$. Then $x \succeq f(t_1, \ldots, t_m)$, $f(t_1, \ldots, t_m) \not\succeq x$ as required.
(ii) $s = f(s_1, \ldots, s_m)$. By induction $s_i \succeq t_i$, $t_i \not\succeq s_i$ for $i = 1, \ldots, m$ and from this $s \succeq t$, $t \not\succeq s$ follows as required. □

Note that the converse does not hold: take incomparable terms, for example c, d.

We obtain the next lemma from Plotkin [6]. It establishes the anti-symmetry of $\succeq$ with respect to the set of term equivalence classes modulo $\equiv_\alpha$.

Lemma 2.14 $s \succeq t \wedge t \succeq s \iff s \equiv_\alpha t$.

3 Machinery

This section introduces some theoretical machinery that will be useful in later sections. We make sequences of terms into syntactic objects and extend some previously defined concepts, notably substitution and generality orders, to these objects. The motivation for this is to be able to manipulate sequences of terms while preserving the order in which they are listed. This again allows for a detailed algorithm, from which an implementation anti-unfying n terms can be derived straightforwardly. We also introduce inverse substitution, the dual of substitution.

3.1 Term sequences

A *term sequence* is a tuple of terms, written $(t_1, \ldots, t_n)$, $n \geq 0$; in particular $()$ denotes the empty term sequence. The symbolic notation for term sequences is derived from that of terms by adding $\bar{}$ to term meta-variables; for example, $\bar{s}, \bar{t}$ denote term sequences. We consider a term t and a term sequence $\bar{t}$ to be unrelated, that is, we may use both meta-variables in an expression without implying any relationship between t and $\bar{t}$. The length $|\bar{t}|$ of a term sequence $\bar{t}$ is a natural number defined thus: $|(t_1, \ldots, t_n)| = n$. While term sequences serve more important purposes we sometimes use them to obtain a more compact notation for terms, writing $f(\bar{t})$ for a term $f(t_1, \ldots, t_n)$.

In addition we allow an equivalent notation for sequences as an abstract datatype. In this notation :: is the infix sequence constructor. As an example we have the following correspondence between the sequence notations: $(s, t) = s :: t :: ()$. This is useful in definitions recursing over sequences.

The set of variables in a term sequence is defined as follows.

Def. 3.1 (Term sequence variables, $\mathcal{V}(\bar{t})$)

$$\begin{aligned}
\mathcal{V}\big(()\big) &= \emptyset \\
\mathcal{V}\big((x,\ldots,x)\big) &= \{x\} \\
\mathcal{V}\big(f(t_1 :: \bar{t}_1),\ldots,f(t_n :: \bar{t}_n)\big) &= \mathcal{V}\big((t_1,\ldots,t_n)\big) \cup \mathcal{V}\big(\big(g(\bar{t}_1),\ldots,g(\bar{t}_n)\big)\big), \\
&\qquad\qquad \mathcal{A}(g) = \mathcal{A}(f) - 1 \\
\mathcal{V}\big(\bar{t}\big) &= \emptyset \qquad \textit{otherwise}
\end{aligned}$$

Note the trick of introducing a new function symbol g on the right-hand side in the equation where the function symbol f is on the left-hand side, and taking $\mathcal{A}(g) = \mathcal{A}(f) - 1$. This allows the definition to be recursive.

Ex. 3.2

$$\begin{aligned}
\mathcal{V}\big((f(x),f(x))\big) &= x & \mathcal{V}\big((f(x),f(y))\big) &= \emptyset \quad \text{if } x \neq y \\
\mathcal{V}\big((\ldots,c,\ldots)\big) &= \emptyset & \mathcal{V}\big((f(\ldots),g(\ldots))\big) &= \emptyset \quad \text{if } f \neq g
\end{aligned} \qquad \diamond$$

Syntactic measures on term sequences, for use in inductive proofs:

Def. 3.3 (Term sequence height, $\mathcal{H}(\bar{t})$; term sequence size, $\mathcal{S}(\bar{t})$)

$$\begin{aligned}
\mathcal{H}\big(()\big) &= 0 \qquad \mathcal{S}\big(()\big) = 0 \\
\mathcal{H}\big((t_1,\ldots,t_n)\big) &= \max\big\{\mathcal{H}(t_1),\ldots,\mathcal{H}(t_n)\big\} \quad (n > 0) \\
\mathcal{S}\big((t_1,\ldots,t_n)\big) &= \mathcal{S}(t_1) + \cdots + \mathcal{S}(t_n) \quad (n > 0)
\end{aligned}$$

Sometimes, we drop the 'term' prefix in 'term sequence' since they are the only kind of sequence considered.

We now define binding and substitution for term sequences. The syntactic objects sequence binding and sequence substitution are analogous to those for terms, but the definition of substitution application requires more care. A *sequence binding* is a pair of a term sequence $\bar{t}$ and a variable x, written $\bar{t}/x$. A *sequence binding set*, denoted $\bar{B}$, is a finite set of sequence bindings $\{\bar{t}_1/x_1,\ldots,\bar{t}_n/x_n\}$ such that variables $x_1,\ldots,x_n$ are distinct.

The definitions of sequence domain, sequence range and the related notation are trivial extensions of those for terms (Def. 2.5). A sequence binding set is a *sequence substitution*, denoted $\bar{\theta}$, if it satisfies the following, called the *well-formedness criterion for sequences substitutions*:

$$\begin{aligned}
&\mathcal{D}(\bar{\theta}) \cap \mathcal{R}(\bar{\theta}) = \emptyset \text{ and} \\
&\text{if } \bar{\theta} = \{\bar{t}_1/x_1,\ldots,\bar{t}_n/x_n\} \text{ then } |\bar{t}_1| = \cdots = |\bar{t}_n|
\end{aligned} \qquad (\bar{2})$$

In particular the empty set is a sequence substitution, called the empty sequence substitution, and written as $\{\}$.[2] We use $\bar{\sigma}, \bar{\rho}$ to denote sequence substitutions.

The notion of length is extended to sequence substitutions by taking $|\{\bar{t}_1/x_1,\ldots,\bar{t}_n/x_n\}| = |\bar{t}_1|$ while $|\{\}|$ is assumed to be such that equation $|\bar{t}| =$

[2] That $\{\}$ is also the empty term substitution does not create problems.

$|\{\}|$ holds for any $\bar{t}$.[3] We say that a term sequence $\bar{t}$ and a sequence substitution $\bar{\theta}$ are *compatible* when $|\bar{t}| = |\bar{\theta}|$. Note that the empty sequence substitution $\{\}$ is compatible with all term sequences.

Def. 3.4 (Term sequence substitution, $\bar{t}\bar{\theta}$) *Expression $\bar{t}\bar{\theta}$ is only well-formed when $\bar{\theta}$ is compatible with $\bar{t}$.*

$$
\begin{aligned}
(x,\ldots,x)\bar{\theta} &= \bar{s} && \text{if } (\bar{s}/x) \in \bar{\theta} \\
(x,\ldots,x)\bar{\theta} &= (x,\ldots,x) && \text{if } (\_/x) \notin \bar{\theta} \\
\big(f(t_1 :: \bar{t}_1),\ldots,f(t_n :: \bar{t}_n)\big)\bar{\theta} &= \big(f(s_1 :: \bar{s}_1),\ldots,f(s_n :: \bar{s}_n)\big) \\
&\quad \text{where } (s_1,\ldots,s_n) = (t_1,\ldots,t_n)\bar{\theta} \\
&\quad \text{and } \big(g(\bar{s}_1),\ldots,g(\bar{s}_n)\big) = \big(g(\bar{t}_1),\ldots,g(\bar{t}_n)\big)\bar{\theta}, \\
&\qquad \mathcal{A}(g) = \mathcal{A}(f) - 1 \\
\bar{t}\bar{\theta} &= \bar{t} && \text{otherwise}
\end{aligned}
$$

The remarks regarding composition and notation following Def. 2.6 also apply here.

Ex. 3.5 This example illustrates that application of a sequence substitution to a sequence $\bar{t}$ cannot in general be reduced to application of corresponding regular substitutions on individual terms of $\bar{t}$.

$$
\begin{aligned}
\big(f(x),f(x)\big)\big\{(c,d)/x\big\} &= \big(f(c),f(d)\big) \\
\big(f(x),g(x)\big)\big\{(c,d)/x\big\} &= \big(f(x),g(x)\big)
\end{aligned}
$$
$\diamond$

We extend $\succeq$ (Def. 2.12) to term sequences as follows:

Def. 3.6 (Term sequence generality orders: $\succeq$)

$$
\begin{aligned}
(s_1,\ldots,s_n) &\succeq (t_1,\ldots,t_n) && \text{if } s_1 \succeq t_1 \wedge \cdots \wedge s_n \succeq t_n \\
\bar{s} &\succ \bar{t} && \text{if } \bar{s} \succeq \bar{t} \wedge \bar{t} \not\succeq \bar{s}
\end{aligned}
$$

We allow the shorthand $s \succeq (t_1,\ldots,t_n)$ meaning $(s,\ldots,s) \succeq (t_1,\ldots,t_n)$.

3.2 Inverse substitution

Informally, inverse substitution takes a term t and a substitution θ and returns a term $t\theta^{-1}$ where, for each binding (s/x) and each occurrence of s in t, that occurrence is replaced with x. Some conditions must be met for inverse substitution to be defined.

Def. 3.7 (Inverse substitution, $t\theta^{-1}$) *Expression $t\theta^{-1}$ is only well-formed when θ is injective and $\mathcal{D}(()\theta) \cap \mathcal{V}(t) = \emptyset$.*

$$
\begin{aligned}
t\theta^{-1} &= x && \text{if } (t/x) \in \theta \\
f(t_1,\ldots,t_n)\theta^{-1} &= f(t_1\theta^{-1},\ldots,t_n\theta^{-1}) && \text{if } (f(t_1,\ldots,t_n)/\_) \notin \theta \\
y\theta^{-1} &= y && \text{if } (y/\_) \notin \theta
\end{aligned}
$$

[3] It can be argued we should replace the length function with a relation; however, this would not any anything significant new insights.

As expected, the following properties hold for inverse substitution:

$$t\{\}^{-1} = t$$
$$s\{t_1/x_1, \ldots, t_n/x_n\}^{-1} = \big(\cdots\big(s\{t_1/x_1\}^{-1}\big)\cdots\big)\{t_n/x_n\}^{-1}$$

The first is trivial. The second follows from well-formedness (2) and injectivity of θ.

Ex. 3.8

$$g(c)\{c/x\}^{-1} = g(x)$$
$$g(c)\{g(c)/x\}^{-1} = x$$
$$f(y,c)\{y/x\}^{-1} = f(x,c)$$
$$f(c,h(c,y))\{c/x\}^{-1} = f(x,h(x,y))$$
$$f(x,c)\{c/x\}^{-1} \text{ undefined}$$

Note that $f(c,g(c))\{c/xg(c)/y\}^{-1} = f(x,y)$ and not $f(x,g(x))$ since the second binding matches before the subterms of g are examined. $\diamond$

We extend the concept of inverse substitution to term sequences.

Def. 3.9 (Inverse term sequence substitution, $\boldsymbol{t\theta^{-1}}$) *Expression $\bar{t}\bar{\theta}^{-1}$ is only well-formed when $\bar{\theta}$ is injective, compatible with $\bar{t}$ and $\mathcal{D}(()\bar{\theta}) \cap \mathcal{V}(\bar{t}) = \emptyset$.*

$$\begin{aligned}
\bar{t}\bar{\theta}^{-1} &= (x,\ldots,x) \qquad \textit{if } (\bar{t}/x) \in \bar{\theta} \\
\big(f(t_1 :: \bar{t}_1),\ldots,f(t_n :: \bar{t}_n)\big)\bar{\theta}^{-1} &= \big(f(s_1 :: \bar{s}_1),\ldots,f(s_n :: \bar{s}_n)\big) \\
&\quad \textit{if } ((f(t_1 :: \bar{t}_1),\ldots,f(t_n :: \bar{t}_n))/_) \notin \bar{\theta} \\
&\quad \textit{where } (s_1,\ldots,s_n) = (t_1,\ldots,t_n)\bar{\theta}^{-1} \\
&\quad \textit{and } \big(g(\bar{s}_1),\ldots,g(\bar{s}_n)\big) = \big(g(\bar{t}_1),\ldots,g(\bar{t}_n)\big)\bar{\theta}^{-1}, \\
&\qquad \mathcal{A}(g) = \mathcal{A}(f) - 1 \\
\bar{t}\bar{\theta}^{-1} &= \bar{t} \qquad \textit{otherwise}
\end{aligned}$$

Similar properties to those for regular inverse substitution hold.

Inverse substitution has many notable properties, especially in relation to substitution. We only need the following, which can be generalised to sequences.

Lemma 3.10 *Let t be arbitrary, let θ be injective and such that $\mathcal{V}(t) \cap \mathcal{D}(()\theta) = \emptyset$. Then $t\theta^{-1} \succeq t$.*

Proof. By induction on $n = \mathcal{H}(t)$.
Basic step: $n = 0$, meaning $t = x$. There are two cases:

(i) $(x/\_) \notin \theta$. Here $x\theta^{-1} = x$ and the lemma trivially holds since $x \succeq x$.
(ii) $(x/y) \in \theta$. Now $x\theta^{-1} = y$ and $y \succeq x$ holds.

Induction hypothesis: The lemma holds for $n \leq k$.
Induction step: Must prove the case $n = k + 1$. Consider $t = f(t_1, \ldots, t_m)$, $\mathcal{H}(t) = k + 1$. Two cases:

(i) $(t/\_) \notin \theta$. We get:[4]

$$\begin{array}{lll} f(t_1, \ldots, t_m)\theta^{-1} & = & \{\text{Def. 3.7}\} \\ f(t_1\theta^{-1}, \ldots, t_m\theta^{-1}) & \succeq & \{\text{induction}\} \\ f(t_1, \ldots, t_m) & & \end{array}$$

(ii) $(t/y) \in \theta$. Now $t\theta^{-1} = y$ and $y \succeq t$ holds. □

4 Anti-unification

We have now developed enough theory to define anti-unification, or the least generalisation, of a set of terms, and to give algorithms to compute it.

4.1 Definition of least generalisation

The next definition and example are adapted from Plotkin [6].

Def. 4.1 (Least generalisation, lg(T)) *Let T be a non-empty set of terms. A term s is a* least generalisation *of T, denoted $s \in \mathrm{lg}(T)$, if the following properties hold:*

$$\forall t \in T(s \succeq t) \qquad \text{(generalisation)}$$

$$\forall u\big(\forall t \in T(u \succeq t) \implies u \succeq s\big) \qquad \text{(least)}$$

Ex. 4.2

$$f\big(z_1, x, g(z_1, z_2)\big) \in \mathrm{lg}\big(\{f\big(c, x, g(c, x)\big)f\big(d, x, g(d, y)\big)\}\big) \qquad \diamond$$

Ex. 4.3 One can have $s_1 \succeq t$ and $s_2 \succeq t$ without s_1, s_2 being comparable by $\succeq$: take $s_1 = f(x, c), s_2 = f(c, y), t = f(c, c)$. ◇

The least condition implies the following.

$$\neg\exists u\big(\forall t \in T(u \succeq t) \wedge s \succ u\big) \qquad (5)$$

That is, there can be no term u such that u is a generalisation of T and a least generalisation s is strictly more general that u. Note that (5) does not imply the least condition; Example 4.3 shows why.

Proof. We use the shorthand $s \succeq T$ for $\forall t \in T(s \succeq t)$.

$$\begin{array}{lll} \forall u(u \succeq T \implies u \succeq s) & \implies & \{\text{Lem. 2.13}\} \\ \forall u(u \succeq T \implies s \not\succ u) & \iff & \{\text{Logic}\} \\ \forall u(u \not\succeq T \vee s \not\succ u) & \iff & \{\text{Logic}\} \\ \neg\exists u\neg(u \not\succeq T \vee s \not\succ u) & \iff & \{\text{Logic}\} \\ \neg\exists u(u \succeq T \wedge s \succ u) & \iff & \{\text{Logic}\} \end{array}$$

□

[4] Inspired by Bird and de Moor [1] we sometimes write proof in a calculational style.

Assumption: $|\bar{t}| = n, n > 0$.

1. $(s_1, \ldots, s_n) := \bar{t}$, $\bar{\theta} := \{\}$.
2. If $s_1 = \cdots = s_n$ then halt with s_1 as the answer.
3. Let w be the leftmost-highest position at which $s_1, \ldots, s_n$ differ. Perform $u_1 := s_1[w]$, ..., $u_n := s_n[w]$.
4. If $((u_1, \ldots, u_n)/x) \in \bar{\theta}$ then perform $s_1[w] := x, \ldots, s_n[w] := x$, and go to 2.
5. Let z be a fresh variable, and perform $\bar{\theta} := \bar{\theta}\{(u_1, \ldots, u_n)/z\}$, $s_1[w] := z, \ldots, s_n[w] := z$, and go to 2.

Figure 1: Imperative anti-unification algorithm.

The next lemma establishes that $\mathrm{lg}(T)$ is unique modulo variable renaming.

Lemma 4.4 (Uniqueness of $\mathrm{lg}(T)$ modulo $\equiv_\alpha$) *Let T be a non-empty term set. If $s_1, s_2 \in \mathrm{lg}(T)$ then $s_1 \equiv_\alpha s_2$.*

Proof. By definition $s_1 \succeq s_2$ and $s_2 \succeq s_1$ so the lemma follows directly from Lemma 2.14. □

4.2 The Plotkin-Reynolds imperative algorithm

Fig. 1 shows the imperative algorithm, adapted from Reynolds [7, Theorem 1]. The notation $s[w]$ means the subterm of s at position w. Leftmost-highest position means the position leftmost and closest to the head of the term.

4.3 The functional algorithm

Let $\bar{t}$ be a term sequence $(t_1, \ldots, t_n), n > 0$. Then the least generalisation of the term set $\{t_1, \ldots, t_n\}$ is $\mathsf{au}(\bar{t})$ where $\mathsf{au}(\bar{t})$ is computed by the functional anti-unification algorithm given in Fig. 2. Expression $\langle a, b\rangle$ denotes a pair of objects a, b, and $\mathrm{fst}, \mathrm{snd}$ are operations for de-constructing pairs: $\mathrm{fst}\big(\langle a, b\rangle\big) = a; \mathrm{snd}\big(\langle a, b\rangle\big) = b$.

The algorithm in Fig. 2 expects the term sequences $\bar{t}$ to consist of ground terms, but it can be adapted to sequences of general terms as follows: Replace all occurrences of a variable x in $\bar{t}$ with a new constant c^x before using the algorithm and, after using the algorithm, replace c^x by x in the result.

Equation (7) covers the case $\mathsf{au}_{\bar{\theta}}((c, \ldots, c), \bar{\theta})$. This equation could be replaced with the more specific

$$\mathsf{au}_{\bar{\theta}}\big((c, \ldots, c), \bar{\theta}\big) = \langle c, \bar{\theta}\rangle$$

but (7) may be more efficient.

Assumptions for $\mathsf{au}(\bar{t})$: $|\bar{t}| > 0$.
Assumptions for $\mathsf{au}_{\bar{\theta}}(\bar{t}, \bar{\theta})$: $|\bar{t}| > 0$, $\bar{\theta}$ injective, $\bar{\theta}$ compatible with $\bar{t}$.

$$\mathsf{au}(\bar{t}) = s \quad \text{where } s = \mathrm{fst}\big(\mathsf{au}_{\bar{\theta}}(\bar{t}, \{\})\big) \tag{6}$$

$$\mathsf{au}_{\bar{\theta}}\big((t, \ldots, t), \bar{\theta}\big) = \langle t, \bar{\theta}\rangle \tag{7}$$

$$\mathsf{au}_{\bar{\theta}}\big(\big(f(t_1 :: \bar{t}_1), \ldots, f(t_n :: \bar{t}_n)\big), \bar{\theta}\big) = \big\langle f(s :: \bar{s}), \bar{\theta}''\big\rangle \tag{8}$$

if (7) does not apply

where $\langle s, \bar{\theta}'\rangle = \mathsf{au}_{\bar{\theta}}\big((t_1, \ldots, t_n), \bar{\theta}\big)$

and $\langle g(\bar{s}), \bar{\theta}''\rangle = \mathsf{au}_{\bar{\theta}}\big(\big(g(\bar{t}_1), \ldots, g(\bar{t}_n)\big), \bar{\theta}'\big), \quad \mathcal{A}(g) = \mathcal{A}(f) - 1$

$$\mathsf{au}_{\bar{\theta}}(\bar{t}, \bar{\theta}) = \langle x, \bar{\theta}\rangle \quad \text{if (7) and (8) do not apply and } (\bar{t}/\_) \in \bar{\theta} \tag{9}$$

where $(\bar{t}/x) \in \bar{\theta}$

$$\mathsf{au}_{\bar{\theta}}(\bar{t}, \bar{\theta}) = \langle z, \bar{\theta}\{\bar{t}/z\}\rangle \quad \text{if (7), (8) and (9) do not apply} \tag{10}$$

where z is a fresh variable

Figure 2: Functional anti-unification algorithm.

Note that the assumption $\bar{\theta}$ injective and condition $(\bar{t}/\_) \in \bar{\theta}$ ensures that $(\bar{t}/x) \in \bar{\theta}$ uniquely determines x in equation (9). Later we prove that the assumptions always hold given that they hold initially, and that all substitutions computed by the algorithm are well-formed (Lemma 5.3, cases i and ii).

5 Correctness of the functional algorithm

In this section we prove that the functional anti-unification algorithm (Fig. 2) complies with the definition of least generalisation (Def. 4.1). To to this we start by proving termination of the algorithm and that it preserves certain invariants (Sec. 5.1), and then we go on to prove the two properties required for a least generalisation (Secs. 5.2, 5.3).

5.1 Termination and invariants

The termination of au follows trivially from the following lemma about $\mathsf{au}_{\bar{\theta}}$.

Lemma 5.1 (Termination) *Let $\bar{t}$ be non-empty and let $\bar{\theta}$ be compatible with $\bar{t}$. Then the computation $\mathsf{au}_{\bar{\theta}}(\bar{t}, \bar{\theta})$ terminates.*

Proof. Note first that $\bar{\theta}$, the second argument to $\mathsf{au}_{\bar{\theta}}$, only determines whether or not (9) applies and does not affect non-termination.

Consider the first argument to $\mathsf{au}_{\bar{\theta}}$. Size, Def. 3.3, induces a well-founded relation on term sequences since term sequences are tuples of finite terms. For $\mathsf{au}_{\bar{\theta}}$ equation (8) is the critical one since this equation only is recursive. Both

recursive calls in (8) apply to sequences with strictly smaller size,

$$\begin{aligned}\mathcal{S}\big((t_1,\ldots,t_n)\big) &< \mathcal{S}\big(\big(f(t_1 :: \bar{t}_1),\ldots,f(t_n :: \bar{t}_n)\big)\big),\\ \mathcal{S}\big(\big(g(\bar{t}_1),\ldots,g(\bar{t}_n)\big)\big) &< \mathcal{S}\big(\big(f(t_1 :: \bar{t}_1),\ldots,f(t_n :: \bar{t}_n)\big)\big).\end{aligned}$$

This means that $\mathsf{au}_{\bar{\theta}}$ cannot be called indefinitely. □

We take advantage of this fact in the following by proving properties of $\mathsf{au}_{\bar{\theta}}$ (and au) by induction over the number of calls to $\mathsf{au}_{\bar{\theta}}$.[5]

We need the following relation on substitutions later.

Def. 5.2 (Substitution order, $\boldsymbol{\theta \geq \sigma}$) *$\theta \geq \sigma$ if $\exists\rho(\theta = \sigma\rho)$.*

The definition of substitution order generalises straightforwardly to term sequences substitutions.[6] The order is transitive by substitution composition.

Next we prove some invariants for arguments to and results from $\mathsf{au}_{\bar{\theta}}$. The purpose of this lemma is to ensure that we may apply the induction hypothesis in the proofs of Lemma 5.4 and Lemma 5.6.

Lemma 5.3 (Invariants) *Let $\bar{t}$ be non-empty and let $\bar{\theta}$ be injective, compatible with $\bar{t}$ and such that $\mathcal{V}(\bar{t}) \cap \mathcal{D}(()\bar{\theta}) = \emptyset$. If $\langle \bar{s}, \bar{\sigma}\rangle = \mathsf{au}_{\bar{\theta}}(\bar{t}, \bar{\theta})$ then the following holds.*

(i) $\bar{\sigma}$ is well-formed and $\bar{\sigma} \geq \bar{\theta}$;
(ii) $\bar{\sigma}$ is injective;
(iii) $\bar{\sigma}$ is compatible with $\bar{t}$;
(iv) $\mathcal{V}(\bar{t}) \cap \mathcal{D}(()\bar{\sigma}) = \emptyset$.

Proof. (Sketch)

(i) Substitution $\bar{\theta}$ is passed as argument to $\mathsf{au}_{\bar{\theta}}$ and a possibly differing substitution is in the returned tuple. Equations (7) and (9) pass $\bar{\theta}$ on directly. Equation (10) passes the composition $\bar{\theta}\{\bar{t}/z\}$. This is a well-formed substitution since z is a fresh variable and $\bar{\theta}$ is well-formed by assumption. Also, $\bar{\theta}\{\bar{t}/z\} \geq \bar{\theta}$. Equation (8) does not itself pass or compute a substitution different from $\bar{\theta}$ but recursive calls may do so; this is handled by induction on the number of calls to $\mathsf{au}_{\bar{\theta}}$.

(ii) By assumption $\bar{\theta}$ is injective. The only equation that adds to $\bar{\theta}$ is (10), and it does so provided that (9) does not apply. In this case $(\bar{t}/\_) \notin \bar{\theta}$ and therefore $\bar{\theta}\{\bar{t}/z\}$ is also injective. An induction on the number of calls to $\mathsf{au}_{\bar{\theta}}$ is needed for (8).

(iii) By assumption $\bar{\theta}$ is compatible with $\bar{t}$. The only equation that adds to $\bar{\theta}$ is (10), and $\bar{\theta}\{\bar{t}/z\}$ is also compatible with $\bar{t}$. Induction on the number of calls to $\mathsf{au}_{\bar{\theta}}$ completes the argument.

[5] Reynolds does the same in his correctness proof (Theorem 2) [7]. Alternatively one could do well-founded induction over $\mathcal{S}(\bar{t})$ as described for instance by Mitchell [4, §1.8.3].

[6] Remark that substitution order here is syntactical, that is, it compares just the substitution objects themselves. A semantical order would be based on the effect of substitutions on terms.

(iv) By assumption $\mathcal{V}(\bar{t}) \cap \mathcal{D}(()\bar{\theta}) = \emptyset$. Since $\bar{\sigma} \geq \bar{\theta}$ and only fresh variables are added, the property holds. □

Observation: The algorithm does not rely on the concrete order in which terms are listed in $\bar{t}$ and $\bar{\theta}$. That is, the order that terms are listed in $\bar{t}$ is immaterial to the result computed by au.

5.2 Generalisation property

We reformulate the generalisation property from Def. 4.1 with sequences instead of sets: Let $\bar{t}$ be non-empty. Then the least generalisation s of $\bar{t}$ must satisfy the following:

$$s \succeq \bar{t} \qquad \text{(generalisation)}$$

A direct inductive proof of our generalisation result, Theorem 5.5, does not work due to the accumulated argument $\bar{\theta}$ in the algorithm. Instead we prove a slightly more general result, Lemma 5.4, and derive the theorem as an easy consequence of the lemma.

Lemma 5.4 *Let $\bar{t}$ be non-empty and let $\bar{\theta}$ be injective, compatible with $\bar{t}$ and such that $\mathcal{V}(\bar{t}) \cap \mathcal{D}(()\bar{\theta}) = \emptyset$. Then* $\mathrm{fst}\big(\mathsf{au}_{\bar{\theta}}(\bar{t}, \bar{\theta})\big) \succeq \bar{t}$.

Proof. Without loss of generality assume that $|\bar{t}| = 2$, that is, $\bar{t} = (t_1, t_2)$ for terms t_1, t_2. Termination of $\mathsf{au}_{\bar{\theta}}$ is already established. The proof is by induction on the number n of recursive calls to $\mathsf{au}_{\bar{\theta}}$.
Basic step: $n = 0$. There are three cases to consider, corresponding to conditions for applying the non-recursive equations (7), (9) and (10). Equation (8) does not apply as it involves recursive calls to $\mathsf{au}_{\bar{\theta}}$, that is, $n > 0$.

(i) $\bar{t} = (t, t)$.

$$\begin{array}{ll} \mathrm{fst}\big(\mathsf{au}_{\bar{\theta}}\big((t,t),\bar{\theta}\big)\big) \succeq (t,t) & = \{(7)\} \\ \mathrm{fst}\big(\langle t, \bar{\theta}\rangle\big) \succeq (t,t) & = \{\text{Def. of fst}\} \\ t \succeq (t,t) & = \{\text{Def. 3.6}\} \\ \text{True} & \end{array}$$

(ii) $\bar{t} = (t_1, t_2), t_1 \neq t_2, (\bar{t}/x) \in \bar{\theta}$.

$$\begin{array}{ll} \mathrm{fst}\big(\mathsf{au}_{\bar{\theta}}(\bar{t},\bar{\theta})\big) \succeq \bar{t} & = \{(9)\} \\ \mathrm{fst}\big(\langle x, \bar{\theta}\rangle\big) \succeq \bar{t} & = \{\text{Def. of fst}\} \\ x \succeq \bar{t} & = \{\text{Def. 3.6}\} \\ \text{True} & \end{array}$$

(iii) $\bar{t} = (t_1, t_2), t_1 \neq t_2, (\bar{t}/\_) \notin \bar{\theta}$, $head(t_1) \neq head(t_2)$. (The latter is not used here.)

$$\begin{array}{ll} \mathrm{fst}\big(\mathsf{au}_{\bar{\theta}}\big(\bar{t},\bar{\theta}\big)\big) \succeq \bar{t} & = \{(10),\ \mathcal{V}(\bar{t}) \cap \mathcal{D}(()\bar{\theta}) = \emptyset\} \\ \mathrm{fst}\big(\langle z, \bar{\theta}\{\bar{t}/z\}\rangle\big) \succeq \bar{t} & = \{\text{Def. of fst}\} \\ z \succeq \bar{t} & = \{\text{Def. 3.6}\} \\ \text{True} & \end{array}$$

Induction hypothesis: The lemma holds for $n \leq k$.
Induction step: Must prove the case $n = k + 1$, that is, computations involving $k + 1$ recursive calls to $\mathsf{au}_{\bar{\theta}}$; only (8) is relevant since it is the only recursive equation. We have $\bar{t} = \big(f(t_1 :: \bar{t}_1), f(t_2 :: \bar{t}_2)\big), f(t_1 :: \bar{t}_1) \neq f(t_2 :: \bar{t}_2)$.

$$
\begin{array}{ll}
 & \mathrm{fst}\big(\mathsf{au}_{\bar{\theta}}\big(\big(f(t_1 :: \bar{t}_1), f(t_2 :: \bar{t}_2)\big), \bar{\theta}\big)\big) \succeq \big(f(t_1 :: \bar{t}_1), f(t_2 :: \bar{t}_2)\big) \\
= & \quad \{\ (8)\ \} \\
 & \mathrm{fst}\big(\big\langle f(s :: \bar{s}), \bar{\theta}''\big\rangle\big) \succeq \big(f(t_1 :: \bar{t}_1), f(t_2 :: \bar{t}_2)\big) \\
 & \quad \mathsf{where}\ \langle s, \bar{\theta}'\rangle = \mathsf{au}_{\bar{\theta}}\big((t_1, t_2), \bar{\theta}\big) \\
 & \quad\quad \mathsf{and}\ \big\langle g(\bar{s}), \bar{\theta}''\big\rangle = \mathsf{au}_{\bar{\theta}}\big(\big(g(\bar{t}_1), g(\bar{t}_2)\big), \bar{\theta}'\big) \\
= & \quad \{\ \text{Def. of fst}\ \} \\
 & f(s :: \bar{s}) \succeq \big(f(t_1 :: \bar{t}_1), f(t_2 :: \bar{t}_2)\big) \\
= & \quad \{\ \text{notation}\ \} \\
 & \big(f(s :: \bar{s}), f(s :: \bar{s})\big) \succeq \big(f(t_1 :: \bar{t}_1), f(t_2 :: \bar{t}_2)\big) \\
= & \quad \{\ \text{Def. 3.6, notation}\ \} \\
 & s \succeq (t_1, t_2) \wedge g(\bar{s}) \succeq \big(g(\bar{t}_1), g(\bar{t}_2)\big) \\
= & \quad \{\ \text{Def. of fst, } s, \bar{s}\ \} \\
 & \mathrm{fst}\big(\mathsf{au}_{\bar{\theta}}\big((t_1, t_2), \bar{\theta}\big)\big) \succeq (t_1, t_2) \\
 & \wedge\ \mathrm{fst}\big(\mathsf{au}_{\bar{\theta}}\big(\big(g(\bar{t}_1), g(\bar{t}_2)\big), \bar{\theta}'\big)\big) \succeq \big(g(\bar{t}_1), g(\bar{t}_2)\big) \\
= & \quad \{\ \text{Lem. 5.3, ind. hyp. \& logic}\ \} \\
 & \mathrm{True}
\end{array}
$$

□

It is possible to prove a more general result with Lemma 5.4 as a special case. That result would tighten the bound on the computed term as follows: Assume $\bar{t}$ non-empty, $\bar{\theta}$ injective and compatible with $\bar{t}$. Then

$$\langle s, \bar{\theta}'\rangle = \mathsf{au}_{\bar{\theta}}(\bar{t}, \bar{\theta}) \implies (s, \ldots, s)\bar{\theta}' \equiv \bar{t}$$

where $\equiv$ is defined on sequences in the obvious way. To prove this a more involved induction hypothesis must be used, similar to that of Lemma 5.6. However, we do not need such generality to derive the theorem below.

We are ready to prove the generalisation property, representing the first part of Def. 4.1.

Theorem 5.5 (Generalisation) *Let $\bar{t}$ be non-empty. Then* $\mathsf{au}(\bar{t}) \succeq \bar{t}$.

Proof. Immediate from (6) and Lemma 5.4 by taking $\bar{\theta} = \{\}$. □

5.3 Least property

We reformulate the least property from Def. 4.1 with sequences instead of sets: Let $\bar{t}$ be non-empty. Then the least generalisation s of $\bar{t}$ must satisfy the following:

$$\forall u(u \succeq \bar{t} \implies u \succeq s) \qquad \text{(least)}$$

As in the previous section we prove a more general result and then derive Theorem 5.7, the theorem stating the least property. One could try to to proceed as in Sec. 5.2: Replacing s with $\mathrm{fst}\big(\mathsf{au}_{\bar\theta}(\bar t,\bar\theta)\big)$ in the above formula, we get

$$\forall u\big(u \succeq \bar t \implies u \succeq \mathrm{fst}\big(\mathsf{au}_{\bar\theta}(\bar t,\bar\theta)\big)\big) \tag{11}$$

and then one may proceed to prove this equation. But taking $\bar t = (c,\ldots,c)$, $\bar\theta = \{\bar t/x\}$ and $u = c$ gives a counterexample to (11). We need to take into account the effect of the initial argument $\bar\theta$ to $\mathsf{au}_{\bar\theta}$. The formulation of the next lemma is inspired by the fact that $\mathrm{fst}\big(\mathsf{au}_{\bar\theta}(\bar t,\bar\theta)\big) \succeq \bar t\bar\theta^{-1}$. Since we do not use this fact, we leave out its proof.

Lemma 5.6 *Let $\bar t$ be non-empty and let $\bar\theta,\bar\sigma$ be injective, compatible with $\bar t$ and such that $\mathcal{V}(\bar t)\cap\mathcal{D}(()\bar\theta) = \emptyset$ and $\mathcal{V}(\bar t)\cap\mathcal{D}(()\bar\sigma) = \emptyset$. If $\langle s,\bar\rho\rangle = \mathsf{au}_{\bar\theta}(\bar t,\bar\theta)$ and $\bar\sigma \geq \bar\rho$ then the following holds:*

$$\forall u\big((u,\ldots,u) \succeq \bar t\bar\sigma^{-1} \implies u \succeq s\big) \tag{$\dagger$}$$

Proof. Without loss of generality assume that $\bar t = (t_1,t_2)$. The proof is by induction on the number n of recursive calls to $\mathsf{au}_{\bar\theta}$. In the various cases we refer to corresponding cases in the proof of Lemma 5.4 for the value of $\langle s,\bar\rho\rangle$. Also, we know from Lemma 5.3, case i that $\bar\rho \geq \bar\theta$.
Basic step: $n = 0$. There are three cases to consider, corresponding to conditions for applying the non-recursive equations (7), (9) and (10).

(i) $\bar t = (t,t)$. From the proof of Lemma 5.4, case i we have $\langle s,\bar\rho\rangle = \langle t,\bar\theta\rangle$ and ($\dagger$) may be specialised:
$\forall u\big((u,u) \succeq (t,t)\bar\sigma^{-1} \implies u \succeq t\big)$ $\quad == \{\text{Def. 3.6}\}$
$\forall u\big((u,u) \succeq (t,t)\bar\sigma^{-1} \implies (u,u) \succeq (t,t)\big)$ $\quad == \{\text{Lem. 3.10 \& trans. of } \succeq\}$
True

(ii) $\bar t = (t_1,t_2), t_1 \neq t_2, (\bar t/x) \in \bar\theta$. By assumption $\bar\sigma$ injective, and since $\bar\rho \geq \bar\theta$ and $\geq$ is transitive, we have $\bar\sigma \geq \bar\theta$. This, together with $(\bar t/x) \in \bar\theta$, implies that $(\bar t/\_) \notin (\bar\sigma \setminus \bar\theta)$. By the proof of Lemma 5.4, case ii we have $\langle s,\bar\rho\rangle = \langle x,\bar\theta\rangle$ and we may specialise ($\dagger$):
$\forall u\big((u,u) \succeq \bar t\bar\sigma^{-1} \implies u \succeq x\big)$ $\quad == \{(\bar t/\_) \notin (\bar\sigma \setminus \bar\theta)\}$
$\forall u\big((u,u) \succeq \bar t\bar\theta^{-1} \implies u \succeq x\big)$ $\quad == \{\text{Def. 3.9}\}$
$\forall u\big((u,u) \succeq (x,x) \implies u \succeq x\big)$ $\quad == \{\text{Def. 3.6, pred. logic}\}$
$\forall u\big(u \succeq x \implies u \succeq x\big)$ $\quad == \{\text{pred. logic}\}$
True

(iii) $\bar t = (t_1,t_2), t_1 \neq t_2, (\bar t/\_) \notin \bar\theta, \mathit{head}(t_1) \neq \mathit{head}(t_2)$. By the proof of Lemma 5.4, case ii $\langle s,\bar\rho\rangle = \langle z,\bar\theta\{\bar t/z\}\rangle$ where z fresh, that is, $z \notin \mathcal{D}(()\bar\theta)$. Now ($\dagger$) specialised is:
$\forall u\big((u,u) \succeq \bar t\bar\sigma^{-1} \implies u \succeq z\big)$ $\quad == \{\bar\sigma \text{ inj., } \bar\sigma \geq \bar\theta\{\bar t/z\},\ z \notin \mathcal{D}(()\bar\theta)\}$
$\forall u\big((u,u) \succeq \bar t\{\bar t/z\}^{-1} \implies u \succeq z\big)$ $\quad == \{\text{Def. 3.9}\}$
$\forall u\big((u,u) \succeq (z,z) \implies u \succeq z\big)$ $\quad == \{\text{Def. 3.6}\}$
$\forall u\big(u \succeq z \implies u \succeq z\big)$ $\quad == \{\text{pred. logic}\}$
True

Induction hypothesis: The lemma holds for $n \leq k$.
Induction step: Must prove the case $n = k+1$. Only the recursive equation (8) is relevant. We have $\bar{t} = \big(f(t_1 :: \bar{t}_1), f(t_2 :: \bar{t}_2)\big), f(t_1 :: \bar{t}_1) \neq f(t_2 :: \bar{t}_2)$. Furthermore:

$$
\begin{array}{ll}
 & \langle s, \bar{\rho} \rangle \\
= & \quad \{ \text{ proof of Lem. 5.4 } \} \\
 & \big\langle f(s' :: \bar{s}'), \bar{\theta}'' \big\rangle \\
 & \quad \mathsf{where}\, \langle s', \bar{\theta}' \rangle = \mathsf{au}_{\bar{\theta}}\big((t_1, t_2), \bar{\theta}\big) \\
 & \quad\quad \mathsf{and}\, \big\langle g(\bar{s}'), \bar{\theta}'' \big\rangle = \mathsf{au}_{\bar{\theta}}\big(\big(g(\bar{t}_1), g(\bar{t}_2)\big), \bar{\theta}'\big) \\
= & \quad \{ \text{ Lem. 5.3, assumptions } \} \\
 & \big\langle f(s' :: \bar{s}'), \bar{\theta}'' \big\rangle \\
 & \quad \mathsf{where}\, \langle s', \bar{\theta}' \rangle = \mathsf{au}_{\bar{\theta}}\big((t_1, t_2), \bar{\theta}\big) \\
 & \quad\quad \wedge\, \bar{\theta}' \text{ inj., comp. to } (t_1, t_2),\ \mathcal{V}((t_1, t_2)) \cap \mathcal{D}(()\bar{\theta}') = \emptyset,\ \bar{\theta}' \geq \bar{\theta} \\
 & \quad\quad \mathsf{and}\, \big\langle g(\bar{s}'), \bar{\theta}'' \big\rangle = \mathsf{au}_{\bar{\theta}}\big(\big(g(\bar{t}_1), g(\bar{t}_2)\big), \bar{\theta}'\big) \\
 & \quad\quad\quad \wedge\, \bar{\theta}'' \text{ inj., comp. to } \big(g(\bar{t}_1), g(\bar{t}_2)\big),\ \mathcal{V}\big(\big(g(\bar{t}_1), g(\bar{t}_2)\big)\big) \cap \mathcal{D}(()\bar{\theta}'') = \emptyset, \\
 & \quad\quad\quad\ \ \bar{\theta}'' \geq \bar{\theta}'
\end{array}
$$

Transitivity of $\geq$ gives $\bar{\theta}'' \geq \bar{\theta}$. By assumption $\bar{\sigma} \geq \bar{\theta}''$ so we have $\bar{\sigma} \geq \bar{\theta}'$ and $\bar{\sigma} \geq \bar{\theta}$. Now (†) specialised is:

$$\forall u\big((u, u) \succeq \big(f(t_1 :: \bar{t}_1), f(t_2 :: \bar{t}_2)\big)\bar{\sigma}^{-1} \implies u \succeq f(s' :: \bar{s}')\big) \tag{12}$$

Observation: In general if $u \succeq f(\bar{t})\theta^{-1}$ then either u is a variable or on the form $f(\bar{u})$. This follows directly from Def. 3.9. Using the observation on the antecedent inside (12) there are two possibilities:

(i) $u = y$. Then we get

$$\forall y\big((y, y) \succeq \big(f(t_1 :: \bar{t}_1), f(t_2 :: \bar{t}_2)\big)\bar{\sigma}^{-1} \implies y \succeq f(s' :: \bar{s}')\big)$$

which, using Def. 2.12, holds since $y \succeq t$ for any t.

(ii) $u = f(v :: \bar{v})$ for some $v, \bar{v}$. Then:

$$
\begin{array}{ll}
 & \forall v, \bar{v}\big(\big(f(v :: \bar{v}), f(v :: \bar{v})\big) \succeq \big(f(t_1 :: \bar{t}_1), f(t_2 :: \bar{t}_2)\big)\bar{\sigma}^{-1} \\
 & \quad\quad \implies f(v :: \bar{v}) \succeq f(s' :: \bar{s}')\big) \\
= & \quad \{ \text{ Def. 3.9, Def. 3.6 } \} \\
 & \forall v, \bar{v}\big((v, v) \succeq (t_1, t_2)\bar{\sigma}^{-1} \wedge \big(g(\bar{v}), g(\bar{v})\big) \succeq \big(g(\bar{t}_1), g(\bar{t}_2)\big)\bar{\sigma}^{-1} \\
 & \quad\quad \implies v \succeq s' \wedge g(\bar{v}) \succeq g(\bar{s}') \\
 & \quad\ \mathsf{where}\, \mathcal{A}(g) = \mathcal{A}(f) - 1\big) \\
= & \quad \{ \text{ Ind. hyp. twice, pred. logic } \} \\
 & \text{True}
\end{array}
$$

□

Now the least property, the second part of Def. 4.1, may be proved straightforwardly.

Theorem 5.7 (Least) *Let $\bar{t}$ be non-empty.*
Then $\forall u\big((u,\ldots,u) \succeq \bar{t} \implies u \succeq \mathsf{au}(\bar{t})\big)$.

Proof.

$$
\begin{array}{ll}
 & \forall u\big((u,\ldots,u) \succeq \bar{t} \implies u \succeq \mathsf{au}(\bar{t})\big) \\
= & \quad \{\ (6)\ \} \\
 & \forall u\big((u,\ldots,u) \succeq \bar{t} \implies u \succeq s\big)\ \mathsf{where}\ \langle s, \_\rangle = \mathsf{au}_{\bar{\theta}}(\bar{t}, \{\}) \\
= & \quad \{\ \text{Lem. 5.6, } \bar{\theta} = \bar{\sigma} = \{\}\ \} \\
 & \text{True}
\end{array}
$$

□

Acknowledgements

A discussion with Arild B. Torjusen inspired Def. 2.1. The reviewers of LOPSTR 2003 provided detailed and valuable feedback (the submission was rejected).

References

1. R. Bird and O. de Moor. *The algebra of programming.* International series in computer science. Prentice Hall, 1996.

2. J.-L. Lassez, M. J. Maher, and K. Marriott. Unification revisited. In J. Minker, editor, *Foundations of deductive databases and logic programming*, chapter 15. Morgan Kaufmann, 1988.

3. B. Meltzer and D. Michie, editors. *Machine Intelligence*, volume 5. Edinburgh University Press, 1970.

4. J. C. Mitchell. *Foundations for Programming Languages.* Foundations of Computing Series. The MIT Press, 1996.

5. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19, 20:629–679, 1994.

6. G. D. Plotkin. A note on inductive generalization. In Meltzer and Michie [3], pages 153–163.

7. J. C. Reynolds. Transformational systems and the algebraic structure of atomic formula. In Meltzer and Michie [3], pages 135–151.

8. U. Schmid. *Inductive Synthesis of Functional Programs – Learning Domain-Specific Control Rules and Abstract Schemes.* Number 2654 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 2003. In press.