Conventional and Uniqueness Typing in Graph Rewrite Systems

Erik Barendsen Sjaak Smetsers
University of Nijmegen*

Abstract

In this paper we describe a Curry-like type system for graphs and extend it with *uniqueness* information to indicate that certain objects are only 'locally accessible'. The correctness of type assignment guarantees that no external access on such an object will take place in the future. We prove that types are preserved under reduction (for both type systems) for a large class of rewrite systems. Adding uniqueness information provides a solution to two problems in implementations of functional languages: efficient space management and interfacing with non-functional operations.

AMS Classification (1991): 68Q42 (68Q05).
CR Classification (1991): F.4.2, D.3.3 (D.3.1, D.1.1, F.4.1).
Keywords and phrases: graph rewrite systems, functional programming languages, typing, higher order functions, uniqueness typing, algebraic types, linear typing, destructive updates.

1. Introduction

There are several models of computation that can be viewed as a basis for functional programming. Traditional examples are the *lambda calculus* and *term rewrite systems*. Graph rewriting is a relatively new concept, providing a model that is sufficiently elegant and abstract, but at the same time incorporates mechanisms that are more realistic with respect to actual implementation techniques.

Graph rewrite systems were introduced in Barendregt *et al.* [1987a]. The present paper deals with a restricted form of GRS's: the so-called *term graph rewrite systems* (TGRS's, see Barendregt *et al.* [1987b] and Barendsen & Smetsers [1992]). TGRS's are very well suited as a basis for (implementation of) functional languages, as is demonstrated by the graph rewrite language *Concurrent Clean*, see Nöcker *et al.* [1991].

Conventional types

The concept of typing in lambda calculus is well known. To study the effect of *patterns* in function definitions on typing, a Curry-like type assignment system on (applicative) term rewriting systems has been developed by van Bakel *et al.* [1992]. The types in this

*Computing Science Institute, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands, e-mail erikb@cs.kun.nl, sjakie@cs.kun.nl, fax +31.80.652525.

system resemble those of the simply typed lambda calculus; the use of type schemes for graph symbols introduces a weak form of polymorphism. Moreover arbitrary type constructors are incorporated.

In the present paper the notion of type assignment to TRS's of van Bakel *et al.* [1992] will be extended to general term graphs (i.e. graphs that are not necessarily trees) in a very natural way. Some aspects of typing are even more convenient in graph theoretical setting. E.g. special contexts for variables are not necessary since multiple occurrences of the same variable are represented by one single node. Moreover, the extra features of graph rewriting (shared c.q. cyclic objects) are treated without extra effort.

As in van Bakel *et al.* [1992], type assignment is not defined using a set of deduction rules but, more directly, by supplying nodes of the graphs and of the rewrite rules with types in a consistent way. The consistency is expressed in a 'local' constraint for each node. Graph symbols are supplied with a type by a so called *type environment*. We show how to incorporate type constructors introduced by *algebraic type specifications*.

Uniqueness types

The underlying motivation for uniqueness types was given by two fundamental problems in practical functional programming and the implementation of functional languages using sharing techniques.

The first problem is the *space behaviour* of functional programs during execution. In a reduction step one often has to construct complicated structures, involving creation of new nodes. One could improve the efficiency of the implementation by re-using the space of obsolete objects of the part of the graph being rewritten, thus performing garbage collection on the spot. It would even be better if one could predict *at compile time* which arguments of a function will become garbage during rewriting. This is called *compile time garbage collection*. This is often the only way to handle complex data structures efficiently.

A second issue is the incorporation of essentially nonfunctional operations in the formalism of graph rewriting, e.g. for dealing with input-output. File updating, for example, is an operation with side-effects, possibly disturbing referential transparency. Such operations are safe, however, if there exists only a single reference to the object being modified, at the moment the modification takes place.

The technique presented here involves incorporation of *locality* or *uniqueness* information in the type system mentioned above. Types are therefore extended with so called uniqueness attributes.

In the type of a function F it can now be indicated that a specific argument should be 'unique'. The intended meaning is that at the moment of evaluation, the corresponding object is local for F, i.e. can only be accessed via F. The type system will allow only applications of F of this kind.

This locality information can be used to solve the problems indicated above. Suppose a function argument is unique. If (part of) the argument is not used in the result, it can be concluded that this part becomes garbage in any concrete application. It is then possible to re-use the space or even the contents of obsolete objects when building the result. The second problem is solved by typing the 'dangerous' updating operations in such a way that they require unique arguments.

In order to achieve this, the notion of type assignment (for conventional typing) is extended: the type correctness of an application FX depends not only on the argument type of F and the type of X, but also on the way X is passed to F: if F expects a 'unique' argument then X should only be accessible by F, e.g. by requiring a reference count of 1. This straightforward reference count approach is rather rough. In practice one usually has a specific evaluation order in mind. In this paper we present a more liberal analysis using this information. The correctness of a function application now depends on the demanded argument type, the offered argument type and the context in which the access takes place. This dependency is formulated in terms of a subtyping relation specifying *coercions*.

We prove that uniqueness typing is preserved under graph rewriting, for a sufficiently large class of graph rewrite systems.

Related work

The weighted reference count analysis, as presented in section 5, is inspired by Guzmán & Hudak [1990]. This paper addresses the mutability problem using a 'single threaded polymorphic lambda calculus' ($poly\text{-}\lambda_{st}$). It uses the operational semantics of lambda-graph reduction of Wadsworth [1971]. In our paper the analysis is performed in the formalism of graph rewriting, which is obviously more direct. The effect of cyclic structures (not present in the paper mentioned above) and general pattern matching are studied in the general graph rewriting setting presented here.

In Wadler [1990], a type system including linear types is developed. The paper also used Wadsworth's lambda reduction. The coercions described in our paper are implicit in Wadler [1990], in a much more restricted form.

In Sastry *et al.* [1993], the update problem is addressed by determining an order of evaluation of expressions via abstract interpretation such that destructive operators can be used instead of non-destructive ones. In our approach the reduction order is fixed, and operators are either destructive or non-destructive. The type system guarantees that all applications are safe. Moreover, Sastry *et al.* [1993] focus on a first-order call-by-value language with flat aggregates (i.e. aggregates only containing non-aggregate values). In the present paper, a higher-order call-by-need language is used with no specific assumptions on the structure of data.

Based on the idea of uniqueness typing, Jacobs [1993] developed a *logical* system explicitly mixing conventional and linear constructive logic. The approach described here is likely to offer a 'propositions-as-types' notion (representing proofs by typed graphs).

A combination of conventional and uniqueness typing has been incorporated in the lazy functional graph rewriting language *Concurrent Clean*. So far, it has been used for the implementation of arrays and of an efficient high-level library for screen and file I/O (see Achten *et al.* [1993]).

Structure of this paper

We start with a brief introduction to graph rewriting. In section 3, we describe the Curry-like basis of graph typing. After an informal introduction to uniqueness typing (section 4), the reference analysis mentioned above is worked out in section 5. The sections 6 and 7 extend the conventional type system with uniqueness information. Algebraic uniqueness types are constructed in section 8, which also describes the uniqueness typing of higher-order functions. The sections 9 and 10 demonstrate that the reference analysis indeed captures the intended uniqueness property. The proof of the fact that typing is preserved during reduction is carried out stepwise in the sections 11–14. We give some examples in section 15 and conclude with directions for future research in section 16.

2. Graph rewriting

Term graph rewrite systems were introduced in Barendregt *et al.* [1987b]. This section summarizes some basic notions for (term) graph rewriting as presented in Barendsen & Smetsers [1992].

Finite sequences

2.1. DEFINITION. (i) A finite sequence (over A) is a tuple $s = (a_1, a_2, \ldots, a_\ell)$ of elements of A. The collection of such sequences is denoted by A^*. For s as above, ℓ is the *length* of s (notation $|s|$). We say that s is *empty* if $|s| = 0$.

(ii) Let s be a sequence. For each $1 \leq i \leq |s|$, the i-th element s is indicated by $(s)_i$.

(iii) We use the following denotation for specific parts of nonempty sequences s.

$$s^- = (a_2, \ldots, a_\ell), \qquad s_- = (a_1, \ldots, a_{\ell-1}).$$

(iv) Let s, t be sequences. Then s and t are *disjoint* (notation $s \# t$) if

$$\forall i, j\ [(s)_i \neq (t)_j].$$

(v) The *concatenation* of s and t is denoted by $s * t$.

(vi) Let s, t be sequences. Then s is a *prefix of* t (notation $s \subseteq t$) if $s * s' = t$ for some s'. Moreover $s \subset t$ denotes that s is a proper prefix of t.

Graphs

The objects of our interest are finite directed graphs in which each node has a specific label. The number of outgoing edges of a node is determined by its label. In the sequel we assume that $\mathcal{N}$ is some basic set of *nodes* (infinite; one usually takes $\mathcal{N} = \mathbb{N}$), and Σ is a (possibly infinite) set of *symbols* with *arity* in $\mathbb{N}$.

2.2. DEFINITION. (i) A *labeled graph* (over $\langle \mathcal{N}, \Sigma \rangle$) is a triple

$$g = \langle N, \mathit{symb}, \mathit{args} \rangle$$

such that

(1) $N \subseteq \mathcal{N}$; N is the set of *nodes* of g;
(2) $\mathit{symb} : N \to \Sigma$; $\mathit{symb}(n)$ is the *symbol* at node n;
(3) $\mathit{args} : N \to N^*$ such that $|\mathit{args}(n)| = \mathrm{arity}(\mathit{symb}(n))$.

Thus $\mathit{args}(n)$ specifies the outgoing edges of n.

(ii) Let $n \in g$ and $i \leq \text{arity}(n)$. The combination $a = (n, i)$ is called a *reference*. The node n is the *root* of a (notation $\mathrm{r}(a)$). Moreover i is the *index* of a (notation $[a]$). The *destination* of a (notation $\mathrm{d}(a)$) is the node to which it refers, i.e. $args(n)_i$. The set of all references of g is indicated by Ref_g.

(iii) A *rooted graph* is a quadruple

$$g = \langle N, symb, args, r \rangle$$

such that $\langle N, symb, args \rangle$ is a labeled graph, and $r \in N$. The node r is called the *root* of the graph g.

(iv) The collection of all finite rooted labeled graphs over $\langle \mathcal{N}, \Sigma \rangle$ is indicated by $\mathbb{G}$.

Convention. (i) $m, n, n', \ldots$ range over nodes; $g, g', h, \ldots$ range over (rooted) graphs; $a, b, a', \ldots$ range over references;.

(ii) If g is a (rooted) graph, then its components are referred to as N_g, $symb_g$, $args_g$ (and r_g) respectively. To simplify notation we write $n \in g$ instead of $n \in N_g$.

Paths

2.3. Definition. (i) Let $a, a' \in Ref_g$. Then a' *succeeds* a if $\mathrm{d}(a) = \mathrm{r}(a')$.

(ii) A *path* in a graph is a sequence p of references such that $(p)_{k+1}$ succeeds $(p)_k$ for all $k < |p|$.

(iii) Let $a \in Ref_g$. A path p is *extendible with* a if either p is empty, or a succeeds $(p)_{|p|}$. The *extension* of p with a is denoted by $p * a$.

(iv) $p : n \rightsquigarrow m$ denotes that p *leads from* n *to* m. More formally,

$$\begin{aligned} () &: n \rightsquigarrow n, \\ p : n \rightsquigarrow m \quad &\Rightarrow \quad p * (m, i) : n \rightsquigarrow args_g(m)_i. \end{aligned}$$

(v) Let $m, n \in g$. Then m is *reachable from* n (notation $n \rightsquigarrow m$) if $p : n \rightsquigarrow m$ for some path p in g.

(vi) Let $n \in g$. We write $n \in p$ if $\mathrm{r}((p)_i) = n$ for some i.

2.4. Definition. Let p be a nonempty path.

(i) The *root* of p (notation $\mathrm{r}(p)$) is the root of its first reference; the *index* of p is the index of this reference. The *destination* of p (notation $\mathrm{d}(p)$) is the destination of its last reference.

(ii) The *node sequence of* p (notation $\widetilde{p}$) is the sequence

$$(\mathrm{r}((p)_1), \ldots, \mathrm{r}((p)_{|p|}), \mathrm{d}((p)_{|p|})).$$

(iii) p is a *cycle* if $\mathrm{r}(p) = \mathrm{d}(p)$.

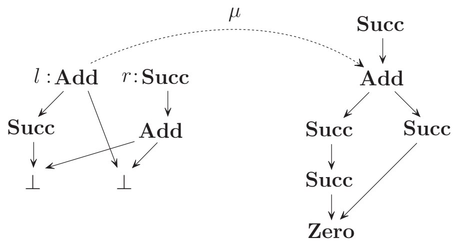
(iv) p is called *cyclic* if it has a cycle as subpath. Moreover p is *root cyclic* if an initial part of p is a cycle.

2.5. Definition. A graph g is a *tree* if for each $n \in g$ there exists a unique path leading from r_g to n. This unique path will be denoted as $\overline{\overline{n}}$. Sometimes the last reference of a nonempty $\overline{\overline{n}}$ plays a special role; we denote it by $\overline{n}$.

2.6. Definition. Let g be a graph and $n \in g$. The *subgraph of* g *at* n (notation $g \mid n$) is the rooted graph $\langle N, symb, args, n \rangle$ where $N = \{m \in g \mid n \rightsquigarrow m\}$, and $symb$ and $args$ are the restrictions (to N) of $symb_g$ and $args_g$ respectively.

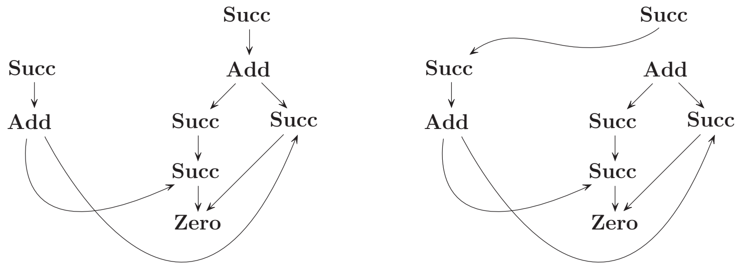
Graph rewriting

Rewrite rules specify transformations of graphs. Each rewrite rule is represented by a special graph containing two roots. These roots determine the left-hand side (the *pattern*) and the right-hand side of the rule. Variables are represented by 'empty nodes', containing the special symbol $\perp$. Let R be some rewrite rule. A graph g can be *rewritten* according to R if R is applicable to g, i.e. the pattern of R *matches* g. A *match* is a mapping from the pattern of R to a subgraph of g that preserves the node structure. The following picture gives the graph representation of the rule $\mathbf{Add}(\mathbf{Succ}(x), y) \to \mathbf{Succ}(\mathbf{Add}(x, y))$ and a match μ.

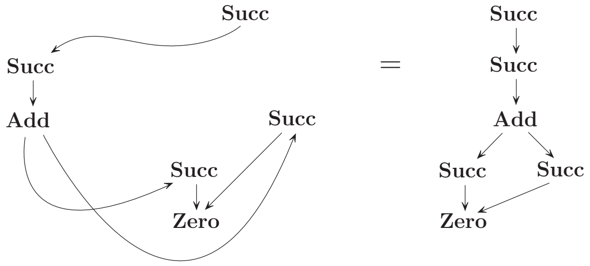


Instead of using names for variables in rewrite rules, multiple occurrence of the same variable is indicated via sharing. E.g. all occurrences of the variable x in the above rule are represented by references to the same empty node.

The combination of a rule and a match is called a *redex*. If a redex has been determined, the graph can be rewritten according to the structure of the right-hand side of the rule involved. This is done in three steps. Firstly, the graph is *extended* with an instance of the right-hand side of the rule. The connections from the new part with the original graph are determined by μ. Then all references to the root of the redex are *redirected* to the root of the right-hand side.



Finally all unreachable nodes are removed by performing *garbage collection*.



This procedure is formalized in the rest of the present section.

2.7. DEFINITION. Let g be a graph.

(i) The set of *empty nodes* of g (notation g°) is the collection

$$g^\circ = \{n \in g \mid symb_g(n) = \bot\}.$$

(ii) The set of *non-empty nodes* (or *interior*) of g is denoted by $g^\bullet$. So $N_g = g^\circ \cup g^\bullet$.

(iii) g is *closed* if $g^\circ = \emptyset$.

The objects on which computations are performed are closed graphs; the others are used as auxiliary objects, e.g. for defining graph rewrite rules.

2.8. DEFINITION. (i) A *term graph rewrite rule* (or *rule* for short) is a triple $R = \langle g, l, r\rangle$ where g is a (possibly open) graph, and $l, r \in g$ (called the *left root* and *right root* of R), such that

(1) $(g \mid l)^\bullet \neq \emptyset$;

(2) $(g \mid r)^\circ \subseteq (g \mid l)^\circ$.

(ii) If $symb_g(l) = F$ then R is said to be a *rule for* F.

(iii) A symbol S is a *pattern symbol* of R if $S = symb_g(n)$ for some $n \in R \mid l, n \neq l$.

(iv) R is *left-linear* if $g \mid l$ is a tree.

(v) R is *extending* if $r \notin (g \mid l)$. Otherwise, R is *projecting*.

In (i) above, condition (1) expresses that the left-hand side of the rewrite rule should not be just a variable. Moreover condition (2) states that all variables occurring on the right-hand side of the rule should also occur on the left-hand side.

NOTATION. We will write $R \mid l$, $R \mid r$ for $g_R \mid l_R$, $g_R \mid r_R$ respectively.

2.9. DEFINITION. Let p, g be graphs.

(i) A *match* is a function $\mu : N_p \to N_g$ such that for all $n \in p^\bullet$

$$\begin{aligned} symb_g(\mu(n)) &= symb_p(n), \\ args_g(\mu(n))_i &= \mu(args_p(n)_i). \end{aligned}$$

In this case we write $\mu : p \underset{\mathrm{m}}{\rightarrow} g$.

(ii) μ is a *rooted match* from p to g (notation $\mu : p \underset{\mathrm{rm}}{\rightarrow} g$) if $\mu : p \underset{\mathrm{m}}{\rightarrow} g$ and $\mu(r_p) = \mu(r_g)$.

(iii) By $p \underset{\mathrm{m}}{\rightarrow} g$ we will denote that there exists a match $\mu : p \underset{\mathrm{m}}{\rightarrow} g$. Analogously we write $p \underset{\mathrm{rm}}{\rightarrow} g$ for rooted matches.

2.10. DEFINITION. The graphs g and g' are *compatible* (notation $g \uparrow g'$) if for some h one has $g \underset{\mathrm{rm}}{\rightarrow} h$ and $g' \underset{\mathrm{rm}}{\rightarrow} h$.

We now formalize the rewrite procedure.

2.11. DEFINITION. Let g be a graph, and $\mathcal{R}$ a set of rewrite rules. An $\mathcal{R}$*-redex* in g (or just *redex*) is a tuple $\Delta = \langle R, \mu\rangle$ where $R \in \mathcal{R}$, and $\mu : (R \mid l) \underset{\mathrm{m}}{\rightarrow} g$.

2.12. DEFINITION. Let N, A be sets of nodes. The *disjoint set extension* of N with A (notation $N \cup^+ A$) is the set $N \cup A^*$, where $A^* = \{a^* \mid a \in A\}$ is a set of fresh nodes associated with A.

2.13. DEFINITION. Let g be a graph. Let $\Delta = \langle R, \mu\rangle$ be a redex in g. Set $N = (R \mid r)^\bullet \backslash (R \mid l)^\bullet$ (the set of new nodes).

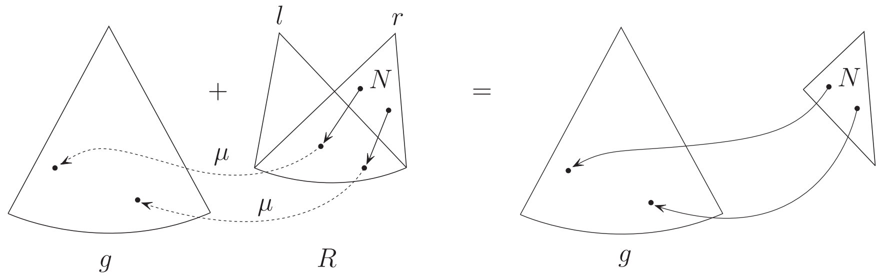
(i) The Δ-*extension* of g (notation $g + \Delta$) is defined as follows. Then $g + \Delta = h$, where

$$\begin{array}{rcll} N_h & = & N_g \cup^+ N; & \\ \mathit{symb}_h(n) & = & \mathit{symb}_g(n) & \text{if } n \in g, \\ & = & \mathit{symb}_R(n) & \text{otherwise;} \\ \mathit{args}_h(n)_i & = & \mathit{args}_g(n)_i & \text{if } n \in g, \\ & = & \mathit{args}_R(n)_i & \text{if } n, \mathit{args}_R(n) \in N, \\ & = & \mu(\mathit{args}_R(n)_i) & \text{otherwise;} \\ r_h & = & r_g. & \end{array}$$

(ii) The *contractum root* of Δ (notation $\mathrm{r}(\Delta)$) is defined as follows.

$$\begin{array}{rcll} \mathrm{r}(\Delta) & = & r_R & \text{if } r_R \in N, \\ & = & \mu(r_R) & \text{otherwise.} \end{array}$$

This construction can be visualized as follows.



A redirection in a graph is an operation which replaces all references to a given node by references to another one.

2.14. DEFINITION. Let g be a graph. Let $n, m \in g$.

(i) The *redirection function* associated with n, m (notation $[n \mapsto m]$) on N_g maps n to m and is the identity elsewhere.

(ii) The result of *redirecting* n to m (notation $g[n := m]$) is the graph

$$\langle N_g, \mathit{symb}_g, \mathit{args}', [n \mapsto m](r_g)\rangle$$

where $\mathit{args}'(x)_i = [n \mapsto m](\mathit{args}_g(x)_i)$.

A rewrite step is concluded by garbage collection.

2.15. DEFINITION. Let g be a rooted graph. The result of performing *garbage collection* on g (notation $\mathrm{GC}(g)$) is the graph obtained from g by deleting all nodes that are not reachable from its root, i.e. $\mathrm{GC}(g) = g \mid r_g$.

The three basic operations extension, redirection, and garbage collection are combined into a single operation. This mechanism corresponds to *substitution* in lambda calculus and term rewriting.

2.16. DEFINITION. Let Δ be a ($\mathcal{R}$-)redex in g. The result h of *contracting* Δ in g is defined by

$$h = \mathrm{GC}((g + \Delta)[\mu(l) := \mathrm{r}(\Delta)]).$$

In this case we write $g \xrightarrow[\mathcal{R}]{\Delta} h$, or just $g \to h$. We call h the Δ-*reduct* of g.

Term graph rewrite systems

A collection of graphs and a set of rewrite rules can be combined into a (term) graph rewrite system. A special class of so-called orthogonal graph rewrite systems is the subject of further investigations.

2.17. DEFINITION. (i) A *term graph rewrite system* (TGRS) is a tuple $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ where $\mathcal{R}$ is a set of rewrite rules, and $\mathcal{G} \subseteq \mathbb{G}$ is a set of closed graphs which is closed under $\mathcal{R}$-reduction.

(ii) $\mathfrak{T}$ is *left-linear* if each $R \in \mathcal{R}$ is left-linear.

(iii) $\mathfrak{T}$ is *regular* if for each $g \in \mathcal{G}$ the $\mathcal{R}$-redexes in g are pairwise disjoint.

(iv) $\mathfrak{T}$ is *orthogonal* if $\mathfrak{T}$ is both left-linear and regular.

It can be shown that for a large class of orthogonal TGRS's (the so-called *interference-free* systems) the Church-Rosser property holds (see Barendsen & Smetsers [1992]).

Applicative graph rewrite systems

In TGRS's, all symbols have a fixed arity. This makes it impossible to use *functions* as arguments or functions as result. The concept of higher order functions, however, can be simulated as follows.

Say we want to model the function $g(x) = \lambda y.f(x, y)$. The idea is to specify f by a TGRS rule

$$\mathbf{F}(x, y) \to \cdots$$

and to associate with $\mathbf{F}$ a so called *Curry variant* $\mathbf{F}_1$ of arity 1. Here $\mathbf{F}_1$ is regarded as a constructor symbol. The intention is that an application $\mathbf{F}_1(X)$ stands for $\lambda y.\mathbf{F}(X, y)$. Moreover one specifies an *application rule* for the special function symbol $\mathbf{Ap}$ (of arity 2) stating

$$\mathbf{Ap}(\mathbf{F}_1(x), y) \to \mathbf{F}(x, y)$$

which supplies the 'partial application' of $\mathbf{F}$ with an extra argument.

In general, one can associate with each function symbol F of nonzero arity (say k) a collection of k Curry variants of arity $0, 1, \ldots, k-1$ respectively. The rules for $\mathbf{Ap}$ look like

$$\begin{array}{rcl}
\mathbf{Ap}(F_0, x_1) & \to & F_1(x_1), \\
\mathbf{Ap}(F_1(x_1), x_2) & \to & F_2(x_1, x_2), \\
 & \cdots & \\
\mathbf{Ap}(F_{k-1}(x_1, \ldots, x_{k-1}), x_k) & \to & F(x_1, \ldots, x_k).
\end{array}$$

The Curry rule for arity j is referred to as Ap_j^F.

In our approach, the basis of a TGRS is formed by proper functional rewrite rules (like the one for **F** above). In the rewrite rules, one might use **Ap** and Curry variants, however. The auxiliary **Ap** rules have the standard shapes as indicated above. We consider *Curry closed* systems, i.e. systems in which there is an **Ap** rule for each occurring Curry variant.

A special class of TGRS's is formed by so-called *Curry complete* TGRS's, in which *all* Curry variants and corresponding application rules are present.

2.18. EXAMPLE. *Combinatory Logic* (CL), originally expressed by

$$\boxed{\begin{array}{rcl} \mathbf{S}\,xyz & = & xz(yz) \\ \mathbf{K}\,xy & = & x \\ \mathbf{I}\,x & = & x \end{array}}$$

can be formulated as a Curry complete TGRS in the following way.

$$\boxed{\begin{array}{rcl} \mathbf{S}(x,y,z) & \to & \mathbf{Ap}(\mathbf{Ap}(x,z),\mathbf{Ap}(y,z)) \\ \mathbf{K}(x,y) & \to & x \\ \mathbf{I}(x) & \to & x \\ \mathbf{Ap}(\mathbf{S}_0,x) & \to & \mathbf{S}_1(x) \\ \mathbf{Ap}(\mathbf{S}_1(x),y) & \to & \mathbf{S}_2(x,y) \\ \mathbf{Ap}(\mathbf{S}_2(x,y),z) & \to & \mathbf{S}(x,y,z) \\ \mathbf{Ap}(\mathbf{K}_0,x) & \to & \mathbf{K}_1(x) \\ \mathbf{Ap}(\mathbf{K}_1(x),y) & \to & \mathbf{K}(x,y) \\ \mathbf{Ap}(\mathbf{I}_0,x) & \to & \mathbf{I}(x) \end{array}}$$

Any Curry complete system is equivalent with a *purely applicative* TGRS (where **Ap** is the only function symbol). For the analysis of typing, however, it will be convenient to focus on the 'functional basis' of a TGRS and view the **Ap** rules as special rules with an induced typing.

2.19. DEFINITION. Let $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ be a TGRS.

(i) The set of $\mathfrak{T}$-*symbols* (notation $\Sigma_{\mathfrak{T}}$) consists of the symbols appearing in $\mathcal{G}$ or in $\mathcal{R}$.

(ii) $\mathcal{R}$ is divided into two parts: $\mathcal{R}_{\mathcal{F}}$ (proper functional rules) and $\mathcal{R}_{\mathcal{C}}$ (Curry rules for **Ap**).

(iii) The set of *proper function symbols* of $\mathfrak{T}$ (notation $\Sigma^{\mathrm{f}}_{\mathcal{F}}(\mathfrak{T})$) consists of the symbols having a rule in $\mathcal{R}_{\mathcal{F}}$. Moreover $\Sigma^{\mathrm{c}}_{\mathcal{F}}(\mathfrak{T})$ denotes the set of their Curry variants. These sets combine into the collection $\Sigma_{\mathcal{F}}(\mathfrak{T})$ of *functional symbols* of $\mathfrak{T}$, i.e. $\Sigma_{\mathcal{F}}(\mathfrak{T}) = \Sigma^{\mathrm{f}}_{\mathcal{F}}(\mathfrak{T}) \cup \Sigma^{\mathrm{c}}_{\mathcal{F}}(\mathfrak{T})$.

(iv) The set of *function symbols* of $\mathfrak{T}$ (notation $\mathrm{Fun}(\mathfrak{T})$) consists of those symbols having a rule in $\mathfrak{T}$. In general one has $\mathrm{Fun}(\mathfrak{T}) = \Sigma^{\mathrm{f}}_{\mathcal{F}} \cup \{\mathbf{Ap}\}$. The other symbols are called *data symbols* (notation $\mathrm{Data}(\mathfrak{T})$). Note that Curry variants are data symbols.

We will only consider *function-data systems*, i.e. systems in which all pattern symbol are included in $\mathrm{Data}(\mathfrak{T})$. We will usually omit '$(\mathfrak{T})$' in the denotations introduced above if $\mathfrak{T}$ is fixed.

3. Conventional typing

In this section we will define a notion of simple type assignment to graphs using a type system based on traditional systems for functional languages. The approach is similar to the one introduced in van Bakel *et al.* [1992]. It is meant to illustrate the concept of 'classical' typing for graphs.

3.1. DEFINITION. Let $\mathbb{V}$ be a set of *type variables*, and $\mathbb{C}$ a set of *type constructors* with *arity* in $\mathbb{N}$.

(i) The set $\mathbb{T}$ of (*graph*) *types over* $\mathbb{C}$ is defined inductively as follows.

$$\begin{aligned} \alpha \in \mathbb{V} &\Rightarrow \alpha \in \mathbb{T}, \\ \sigma, \tau \in \mathbb{T} &\Rightarrow \sigma \to \tau \in \mathbb{T}, \\ T \in \mathbb{C}, \sigma_1, \ldots, \sigma_k \in \mathbb{T} &\Rightarrow T(\sigma_1, \ldots, \sigma_k) \in \mathbb{T}. \end{aligned}$$

(ii) $\mathrm{TV}(\sigma)$ denotes the set of type variables occurring in σ. Let $\vec{\alpha} \in \mathbb{V}$. Then $\mathbb{T}(\vec{\alpha})$ denotes the set of types σ with $\mathrm{TV}(\sigma) \subseteq \{\vec{\alpha}\}$.

(iii) The set $\mathbb{T}_{\mathrm{S}}$ of *symbol types* is defined as

$$\sigma_1, \ldots, \sigma_k, \tau \in \mathbb{T} \Rightarrow (\sigma_1, \ldots, \sigma_k) \rightarrowtail \tau \in \mathbb{T}_{\mathrm{S}}, \qquad k \geq 0.$$

We will usually abbreviate $() \rightarrowtail \tau$ to τ and $(\sigma) \rightarrowtail \tau$ to $\sigma \rightarrowtail \tau$.

CONVENTION. In the sequel, $\alpha, \beta, \alpha_1, \ldots$ range over type variables; $\sigma, \tau, \tau_1, \ldots$ range over (symbol) types.

3.2. DEFINITION. (i) A *substitution* is a function $* : \mathbb{V} \to \mathbb{T}$. This induces an operation on $\mathbb{T}$ (and $\mathbb{T}_{\mathrm{S}}$) in a straightforward way.

(ii) σ is an *instance* of τ (notation $\sigma \subseteq \tau$) if there exists a substitution $*$ such that $\tau^* = \sigma$.

(iii) Let $\sigma, \tau \in \mathbb{T}$. A *unifier* for σ and τ is a substitution $*$ such that $\sigma^* = \tau^*$.

(iv) The above substitution $*$ is a *most general unifier* if for all $*_1$

$$\sigma^{*_1} = \tau^{*_1} \Rightarrow *_1 = *_2 \circ * \text{ for some } *_2 .$$

Algebraic type systems

In our system we consider types which are built up from type variables and type constructors. The standard type constructor is $\to$ for function spaces. We will use the infix denotation $\sigma \to \tau$ for function types.

The other type constructors are assumed to be given by a so called *algebraic type system*. The idea is to specify the *canonical* inhabitants of each constructor type $T\vec{\sigma}$ by a declaration of the form

$$T\vec{\alpha} = C_1\vec{\sigma_1} \mid C_2\vec{\sigma_2} \mid \cdots$$

where the length of $\vec{\alpha}$ indicates the arity of the defined type constructor T. A subexpression $C_i\vec{\sigma_i}$ is called a *clause* in the definition of $T\vec{\alpha}$. The free variables in σ_i are among the $\vec{\alpha}$. Moreover C_i is called an *algebraic constructor*.

An *algebraic type system* is a collection $\mathcal{A}$ of constructor declarations. These might be (directly or indirectly) recursive.

3.3. Example. The system

$$\begin{array}{rcl}\mathrm{Nat} & = & \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \cdots \\ \mathrm{List}(\alpha) & = & \mathbf{Cons}(\alpha, \mathrm{List}(\alpha)) \mid \mathbf{Nil}\end{array}$$

specifies that the (canonical) objects of type Nat are the constants $\mathbf{0}$, $\mathbf{1}$, $\mathbf{2}$, etc. Moreover an object of type $\mathrm{List}(\sigma)$ is either equal to $\mathbf{Nil}$ or to a graph of the form

$$\begin{tikzcd} & \mathbf{Cons} \arrow[dl] \arrow[dr] & \\ g & & h \end{tikzcd}$$

where g is an object of type σ, and h is again an object of type $\mathrm{List}(\sigma)$.

If $\mathcal{A}$ is an algebraic type system, then $\mathbb{C}_{\mathcal{A}}$ denotes the set of type constructors specified in $\mathcal{A}$. Moreover $\Sigma_{\mathcal{A}}$ is the set of (graph) constructors introduced in $\mathcal{A}$. In the above example one has

$$\mathbb{C}_{\mathcal{A}} = \{\mathrm{Nat}, \mathrm{List}\}$$

and

$$\Sigma_{\mathcal{A}} = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots, \mathbf{Cons}, \mathbf{Nil}\}.$$

It is assumed that each algebraic constructor is used only once in $\mathcal{A}$.

We restrict ourselves to the following combinations of algebraic type systems and TGRS's. We write Σ_0 for the set of *basic symbols* $\perp$ and $\mathbf{Ap}$.

3.4. Definition. Let $\mathcal{A}$ be an algebraic type system, and let $\mathfrak{T}$ be a TGRS. Then $\mathfrak{T}$ is *applicative TGRS over* $\mathcal{A}$ if $\mathfrak{T}$ is Curry closed, and $\mathrm{Data}(\mathfrak{T}) \subseteq \Sigma_0 \cup \Sigma^{\mathrm{c}}_{\mathcal{F}}(\mathfrak{T}) \cup \Sigma_{\mathcal{A}}$.

We restrict our types accordingly by setting $\mathbb{C} = \mathbb{C}_{\mathcal{A}}$.

Type assignment for graphs

In the rest of this section we describe how types can be assigned to graphs given a type assignment to the symbols by a so called *environment*.

3.5. Definition. (i) Let $\Gamma \subseteq \Sigma$ be a set of symbols. A *type environment for* Γ is a function $\mathcal{E} : \Gamma \to \mathbb{T}$.

(ii) The *basic type environment* $\mathcal{E}_0$ is the following environment for Σ_0.

$$\begin{array}{rcl}\mathcal{E}_0(\perp) & = & \alpha, \\ \mathcal{E}_0(\mathbf{Ap}) & = & (\alpha \to \beta, \alpha) \rightarrowtail \beta.\end{array}$$

These environment types are regarded as *type schemes*, i.e. in any concrete application of a symbol one uses an instance of its environment type. An alternative view is to consider the environment types as universally quantified on the topmost level, e.g. $\forall\alpha\forall\beta.(\alpha \to \beta, \alpha) \rightarrowtail \beta$.

3.6. Definition. Let $g = \langle N, \mathit{symb}, \mathit{args}\rangle$ be a graph.

(i) A *type assignment* to g (or *typing for* g) is a function $\mathcal{T} : N \to \mathbb{T}$.

(ii) Let $\mathcal{T}$ be a typing for g, and $n \in g$; say $k = \text{arity}(n)$. The *symbol type* of n according to $\mathcal{T}$ (notation $\mathcal{F}_{\mathcal{T}}(n)$) is defined by

$$\mathcal{F}_{\mathcal{T}}(n) = (\mathcal{T}(\mathit{args}(n)_1), \ldots, \mathcal{T}(\mathit{args}(n)_k)) \rightarrowtail \mathcal{T}(n).$$

(iii) Let $\mathcal{E}$ be an environment. Then $\mathcal{T}$ is an *$\mathcal{E}$-typing for g* if for each $n \in g$ one has

$$\mathcal{F}_{\mathcal{T}}(n) \subseteq \mathcal{E}(\mathit{symb}(n)).$$

(iv) g is *$\mathcal{E}$-typable with type σ* (notation $\mathcal{E} \vdash g : \sigma$) if there exists an $\mathcal{E}$-typing $\mathcal{T}$ for g with $\mathcal{T}(r_g) = \sigma$.

(v) If $\mathcal{E} \vdash g : \sigma$ is a typing statement, then g is called the *subject* of the statement; moreover σ is its *predicate*.

3.7. Example. Let $\mathcal{E}$ be an environment containing the type declarations

$$\boxed{\begin{array}{lcl} \mathbf{F} & : & \text{List}(\beta) \rightarrowtail \beta, \\ \mathbf{Cons} & : & (\alpha, \text{List}(\alpha)) \rightarrowtail \text{List}(\alpha), \\ \mathbf{Nil} & : & \text{List}(\alpha), \\ \mathbf{3} & : & \text{Int}. \end{array}}$$

The following type assignment shows that $\mathcal{E} \vdash g : \text{Int}$ for the the displayed graph g.

$$\begin{tikzcd}[column sep=tiny] & \mathbf{F} : \text{Int} \arrow[d] & \\ & \mathbf{Cons} : \text{List(Int)} \arrow[dl] \arrow[dr] & \\ \mathbf{3} : \text{Int} & & \mathbf{Nil} : \text{List(Int)} \end{tikzcd}$$

Type assignments are substitutive. This is formulated as follows.

3.8. Instantiation Lemma. *Let g be a graph, and $\mathcal{E}$ an environment. Let $*$ be a substitution.*

(i) *Let $\mathcal{T}$ be an $\mathcal{E}$-typing for g. Then $\mathcal{T}^*$ is an $\mathcal{E}$-typing for g.*

(ii) *For each $\sigma \in \mathbb{T}$*

$$\mathcal{E} \vdash g : \sigma \;\Rightarrow\; \mathcal{E} \vdash g : \sigma^*.$$

Proof. Easy. $\square$

Principal types

In this subsection we will show that type assignment has the principal type property, i.e. if a graph g is typable then there exists a most general typing for g. In the sequel, some fixed environment $\mathcal{E}$ is assumed. The presentation below is inspired by Barendregt [1992].

3.9. Definition. Let $E = \{\sigma_1 = \tau_1, \ldots, \sigma_n = \tau_n\}$ be a finite set of equations between types. A *solution* for E is a substitution $*$ such that

$$\sigma_1^* = \tau_1^* \;\;\&\;\; \cdots \;\;\&\;\; \sigma_n^* = \tau_n^*.$$

In that case one writes $* \models E$. Analogously to the notion of most general unifier, one defines the notion of *most general solution* for E.

3.10. Unification Theorem. *There exists a recursive function* Unify *having as input finite sets of equations between types such that*

$$\begin{array}{rcl} E \textit{ has a solution} & \Rightarrow & \text{Unify}(E) \textit{ is the most general solution for } E; \\ E \textit{ has no solution} & \Rightarrow & \text{Unify}(E) = \mathsf{fail}. \end{array}$$

Proof. See Robinson [1965]. $\square$

3.11. Proposition. *Let g be a graph, and $\mathcal{T}$ a typing for g. Then there exist a finite set of equations $E = E_{\mathcal{E}}(\mathcal{T}, g)$ such that for all substitutions $*$ one has*
(1) $* \models E \quad \Rightarrow \quad \mathcal{T}^*$ *is an $\mathcal{E}$-typing for g.*
(2) $\mathcal{T}^*$ *is a typing for g* $\quad \Rightarrow \quad *_1 \models E$ *for some $*_1$ such that $\mathcal{T}^{*_1} = \mathcal{T}^*$.*

Proof. Define

$$\begin{array}{rcl} E_{\mathcal{E}}(\mathcal{T}, n) & = & [\, \mathcal{F}_{\mathcal{T}}(n) = \mathcal{E}(symb(n))\,] \\ E_{\mathcal{E}}(\mathcal{T}, g) & = & \{E_{\mathcal{E}}(\mathcal{T}, n) \mid n \in g\}. \end{array}$$

We assume that in each equation of $E_{\mathcal{E}}(\mathcal{T}, g)$ a 'fresh' instance of $\mathcal{E}(symb(n))$ is chosen, i.e. an instance having no variables in common with any other type appearing in $E_{\mathcal{E}}(\mathcal{T}, g)$.
(1) Obvious.
(2) Since $\mathcal{T}^*$ is an $\mathcal{E}$-typing for g, for each $n \in g$ there exists a substitution $*_n$ such that

$$(\mathcal{F}_{\mathcal{T}}(n))^* = \mathcal{E}(symb(n))^{*_n}.$$

Each substitution $*_n$ has only on effect on the corresponding instance of $\mathcal{E}(symb(n))$. Hence one can write $*_1$ as a composition of $*$ and all substitutions $*_n$ for $n \in g$. $\square$

3.12. Definition. Let g be a graph, and $\mathcal{T}$ an $\mathcal{E}$-typing for g. $\mathcal{T}$ is a *principal* $\mathcal{E}$-typing for g if

$$\mathcal{T}' \text{ is an } \mathcal{E}\text{-typing for } g \ \Rightarrow\ \mathcal{T}' = \mathcal{T}^* \text{ for some } *.$$

3.13. Principal Type Theorem. *There exists a recursive function pt such that*

$$\begin{array}{rcl} g \textit{ is } \mathcal{E}\textit{-typable} & \Rightarrow & pt(g) \textit{ is a principal } \mathcal{E}\textit{-typing for } g; \\ g \textit{ is not } \mathcal{E}\textit{-typable} & \Rightarrow & pt(g) = \mathsf{fail}. \end{array}$$

Proof. Set $\mathcal{T}_0(n) = \alpha_n$ where α_n is fresh. Define

$$\begin{array}{rcll} pt(g) & = & \mathcal{T}_0^* & \text{if Unify}(E_{\mathcal{E}}(\mathcal{T}_0, g)) = *, \\ & = & \mathsf{fail} & \text{if Unify}(E_{\mathcal{E}}(\mathcal{T}_0, g)) = \mathsf{fail}. \end{array}$$

Suppose g is $\mathcal{E}$-typable, and $\mathcal{T}$ is an $\mathcal{E}$-typing for g. Obviously, $\mathcal{T} = \mathcal{T}_0^{*_0}$ for some $*_0$. By proposition 3.11 (2), there exist an $*_1$ such that $*_1 \models E_{\mathcal{E}}(\mathcal{T}_0, g)$ and $\mathcal{T}_0^{*_1} = \mathcal{T}_0^{*_0}$. Consequently, $\text{Unify}(E_{\mathcal{E}}(\mathcal{T}_0, g)) = *$ is defined and $\mathcal{T}_0^*$ is a typing for g. Since $*$ is the most general unifier of $E_{\mathcal{E}}(\mathcal{T}_0, g)$ it follows that $*_1 = *_2 \circ *$ for some $*_2$. Now

$$(\mathcal{T}_0^*)^{*_2} = \mathcal{T}_0^{*_1} = \mathcal{T}_0^{*_0} = \mathcal{T}.$$

If g is not typable, then there exists no $*$ such that $* \models E_{\mathcal{E}}(\mathcal{T}_0, g)$, and hence

$$\text{Unify}(E_{\mathcal{E}}(\mathcal{T}_0, g)) = \mathsf{fail} = pt(g). \ \square$$

Type assignment for rewrite rules

3.14. Definition. (i) Let $g = \langle h, l, r\rangle$ be a graph with two roots. A *balanced $\mathcal{E}$-typing* for g is an $\mathcal{E}$-typing $\mathcal{T}$ for h such that

$$\mathcal{T}(l) = \mathcal{T}(r).$$

(ii) The denotation $\mathcal{E} \vdash R : \vec{\sigma} \rightarrowtail \tau$ indicates that there exists a balanced $\mathcal{E}$-typing $\mathcal{T}$ for g_R with $\mathcal{F}_{\mathcal{T}}(l) = \vec{\sigma} \rightarrowtail \tau$.

3.15. Definition. (i) Let R be a rewrite rule. An *$\mathcal{E}$-typing* for R is a type assignment $\mathcal{T}$ to g that meets the following requirements.

(1) $\mathcal{T}$ is a balanced $\mathcal{E}$-typing $\mathcal{T}$ for R;
(2) $\mathcal{T} \restriction (R \mid l) = pt(R \mid l)$.

(ii) A set $\mathcal{R}$ of rewrite rules is *$\mathcal{E}$-typable* if each $R \in \mathcal{R}$ has an $\mathcal{E}$-typing.

The requirement for mentioned in (2) ensures that the actual typing of a graph (restricted to the matching part) is an instance of the typing of R (restricted to $R \mid l$) whenever R is applicable.

Type environments

In this subsection we will describe two standard constructions for type environments: an environment $\mathcal{E}_{\mathcal{A}}$ based on the algebraic type system $\mathcal{A}$, and an environment $\mathcal{E}_{\mathcal{F}}$ for functional symbols, based on a type assignment to the proper function symbols.

3.16. Definition. Let $\mathcal{A}$ be an algebraic type system. The *algebraic environment* for $\Sigma_{\mathcal{A}}$ associated with $\mathcal{A}$ (notation $\mathcal{E}_{\mathcal{A}}$) is obtained by setting for each declaration $T\vec{\alpha} = C_1\vec{\sigma_1} \mid C_2\vec{\sigma_2} \mid \cdots$ and each i

$$\mathcal{E}_{\mathcal{A}}(C_i) = \vec{\sigma_i} \rightarrowtail T\vec{\alpha}.$$

3.17. Definition. For each symbol type $\sigma = (\sigma_1, \ldots, \sigma_k) \rightarrowtail \tau$ and $j \leq k$, the *j-th Curried version* of σ (notation $\sigma_j^{\mathcal{C}}$) is the type

$$(\sigma_1, \ldots, \sigma_j) \rightarrowtail \sigma_{j+1} \rightarrow (\sigma_{j+2} \rightarrow (\cdots \rightarrow (\sigma_k \rightarrow \tau)\cdot\cdot)).$$

3.18. Definition. (i) A *function type environment* is a map $\mathcal{F} : \Sigma^{\mathrm{f}}_{\mathcal{F}} \rightarrow \mathbb{T}_{\mathrm{S}}$.

(ii) Such a function type environment induces a type environment for $\Sigma_{\mathcal{F}}$ (notation $\mathcal{E}_{\mathcal{F}}$) in the following way.

$$\begin{aligned} \mathcal{E}_{\mathcal{F}}(F) &= \mathcal{F}(F), \\ \mathcal{E}_{\mathcal{F}}(F_j) &= \mathcal{F}(F)^{\mathcal{C}}_j. \end{aligned}$$

These constructions are combined in the following way.

3.19. Definition. Let $\mathfrak{T}$ be an applicative TGRS over $\mathcal{A}$, and let $\mathcal{F}$ be a function environment.

(i) The *combined type environment* $\mathcal{E}_{\mathcal{F},\mathcal{A}}$ is obtained by setting

$$\mathcal{E}_{\mathcal{F},\mathcal{A}} = \mathcal{E}_0 \cup \mathcal{E}_{\mathcal{F}} \cup \mathcal{E}_{\mathcal{A}}.$$

(ii) Typability in $\mathcal{F}, \mathcal{A}$ is denoted by writing $\mathcal{F}, \mathcal{A} \vdash g : \sigma$ instead of $\mathcal{E}_{\mathcal{F},\mathcal{A}} \vdash g : \sigma$.

Subject reduction

In this subsection we will show that typing is preserved under rewriting.

First, we will derive some properties concerning the interaction between typing and the basic graph operations matching, extension, redirection and garbage collection. For the moment, fix some type environment $\mathcal{E}$.

3.20. Matching Lemma. *Let p, g be graphs, and let $\mu : p \underset{\mathrm{m}}{\to} g$ be a match. Furthermore, let $\mathcal{T}$ be a typing for g. Then $\mathcal{T} \circ \mu$ is a typing for p.*

Proof. Let $n \in p$. The case $n \in p^{\circ}$ is trivial. Suppose $n \in p^{\bullet}$. Then

$$\mathcal{F}_{\mathcal{T}\circ\mu}(n) \subseteq \mathcal{E}(symb(\mu(n))) = \mathcal{E}(symb(n)). \ \square$$

3.21. Extension Lemma. *Let $\Delta = \langle R, \mu \rangle$ be a redex in g. Let $\mathcal{T}_g, \mathcal{T}$ be typings for g and $R \mid r$ respectively. Set $O = (R \mid r) \cap (R \mid l)$. Then*

$$\mathcal{T} \restriction O = (\mathcal{T}_g \circ \mu) \restriction O \ \Rightarrow \ \mathcal{T}_g + \mathcal{T} \text{ is a typing for } g + \Delta.$$

Here $\mathcal{T}_g + \mathcal{T}$ denotes the extension of $\mathcal{T}_g$ with $\mathcal{T}$.

Proof. By a case distinction. The case $n \in g$ is trivial. Suppose $n \in (R \mid r)^{\bullet} \backslash (R \mid l)^{\bullet}$. Observe that $\mathcal{T}_g + \mathcal{T}(args_{g+\Delta}(n)_i) = \mathcal{T}(args_R(n)_i)$ for any i. Hence $\mathcal{F}_{\mathcal{T}_g+\mathcal{T}}(n) \subseteq \mathcal{E}(symb(n))$. $\square$

3.22. Redirection Lemma. *Let $\mathcal{T}$ be a typing for g. Furthermore, let $m, n \in g$. If $\mathcal{T}(m) = \mathcal{T}(n)$ then $\mathcal{T}$ is a typing for $g[m := n]$.*

Proof. Trivial. $\square$

3.23. Garbage Collection Lemma. *Let $\mathcal{T}$ be a typing for g. Then $\mathcal{T} \restriction N_{\mathrm{GC}(g)}$ is a typing for* $\mathrm{GC}(g)$.

Proof. Trivial. $\square$

3.24. Subject Reduction Theorem. *Let $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$. Suppose $\mathcal{R}$ is $\mathcal{E}$-typable. Then for any $g, h \in \mathcal{G}$*

$$\mathcal{E} \vdash g : \sigma, \ g \to h \ \Rightarrow \ \mathcal{E} \vdash h : \sigma.$$

Proof. Let $\mathcal{T}$ be an $\mathcal{E}$-typing for g. Let $\mathcal{T}_R$ be an $\mathcal{E}$-typing for R. Consider $\mathcal{T} \circ \mu$. By the Matching Lemma this is a typing for $R \mid l$. Hence $\mathcal{T} \circ \mu = (\mathcal{T}_R \restriction (R \mid l))^*$ for some $*$, by the principal type property. Then $\mathcal{T}_R^*$ is a typing for $R \mid r$ by the Instantiation Lemma 3.8. Set $\mathcal{T}_h = \mathcal{T} + \mathcal{T}_R^*$. This is a typing for $g + \Delta$, by the Extension Lemma. Note that $\mathcal{T}_h(r(\Delta)) = \mathcal{T}(\mu(l))$. Now we are done by the Redirection Lemma and the Garbage Collection Lemma. $\square$

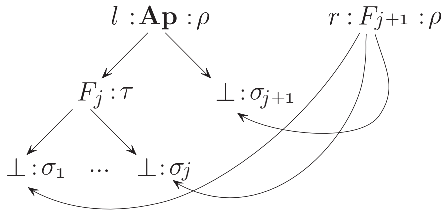
We now consider typings for applicative TGRS's. Until now, rules in $\mathcal{R}_{\mathcal{F}}$ and $\mathcal{R}_{\mathcal{C}}$ have been treated in the same way. It is intuitively clear that the typing for proper function symbols and their Curry variants are related. For applicative systems we will consider environments with the following property.

3.25. Definition. Let $\mathcal{E}$ be a type environment for $\mathfrak{T}$. Then $\mathcal{E}$ is said to be *applicative* if for any $F \in \Sigma^{\mathrm{f}}_{\mathcal{F}}$ and $j < \operatorname{arity}(F)$

$$\mathcal{E}(F_j) = \mathcal{E}(F)^{\mathcal{C}}_j.$$

In such an applicative environment the typing notion for $\mathcal{R}_{\mathcal{C}}$ is straightforward.

3.26. Definition. Let $F \in \Sigma_{\mathcal{F}}$ with arity $k \geq 1$. Say $\mathcal{F}(F) = (\sigma_1, \ldots, \sigma_k) \rightarrowtail \sigma_{k+1}$. Let $j < k$. Set $\rho = \sigma_{j+2} \to \cdots \to \sigma_{k+1}$, and $\tau = \sigma_{j+1} \to \rho$. The *standard typing* for Ap^F_j is the type assignment indicated by the following picture. (For simplicity we write F as F_k.)



3.27. Lemma. *Typing in applicative environments satisfies the subject reduction property for Curry reductions.*

Proof. Obvious since the above typing is a rule typing for each Ap^F_j. $\square$

In the remainder of this subsection we will formulate a criterion for balanced typings of functional rewrite rules which implies the subject reduction property for the complete applicative TGRS. It is possible to express a sufficient typing condition avoiding the principal type property. The results apply to a large class of TGRS's and applicative type environments, notably those related to functional programming languages.

3.28. Definition. Let $\mathcal{E}$ be an environment for $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$. Then $\mathcal{E}$ is called *principal* for $\mathfrak{T}$ if for each $R \in \mathcal{R}_{\mathcal{F}}$ (say for F) one has

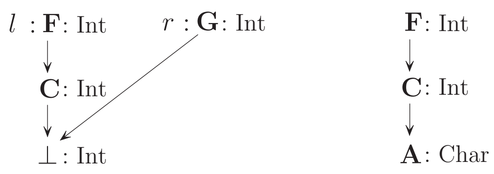
$$\mathcal{E} \vdash R : \mathcal{E}(F).$$

Requiring that $\mathcal{E}$ is principal is not sufficient to guarantee preservance of typing under reduction.

3.29. Example. Let $\mathcal{E}$ contain the following type declarations.

$\mathbf{F}$	:	$\mathrm{Int} \rightarrowtail \mathrm{Int}$,
$\mathbf{C}$	:	$\alpha \rightarrowtail \mathrm{Int}$,
$\mathbf{G}$	:	$\mathrm{Int} \rightarrowtail \mathrm{Int}$,
$\mathbf{A}$	:	Char.

The respective typings of the rule $\mathbf{F}(\mathbf{C}(x)) \to \mathbf{G}(x)$ and the graph $g = \mathbf{F}(\mathbf{C}(\mathbf{A}))$ are



However, rewriting g results in the graph $\mathbf{G}(\mathbf{A})$ which is not typable.

The problem arises from the fact that the typing of the matching part of the object graph cannot be extended to a typing of the right-hand side of the rule. In the example, observe that typing the right-hand side of the rule for **F** results in a type for $\bot$ (namely Int) that is not induced by the environment type for **F**. This leaves some freedom, admitting type-correct object graphs conflicting the assumed left-hand side typing in lower pattern nodes. The idea is to force encoding of pattern types in the environment type for **F**; thus the type structure of any matching part is uniquely determined by this environment type. This is established by a requirement on environment types of pattern symbols.

3.30. Definition. (i) A symbol type $(\sigma_1, \ldots, \sigma_k) \rightarrowtail \tau$ is *argument preserving* if

$$\forall i \leq k \; [\, \mathrm{TV}(\sigma_i) \subseteq \mathrm{TV}(\tau) \,].$$

(ii) Let $\mathcal{E}$ be an environment, and $S \in \Sigma$. Then $\mathcal{E}$ is *argument preserving* w.r.t. S if $\mathcal{E}(S)$ is argument preserving.

(iii) Let $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ be a TGRS. Then $\mathcal{E}$ is *safe* for $\mathfrak{T}$ if $\mathcal{E}$ is principal for $\mathfrak{T}$ and $\mathcal{E}$ is argument preserving w.r.t. each pattern symbol of $\mathcal{R}_{\mathcal{F}}$.

In order to prove that any $\mathcal{E}$-typing for $\mathfrak{T}$ satisfies the subject reduction property whenever $\mathcal{E}$ safe for $\mathfrak{T}$, it will be sufficient to show that the type variables occurring in a pattern typing are propagated upwards.

3.31. Variable Propagation Lemma. *Let $\mathcal{E}$ be safe for $\mathfrak{T}$. Let $R \in \mathcal{R}$. Let $\mathcal{T}$ be an $\mathcal{E}$-typing for $R \mid l$.*

(i) *For any pattern node n of R and any $i \leq \mathrm{arity}(n)$*

$$\mathrm{TV}(\mathcal{T}(\mathit{args}(n)_i)) \subseteq \mathrm{TV}(\mathcal{T}(n)).$$

(ii) *For any node $n \in R \mid l$*

$$\mathrm{TV}(\mathcal{T}(n)) \subseteq \mathrm{TV}(\mathcal{F}_{\mathcal{T}}(l)).$$

Proof. (i) Straightforward.

(ii) By (i). $\square$

We can now show that any functional rewrite rule is typable according in the original sense (cf. definition 3.15).

3.32. Proposition. *Let $\mathcal{E}$ be safe for $\mathfrak{T}$. Let $R \in \mathcal{R}_{\mathcal{F}}$. Then R is typable.*

Proof. Since $\mathcal{E}$ is principal there exists a balanced $\mathcal{E}$-typing for R, say $\mathcal{T}$, with $\mathcal{F}_{\mathcal{T}}(l) = \mathcal{E}(\mathit{symb}(l))$. We will show that $\mathcal{T} \restriction (R \mid l) = \mathit{pt}(R \mid l)$. Then we are done, since $\mathcal{T}$ is a rule typing. Indeed, set $\mathcal{T}' = \mathcal{T} \restriction (R \mid l)$. Say $\mathcal{T}' = \mathit{pt}^*(R \mid l)$. Since $\mathcal{F}_{\mathit{pt}(R|l)}(l) \subseteq \mathcal{E}(\mathit{symb}(l))$ one has $\mathcal{F}_{\mathcal{T}'}(l) = \mathcal{F}_{\mathit{pt}(R|l)}(l)$. Now by the Variable Propagation Lemma (ii) the substitution $*$ is the identity function. $\square$

3.33. Corollary. *Let $\mathfrak{T}$ be a TGRS. Let $\mathcal{E}$ be an applicative type environment for $\mathfrak{T}$. If $\mathcal{E}$ is safe for $\mathfrak{T}$, then $\mathcal{E}$-typing satisfies the subject reduction property.*

PROOF. By lemma 3.27 and proposition 3.32. □

This immediately leads to a result on $\mathcal{F}, \mathcal{A}$-typings.

3.34. DEFINITION. Let $\mathcal{F}$ be a function type environment $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$.

(i) $\mathcal{F}$ is called *Curry safe* for $\mathfrak{T}$ if for any Curry variant F_j occurring as pattern symbol in $\mathcal{R}_{\mathcal{F}}$ the following holds. Say $\mathcal{F}(F) = (\sigma_1, \ldots, \sigma_k) \rightarrowtail \sigma_{k+1}$. Then

$$\mathrm{TV}(\sigma_1) \cup \cdots \cup \mathrm{TV}(\sigma_j) \subseteq \mathrm{TV}(\sigma_{j+1}) \cup \cdots \cup \mathrm{TV}(\sigma_{k+1}).$$

(ii) Let $\mathcal{A}$ be an algebraic type system. Then $\mathcal{F}$ is said to be *principal* for $\mathfrak{T}$ with respect to $\mathcal{A}$ if $\mathcal{E}_{\mathcal{F},\mathcal{A}}$ is principal for $\mathfrak{T}$.

3.35. THEOREM. *Let $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ be applicative over $\mathcal{A}$. Let $\mathcal{F}$ be a function type environment for $\mathfrak{T}$. Suppose $\mathcal{F}$ is Curry safe and principal for $\mathfrak{T}$ (w.r.t. $\mathcal{A}$). Then for any $g, h \in \mathcal{G}$*

$$\mathcal{F}, \mathcal{A} \vdash g : \sigma, \; g \rightarrow h \;\; \Rightarrow \;\; \mathcal{F}, \mathcal{A} \vdash h : \sigma.$$

PROOF. Let $\mathcal{F}$ is Curry safe and principal for $\mathfrak{T}$ (w.r.t. $\mathcal{A}$). First note that $\mathcal{E}_{\mathcal{F},\mathcal{A}}$ is applicative. Moreover $\mathcal{E}_{\mathcal{F},\mathcal{A}}$ is principal for $\mathfrak{T}$. Moreover $\mathcal{E}_{\mathcal{F},\mathcal{A}}$ is argument preserving by Curry safety of $\mathcal{F}$ and the restriction on variable occurrences in algebraic type definitions. Hence $\mathcal{E}_{\mathcal{F},\mathcal{A}}$ is safe for $\mathfrak{T}$ and we are done by corollary 3.33. □

In the functional language *Miranda* only algebraic constructors are allowed as pattern symbols. Moreover, the type system of *Miranda* combines the systems of the type systems of Milner and Mycroft. Milner's type system is used for inference of types for functions for which no type is specified. Mycrofts's type system is applied to verify user-specified function types. This leads to a principal function type environment. Without describing the way to interpret *Miranda* scripts as graph rewrite systems, we conclude the following.

3.36. COROLLARY. *Typing in Miranda is preserved under evaluation.*

4. Introduction to uniqueness typing

In the sequel of this paper, conventional typing will be extended with so called *uniqueness information*. This will enable us to indicate constraints on the *reference structure* of function-argument combinations. Consequently, the validity of an application of a function will depend on the *context* in which it appears, notably on the number of (external) references to its arguments.

This is best explained by an example. Suppose $\mathbf{F}$ is a unary operation, and we wish to require that in any application of $\mathbf{F}$, the access of $\mathbf{F}$ to its argument is *unique*, i.e. the reference count of that argument object is 1. In our approach, this is done by dressing the conventional environment type of $\mathbf{F}$, say $\sigma \rightarrowtail \tau$, with a uniqueness attribute on the argument type: $\overset{\bullet}{\sigma} \rightarrowtail \tau$. An application of $\mathbf{F}$ is then correct if the actual argument of $\mathbf{F}$ is of the right (conventional) type, and moreover the reference count of this object is 1. This is, however, not sufficient to maintain the correctness of the **F**-application: the uniqueness of **F**'s argument should be preserved during reduction.

$$\begin{tikzcd} & \mathbf{F} \arrow[d] & \\ & \mathbf{G} \arrow[dl] \arrow[dr] & \\ {} & & {} \end{tikzcd}$$

In the above example, evaluation of **G** should not increase the reference count of **F**'s argument.

The idea is to encode this so called *access stability* in the result types of functions, again by a uniqueness attribute, e.g. $\mathbf{G} : (\rho_1, \rho_2) \rightarrowtail \stackrel{\bullet}{\sigma}'$. The typing requirements for rewrite rules will ensure that any application of such **G** is access stable: the reference count of the result is not greater than the reference count of the application. One could say that such an application is *potentially unique*: uniqueness of the application and its reducts is guaranteed whenever the present reference count is 1.

Observe that the potential uniqueness of an object can be handled in two ways, when offered as an argument of a function **F**. The information can either be used (if **F** expects a unique argument, and the reference count if 1) or discarded (if **F** does not require any uniqueness, or the reference count is greater then 1). In the former case, the potentially unique object is *actually unique* for **F**; in the latter it *loses* its uniqueness. This relation between offered and demanded type, determined by the reference situation, is expressed by *coercion relations*. Roughly spoken, potentially unique objects can be coerced to actually unique or to nonunique ones.

For some objects, however, one explicitly requires that they are *guaranteed unique* rather than just potentially unique. This is the case for curried function applications. We reserve the attribute $\vartriangle$ for such objects.

Pattern matching and algebraic types

The treatment of functions using pattern matching (through algebraic constructors) is somewhat delicate. Suppose **F** has a rewrite rule with the following pattern.

$$\begin{array}{c}\mathbf{F}\\ \downarrow\\ \mathbf{C}\\ \downarrow\\ \perp\end{array}$$

Moreover suppose we wish the argument of **C** in any matching part of a graph to be unique for **F**, i.e. there is only one path from **F** to this argument (say X), and X is only reachable via **F**.

This requirement needs to be decomposed into referencewise dependencies, since an actual application of **F** might not yet be a redex, but could become one after some reduction steps. It is needed to anticipate on future completion of the pattern.

The desired property of X splits into two requirements: X should be unique for **C**, and the **C**-node should be unique for **F**. The idea is to encode this information entirely into the type of **F**: both the uniqueness of $\perp$ and the uniqueness of **C** are indicated in the corresponding components of **F**'s argument type. This looks like $\mathbf{F} : (\cdots \overset{\bullet}{\sigma} \cdots)^{\bullet} \rightarrowtail \tau$, where σ is the type of the $\perp$-argument. The result is that any partially matching application

$$\begin{tikzcd} & \mathbf{F} \arrow[d] & \\ & \mathbf{H} \arrow[dl] \arrow[dr] & \\ {} & & {} \end{tikzcd}$$

has a unique first reference, and the type of the **H**-application is of the form $(\cdots \overset{\bullet}{\sigma'} \cdots)^{\bullet}$. The outermost uniqueness attribute expresses access stability whereas the other uniqueness attribute ensures uniqueness of the second reference as soon as the pattern is completed.

The conventional types for algebraic constructors are furnished with uniqueness information. This leads to several uniqueness variants of these constructor types.

For the constructor **Cons** of lists, e.g., the possible variants include

$$\boxed{\begin{array}{lcll lcll} \mathbf{Cons} & : & (\alpha, \mathrm{List}(\alpha)) \rightarrowtail \mathrm{List}(\alpha) & (1) & \mathbf{Cons} & : & (\overset{\bullet}{\alpha}, \mathrm{List}(\overset{\bullet}{\alpha})) \rightarrowtail \mathrm{List}(\overset{\bullet}{\alpha}) & (3) \\ \mathbf{Cons} & : & (\alpha, \overset{\bullet}{\mathrm{List}}(\alpha)) \rightarrowtail \overset{\bullet}{\mathrm{List}}(\alpha) & (2) & \mathbf{Cons} & : & (\overset{\bullet}{\alpha}, \overset{\bullet}{\mathrm{List}}(\overset{\bullet}{\alpha})) \rightarrowtail \overset{\bullet}{\mathrm{List}}(\overset{\bullet}{\alpha}) & (4) \end{array}}$$

With (1) ordinary lists can be built. (2) can be used for lists of which the 'spine' is unique, (3) for lists containing unique elements, and (4) for lists of which both the spine and the elements are unique.

The abovementioned encoding of uniqueness of 'deeper' arguments in the pattern of **F** is achieved by making the types of algebraic constructors *uniqueness propagating*: if one of their arguments is unique then the result is unique as well. This entails that the uniqueness of the $\perp$-argument is 'propagated upwards', leading to the uniqueness of the **C**-node. For the **Cons** example this means that variant (3) is rejected.

Uniqueness propagation imposes a restriction on the occurrences of type constructors: e.g. the type $\mathrm{List}(\overset{\bullet}{\mathrm{Int}})$ does not make sense. This is plausible if one realises that in an application

$$\begin{tikzcd} & \mathbf{F} \arrow[d] & {} \arrow[dl] \\ & \mathbf{Cons} \arrow[dl] \arrow[dr] & \\ \mathbf{3} & & g \end{tikzcd}$$

neither of the bottommost nodes can be considered unique (when accessed through **Cons**) if the **Cons** node itself is not potentially unique. The argument of the type constructor List is said to be a *uniqueness propagating position*. The uniqueness propagating positions of type constructors are deduced by analyzing their algebraic definition.

The coercion relation mentioned above introduces a notion of *subtyping* on uniqueness variants. Intuitively, a coercion statement $\sigma \leq \tau$ means that every σ-object can be regarded as a τ-object. If the definition of T is

$$T\,\vec{\alpha} = C_1\,\vec{\sigma_1} \mid C_2\,\vec{\sigma_2} \mid \cdots$$

then we would like

$$T\vec{\rho} \leq T\vec{\rho'} \tag{$*$}$$

if and only if for any i

$$\vec{\sigma_i}[\vec{\alpha} := \vec{\rho}] \leq \vec{\sigma_i}[\vec{\alpha} := \vec{\rho'}].$$

The idea is that an object of type $T\vec{\rho}$ (say built by C_i) can be considered as a $T\vec{\rho'}$ object if the respective arguments of C_i can be coerced to each other. The expression $(*)$ is reduced to componentwise coercion relations between $\vec{\rho}$ and $\vec{\rho'}$, determined by analyzing the occurrences of the variables $\vec{\alpha}$ in the algebraic specification of $T\vec{\alpha}$.

Under special circumstances it is possible to use nonalgebraic constructors in patterns of rewrite rules. We will go into this at the end of the paper.

Refined reference count and locality

The reference count approach described above is rather rough. An object is considered nonunique if its reference count is greater than 1.

In practice one often uses a graph rewrite system together with a specific reduction strategy. The idea is that multiple references to a node are harmless if one knows that only one of them remains at the moment of evaluation. E.g. the standard evaluation of a conditional statement **If** c **Then** t **Else** e causes first the evaluation of the c part, and subsequently evaluation of either t or e, but not both. Hence, a single access to a node n in t combined with a single access to n in e would overall still result in a 'unique' access to n.

This idea is generalized to arbitrary symbols by classifying the arguments according to the intended evaluation order. Now suppose p, q are paths from m to n such that p and q are disjoint between start and end point. Whether destructive access to n via p and q respectively is considered harmful depends on the argument classification of m.

This contrasts the reference count approach, which treats all references to a given node (the so called *access set* of that node) in the same way: the uniqueness of an argument only depends on the size of the corresponding access set. In the refined approach, some access references are considered harmful, others not. This makes it necessary to distinguish between references in an access set. In the paper we do this by a labeling mechanism.

If the above p is indeed 'dangerous', destructive access to n via p will be prevented by *labeling* some reference in a tail part of p (containing only data nodes) as 'read-only' ($\otimes$). Possible 'write' access is indicated by $\odot$.

The refined approach will be such that the access stability information (encoded in result types of functions) is still valid. The notion of uniqueness is modified as follows. A node n is unique for a node m if n is *local* for m, i.e. m is only reachable via n. Note that this is indeed a weaker notion than the previous one. It is, however, still usable to model destructive usage.

For the reference analysis it is convenient if the root of the graph is not involved in the rewrite process. For this reason we consider only TGRS's in which each object graph has the following standard shape: the root has in-degree 0 and contains a fixed data symbol **Root** of arity 1 (in Σ_0). For the analysis of other graphs g (notably the right-hand sides of rewrite rules) one temporarily attaches such an extra root to g. The resulting graph is denoted as g^+.

5. Usage analysis

As mentioned before, one can refine the reference count approach by taking a specific reduction order into account. As a first step, the idea is to characterize (as precisely as possible) an area in each graph containing the next redex to be contracted. The evaluation process of a graph (leading to the selection of a redex) usually traverses the graph from its root downwards, directed by the function-argument structure. This motivates the following classification, which translates the chosen evaluation directions into a (context independent) division of direct symbol arguments.

5.1. DEFINITION. (i) A *pre-classification* (for a symbol set Γ) is a function $\mathcal{P}$ such that for any S one has $\mathcal{P}(S) \subseteq \{1, \ldots, \text{arity}(S)\}$.
(ii) S is called *simple* if $\mathcal{P}(S) = \{1, \ldots, \text{arity}(S)\}$.

Arguments of S occurring in $\mathcal{P}(S)$ are called *primary arguments* of S. The others are *secondary arguments*. The above characterization can now be formalized by describing the 'active area' in a graph, consisting of nodes reachable from the root by descending via primary arguments. These may appear as redex roots.

5.2. DEFINITION. Let g be a graph.
(i) A reference $(n, i) \in \mathit{Ref}_g$ is a *primary reference* if i is a primary of $\mathit{symb}_g(n)$.
(ii) A path p is a *primary path* if each reference on p is primary reference.
(iii) A node $n \in g$ is a *primary node* if $p : r_g \rightsquigarrow n$ for some primary path p.
(iv) Let $\Delta = \langle R, \mu \rangle$ be a redex in g. Then Δ is a *primary redex* if $\mu(l)$ is a primary node. The associated reduction relation is denoted by $\underset{\mathcal{P}}{\rightarrow}$.

In this paper we concentrate on $\underset{\mathcal{P}}{\rightarrow}$. The idea is to classify multiple access within the active area as harmful. Analogously multiple access within the nonactive area is considered dangerous. When a combination of accesses (active and nonactive) occurs, only the active access is marked as unsafe.

The analysis of the nonactive accesses can be refined since in many cases (notably case distinctions c.q. conditionals) one can subdivide the secondary arguments into mutually exclusive groups such that accesses from different groups do not conflict.

5.3. DEFINITION. A *classification* for Γ is a pair $(\mathcal{P}, \mathcal{S})$ such that $\mathcal{P}$ is a pre-classification, and each $\mathcal{S}(S)$ is a partition of the secondary arguments $\{1, \ldots, \text{arity}(S)\} \backslash \mathcal{P}(S)$. We usually indicate the resulting sets by $\mathcal{P}^S, \mathcal{S}_1^S, \ldots, \mathcal{S}_n^S$, where n is the size of $\mathcal{S}(S)$.

A classification $(\mathcal{P}, \mathcal{S})$ induces a dependency relation on references and paths.

5.4. DEFINITION. (i) Let $S \in \Sigma$. Say S has arity ℓ. The relations $\sim_S, \lhd_S$ and $\lesssim_S$ on $\{1, \ldots, \ell\}$ are defined by

$$
\begin{aligned}
i \lhd_S j \quad &\Leftrightarrow \quad i \in \mathcal{P}^S \text{ and } j \notin \mathcal{P}^S, \\
i \sim_S j \quad &\Leftrightarrow \quad i, j \in \mathcal{P}^S \text{ or } i, j \in \mathcal{S}_k^S \text{ for some } k, \\
i \trianglelefteq\!\!\sim_S j \quad &\Leftrightarrow \quad i \lhd_S j \text{ or } i \sim_S j.
\end{aligned}
$$

(ii) Let g be a graph. The relation $\underset{\sim}{\lhd}$ is defined on nonempty paths in g starting with the same node n, by

$$p \underset{\sim}{\lhd} q \;\Leftrightarrow\; [p] \underset{\sim}{\lhd}_{symb_g(n)} [q].$$

The dependency relation for direct arguments can be translated into a (global) dependency relation on all references in a graph.

5.5. Definition. Let p, q be paths in g.

(i) p and q *diverge* (notation $p \wedge q$) if p, q start in the same node and are distinct elsewhere. More precisely, $p \wedge q$ if either $p = ()$ or $q = ()$ or

$$\mathrm{r}(p) = \mathrm{r}(q) \quad \text{and} \quad (\tilde{p})^- \# (\tilde{q})^-.$$

(ii) Let $a, b \in Ref_g$. By $(p, a) \wedge (q, b)$ we denote that $p \wedge q$ and p, q are extendible with a, b respectively.

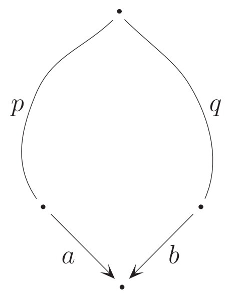
The relation $\underset{\sim}{\lhd}$ on diverging paths induces the following relation on Ref_g and N_g respectively. Intuitively, $a \underset{\sim}{\lhd} b$ means that a might be used before b.

5.6. Definition. Let g be a graph, and $a, b \in Ref_g$. Then

$$a \underset{\sim}{\lhd} b \;\Leftrightarrow\; p * a \underset{\sim}{\lhd} q * b \text{ for some acyclic } p, q \text{ with } (p, a) \wedge (q, b).$$

The dependency relation $\underset{\sim}{\lhd}$ provides a weighted reference count analysis.

5.7. Definition. A *critical path combination* is a quadruple p, a, q, b such that $(p, a) \wedge (q, b)$, the paths p, q are acyclic, $a \neq b$, and $\mathrm{d}(a) = \mathrm{d}(b)$.



The common start node of $p * a$ and $q * b$ is called the *joining node* of the critical path pair.

Suppose $(p, a) \wedge (q, b)$ is a critical path combination. If $p * a \underset{\sim}{\lhd} q * b$, then the reference a to $\mathrm{d}(a)$ might be used before b (in the $\lhd$ case) or a, b might be used in any order. The idea is now that $\mathrm{d}(a)$ is not allowed to be used destructively via $p * a$. This will be indicated by a suitable *labeling* of references with *use attributes* $\odot$ (for 'write use allowed') or $\otimes$ ('read access only').

A straightforward approach would label the reference a above with $\otimes$ (see example 5.10). However, the usage information considered here is only important for *function* applications, in particular for parts of the graph matching the left-hand side of a rewrite rule. Since we consider systems with patterns (containing data nodes) we can be more liberal: in the above case it is sufficient that $p * a$ contains a reference labeled $\otimes$ anywhere in its 'data tail', to be made explicit below. The typing system will be such that this suffices to prevent destructive use of $\mathrm{d}(a)$ via the indicated path.

5.8. DEFINITION. (i) p is a *data path* if each $n \in p$ is a data node. (Note that $\mathrm{d}(p)$ needs not be a data node.)

(ii) The set of *use marking attributes* is $\mathbb{M} = \{\odot, \otimes\}$.

(iii) Let $use : Ref_g \to \mathbb{M}$ be a labeling. A path p in g is *marked* if there exist paths p_1, p_2 and a reference a such that $p = p_1 * a * p_2$, p_2 is a data path, and $use(a) = \otimes$.

(iv) A labeling use is a *marking* for g if for each critical path combination $(p, a) \wedge (q, b)$ one has

$$p * a \precsim q * b \;\Rightarrow\; p * a \text{ is marked.}$$

Note that our analysis only considers non-overlapping paths, and does not distinguish data nodes from function nodes. We will not make any specific assumptions on the classification of data symbols, i.e. all data symbols are considered as simple. In order to assure that the reference analysis is stable during reduction, we require that the classification is consistent with the way arguments are passed from the left-hand to the right-hand side of the rewrite rules in the TGRS in question.

5.9. DEFINITION. Let $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ be an applicative TGRS. A *classification for* $\mathfrak{T}$ is a classification for $\Sigma^{\mathrm{f}}_{\mathcal{F}}(\mathfrak{T})$. This induces a classification for $\Sigma_{\mathfrak{T}}$ by regarding the data symbols and **Ap** as simple. It is required that the classification is consistent with each $R \in \mathcal{R}$, i.e. for each pair of border references a, b of R one has

$$a \precsim b \text{ in } R \,|\, r \;\Rightarrow\; \overline{a} \precsim \overline{b} \text{ in } R \,|\, l.$$

Note that we have abstracted from any concrete reduction strategy and consider only $\underset{\mathcal{P}}{\rightarrow}$ reductions. In general one would like a suitable approximating classification for a given strategy in order to perform the analysis in this paper. Informally, a (one step) *reduction strategy* (for $\mathfrak{T}$) is a subset $\underset{s}{\rightarrow}$ of $\underset{\mathcal{R}}{\rightarrow}$. We say that a classification $(\mathcal{P}, \mathcal{S})$ *approximates* this strategy if $\underset{s}{\rightarrow} \subseteq \underset{\mathcal{P}}{\rightarrow}$. The trivial classification (considering all symbols simple) is of course always a safe approximation. However, it is desirable to have a classification such that $\mathcal{P}(S)$ is as small as possible, and subsequently $\mathcal{S}(S)$ consists of a maximal number of distinct classes. In practice this is difficult (or even impossible), but sometimes the way to define a strategy is close to an argument classification.

The *functional reduction strategy* (used in *Concurrent Clean*) is obtained by considering the following lazy evaluation procedure. First the rules are ordered. During evaluation, the graph is examined from the root downwards. If a function symbol is encountered, it will be tried to complete a pattern by reducing arguments of that function. The ordering of the rules gives preferences. If the pattern is completed, the appropriate rule is applied and evaluation continues. This gives a natural way for determining a classification: initially, the function arguments containing a non-$\perp$ pattern in $\mathcal{R}$ are classified as primary, and all others are in singleton secondary groups. The classification is completed by determining an '$\mathcal{R}$-consistent' closure via a fixed point construction. For the conditional **If** with the obvious rules

$$\begin{array}{rcl} \mathbf{If}(\mathbf{True}, t, e) & \to & t, \\ \mathbf{If}(\mathbf{False}, t, e) & \to & e, \end{array}$$

this gives $\mathcal{P}^{\mathbf{If}} = \{1\}$, $\mathcal{S}^{\mathbf{If}}_1 = \{2\}$ and $\mathcal{S}^{\mathbf{If}}_2 = \{3\}$. The secondary classification consists of two classes since the rules are discarding either the second or the third argument.

There are two important examples of marking functions.

5.10. Example. Let g be a graph.

(i) 'Last reference' marking is done by defining use^{ℓ} as follows. Let $n \in g$. Then for each $a \in acc_g(n)$

$$\begin{array}{rcll} use^{\ell}(a) & = & \otimes & \text{if } a \mathrel{\underset{\sim}{\lhd}} b \text{ for some } b \in acc_g(n) \text{ with } b \neq a, \\ & = & \odot & \text{otherwise.} \end{array}$$

Note that this definition completely specifies the function use^{ℓ}.

(ii) Straightforward reference counting is done by considering each symbol as simple and performing last reference marking. More directly, this labeling is obtained by setting

$$\begin{array}{rcll} use^{\mathrm{rc}}(a) & = & \odot & \text{if } \mathrm{d}(a) \text{ has reference count 1,} \\ & = & \otimes & \text{otherwise.} \end{array}$$

Using the standard classification of arguments of the conditional **If**, and no specific assumptions about other symbols, the above two examples give the following markings of the displayed graph.

$$\begin{tikzcd} & \mathbf{If} \arrow[dl, "\odot"'] \arrow[d, "\odot"'] \arrow[d, bend left, "\odot"] \\ \mathbf{G} \arrow[dr, "\otimes"'] & \mathbf{H} \arrow[d, "\odot"'] \\ & \mathbf{C} \end{tikzcd}$$

use^{ℓ}

$$\begin{tikzcd} & \mathbf{If} \arrow[dl, "\odot"'] \arrow[d, "\otimes"'] \arrow[d, bend left, "\otimes"] \\ \mathbf{G} \arrow[dr, "\otimes"'] & \mathbf{H} \arrow[d, "\otimes"'] \\ & \mathbf{C} \end{tikzcd}$$

use^{rc}

6. Uniqueness types

In this section we will extend the notion of typing (introduced in section 3) with uniqueness information. We therefore consider the set of types $\mathbb{T}$ and attach to each subtype a so called *uniqueness attribute* which may be $\times$ (for 'ordinary' or 'nonunique'), $\bullet$ (for 'unique') or $\vartriangle$ (for 'necessarily unique').

6.1. Definition. (i) The set of *uniqueness attributes* is $\mathbb{U} = \{\times, \bullet, \vartriangle\}$.

(ii) The partial ordering $\leq$ on $\mathbb{U}$ is defined by

$$\bullet \leq \bullet, \quad \bullet \leq \times, \quad \times \leq \times, \quad \vartriangle \leq \vartriangle.$$

This relation mirrors the intention that potentially unique objects may be used either as unique objects or as nonunique ones. Necessarily unique objects remain unique.

Uniqueness types are built from uniqueness variables, using the standard type constructor $\rightarrow$ and type constructors defined by an algebraic type system. In the sequel, let $\mathcal{A}$ be such an system. The uniqueness propagation mentioned in section 4 restricts the applications of uniqueness variants of the type constructors in $\mathbb{C}_{\mathcal{A}}$. The dependency between the uniqueness of a type $T\vec{\sigma}$ and the uniqueness of the σ_i is determined by analyzing the dependencies in $\mathcal{A}$. Since $\mathcal{A}$ may contain (direct or indirect) recursion this is done by a fixedpoint construction.

We say that an occurrence of α in σ is *uniqueness propagating* if uniqueness of α induces uniqueness of σ. The idea is to define the notions 'uniqueness propagating occurrence' $unioccA_{\mathcal{A}}(\alpha,\sigma)$ and 'uniqueness propagating position' $uniprop_{\mathcal{A}}(T)_i$ simultaneously as the least solution of some predicate equations. This amounts to solving a least fixedpoint equation with respect to the ordering 'false' $\leq$ 'true'.

6.2. Definition. The predicates $uniocc_{\mathcal{A}}$ and $uniprop_{\mathcal{A}}$ are specified by mutual induction, as follows. For the standard constructor $\rightarrow$ both $uniprop_{\mathcal{A}}(\rightarrow)_1$ and $uniprop_{\mathcal{A}}(\rightarrow)_2$ are set 'false'.

$$\begin{array}{rcl} uniocc_{\mathcal{A}}(\alpha,\beta) & \Leftrightarrow & \alpha=\beta, \\ uniocc_{\mathcal{A}}(\alpha,T(\sigma_1,\ldots,\sigma_k)) & \Leftrightarrow & \bigvee_{i\leq k} uniprop_{\mathcal{A}}(T)_i \;\&\; uniocc_{\mathcal{A}}(\alpha,\sigma_i). \end{array}$$

Moreover for each declaration $T\vec{\alpha} = C_1\vec{\sigma_1} \mid C_2\vec{\sigma_2} \mid \cdots$

$$uniprop_{\mathcal{A}}(T)_i \;\Leftrightarrow\; \bigvee_{n} \bigvee_{j\leq \mathrm{arity}(C_n)} uniocc_{\mathcal{A}}(\alpha_i,\sigma_{nj}).$$

Using the predicates $uniprop_{\mathcal{A}}$, the collection of well-formed uniqueness types over $\mathcal{A}$ can be defined. E.g. $\overset{\bullet}{\mathrm{List}}(\overset{\bullet}{\mathrm{Int}})$ is correct, whereas $\overset{\times}{\mathrm{List}}(\overset{\bullet}{\mathrm{Int}})$ is not. The attribute $\vartriangle$ has the highest priority for propagation.

To simplify notation, we usually omit $\mathcal{A}$ in $uniprop_{\mathcal{A}}$ and $uniocc_{\mathcal{A}}$, and leave the algebraic system implicit.

6.3. Definition. Let $u_1,\ldots,u_k \in \mathbb{U}$.

(i) Let $I \subseteq \{1\ldots,k\}$. The *cumulative uniqueness attribute of $\vec{u}$ over I* (notation $\Sigma_I\vec{u}$) is defined by

$$\begin{array}{rcll} \Sigma_I\vec{u} & = & \vartriangle & \text{if } u_i=\vartriangle \text{ for some } i\in I\text{; else:} \\ & = & \bullet & \text{if } u_i=\bullet \text{ for some } i\in I, \\ & = & \times & \text{otherwise.} \end{array}$$

We omit I if it is equal to $\{1\ldots,k\}$.

(ii) Let $T\in\mathbb{C}_{\mathcal{A}}$ with arity k. The *T-relativized cumulative uniqueness attribute* of $\vec{u}$ (notation $\Sigma_T\vec{u}$) is $\Sigma_I\vec{u}$, where $I=\{i \mid uniprop(T)_i\}$.

6.4. Definition. Let $T\in\mathbb{C}_{\mathcal{A}}$, say with arity k. Let $v,u_1,\ldots,u_k\in\mathbb{U}$. Then v is said to be *T-admissible* for $u_1,\ldots,u_k$ if $v=\Sigma_T\vec{u}$ whenever $\Sigma_T\vec{u}\neq\times$.

6.5. Definition. (i) For each attribute $u\in\mathbb{U}$ the set of *uniqueness variables over u* is defined by

$$\overset{u}{\mathbb{V}} = \{\overset{u}{\alpha} \mid \alpha\in\mathbb{V}\}.$$

The collection of uniqueness variables is

$$\widehat{\mathbb{V}} = \overset{\bullet}{\mathbb{V}} \cup \overset{\times}{\mathbb{V}} \cup \overset{\vartriangle}{\mathbb{V}}.$$

(ii) For each $u \in \mathbb{U}$, the set of *uniqueness types over* u is defined by simultaneous induction as follows. Below, $u, v, u_1, \dots$ range over $\mathbb{U}$, and T over $\mathbb{C}_{\mathcal{A}}$.

$$\begin{array}{rcl}
\alpha \in \overset{u}{\mathbb{V}} & \Rightarrow & \alpha \in \overset{u}{\mathbb{T}}, \\
\sigma \in \overset{v}{\mathbb{T}}, \tau \in \overset{w}{\mathbb{T}} & \Rightarrow & \sigma \xrightarrow{u} \tau \in \overset{u}{\mathbb{T}}, \\
\left.\begin{array}{r} \sigma_1 \in \overset{u_1}{\mathbb{T}}, \dots, \sigma_k \in \overset{u_k}{\mathbb{T}}, \\ u \text{ is } T\text{-admissible for } u_1, \dots, u_k \end{array}\right\} & \Rightarrow & \overset{u}{T}(\sigma_1, \dots, \sigma_k) \in \overset{u}{\mathbb{T}}.
\end{array}$$

Set $\widehat{\mathbb{T}} = \overset{\times}{\mathbb{T}} \cup \overset{\bullet}{\mathbb{T}} \cup \overset{\triangle}{\mathbb{T}}$.

(iii) For each $\sigma \in \widehat{\mathbb{T}}$ the *attribute* of σ (notation $[\sigma]$) is defined by

$$[\sigma] = u \text{ if } \sigma \in \overset{u}{\mathbb{T}}.$$

(iv) The set of *uniqueness symbol types* (notation $\widehat{\mathbb{T}}_{\mathrm{S}}$) is defined by

$$\sigma_1, \dots, \sigma_k, \tau \in \widehat{\mathbb{T}} \;\Rightarrow\; (\sigma_1, \dots, \sigma_k) \rightarrowtail \tau \in \widehat{\mathbb{T}}_{\mathrm{S}}.$$

(v) For $\sigma \in \widehat{\mathbb{T}}, \widehat{\mathbb{T}}_{\mathrm{S}}$, let $|\sigma|$ denote the *stripped version* of σ, i.e. σ without any uniqueness attributes. Note that $|\sigma| \in \mathbb{T}, \mathbb{T}_{\mathrm{S}}$.

Notation. Let $\vec{\sigma} \in \widehat{\mathbb{T}}$. Then $\Sigma\vec{\sigma}$ denotes the cumulative uniqueness attribute $\Sigma[\vec{\sigma}]$. Analogously we use the T-relativized version.

As was mentioned in section 4, the description of type assignment uses coercion relations, depending on the kind of reference via which an argument is accessed by a function or constructor.

We introduce coercion relations for access via $\odot$-references and $\otimes$-references respectively. The idea is that unique objects keep their uniqueness while being passed via $\odot$-arcs, and lose it when they are accessed via $\otimes$-references. Furthermore, a unique object can be coerced to a nonunique one (if such a nonunique object is demanded in a function application). Necessarily unique objects remain unique in all cases.

Intuitively, a coercion statement $\sigma \leq \tau$ means that every σ-object can be regarded as a τ-object. For function types this view leads to the rule

$$\sigma' \leq \sigma, \tau \leq \tau' \;\Rightarrow\; \sigma \rightarrow \tau \leq \sigma' \rightarrow \tau'.$$

Note the so called contravariance in the first argument of the type constructor $\rightarrow$: one says that the first argument occurs on a *negative* position. This is generalized to arbitrary constructors: the coercion rules are made dependent on a 'negative/positive' classification of constructor arguments.

First we will explain how an algebraic type system $\mathcal{A}$ determines a classification function $sign_{\mathcal{A}}$ for type constructors. The idea is that the classification of a type constructor is deduced from the syntactical form of the types in each clause of its algebraic definition. The technique is analogous to the specification method for *uniprop*.

6.6. Definition. (i) The *sign classification lattice* $\mathbb{S}$ is the set $\{\bot, \oplus, \ominus, \top\}$, partially ordered by $\sqsubseteq$, as follows.

$$\begin{tikzcd}[column sep=small] & \top \arrow[dl, dash] \arrow[dr, dash] & \\ \oplus \arrow[dr, dash] & & \ominus \arrow[dl, dash] \\ & \bot & \end{tikzcd}$$

(ii) The binary operation $\cdot$ on $\mathbb{S}$ is defined as follows. Here s ranges over $\mathbb{S}$.

$$\begin{aligned} \bot \cdot s = s \cdot \bot &= \bot, \\ \top \cdot s = s \cdot \top &= \top \quad \text{if } s \neq \bot, \\ \oplus \cdot \oplus = \ominus \cdot \ominus &= \oplus, \\ \oplus \cdot \ominus = \ominus \cdot \oplus &= \ominus. \end{aligned}$$

The classification of type constructor arguments is determined using an auxiliary function *occ*, determining the kind of occurrences of each variable in a given type.

6.7. Definition. The functions $occ\ (= occ_{\mathcal{A}}) : \mathbb{V} \times \mathbb{T} \to \mathbb{S}$ and $sign\ (= sign_{\mathcal{A}})$ are specified below by simultaneous induction. For the moment we consider $\to$ as an ordinary constructor with $sign(\to)_1 = \ominus$ and $sign(\to)_2 = \oplus$.

$$\begin{aligned} occ(\alpha, \alpha) &= \oplus, \\ occ(\alpha, \beta) &= \bot \quad \text{if } \beta \neq \alpha, \\ occ(\alpha, T(\sigma_1, \ldots, \sigma_k)) &= \bigsqcup_{i \leq k} sign(T)_i \cdot occ(\alpha, \sigma_i). \end{aligned}$$

Moreover for each declaration $T\vec{\alpha} = C_1\vec{\sigma_1} \mid C_2\vec{\sigma_2} \mid \cdots$

$$sign(T)_i = \bigsqcup_n \bigsqcup_{j \leq \text{arity}(C_n)} occ(\alpha_i, \sigma_{nj}).$$

6.8. Example. If $\mathcal{A}$ contains

$$\begin{aligned} \mathrm{List}(\alpha) &= \mathbf{Cons}(\alpha, \mathrm{List}(\alpha)) \mid \mathbf{Nil}, \\ \mathrm{T}(\alpha, \beta, \gamma) &= \mathbf{C}(\mathrm{List}(\beta \to \alpha)) \mid \mathbf{D}(\gamma \to \mathrm{List}(\gamma)), \end{aligned}$$

then

$$\begin{aligned} sign(\mathrm{List}) &= \oplus, \\ sign(\mathrm{T}) &= (\oplus, \ominus, \top). \end{aligned}$$

We say that α *occurs essentially* in σ if $occ(\alpha, \sigma) \neq \bot$. If T is a constructor with $sign(T)_i = \bot$, then apparently α_i does not occur essentially in any of the clauses in the definition of $T\vec{\alpha}$. Observe that this happens if α_i either is absent in these clauses or occurs only in a (direct or indirect) recursion of T. Thus the i-th position of a type $T\vec{\sigma}$ is nonessential. Without loss of generality, we can assume that such nonessential positions of type constructors T do not exist: this does not impose a serious restriction on the expressive power of the system. An important consequence is the following.

6.9. LEMMA. $occ(\alpha, \sigma) = \bot \ \Rightarrow\ \alpha \notin \mathrm{TV}(\sigma)$.

The classifications *sign* and *uniprop* are related in the following way.

6.10. LEMMA. $uniprop(T)_i \ \Rightarrow\ sign(T)_i \sqsupseteq \oplus$.

Now the coercion properties of constructor applications are defined using the deduced classification of constructor arguments. For convenience we introduce the following notation.

NOTATION. Let A be a set, and R a relation on A.

(i) For each $s \in \mathbb{S}$, the s-variant of R is defined by

$$\begin{array}{rll} x \ {}^{\oplus}\!R\ y & \text{if} & x\ R\ y, \\ x \ {}^{\ominus}\!R\ y & \text{if} & y\ R\ x, \\ x \ {}^{\top}\!R\ y & \text{if} & x \ {}^{\oplus}\!R\ y \text{ and } x \ {}^{\ominus}\!R\ y, \\ (x \ {}^{\bot}\!R\ y & \text{if} & x \ {}^{\oplus}\!R\ y \text{ or } x \ {}^{\ominus}\!R\ y). \end{array}$$

(ii) This denotation naturally extends to sequences of types:

$$\vec{x} \ {}^{\vec{s}}\!R\ \vec{y} \quad \text{if} \quad x_i \ {}^{s_i}\!R\ y_i \text{ for each } i.$$

6.11. DEFINITION. The *general coercion relation* $\leq$ on $\widehat{\mathbb{T}}$ is defined inductively as follows.

$$\overset{u}{\alpha} \leq \overset{u}{\alpha},$$
$$u \leq u',\ \ \vec{\sigma} \ {}^{sign(T)}\!\!\leq \vec{\sigma'} \ \Rightarrow\ \overset{u}{T}\vec{\sigma} \leq \overset{u'}{T}\vec{\sigma'}.$$

Coercions along $\odot$ and $\otimes$ references can now be made explicit.

6.12. DEFINITION. The coercion relations $\leq^{\odot}$ and $\leq^{\otimes}$ are defined by setting

$$\begin{array}{rcl} \sigma \leq^{\odot} \tau & \Leftrightarrow & \sigma \leq \tau, \\ \sigma \leq^{\otimes} \tau & \Leftrightarrow & \sigma \leq \tau \text{ and } [\tau] = \times. \end{array}$$

In this paper we will consider uniqueness symbol types which are *uniformly attributed* variants of conventional symbol types, i.e. throughout a type, uniqueness attributes of variables are equal whenever the underlying variables are. This is no essential restriction but will simplify the description and analysis of the system.

6.13. DEFINITION. (i) Let $\sigma \in \mathbb{T}$. A *variable attribute environment* for σ is a function $\varphi : \mathrm{TV}(\sigma) \to \mathbb{U}$.

(ii) Let $\sigma \in \widehat{\mathbb{T}}, \widehat{\mathbb{T}}_{\mathbb{S}}$. Then φ_σ denotes the corresponding variable attribute environment for $|\sigma|$. This is uniquely determined if σ is uniformly attributed.

Uniform attribution enables us to view substitutions as follows.

6.14. DEFINITION. (i) A *substitution* is a function $* : \mathbb{V} \to \widehat{\mathbb{T}}$.

(ii) $*$ is a *substitution for* σ if it respects the attributes in σ, i.e. for each $\alpha \in \mathrm{TV}(|\sigma|)$

$$[*(\alpha)] = \varphi_\sigma(\alpha).$$

The result of applying $*$ to σ is denoted by σ^*.

(iii) The notion of *instance* ($\subseteq$) is modified accordingly.

The coercions are defined in such a way that the expected substitutivity results hold.

6.15. Lemma. (i) *$\leq^{\odot}$ is reflexive on $\widehat{\mathbb{T}}$.*

(ii) *$\leq^{\otimes}$ is reflexive on $\overset{\times}{\mathbb{T}}$.*

Proof. Easy induction. □

6.16. Lemma. *Let $\sigma, \tau \in \widehat{\mathbb{T}}$, and let $*$ be a substitution.*

(i) $\sigma \leq^{\odot} \tau \;\Rightarrow\; \sigma^* \leq^{\odot} \tau^*$.

(ii) $\sigma \leq^{\otimes} \tau \;\Rightarrow\; \sigma^* \leq^{\otimes} \tau^*$.

Proof. By induction on the generation of the coercion relations, using lemma 6.15 in the variable-to-variable case. □

7. Type assignment

In our new environments we allow symbols to have more than one type. This is needed, e.g., for incorporation of multiple variants of the types of algebraic constructors, and for the basic symbols.

7.1. Definition. (i) Let $\Gamma \subseteq \Sigma$ be a set of symbols. A *uniqueness type environment for* Γ is a function $\mathcal{E} : \Gamma \to \wp(\widehat{\mathbb{T}}_{\mathrm{S}})$ such that for any $S \in \Gamma$ one has the following.

(1) Each $\sigma \in \mathcal{E}(S)$ is uniformly attributed.

(2) $|\sigma| = |\sigma'|$ for all $\sigma, \sigma' \in \mathcal{E}(S)$.

(ii) A *uniqueness types environment for* $\mathfrak{T}$ is an environment for $\Sigma_{\mathfrak{T}}$.

(iii) The *basic uniqueness type environment* $\mathcal{E}_0$ is the following environment for Σ_0.

$$\begin{array}{rcl}
\mathcal{E}_0(\bot) & = & \{\overset{u}{\alpha} \mid u \in \mathbb{U}\}, \\
\mathcal{E}_0(\mathbf{Ap}) & = & \{(\overset{s}{\alpha} \overset{t}{\to} \overset{u}{\beta}, \overset{s}{\alpha}) \rightarrowtail \overset{u}{\beta} \mid s, u \in \{\times, \bullet, \vartriangle\}, t \in \{\vartriangle, \times\}\}, \\
\mathcal{E}_0(\mathbf{Root}) & = & \{\overset{u}{\alpha} \rightarrowtail \overset{u}{\alpha} \mid u \in \mathbb{U}\}.
\end{array}$$

(iv) For any uniqueness environment $\mathcal{E}$, the underlying conventional environment is denoted by $|\mathcal{E}|$.

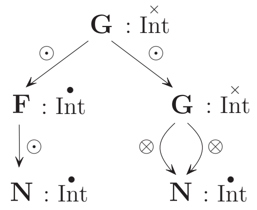
7.2. Definition. Let $g = \langle N, \mathit{symb}, \mathit{args}\rangle$ be a graph, and $\mathcal{E}$ an environment.

(i) Let $\mathcal{U} : N \to \widehat{\mathbb{T}}$ be a uniqueness type assignment, and *use* a marking function for g. Then $\mathcal{U}$ is an *$\mathcal{E}$-uniqueness typing for g according to use* if for each $n \in g$ there exist $\sigma \in \mathcal{E}(\mathit{symb}(n))$ and $\tau_1, \ldots, \tau_k \in \widehat{\mathbb{T}}$ such that

$$\begin{array}{c}
\mathcal{U}(\mathit{args}(n)_i) \leq^{\mathit{use}(n)_i} \tau_i \quad \text{for any } i \leq k = \mathrm{arity}(n), \\
(\tau_1, \ldots, \tau_k) \rightarrowtail \mathcal{U}(n) \subseteq \sigma.
\end{array}$$

(ii) g is *$\mathcal{E}$-typable with type* σ (notation $\mathcal{E} \vdash g : \sigma$) if there exists a marking function *use* and a uniqueness typing $\mathcal{U}$ for g according to *use* such that $\mathcal{U}(r_g) = \sigma$. We write $\mathcal{E} \vdash_{use} g : \sigma$ if we wish to indicate the applied marking explicitly.

7.3. EXAMPLE. The following gives (parts of) a well-typed graph and the corresponding environment.



$$\boxed{\begin{array}{lcl} \mathbf{F} & : & \dot{\alpha} \rightarrowtail \dot{\alpha}, \\ \mathbf{G} & : & (\overset{\times}{\beta}, \overset{\times}{\beta}) \rightarrowtail \overset{\times}{\beta}, \\ \mathbf{N} & : & \dot{\mathrm{Int}} \end{array}}$$

7.4. DEFINITION. Let $g = \langle N, symb, args \rangle$ be a graph, and $\mathcal{E}$ be an environment. Furthermore, let $\mathcal{U} : N \to \widehat{\mathbb{T}}$ be a uniqueness type assignment.

(i) Let $n \in g$. The *plain function type* of n according to $\mathcal{U}$ (notation $\mathcal{F}_{\mathcal{U}}(n)$) is

$$\mathcal{F}_{\mathcal{U}}(n) = (\mathcal{U}(n_1), \ldots, \mathcal{U}(n_k)) \rightarrowtail \mathcal{U}(n),$$

where $k = \mathrm{arity}(n)$, and $n_i = args(n)_i$.

(ii) $\mathcal{U}$ is an *plain $\mathcal{E}$-uniqueness typing for* g if for each $n \in g$ there exists $\sigma \in \mathcal{E}(symb(n))$ such that

$$\mathcal{F}_{\mathcal{U}}(n) \subseteq \sigma.$$

Again, we will omit $\mathcal{E}$ if it is clear from the context.

The notion of uniqueness typing for rewrite rules distinguishes functional and Curry rules. The presentation is inspired by the analysis at the end of section 3.

7.5. DEFINITION. Let $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ be a TGRS. Let R be a rewrite rule.

(i) A *marking function for* R is a marking function for $(R \,|\, r)^+$. If *use* is a marking function for R then the *root mark* of R (notation $use(R)$) is the value of *use* in the root reference of $(R \,|\, r)^+$.

(ii) Let *use* be a marking for R. Let $\sigma \in \widehat{\mathbb{T}}_{\mathrm{S}}$. A function $\mathcal{U} : g_R \to \widehat{\mathbb{T}}$ is an *$\mathcal{E}$-uniqueness typing* for R (according to *use*) if the following requirements are satisfied.

(1) $\mathcal{U}$ is a plain uniqueness typing for $R \,|\, l$,

(2) $\mathcal{U}$ is a uniqueness typing (according to *use*) for $R \,|\, r$,

(3) $\mathcal{U}(r) \leq^{use(R)} \mathcal{U}(l)$.

(iii) R is *$\mathcal{E}$-typable with function type* $\vec{\sigma} \rightarrowtail \tau$ (notation $\mathcal{E} \vdash R : \vec{\sigma} \rightarrowtail \tau$) if there exist a marking *use* and a typing $\mathcal{U}$ for R according to *use* such that $\mathcal{F}_{\mathcal{U}}(l) = \vec{\sigma} \rightarrowtail \tau$.

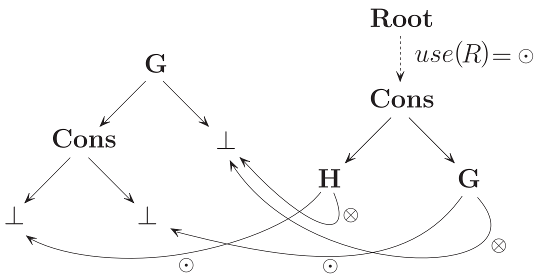
(iv) A rule R for F is *$\mathcal{E}$-uniqueness typable* if $\mathcal{E} \vdash R : \sigma$ for each $\sigma \in \mathcal{E}(F)$.

(v) $\mathcal{R}$ is *uniqueness typable* if each $R \in \mathcal{R}_{\mathcal{F}}$ is $\mathcal{E}$-uniqueness typable.

Some explanations are in place. The left-hand side is to be typed plainly, to establish the encoding of uniqueness of 'deeper' arguments into the function type (cf. section 4).

The coercion condition (3) for r is included to account for the effect of redirection in the construction of the contractum. This will become clear in the proof of the subject reduction property for this system. Moreover the marking function for R is only essential in the case of an extending rewrite rule.

7.6. EXAMPLE. Consider the following (recursive) rule for **G**.



This rule is uniqueness typable using the following environment.

$$\boxed{\begin{array}{rcl}
\mathbf{G} & : & (\overset{\bullet}{\mathrm{List}}(\overset{\times}{\alpha}), \overset{\times}{\alpha}) \rightarrowtail \overset{\bullet}{\mathrm{List}}(\overset{\times}{\alpha}), \\
\mathbf{H} & : & (\overset{\times}{\alpha}, \overset{\times}{\alpha}) \rightarrowtail \overset{\times}{\alpha}, \\
\mathbf{Cons} & : & (\overset{\times}{\alpha}, \overset{\bullet}{\mathrm{List}}(\overset{\times}{\alpha})) \rightarrowtail \overset{\bullet}{\mathrm{List}}(\overset{\times}{\alpha}).
\end{array}}$$

8. Uniqueness type environments

In this section we will give two standard constructions for type environments concerning data symbols: an environment $\mathcal{E}_{\mathcal{A}}$ for algebraic constructors introduced in $\mathcal{A}$, and an environment $\mathcal{E}_{\mathcal{F}}$ for function symbols and their Curry variants.

In order to describe coercion and uniqueness properties of environments we introduce some terminology.

8.1. DEFINITION. (i) The *schematic coercion relation* $\precsim$ is defined as $\leq$ (see definition 6.11), extended with the clause

$$u \leq v \;\Rightarrow\; \overset{u}{\alpha} \precsim \overset{v}{\alpha}.$$

(ii) Let $\sigma_1, \sigma_2 \in \widehat{\mathbb{T}}$, each uniformly attributed. Suppose $|\sigma_1| = |\sigma_2|$ $(= \sigma$, say). Let $*_1, *_2$ be substitutions for σ_1, σ_2 respectively. The *combined coercion relation* is defined by

$$(\sigma_1, *_1) \leq (\sigma_2, *_2) \;\Leftrightarrow\; \sigma_1 \precsim \sigma_2 \text{ and } \forall \alpha \in \sigma\;\; [*_1(\alpha) \overset{occ(\alpha,\sigma)}{\leq} *_2(\alpha)].$$

8.2. LEMMA. *Let $\sigma_1, \sigma_2 \in \widehat{\mathbb{T}}$, each uniformly attributed. Suppose $|\sigma_1| = |\sigma_2|$. Furthermore, let $*_1, *_2$ be substitutions for σ_1, σ_2. Then*

$$\sigma_1^{*_1} \leq \sigma_2^{*_2} \;\Leftrightarrow\; (\sigma_1, *_1) \leq (\sigma_2, *_2).$$

PROOF. By induction on the structure of $\sigma = |\sigma_1|$.

The case $\sigma = \alpha$ is easy.

Suppose $\sigma = T(\tau_1, \ldots, \tau_k)$. Say $\sigma_1 = \overset{u_1}{T}(\tau_{11}, \ldots, \tau_{1k})$, $\sigma_2 = \overset{u_2}{T}(\tau_{21}, \ldots, \tau_{2k})$, and $s_i = sign(T)_i$.

($\Rightarrow$) By assumption

$$u_1 \leq u_2 \text{ and } (\tau_{1i})^{*_1} \ {}^{s_i}\!\!\leq (\tau_{2i})^{*_2}.$$

Using the induction hypothesis one obtains

$$(\tau_{1i}, *_1) \ {}^{s_i}\!\!\leq (\tau_{2i}, *_2).$$

Hence $(\overset{u_1}{T}(\tau_{11}, \ldots, \tau_{1k}), *_1) \leq (\overset{u_2}{T}(\tau_{21}, \ldots, \tau_{2k}), *_2)$

($\Leftarrow$) Similarly. $\square$

8.3. Definition. Let $\mathcal{E}$ be a type environment, and let $S \in \Sigma$.

(i) $\mathcal{E}$ is *coercion consistent* w.r.t. S if for all $\vec{\sigma} \rightarrowtail \tau, \vec{\sigma'} \rightarrowtail \tau' \in \mathcal{E}(S)$ one has

$$\tau \precsim \tau' \ \Rightarrow\ \vec{\sigma} \precsim \vec{\sigma'}.$$

(ii) A symbol type $(\sigma_1, \ldots, \sigma_k) \rightarrowtail \tau$ is *occurrence increasing* if for all $\alpha \in \mathbb{V}$ and $i \leq k$

$$occ(\alpha, \sigma_i) \sqsubseteq occ(\alpha, \tau).$$

(iii) $\mathcal{E}$ is *occurrence increasing* w.r.t. S if $|\mathcal{E}|(S)$ is occurrence increasing.

(iv) $\mathcal{E}$ is *coercion safe* w.r.t. S if $\mathcal{E}$ is both coercion consistent and occurrence increasing w.r.t. S.

(v) $\mathcal{E}$ is *uniqueness propagating* w.r.t. S if for any $\vec{\sigma} \rightarrowtail \tau \in \mathcal{E}(S)$

$$\Sigma\vec{\sigma} \neq \times \ \Rightarrow\ [\tau] \neq \times.$$

(vi) $\mathcal{E}$ is *uniqueness safe* w.r.t. S if $\mathcal{E}$ is coercion safe and uniqueness propagating w.r.t. S.

(vii) Let $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ be a TGRS. $\mathcal{E}$ is *uniqueness safe* for $\mathfrak{T}$ if $\mathcal{E}$ is uniqueness safe with respect to all pattern symbols appearing in $\mathcal{R}_{\mathcal{F}}$.

8.4. Remark. Let $\sigma = (\sigma_1, \ldots, \sigma_k) \rightarrowtail \tau$ be uniformly attributed and occurrence increasing. Then any substitution for τ is a substitution for the σ_i by lemma 6.9.

Coercion safety for environment types implies the following coercion property for their instances.

8.5. Proposition. *Let $\mathcal{E}$ be coercion safe w.r.t. S. Let $\vec{\sigma} \rightarrowtail \tau, \vec{\sigma'} \rightarrowtail \tau' \in \mathcal{E}(S)$. Then for any pair $*, *'$ of substitutions for τ, τ' and any i*

$$\tau^* \leq (\tau')^{*'} \ \Rightarrow\ (\sigma_i)^* \leq (\sigma'_i)^{*'}.$$

Proof. Let $*, *'$ be substitutions. Then for any i

$$\begin{aligned} \tau^* \leq (\tau')^{*'} \quad &\Rightarrow \quad (\tau, *) \leq (\tau', *'), \quad \text{by lemma 8.2} \\ &\Rightarrow \quad (\sigma_i, *) \leq (\sigma'_i, *'), \quad \text{by coercion safety} \\ &\Rightarrow \quad \sigma_i^* \leq (\sigma'_i)^{*'}, \quad \text{by lemma 8.2. } \square \end{aligned}$$

Algebraic type environments

In our treatment of algebraic type systems, we will assume that the definitions are *monotonic* when viewed as inductive operators. This corresponds to the semantical intuition where algebraic definitions are viewed as inductive definitions. This boils down to the assumption that in any definition of the form

$$T\vec{\alpha} = C_1\vec{\sigma_1} \mid C_2\vec{\sigma_2} \mid \cdots$$

all occurrences the defined constructor T in the σ_i are positive. It is possible, however, to admit nonmonotonic algebraic definitions if one restricts the coercion relation for constructor types. This will not be worked out here.

We will now explain how the types of algebraic constructors are derived from an algebraic type system. One could consider the types that are obtained by consistently attributing the variables and constructor symbols in an algebraic definition. For lists this would result in nine uniqueness variants for **Cons**:

$$\boxed{\begin{array}{lcll}
\mathbf{Cons} & : & (\overset{\times}{\alpha}, \overset{\times}{\mathrm{List}}(\overset{\times}{\alpha})) \rightarrowtail \overset{\times}{\mathrm{List}}(\overset{\times}{\alpha}) & (1) \\
\mathbf{Cons} & : & (\overset{\times}{\alpha}, \overset{\bullet}{\mathrm{List}}(\overset{\times}{\alpha})) \rightarrowtail \overset{\bullet}{\mathrm{List}}(\overset{\times}{\alpha}) & (2) \\
\mathbf{Cons} & : & (\overset{\bullet}{\alpha}, \overset{\times}{\mathrm{List}}(\overset{\bullet}{\alpha})) \rightarrowtail \overset{\times}{\mathrm{List}}(\overset{\bullet}{\alpha}) & (3) \\
\mathbf{Cons} & : & (\overset{\bullet}{\alpha}, \overset{\bullet}{\mathrm{List}}(\overset{\bullet}{\alpha})) \rightarrowtail \overset{\bullet}{\mathrm{List}}(\overset{\bullet}{\alpha}) & (4) \\
\mathbf{Cons} & : & (\overset{\triangle}{\alpha}, \overset{\triangle}{\mathrm{List}}(\overset{\triangle}{\alpha})) \rightarrowtail \overset{\triangle}{\mathrm{List}}(\overset{\triangle}{\alpha}) & (5) \\
 & \vdots & &
\end{array}}$$

Since the type constructor list List is uniqueness propagating in its argument, some of the combinations (e.g. (3)) are illegal.

We will make the above idea more explicit by giving a method for consistently attributing the constructor types associated with an algebraic type definition of $T\vec{\alpha}$. The starting point will be an attribute assignment to the variables $\vec{\alpha}$, and an admissible attribute t for T. The idea is to attribute all type constructors with $\times$, unless another attribute is necessary by propagation considerations. An exception is made for direct recursion, i.e. for the occurrences of the defined constructor T. This allows e.g. the construction of spine-unique lists.

8.6. Definition. Let T be a type constructor with arity k.

(i) A $T\vec{\alpha}$ attribution is a pair consisting of a variable attribute environment φ for $\vec{\alpha}$ and an attribute t such that t is T-admissible for $\varphi(\vec{\alpha})$.

(ii) For each $T\vec{\alpha}$ attribution φ, t, and each $\sigma \in \mathbb{T}(\vec{\alpha})$, the the *$T$-type attribution* of σ (notation $[\![\sigma]\!]^T_{\varphi,t}$) is defined as follows.

$$\begin{array}{rcl}
[\![\alpha]\!]^T_{\varphi,t} & = & \overset{\varphi(\alpha)}{\alpha}, \\
[\![T(\sigma_1,\ldots,\sigma_k)]\!]^T_{\varphi,t} & = & \overset{u}{T}([\![\sigma_1]\!]^T_{\varphi,t},\ldots,[\![\sigma_k]\!]^T_{\varphi,t}), \\
 & & \begin{array}{lcll} \text{where } u & = & \triangle & \text{if } \Sigma_T[\![\vec{\sigma}]\!]^T_{\varphi,t} = \triangle \text{ or } t = \triangle;\ \text{else:} \\ & = & \bullet & \text{if } \Sigma_T[\![\vec{\sigma}]\!]^T_{\varphi,t} = \bullet \text{ or } t = \bullet, \\ & = & \times & \text{otherwise.} \end{array} \\
[\![U(\sigma_1,\ldots,\sigma_k)]\!]^T_{\varphi,t} & = & \overset{u}{U}([\![\sigma_1]\!]^T_{\varphi,t},\ldots,[\![\sigma_k]\!]^T_{\varphi,t}), \quad \text{where } u = \Sigma_U[\![\vec{\sigma}]\!]^T_{\varphi,t}.
\end{array}$$

Note that $[\![\sigma]\!]^T_{\varphi,t}$ is well defined for all appropriate σ, i.e. all attributes assigned to type constructors are admissible.

As with conventional types, a set $\mathcal{A}$ of algebraic type definitions induces a uniqueness type environment $\mathcal{E}_{\mathcal{A}}$ for all constructors. This is done in the following way.

8.7. DEFINITION. The *uniqueness type environment* $\mathcal{E}_{\mathcal{A}}$ *associated with* $\mathcal{A}$ is defined by setting for each declaration $T\vec{\alpha} = C_1\vec{\sigma_1} \mid C_2\vec{\sigma_2} \mid \cdots$

$$\mathcal{E}_{\mathcal{A}}(C_n) = \{[\![\vec{\sigma_n} \rightarrowtail T\vec{\alpha}]\!]^T_{\varphi,t} \mid (\varphi,t) \text{ is a } T\vec{\alpha} \text{ attribution}\}.$$

Observe that $\mathcal{E}_{\mathcal{A}}$ is indeed a proper uniqueness type environment (see definition 7.1).

In the remainder of this subsection we will show that $\mathcal{E}_{\mathcal{A}}$ is both coercion safe and uniqueness propagating.

8.8. PROPOSITION. $\mathcal{E}_{\mathcal{A}}$ *is occurrence increasing.*

PROOF. This follows directly from the way the functions *sign* and *occ* are determined by the algebraic type system $\mathcal{A}$. $\Box$

We now analyze the variants for the clauses in the definition of $T\vec{\alpha}$ in $\mathcal{A}$, obtained by the attribution function $[\![\ ]\!]^T$.

8.9. DEFINITION. For each $\sigma \in \mathbb{T}(\vec{\alpha})$, the relation $\leq^{\sigma,T}$ on $T\vec{\alpha}$ attributions is defined as follows.

$$(\varphi,t) \leq^{\sigma,T} (\varphi',t') \;\Leftrightarrow\; \begin{cases} \forall\alpha \in \sigma \;\; [\varphi(\alpha) \;{}^{occ(\alpha,\sigma)}\!\!\leq \varphi'(\alpha)], \\ t \;{}^{occ(T,\sigma)}\!\!\leq t'. \end{cases}$$

8.10. LEMMA. $(\varphi,t) \leq^{\sigma,T} (\varphi',t') \;\Rightarrow\; [\![\sigma]\!]^T_{\varphi,t} \precsim [\![\sigma]\!]^T_{\varphi',t'}$.

PROOF. By induction on the structure of σ.

If $\sigma = \alpha$ this is simple.

Suppose $\sigma = T(\sigma_1,\ldots,\sigma_k)$, and $(\varphi,t) \leq^{\sigma,T} (\varphi',t')$. By definition of *occ* one has $(\varphi,t) \;{}^{sign(T)_i}\!\!\leq^{\sigma_i,T} (\varphi',t')$ for each $i \leq k$. Hence by the induction hypothesis (k times)

$$[\![\sigma_i]\!]^T_{\varphi,t} \;{}^{sign(T)_i}\!\!\precsim [\![\sigma_i]\!]^T_{\varphi',t'}.$$

It remains to show that $[[\![T\vec{\sigma}]\!]^T_{\varphi,t}] \leq [[\![T\vec{\sigma}]\!]^T_{\varphi',t'}]$. By the induction hypothesis and lemma 6.10 one has $\Sigma_T[\![\vec{\sigma}]\!]^T_{\varphi,t} \leq \Sigma_T[\![\vec{\sigma}]\!]^T_{\varphi',t'}$. Combined with $t \leq t'$ this yields the result.

The case $\sigma = U(\sigma_1,\ldots,\sigma_k)$ with $U \neq T$ is treated similarly. $\Box$

8.11. PROPOSITION. *Let* $C\vec{\sigma}$ *be a clause in the definition of* $T\vec{\alpha}$. *Then for any* $T\vec{\alpha}$ *attributions* (φ,t) *and* (φ',t') *one has*

$$[\![T\vec{\alpha}]\!]^T_{\varphi,t} \precsim [\![T\vec{\alpha}]\!]^T_{\varphi',t'} \;\Rightarrow\; [\![\sigma_i]\!]^T_{\varphi,t} \precsim [\![\sigma_i]\!]^T_{\varphi',t'}.$$

PROOF. Suppose $[\![T\vec{\alpha}]\!]^T_{\varphi,t} \precsim [\![T\vec{\alpha}]\!]^T_{\varphi',t'}$. Then $(\varphi,t) \leq^{T\vec{\alpha},T} (\varphi',t')$. Let $i \leq k$. By construction of *occ*, *sign* and the fact that T occurs only positively in σ_i one has $(\varphi,t) \leq^{\sigma_i,T} (\varphi',t')$, and we are done by lemma 8.10. $\Box$

8.12. COROLLARY. (i) *$\mathcal{E}_{\mathcal{A}}$ is coercion consistent.*

(ii) *$\mathcal{E}_{\mathcal{A}}$ is coercion safe.*

It remains to show that algebraic type environments are uniqueness propagating.

8.13. LEMMA. *Let (φ, t) be a $T\vec{\alpha}$ attribution, and $\sigma \in \mathbb{T}(\vec{\alpha})$. Then for any $u \in \mathbb{U}$ with $u \neq \times$*

$$[\![\![\sigma]\!]^{T}_{\varphi,t}] = u \;\Rightarrow\; \begin{array}{l} t = u \textit{ or} \\ \varphi(\alpha) = u \textit{ for some } \alpha \textit{ with } uniocc(\alpha, \sigma). \end{array}$$

PROOF. Easy. □

8.14. PROPOSITION. *$\mathcal{E}_{\mathcal{A}}$ is uniqueness propagating.*

PROOF. Let $C \in \Sigma_{\mathcal{A}}$. Say $C\vec{\sigma}$ is the clause for C in the definition of $T\vec{\alpha}$. Let (φ, t) be a $T\vec{\alpha}$ attribution. If $\Sigma[\![\vec{\sigma}]\!]^{T}_{\varphi,t} = \times$ we are done. Suppose $\Sigma[\![\vec{\sigma}]\!]^{T}_{\varphi,t} = \vartriangle$. Say $[[\![\sigma_i]\!]^{T}_{\varphi,t}] = \vartriangle$. Then by lemma 8.13 and the observation that for any j, $uniocc(\alpha_j, \sigma_i)$ implies $uniprop(T)_j$ one obtains $[[\![T\vec{\alpha}]\!]^{T}_{\varphi,t}] = t = \vartriangle$. The $\bullet$ case is treated similarly. □

Curry environments

First we describe how the types for each Curry variant F_i of F is obtained from the environment type for F. With respect to the underlying conventional types, the Currying operation for uniqueness types is essentially the same as in the conventional case. In addition, one has to keep track of uniqueness attributes while constructing the nested function types. If one of the arguments of a Curry variant is unique (i.e. $\bullet$ or $\vartriangle$) this uniqueness is *protected* by making the result of this partial application 'necessarily unique' ($\vartriangle$). This will prevent harmful copying of these applications; see the example below.

8.15. DEFINITION. (i) Let $u \in \mathbb{U}$ and $\vec{\sigma} \in \widehat{\mathbb{T}}$. Then u is said to be *uniqueness fixating for $\vec{\sigma}$* if $u = \vartriangle$ whenever $\Sigma\vec{\sigma} \neq \times$; otherwise $u \in \{\bullet, \times\}$.

(ii) For each symbol type $\sigma = (\sigma_1, \ldots, \sigma_k) \rightarrowtail \sigma_{k+1}$ and $0 \leq j \leq k$, the set of *Curried versions of arity j* (notation $\sigma^{\mathcal{C}}_j$) is

$$\{(\sigma_1, \ldots, \sigma_j) \rightarrowtail \sigma_{j+1} \stackrel{u_j}{\rightarrow} \cdots \stackrel{u_{k-1}}{\rightarrow} \sigma_{k+1} \mid \text{each } u_i \text{ is uniqueness fixating for } (\sigma_1, \ldots, \sigma_i)\}.$$

Note that the function attributes are fixed as soon as one of the preceding σ_i is unique; in other cases there is some liberty. It will be convenient to isolate a *minimal* and a *maximal* result type of the j-th Curried version: the minimal variant is obtained by choosing $u_i = \bullet$ whenever possible; the maximal variant has $\times$ on such spots.

We consider type environments where each function symbol has exactly one type. This type determines the possible types for its Curry variants.

8.16. DEFINITION. (i) A *function type environment* (for $\mathfrak{T}$) is a map $\mathcal{F} : \Sigma^{\mathrm{f}}_{\mathcal{F}} \rightarrow \widehat{\mathbb{T}}_{\mathrm{S}}$.

(ii) Such a function type environment induces a uniqueness type environment for $\Sigma^{\mathrm{f}}_{\mathcal{F}}$ and $\Sigma^{\mathrm{c}}_{\mathcal{F}}$ (notation $\mathcal{E}^{\mathrm{f}}_{\mathcal{F}}$ and $\mathcal{E}^{\mathrm{c}}_{\mathcal{F}}$ respectively) in the following way.

$$\begin{array}{rcl} \mathcal{E}^{\mathrm{f}}_{\mathcal{F}}(F) & = & \{\mathcal{F}(F)\}, \\ \mathcal{E}^{\mathrm{c}}_{\mathcal{F}}(F_j) & = & \mathcal{F}(F)^{\mathcal{C}}_j. \end{array}$$

(iii) $\mathcal{E}_{\mathcal{F}}$ denotes the combination of these environments for $\Sigma_{\mathcal{F}}$, i.e. $\mathcal{E}_{\mathcal{F}} = \mathcal{E}^{\mathrm{f}}_{\mathcal{F}} \cup \mathcal{E}^{\mathrm{c}}_{\mathcal{F}}$.

8.17. LEMMA. *Let $\mathcal{F}$ be a function environment.*
(i) *$\mathcal{E}^{\mathrm{c}}_{\mathcal{F}}$ is coercion consistent.*
(ii) *$\mathcal{E}^{\mathrm{c}}_{\mathcal{F}}$ is uniqueness propagating.*

PROOF. Easy. □

8.18. EXAMPLE. Let $\mathcal{F}(\mathbf{F}) = (\overset{\bullet}{\sigma}, \overset{\times}{\tau}) \rightarrowtail \overset{\times}{\rho}$. Then

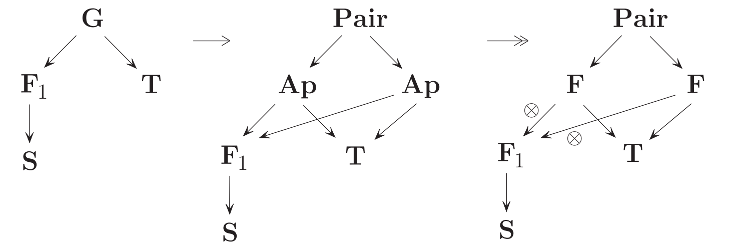
$$\begin{array}{rcl}
\mathcal{E}_{\mathcal{F}}(\mathbf{F}) & = & \{(\overset{\bullet}{\sigma}, \overset{\times}{\tau}) \rightarrowtail \overset{\times}{\rho}\}, \\
\mathcal{E}_{\mathcal{F}}(\mathbf{F}_1) & = & \{\overset{\bullet}{\sigma} \rightarrowtail \overset{\times}{\tau} \overset{\triangle}{\to} \overset{\times}{\rho}\}, \\
\mathcal{E}_{\mathcal{F}}(\mathbf{F}_0) & = & \{\ \overset{\bullet}{\sigma} \overset{\bullet}{\to} \overset{\times}{\tau} \overset{\triangle}{\to} \overset{\times}{\rho}, \\
 & & \quad \overset{\bullet}{\sigma} \overset{\times}{\to} \overset{\times}{\tau} \overset{\triangle}{\to} \overset{\times}{\rho}\ \}.
\end{array}$$

The choice of the attribute $\triangle$ for the result type of $\mathbf{F}_1$ (instead of just $\bullet$) becomes clear if one realizes that uniqueness of the σ object may be destroyed if one allows the result type to be coerced to a $\times$-type. This is illustrated in the following.

8.19. NONEXAMPLE. Consider $\mathcal{F}$ with $\mathcal{F}(\mathbf{F})$ as in the above example, and $\mathcal{F}(\mathbf{G}) = (\overset{\times}{\tau} \overset{\times}{\to} \overset{\times}{\rho}, \overset{\times}{\tau}) \rightarrowtail \overset{\bullet}{\mathrm{Prod}}(\overset{\times}{\rho}, \overset{\times}{\rho})$. Suppose one allows $\overset{\bullet}{\sigma} \rightarrowtail \overset{\times}{\tau} \overset{\bullet}{\to} \overset{\times}{\rho}$ as a type for $\mathbf{F}_1$. Then the rule

$$\mathbf{G}(f, x) \to \mathbf{Pair}(\mathbf{Ap}(f, x), \mathbf{Ap}(f, x))$$

is typable taking $\mathbf{Pair} : (\overset{\times}{\alpha}, \overset{\times}{\beta}) \rightarrowtail \overset{\times}{\mathrm{Prod}}(\overset{\times}{\alpha}, \overset{\times}{\beta})$, and using the fact that $\overset{\times}{\tau} \overset{\bullet}{\to} \overset{\times}{\rho} \leq^{\otimes} \overset{\times}{\tau} \overset{\times}{\to} \overset{\times}{\rho}$. Applying this rule to the first (well-typed) graph below (assuming $\mathbf{S} : \overset{\bullet}{\sigma}$ and $\mathbf{T} : \overset{\times}{\tau}$) leads to the second graph, and via two applicative reduction steps to the third, which is obviously not well-typed, since $\mathbf{S}$ is no longer unique.



Combining environments

In the rest of this paper we will consider environments given in the following way.

8.20. DEFINITION. Let $\mathfrak{T}$ be an applicative TGRS over $\mathcal{A}$, and let $\mathcal{F}$ be a function environment.

(i) The *combined uniqueness environment* $\mathcal{E}_{\mathcal{F},\mathcal{A}}$ is obtained by setting

$$\mathcal{E}_{\mathcal{F},\mathcal{A}} = \mathcal{E}_0 \cup \mathcal{E}_{\mathcal{F}} \cup \mathcal{E}_{\mathcal{A}}.$$

(ii) Typability in $\mathcal{F}, \mathcal{A}$ is denoted by writing $\mathcal{F}, \mathcal{A} \vdash g : \sigma$ instead of $\mathcal{E}_{\mathcal{F},\mathcal{A}} \vdash g : \sigma$.

We allow Curry variants in functional rewrite rules only in a restricted way.

8.21. DEFINITION. Let $\mathcal{F}$ be a function environment for $\mathfrak{T}$. Then $\mathfrak{T}$ is said to be *Curry safe* for $\mathcal{F}$ if for any F_j occurring as pattern symbol in $\mathcal{R}_{\mathcal{F}}$ the following holds. Say $|\mathcal{F}(F)| = (\sigma_1, \dots, \sigma_k) \rightarrowtail \tau$. For any $i \leq j$ and $\alpha \in \mathbb{V}$

$$occ(\alpha, \sigma_i) \sqsubseteq \ominus \cdot occ(\alpha, \sigma_{j+1}) \sqcup \cdots \sqcup \ominus \cdot occ(\alpha, \sigma_k) \sqcup occ(\alpha, \tau).$$

Note that this is equivalent to '$\mathcal{E}^{\mathrm{c}}_{\mathcal{F}}$ is occurrence increasing w.r.t. F_j'.

8.22. ENVIRONMENT THEOREM. *Let $\mathfrak{T}$ be a TGRS over $\mathcal{A}$, and let $\mathcal{F}$ be a function environment for $\mathfrak{T}$. Suppose $\mathfrak{T}$ is Curry safe for $\mathcal{F}$. Then $\mathcal{E}_{\mathcal{F},\mathcal{A}}$ is uniqueness safe for $\mathfrak{T}$.*

PROOF. By corollary 8.12, proposition 8.14 and lemma 8.17. $\square$

9. Typing redexes

In this section we describe the relation between typing and matching. If R is a functional rule that applies to g, then the uniqueness typing of the matching part in g is shown to be coercible to an instance of the R-typing. This has two consequences: the reduction result is typable with the corresponding instance of the right-hand side, and the uniqueness assumptions made in the left-hand side of R translate into locality properties of the matching nodes. The former is important in the proof of the subject reduction property (see section 14); the latter shows that the typing system indeed guarantees the intended notion of 'uniqueness' (see section 10).

The typing of a function symbol induces a typing notion for the corresponding Curry rules. This standard type construction will be described in the second part of this section. It will turn out that this typing also satisfies the above properties. Therefore, in the analysis performed in the rest of this paper, functional and applicative rewrite steps can be handled in a uniform way.

Fix a TGRS $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ over $\mathcal{A}$, and a function environment $\mathcal{F}$ for $\mathfrak{T}$. We assume that $\mathcal{R}$ is $\mathcal{F}, \mathcal{A}$-typable, and $\mathfrak{T}$ is Curry safe for $\mathcal{F}$.

Typing for functional redexes

We start with a general result.

9.1. LEMMA. *Let $\mu : g \xrightarrow[\mathrm{m}]{} h$ be a match. Suppose $\mathcal{U}_g$ is a plain $\mathcal{E}$-uniqueness typing for g, and $\mathcal{U}_h$ is an $\mathcal{E}$-uniqueness typing for h according to use_h. Let $n \in g$. Suppose $\mathcal{E}$ is uniqueness safe w.r.t. $symb(n)$. Set $n_i = args_g(n)_i$.*

(i) *If $\mathcal{U}_h(\mu(n)) \leq \mathcal{U}_g(n)$, then $\mathcal{U}_h(\mu(n_i)) \leq \mathcal{U}_g(n_i)$.*

(ii) *If $[\mathcal{U}_g(n_i)] \neq \times$, then $[\mathcal{U}_g(n)] \neq \times$ and $use_h(\mu(n), i) = \odot$.*

PROOF. Say $\vec{\sigma} \rightarrowtail \tau, \vec{\sigma'} \rightarrowtail \tau' \in \mathcal{E}(symb(n))$ such that, say, $\mathcal{U}_h(\mu(n)) = \tau^*$, $\mathcal{U}_g(n) = (\tau')^{*'}$ and $\mathcal{U}_h(\mu(n_i)) \leq^{use_h(\mu(n),i)} \sigma_i^*$, $\mathcal{U}_g(n_i) = (\sigma_i')^{*'}$. Then (i) follows since $\sigma_i^* \leq (\sigma_i')^{*'}$ by proposition 8.5. If $[\mathcal{U}_g(n_i)] = [(\sigma_i')^{*'}] \neq \times$ then also $[\sigma_i^*] \neq \times$ by uniqueness propagation, and hence $use_h(\mu(n), i) = \odot$. This shows (ii). $\square$

We now investigate typing according to $\mathcal{F}, \mathcal{A}$.

9.2. Lemma. *Let $g \in \mathcal{G}$, and let $\Delta = \langle R, \mu\rangle$ be a redex in g with $R \in \mathcal{R}_{\mathcal{F}}$. Let $\mathcal{U}_g$ be a uniqueness typing for g according to use_g, and let $\mathcal{U}_R$ be a uniqueness typing for R. Set $l_i = args_R(l)_i$ for each appropriate i.*

(i) *There exists a substitution $*$ such that*

$$\mathcal{U}_g(\mu(l)) = (\mathcal{U}_R(l))^*,$$
$$\mathcal{U}_g(\mu(l_i)) \leq (\mathcal{U}_R(l_i))^*.$$

(ii) *If $[\mathcal{U}_R(l_i)] \neq \times$ then $use_g(\mu(l), i) = \odot$.*

Proof. Say $\mathcal{F}(symb(n)) = \vec{\sigma} \rightarrowtail \tau$. Since $\mathcal{U}_g$ is a uniqueness typing one has

$$\mathcal{U}_g(\mu(l)) = \tau^*,$$
$$\mathcal{U}_g(\mu(l)_i) \leq^{use_g(\mu(l),i)} \sigma_i^*$$

for some $*$.

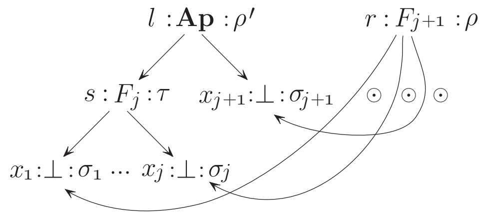
(i) Observe that $\mathcal{U}_g(\mu(l)) = (\mathcal{U}_R(l))^*$ by definition of rule typing. Moreover $\mathcal{U}_g(l_i) \leq (\mathcal{U}_R(l_i))^*$ since $\mathcal{U}_R$ is a plain uniqueness typing for $R \mid l$.

(ii) Suppose $[\mathcal{U}_R(l_i)] \neq \times$. Then also $[\sigma_i^*] \neq \times$ since $\mathcal{U}_R$ is a plain uniqueness typing. Hence $use_g(\mu(l), i) = \odot$. $\square$

Typing for Curry redexes

The environment $\mathcal{F}$ induces the following typing notion for applicative redexes. Let $F \in \Sigma_{\mathcal{F}}$ with arity $k \geq 1$; say $\mathcal{F}(F) = \sigma = (\sigma_1, \ldots, \sigma_k) \rightarrowtail \sigma_{k+1}$.

The *standard uniqueness typing* for Ap_j^F (notation $\mathcal{U}_j^F$) is the type assignment indicated by the following picture. Here F_j, F_{j+1} are respectively the j-th and $j{+}1$-th Curry variant of F. Moreover, τ is the maximal result type of the j-th Curried version of σ, and ρ the minimal result type of the $j{+}1$-th Curried version. Say $\tau = \sigma_{j+1} \xrightarrow{u} \rho'$.



9.3. Lemma. *$\mathcal{U}_j^F$ is a uniqueness typing for Ap_j^F (according to the marking indicated above).*

Proof. Obvious, since $\rho \leq \rho'$. $\square$

9.4. Lemma. *Let $\Delta = \langle \mathrm{Ap}_j^F, \mu\rangle$ be an (applicative) redex in g. Suppose $\mathcal{U}$ is a uniqueness typing for g, according to use_g. Then one has the following.*

(i) *There exists a substitution* $*$ *such that*

$$\mathcal{U}(\mu(s)) \leq (\mathcal{U}_j^F(s))^*,$$
$$\mathcal{U}(\mu(x_i)) \leq (\mathcal{U}_j^F(x_i))^* \quad \textit{for all } i \leq j+1,$$
$$(\mathcal{U}_j^F(r))^* \leq \mathcal{U}(\mu(l)).$$

(ii) *For any node* $n = s, x_1, \ldots, x_{j+1}$

$$[\mathcal{U}_j^F(n)] \neq \times \;\Rightarrow\; \mu(\overline{\overline{n}}) \textit{ is not marked.}$$

PROOF. Since $\mathcal{U}$ is a uniqueness typing we have

$$\mathcal{U}(\mu(s)) = \sigma_{j+1}^* \xrightarrow{v} (\rho'')^*,$$
$$\mathcal{U}(\mu(x_i)) \leq^{use_g(\mu(s),i)} \sigma_i^* \text{ for all } i \leq j$$

for some substitution $*$, attribute v, and type ρ''. Note that $\rho'' \leq \rho'$ and $v \leq u$, since τ is maximal. Say $(\varphi \xrightarrow{w} \mathcal{U}(\mu(l)), \varphi) \rightarrowtail \mathcal{U}(\mu(l))$ is the used type instance of **Ap**. Then

$$\mathcal{U}(\mu(x_{j+1})) \leq^{use_g(\mu(l),2)} \varphi,$$
$$\sigma_{j+1}^* \xrightarrow{v} (\rho'')^* \leq \varphi \xrightarrow{w} \mathcal{U}(\mu(l)),$$
$$\mathcal{U}(\mu(l)) \leq (\rho'')^*.$$

As to (i), the result for $\mu(s)$ follows by maximality of τ. Moreover note that $\mathcal{U}(\mu(x_i)) \leq \sigma_i^*$ for any $i \leq j$, and $\mathcal{U}(\mu(x_{j+1})) \leq \varphi \leq \sigma_{j+1}^*$. The third result follows by minimality of ρ.

As to (ii), first note that $[\mathcal{U}_j^F(s)]$ $(= [\tau]) = \vartriangle$ if it is unique. Hence $use_g(\mu(l), i) = \odot$. For the x_i we distinguish two cases. Suppose $[\mathcal{U}_j^F(x_i)]$ $(= \sigma_i) \neq \times$.

Case $i \leq j$. Since $\mathcal{U}(\mu(x_i)) \leq^{use_g(\mu(s),i)} \sigma_i^*$ one has $use_g(\mu(s), i) = \odot$. Moreover, observe that

$$[\sigma_i] \neq \times \;\Rightarrow\; [\tau] = \vartriangle.$$

Hence again $use_g(\mu(l), 1) = \odot$. Thus, $\mu(\overline{\overline{x_i}})$ is not marked.

Case $i = j+1$. Now we can use that

$$\mathcal{U}(\mu(x_{j+1})) \leq^{use_g(\mu(l),2)} \varphi \leq \sigma_{j+1}^*.$$

Hence $use(\mu(l), 2) = \odot$. $\square$

Matching and extension

9.5. MATCHING THEOREM. *Let* $g \in \mathcal{G}$, *and let* $\Delta = \langle R, \mu \rangle$ *be a redex in* g *with* $R \in \mathcal{R}$. *Let* $\mathcal{U}_g$ *be an* $\mathcal{F}, \mathcal{A}$*-uniqueness typing for* g *according to* use_g, *and let* $\mathcal{U}_R$ *be a* $\mathcal{F}, \mathcal{A}$*-typing for* R.

(i) *There exists a substitution* $*$ *such that*

$$(\mathcal{U}_R(r))^* \leq^{use(R)} \mathcal{U}_g(\mu(l)),$$
$$\mathcal{U}_g(\mu(n)) \leq (\mathcal{U}_R(n))^*, \textit{ for any } n \in R \,|\, l \textit{ with } n \neq l.$$

(ii) *For any* $n \in R \,|\, l$ *with* $n \neq l$

$$[\mathcal{U}_R(n)] \neq \times \;\Rightarrow\; \mu(\overline{\overline{n}}) \textit{ is not marked by } use_g.$$

Proof. Distinguish cases as to the form of the rewrite rule.

Case 1. R is functional. Using lemma 9.2 (i), determine $*$ such that $\mathcal{U}_g(\mu(l)) = (\mathcal{U}_R(l))^*$ and $\mathcal{U}_g(\mu(l_i)) \leq (\mathcal{U}_R(args_R(l)_i))^*$. As to (i), by substitutivity of the coercion relations (lemma 6.16)

$$(\mathcal{U}_R(r))^* \leq^{use(R)} (\mathcal{U}_R(l))^* = \mathcal{U}_g(\mu(l)).$$

The second property holds by lemma 9.1 (i) (applied repeatedly). Moreover (ii) follows by repeated application of lemma 9.1 (ii) and lemma 9.2 (ii).

Case 2. R is applicative. Then we are done by lemma 9.4. □

9.6. Corollary (Extension Typing). *Let* $g \in \mathcal{G}$, *and let* $\Delta = \langle R, \mu \rangle$ *be a redex in* g *with* $R \in \mathcal{R}$. *Let* $\mathcal{U}_g$ *be an* $\mathcal{F}, \mathcal{A}$*-uniqueness typing for* g, *according to* use_g. *Then there exists a marking use for* $R \,|\, r$ *and a uniqueness typing* $\mathcal{U}$ *for* $R \,|\, r$ *(according to use) such that*

(1) $\mathcal{U}(r) \leq^{use(R)} \mathcal{U}_g(\mu(l))$

and for any $n \in (R \,|\, l) \cap (R \,|\, r)$

(2) $\mathcal{U}_g(\mu(n)) \leq \mathcal{U}(n)$,

(3) $[\mathcal{U}(n)] \neq \times \;\Rightarrow\; \mu(\overline{\overline{n}})$ *is not marked by* use_g.

Proof. Let $\mathcal{U}_R$ be a uniqueness typing for R. Set $use = use_R$. Set $\mathcal{U} = \mathcal{U}_R^* \restriction (R \,|\, r)$, where $*$ is obtained by the Matching Theorem. Then (1), (2) and (3) hold. Moreover $\mathcal{U}$ is a uniqueness typing for $R \,|\, r$ since the coercion relations are substitutive (lemma 6.16). □

10. Locality properties of primary redexes

The marking principle was introduced to analyze access dependencies in graphs. The intention is that any actual argument in a primary redex is *local* for the root of the redex whenever its counterpart in the pattern of the rule is typed 'unique'. This is proved by translating the uniqueness information in the rule pattern into a marking property of the path connecting the redex root and the argument in question.

10.1. Definition. Let g be a graph, and $m, n \in g$

(i) n is *local for* m (in g) if either $m = n$ or

$$\forall p : r_g \rightsquigarrow n \; [m \in p].$$

(ii) n is *singly connected* to m if there exist exactly one path $p : m \rightsquigarrow n$.

(iii) n is *unique* for m is n is local for m and n is singly connected to m.

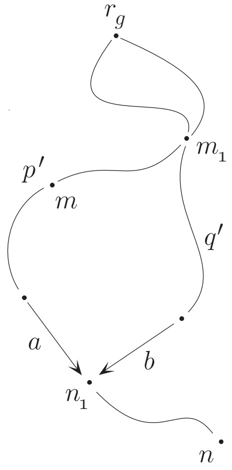
10.2. Definition. Let p be a path in g. Then p is a *function-data path* if $\mathrm{r}(p)$ is a function node, and p^- is a data path.

10.3. LEMMA. *Let g be a graph, and let use be a marking function for g. Let $p : m \rightsquigarrow n$ be a function-data path. Suppose m is a primary node. If p is not marked by use then one has the following.*

(i) *n is local for m in g.*

(ii) *If m is a primary node then n is singly connected to m.*

PROOF. (i) If $m = n$ then we are done immediately, so assume $m \neq n$. Say $p_0 : r_g \rightsquigarrow m$ be an acyclic primary path. Let $q : r_g \rightsquigarrow n$. We have to show that $m \in q$. We can assume that q is acyclic. Suppose, towards a contradiction, that $m \notin q$. Then there exists a node n_1 on $\widetilde{p}$ and $a, b \in acc(n_1)$ such that $a \neq b$ and $a \in p$, $b \in q$. Now there are subpaths p' of $p_0 * p$ and q' of q such that $(p', a) \wedge (q', b)$ is a critical path combination, say with joining node m_1.



We claim that $p' * a \lesssim q' * b$. Then $p' * a$ is marked, contradicting the assumption that p is not marked.

Proof of the claim. Consider the position of m_1 on $p_0 * p$.

Case 1. $m_1 \in p$. Note that $m_1 \neq m$ by assumption. Then m_1 is a data node, so $p' * a \sim q' * b$.

Case 2. $m_1 \in p_0$. Then $p' * a$ starts with a primary reference, so $p' * a \lesssim q' * b$.

(ii) Let $q : m \rightsquigarrow n$. Suppose, towards a contradiction, that $q \neq p$. Say a the first reference on q such that $a \notin p$. Note that $a \lesssim b$. Then a and b are the top references of some critical path combination causing a mark on p. Contradiction. □

10.4. UNIQUENESS THEOREM. *Let $\mathfrak{T}$ be Curry safe for $\mathcal{F}$. Let g be a $\mathcal{F}, \mathcal{A}$-typable graph. Let $\Delta = \langle R, \mu \rangle$ be a primary redex in g with $\mathcal{F}, \mathcal{A}$-uniqueness typing $\mathcal{U}$. Let $n \in R \mid l$, $n \neq l$ be a primary node. Then*

$$[\mathcal{U}(n)] \neq \times \;\Rightarrow\; \mu(n) \textit{ is unique for } \mu(l) \textit{ in } g.$$

PROOF. Say use_g is an appropriate marking function. By the Matching Theorem 9.5, the function-data path $\mu(\overline{\overline{n}})$ is not marked by use_g. Now the result follows by lemma 10.3. □

Now the compile time garbage collection (mentioned in the Introduction) can be made explicit. If a (primary) left-hand side node is typed 'unique' and it is not re-used in the right-hand side, then the corresponding node in the object graph will certainly become garbage.

11. Saturated markings

In the sections 12, 13 and 14, it will be shown that uniqueness typability is preserved during reduction. This splits into two parts. Firstly, if $g \xrightarrow{\Delta} h$ one has to prove that the markings use_g and use_R (of g and the applied rewrite rule respectively) can be combined into a proper marking of h. Furthermore it needs to be verified that this marking admits a suitable uniqueness typing.

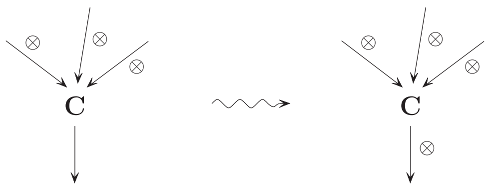
It will be convenient to transform a given marking function for g into a *saturated* one, i.e. a marking that exhibits a 'maximal' labeling. This transformation is based on the following observation. If all paths (from r_g) to a certain data node are marked, then each direct argument of this node cannot be used destructively by any function accessing this argument via the data node in question. In other words: access via this data node to such a direct argument will necessarily be considered 'read-only'. The corresponding references can therefore safely be marked with $\otimes$.

11.1. Definition. A marking *use* for g is *saturated* if for any data node $n \in g$ the following holds. If every path $p : r_g \rightsquigarrow n$ is marked, then for all $i \leq \text{arity}(symb_g(n))$

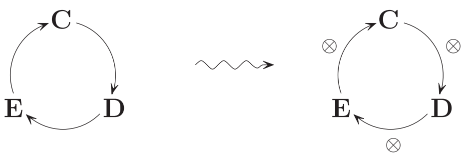
$$use((n,i)) = \otimes.$$

In this section we will present a method for transforming a marking function into a saturated one, in such a way that typability is maintained. This will be done via two operations.

The first operation is $\otimes$-*extension* for data nodes to which all references are labeled $\otimes$.



Secondly, constructor cycles can be marked completely.



We first prove some type theoretical results. For certain uniqueness types σ one can construct a 'nonunique variant' $\langle\sigma\rangle$ such that $\sigma \leq \langle\sigma\rangle$.

11.2. Definition. (i) The *uniqueness removal map* $\langle\,\rangle : \widehat{\mathbb{T}} \to \overset{\times}{\mathbb{T}} \cup \{\uparrow\}$ (where $\uparrow$ represents 'undefined') is specified inductively as follows. (Here $\to$ is again regarded as an ordinary constructor.)

$$\begin{array}{rcll} \langle\sigma\rangle & = & \sigma & \text{if } [\sigma] = \times, \\ \langle\sigma\rangle & = & \uparrow & \text{if } [\sigma] = \triangle, \end{array}$$

$$
\begin{array}{rcl}
\langle \overset{\bullet}{\alpha} \rangle & = & \uparrow, \\
\langle \overset{\bullet}{T}(\sigma_1, \ldots, \sigma_k) \rangle & = & \overset{\times}{T}(\sigma_1', \ldots, \sigma_k'), \\
& & \text{where } \; \sigma_i' \;=\; \langle \sigma_i \rangle \quad \text{if } \mathit{uniprop}(T)_i, \\
& & \qquad\quad\;\; = \; \sigma_i \qquad \text{otherwise.}
\end{array}
$$

It is understood that $\langle \overset{\bullet}{T}(\sigma_1, \ldots, \sigma_k) \rangle$ is $\uparrow$ whenever one of the σ_i' equals $\uparrow$.

(ii) We will write $\langle \sigma \rangle\!\downarrow$ to indicate that $\langle \sigma \rangle \neq \uparrow$.

11.3. Lemma. *$\langle\ \rangle$ is well defined, i.e. for all $\sigma \in \widehat{\mathbb{T}}$ one has $\langle \sigma \rangle \in \widehat{\mathbb{T}}$ whenever $\langle \sigma \rangle\!\downarrow$.*

Proof. One easily checks that $\langle \sigma \rangle$ satisfies the admissibility requirements for type attributes. $\square$

11.4. Lemma. (i) *σ is $\leq^{\otimes}$-coercible $\Rightarrow$ $\langle \sigma \rangle\!\downarrow$.*

(ii) *$\sigma \leq^{\otimes} \tau \;\Rightarrow\; \sigma \leq \langle \sigma \rangle \leq^{\otimes} \tau$.*

Proof. Since the arguments are similar we will prove the statements (i) and (ii) simultaneously by induction on the structure of σ. First observe that $[\sigma] \neq \vartriangle$. If $[\sigma] = \times$ then we are done. Furthermore, $\sigma = \overset{\bullet}{\alpha}$ is impossible. Now suppose $\sigma = \overset{\bullet}{T}(\sigma_1, \ldots, \sigma_k)$. Then $\tau = \overset{\times}{T}(\tau_1, \ldots, \tau_k)$ for some $\tau_1, \ldots, \tau_k$ with $\vec{\sigma} \;{}^{sign(T)}\!\!\leq \vec{\tau}$. As in definition 11.2, write $\langle \sigma \rangle = \overset{\times}{T}\vec{\sigma'}$. We claim that $\sigma_i'\!\downarrow$ and $\sigma_i \leq \sigma_i' \leq \tau_i$ for all $i \leq k$. Note that this implies the desired result. Indeed, if not $\mathit{uniprop}(T)_i$ then $\sigma_i' = \sigma_i$ so we are done. Otherwise $[\tau_i] = \times$ and $\sigma_i' = \langle \sigma_i \rangle$. Moreover, $\mathit{uniprop}(T)_i$ implies $\mathit{sign}(T)_i \sqsupseteq \oplus$. In case $\mathit{sign}(T)_i = \top$ one has $\sigma_i \;{}^{\top}\!\!\leq \tau_i$. Hence $[\sigma_i] = \times$ so again $\sigma_i' = \sigma_i$. The remaining case is $\mathit{sign}(T)_i = \oplus$. Since $[\tau_i] = \times$ one has $\sigma_i \leq^{\otimes} \tau_i$. Now the induction hypothesis applies, so both $\sigma_i'\!\downarrow$ and $\sigma_i \leq \sigma_i' \leq \tau_i$ for all $i \leq k$. $\square$

11.5. Proposition. *Let C be a data symbol. Let $\vec{\sigma} \rightarrowtail \tau \in \mathcal{E}_{\mathcal{F},\mathcal{A}}(C)$. Let $*$ be a substitution for this type. Suppose τ^* is $\leq^{\otimes}$-coercible. Then there exists $\vec{\sigma'} \rightarrowtail \tau' \in \mathcal{E}_{\mathcal{F},\mathcal{A}}(C)$ and an appropriate substitution $*'$ such that*

$$\tau^* \leq (\tau')^{*'}$$

and for each i

$$\sigma_i^* \leq^{\otimes} (\sigma_i')^{*'}$$

and moreover for each ρ with $\tau^* \leq^{\otimes} \rho$

$$(\tau')^{*'} \leq^{\otimes} \rho.$$

Proof. Note that $\langle \tau^* \rangle\!\downarrow$ by lemma 11.4 (i).

Case 1. C is an algebraic constructor. Then $\langle \tau^* \rangle = (\tau')^{*'}$ with $\vec{\sigma'} \rightarrowtail \tau' \in \mathcal{E}_{\mathcal{A}}(C)$ for some $\vec{\sigma'}$, since $\mathcal{E}_{\mathcal{A}}(C)$ contains all $[\![\ ]\!]^T_{\varphi,t}$-variants of constructor types for admissible φ, t. Observe that $\tau^* \leq (\tau')^{*'}$ by lemma 11.4 (ii). Moreover $\sigma_i^* \leq (\sigma_i')^{*'}$ by proposition 8.5. Note that $[(\tau')^{*'}] = \times$. Hence by uniqueness propagation of $\mathcal{E}_{\mathcal{A}}$ (proposition 8.14) one has $[(\sigma_i')^{*'}] = \times$ for each i, so $\sigma_i^* \leq^{\otimes} (\sigma_i')^{*'}$. The third property follows from lemma 11.4 (iii).

Case 2. C is a Curry variant. Then τ is of the form $\tau_1 \xrightarrow{u} \tau_2$. Note that $\langle \tau \rangle = \tau_1 \xrightarrow{\times} \tau_2$ since neither of the $\rightarrow$ positions is uniqueness propagating. Moreover $\langle \tau \rangle$ is a valid result type for the C-variant. Hence one can take $\vec{\sigma'} = \vec{\sigma}$, $\tau' = \langle \tau \rangle$, and $*' = *$. $\square$

NOTATION. (i) If f is a function, then $f[x \mapsto p]$ denotes the function f' such that

$$\begin{array}{rcll} f'(x) & = & p, & \\ f'(y) & = & f(y) & \text{if } y \neq x. \end{array}$$

Multiple assignments of the form $f[x \mapsto p, y \mapsto q]$ are also used.

(ii) The ordering $\leq$ on uniqueness typings for a graph g is interpreted pointwise, by

$$\mathcal{U} \leq \mathcal{U}' \;\Leftrightarrow\; \forall n \in g \;\; [\mathcal{U}(n) \leq \mathcal{U}'(n)].$$

The following gives a justification for the $\otimes$-extension operation.

11.6. PROPOSITION. *Let $\mathcal{U}$ be a $\mathcal{F},\mathcal{A}$-uniqueness typing for g according to use. Let $n \in g$, $n \neq r_g$ be a data node (say with arity k) such that $use(a) = \otimes$ for all $a \in acc(n)$. Set $use' = use[(n,1) \mapsto \otimes, \ldots, (n,k) \mapsto \otimes]$. Then there exists a $\mathcal{F},\mathcal{A}$-uniqueness typing $\mathcal{U}'$ according to use' such that $\mathcal{U} \leq \mathcal{U}'$ and $\mathcal{U}'(r_g) = \mathcal{U}(r_g)$.*

PROOF. Set $n_i = args(n)_i$. Say $\vec{\sigma} \rightarrowtail \tau \in \mathcal{E}_{\mathcal{F},\mathcal{A}}(symb(n))$, such that

$$\begin{array}{rl} \mathcal{U}(n) = \tau^*, & \\ \mathcal{U}(n_i) \leq^{use(n,i)} \sigma_i^* & \quad \text{for all } i. \end{array}$$

By proposition 11.5 there exists $\vec{\sigma'} \rightarrowtail \tau' \in \mathcal{E}_{\mathcal{F},\mathcal{A}}(symb(n))$ and a substitution $*'$ such that

$$\begin{array}{rl} \tau^* \leq (\tau')^{*'}, & \\ \sigma_i^* \leq^{\otimes} (\sigma_i')^{*'} & \quad \text{for all } i. \end{array}$$

Now take $\mathcal{U}' = \mathcal{U}[n \mapsto (\tau')^{*'}]$. Then

$$\mathcal{U}'(n_i) \leq^{use(n,i)} \sigma_i^* \leq^{\otimes} (\sigma_i')^{*'}$$

so by transitivity of $\leq$

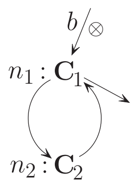
$$\mathcal{U}'(n_i) \leq^{\otimes} (\sigma_i')^{*'}$$

and we are done. $\square$

The following justifies the cycle-markings described above.

11.7. PROPOSITION. *Let $\mathcal{U}$ be a $\mathcal{F},\mathcal{A}$-uniqueness typing for g, according to use. Let $p : n \rightsquigarrow n$ be a data cycle; write $p = (a_1, \ldots, a_n)$. Set $use' = use[a_1 \mapsto \otimes, \ldots, a_n \mapsto \otimes]$. Then there exists a $\mathcal{F},\mathcal{A}$-uniqueness typing $\mathcal{U}'$ according to use' such that $\mathcal{U} \leq \mathcal{U}'$ and $\mathcal{U}'(r_g) = \mathcal{U}(r_g)$.*

PROOF. To avoid tedious denotational matters, we treat an example with two data nodes. From this the general idea will become clear. Consider the following situation.



There exists an external reference b to this cycle, say to n_1. Since $use(b) = \otimes$ the type $\mathcal{U}(n_1)$ is $\leq^{\otimes}$-coercible. Say in $\mathcal{E}_{\mathcal{F},\mathcal{A}}$ one has

$$\begin{aligned} \mathbf{C}_1 &: (\sigma, \tau) \rightarrowtail \rho, \\ \mathbf{C}_2 &: \chi \rightarrowtail \psi, \end{aligned}$$

such that $\mathcal{U}(n_1) = \rho^{*_1}$, $\mathcal{U}(n_2) = \psi^{*_2}$ and $\psi^{*_2} \leq \sigma^{*_1}$, $\rho^{*_1} \leq \chi^{*_2}$ respectively. By proposition 11.5 there exists an environment type $(\sigma', \tau') \rightarrowtail \rho'$ for $\mathbf{C}_1$ and a substitution $*'_1$ such that

$$\begin{aligned} \rho^{*_1} &\leq (\rho')^{*'_1}, \\ \sigma^{*_1} &\leq^{\otimes} (\sigma')^{*'_1}. \end{aligned}$$

Hence ψ^{*_2} $(\leq \sigma^{*_1})$ is $\leq^{\otimes}$-coercible so again by proposition 11.5 is an environment type $\chi' \rightarrowtail \psi'$ for $\mathbf{C}_2$ and a substitution $*'_2$ such that

$$\begin{aligned} \psi^{*_2} &\leq (\psi')^{*'_2}, \\ \chi^{*_2} &\leq^{\otimes} (\chi')^{*'_2}. \end{aligned}$$

Define $\mathcal{U}' = \mathcal{U}[n_1 \mapsto (\rho')^{*'_1}, n_2 \mapsto (\chi')^{*'_2}]$. It remains to show that $\mathcal{U}'$ is a uniqueness typing according to use'. Since the references to n_1, n_2 not occurring on p are $\otimes$-labeled, coercions along these references remain valid by proposition 11.5. In order to show that $\mathcal{U}'(n_2) \leq^{\otimes} (\sigma')^{*'_1}$, it is sufficient to prove that $\psi^{*_2} \leq^{\otimes} (\sigma')^{*'_1}$. Indeed,

$$\begin{aligned} \psi^{*_2} &\leq \sigma^{*_1} \\ &\leq^{\otimes} (\sigma')^{*'_1} \end{aligned}$$

and we are done. Similarly one shows $\mathcal{U}'(n_1) \leq^{\otimes} (\chi')^{*'_2}$. □

This provides a method for constructing saturated marking functions.

11.8. Definition. Let *use* be a marking function for g. Then use^{+} is the marking function obtained by adjusting *use* in the following ways.

(1) Each data cycle is completely marked.
(2) $\otimes$-extension is performed repeatedly while possible.

11.9. Theorem. *Let use be a marking function for g. Then use^{+} is saturated.*

Proof. Suppose n is a data node in g such that every path $p : r_g \rightsquigarrow n$ is marked. We will show that $use^{+}(a) = \otimes$ for any $a \in acc(n)$. Then we are done since use^{+} is closed under $\otimes$-extension.

It will be shown that there exists no nonempty unmarked acyclic path to n. Note that this implies the above statement. Suppose, towards a contradiction, q is such a path of maximal length. Write $q = (m, i) * q'$ with q' a data path. Note that m is not a function node (otherwise there is an unmarked path from r to n).

Claim. $use^{+}(a) = \otimes$ for all $a \in acc(m)$. This contradicts the assumption that q is not marked.

Proof of the claim. Let $a = (m', j) \in acc(m)$. If m' is a function node then $use^{+}(a) = \otimes$ since every path from r to n is marked. Suppose m' is a data node. If $m' \in q$ then $use^{+}(a) = \otimes$ since each data cycle is completely marked by use^{+}. If, on the other hand, $a * q$ is acyclic then $use^{+}(a) = \otimes$ by maximality of q. This proves the claim. □

11.10. SATURATION THEOREM. Let g be a graph. Suppose $\mathcal{F}, \mathcal{A} \vdash_{use} g : \sigma$. Then there exists a marking function use' such that

(1) use' is saturated;

(2) $\mathcal{F}, \mathcal{A} \vdash_{use'} g : \sigma$.

PROOF. By theorem 11.9 and the propositions 11.6 and 11.7. $\square$

12. Marking reducts

In this section we will give a method for constructing a marking function for a reduction result, based on a (saturated) marking for the object graph and a marking for the right-hand side of the applied rewrite rule. We distinguish to cases as to the form of the rewrite rule. Note that application rules are extending.

Markings for extending reductions

In the sequel, let $g \in \mathcal{G}$, and let $\Delta = \langle R, \mu \rangle$ be an extending $\mathcal{R}$-redex in g. Say $g \xrightarrow{\Delta} h$.

Before making this precise we introduce some terminology.

12.1. DEFINITION. (i) A reference $a \in R \,|\, r$ is called a *border reference* if $a \notin R \,|\, l$ and $\mathrm{d}(a) \in R \,|\, l$. The collection of border references in R is indicated by B_R.

(ii) Let $a \in B_R$. Then $\overline{\overline{a}}$ denotes the path $\overline{\overline{\mathrm{d}(a)}}$ in the tree $R \,|\, l$. Similarly we use $\overline{a}$.

12.2. DEFINITION. Let a be a reference in h.

(i) a is called *new* if a is a reference in $R \,|\, r$ but not in $R \,|\, l$. Note that the starting node of a is a new node in h. Other references are called *old*. This terminology carries over to paths: a path is *new* if it contains a new reference; otherwise it is *old*.

(ii) a is *redirected* if $\mathrm{d}_g(a) = \mu(l)$ (and consequently $\mathrm{d}_h(a) = r_R$).

First we describe how a marking for h can be obtained from the respective markings for g and R. Since the part of g matching the left-hand side of R may contain more sharing than $R \,|\, l$, one could expect that the marking of $R \,|\, r$ might be too liberal: it may contain too few $\otimes$-labeled references in order to result in a proper marking for the reduct. However, arbitrary changing $\odot$-labels into $\otimes$ reduces coercion possibilities. It will turn out to be safe to add a $\otimes$ mark to any border reference a for which $\mu(\overline{\overline{a}})$ is marked.

For convenience, we introduce some auxiliary operations concerning marking functions.

12.3. DEFINITION. (i) Let p be a path in a graph g, and use a marking for g. Then

$$\begin{aligned} use(p) &= \otimes && \text{if } p \text{ is marked,} \\ &= \odot && \text{otherwise.} \end{aligned}$$

(ii) The operation $+$ on $\mathbb{M}$ is defined by

$$\begin{aligned} u + v &= \otimes && \text{if } u = \otimes \text{ or } v = \otimes, \\ &= \odot && \text{otherwise.} \end{aligned}$$

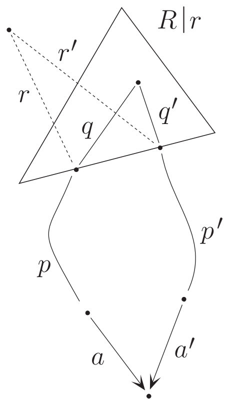
The following describes the construction of a marking for h based on markings for g and R.

12.4. Definition. Let *use* be a marking for g, and use_R the rule marking of R. The *combined use function* use^{Δ} on Ref_h is defined as follows.

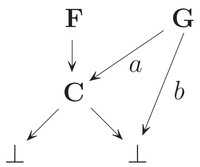
$$
\begin{array}{rcll}
use^{\Delta}(a) & = & use(a) & \text{if } a \text{ is old and } a \text{ is not redirected,} \\
 & = & use(a) + use_R(R) & \text{if } a \text{ is redirected,} \\
 & = & use_R(a) & \text{if } a \text{ is new and } a \text{ is not a border reference,} \\
 & = & use_R(a) + use(\mu(\overline{\overline{a}})) & \text{if } a \text{ is a border reference.}
\end{array}
$$

Note that two corrections are made on *use* and use_R: a border reference a is marked if either it is marked in R or $\mu(\overline{\overline{a}})$ is marked in g. Moreover redirected references are corrected according to the root attribute of $R \mid r$.

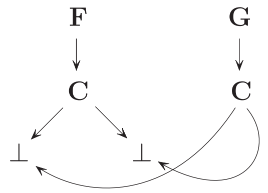
As said before, a marking function for $R \mid r$ might be an insufficient marking for the extension part (containing the new nodes) of h, e.g. in the case of sharing (in h) 'below' $R \mid r$; see the following picture.



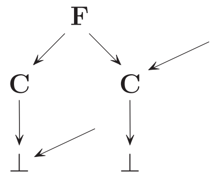
Suppose this introduces a critical path combination in which one of the paths needs to be marked (e.g. $q \precsim q'$), and neither *use* nor use_R gives a $\otimes$-label on this path. Now consider the paths $r * p * a$ and $r' * p' * a'$ both starting in $\mu(R \mid l)$. If r and r' do not coincide then consistency of R w.r.t. the argument classification implies that r is marked in g. Consequently, this marking is copied onto $R \mid r$ by the correction of border references. However, if r and r' coincide this does not work. This occurs if border references appear in a configuration of the following form.



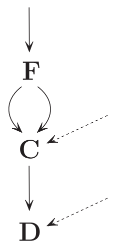
Here a and b are said to be *stacked*: $\overline{\overline{a}}$ is an initial part of $\overline{\overline{b}}$. We do not allow such stacking of border references in rewrite rules. Note that this does not really reduce the expressive power of graph rewriting: one could replace the above rule by the following.



Unfortunately, there are pathological cases in which this restriction is insufficient. This will appear in a special case in the correctness proof. The problem is caused by pairs of border references which point into two distinct but compatible subpatterns of R (let us call them *target patterns*), in such a way that they are 'stacked modulo compatibility', as shown in the following figure.



(Applying this rule to a graph of the form



results in the creation of new references that are stacked.) Call the index of the reference from l to a subpattern the *index* of that subpattern. Say P is one of the target patterns in question. If this stack suspect situation appears, then it is necessary that the index of *any* P-compatible subpattern to be $\underset{\sim}{\lhd}$-related with the index of P. In order to make this precise, some terminology is introduced.

12.5. Definition. Let g be a graph.

(i) Let $p = ((n_1, i_1), \ldots, (n_\ell, i_\ell))$ be a path in g. The *route* of p is the sequence $(i_1, i_2, \ldots, i_\ell)$.

(ii) If g is a tree, then *address* of n (in g) is the route of the path from r_g to n.

12.6. Definition. Let R be a rewrite rule of arity k.

(i) Let $a \in B_R$. We use the denotation $\widehat{a}$ to indicate the address of the node $\mathrm{d}(a)$ (in $R \,|\, l$). Note that $(\widehat{a})_1$ is the index of the subpattern p containing $\mathrm{d}(a)$ whereas $(\widehat{a})^-$ is the address of $\mathrm{d}(a)$ in p.

(ii) Let $i, j \le k$. Then i and j are *pattern overlapping* (notation $i \uparrow j$) if

$$(R \,|\, args(l)_i) \uparrow (R \,|\, args(l)_j).$$

(iii) Let $a, b \in B_R$. Then a is *potentially stacked on* b if

$$(\widehat{a})_1 \uparrow (\widehat{b})_1 \text{ and } (\widehat{a})^- \subset (\widehat{b})^-.$$

If a is potentially stacked on b then both a and b are called *stack suspect*.

(iv) R is *stack safe* if for each stack suspect $a \in B_R$ one has the following. Set $j = (\widehat{a})_1$.

$$\forall i \leq k\ [i \uparrow j \ \Rightarrow\ i \precsim j].$$

In particular rewrite rules for simple function symbols are stack safe.

12.7. Lemma. *Let $\Delta = \langle R, \mu \rangle$ be a redex in g. Suppose R is stack safe. Moreover let a be stack suspect; say $(\widehat{a})_1 = j$. If $\mu(args(l)_i) = \mu(args(l)_j)$, then $i \precsim j$.*

Proof. Set $a_i = args(l)_i$ and $a_j = args(l)_j$. Observe that $\mu : R \mid a_i \underset{\mathrm{rm}}{\rightrightarrows} g \mid \mu(a_i)$ and $\mu : R \mid a_j \underset{\mathrm{rm}}{\rightrightarrows} g \mid \mu(a_j)$. So $i \uparrow j$ and hence by stack safety one has $i \precsim j$. $\square$

In the sequel, we assume that any $R \in \mathcal{R}$ is stack safe.

12.8. Theorem. *Let use_g be a saturated marking for g, and use_R a marking for R. Then the labeling use_g^{Δ} is a marking for h.*

Proof. See section 13. $\square$

Markings for projections

For this subsection, let $g \in \mathcal{G}$, and let $\Delta = \langle R, \mu \rangle$ be a projecting $\mathcal{R}$-redex in g. Say $g \xrightarrow{\Delta} h$.

12.9. Definition. Let a be a reference in h. Then a is called *new* if $\mathrm{d}_g(a) = \mu(l)$, and consequently $\mathrm{d}_h(a) = \mu(r)$. Otherwise a is called *old*.

Now we are ready to describe the construction of a marking function for h, based on a marking for g.

12.10. Definition. Let *use* be a marking for g. The *redirection marking* use^{Δ} on Ref_h is defined by

$$\begin{array}{rcll} use^{\Delta}(a) & = & use(a) & \text{if } a \text{ is old,} \\ & = & use(a) + use_g(\mu(\overline{\overline{r}})) & \text{if } a \text{ is new.} \end{array}$$

In this case only one correction on *use* is made, depending on the presence of a mark on $\mu(\overline{\overline{r}})$.

12.11. Theorem. *Let use_g be a saturated marking for g. Then the labeling use_g^{Δ} is a marking for h.*

Proof. See section 13. $\square$

13. Correctness of reduct markings

We suggest that the reader skips the (very technical) correctness proofs at first reading. The complexity is mainly caused by the presence of cycles.

In the sequel, let g be a graph, and let *use* be a marking function for g. We will first prove some technical results concerning marking functions. The main part involves an analysis of the relative positions of path pairs and a treatment of cycles. As a warming-up, we start with some trivialities.

13.1. Remark. Let $p = p_1 * p_2$ be a path.

(i) If p_2 is marked then p is marked.

(ii) Let p_1' be a marked path which is extendible with p_2. If p is marked then $p_1' * p_2$ is marked.

(iii) If p_1 is marked and p_2 is a data path then p is marked.

Some terminology concerning paths is necessary.

13.2. Definition. Let p, q be paths.

(i) p and q *form a diamond* (notation $p \Diamond q$) if p and q both diverge and converge, i.e. either $p = ()$ or $q = ()$ or

$$\mathrm{r}(p) = \mathrm{r}(q) \text{ and } \mathrm{d}(p) = \mathrm{d}(q) \text{ and } (\widetilde{p})^{-}_{-} \# (\widetilde{q})^{-}_{-}.$$

We use $p \dot{\Diamond} q$ to indicate that the diamond is nontrivial, i.e. $p \Diamond q$ and at least one of the paths p, q is nonempty.

(ii) p and q are *parallel* (notation $p \parallel q$) if $\widetilde{p} = \widetilde{q}$.

13.3. Diamond Factorization. Let $p, q : n \rightsquigarrow m$ be paths in g such that $p \neq q$. Then there exist paths p_0, q_0, p_1, q_1, r with $p_1 \dot{\Diamond} q_1$ such that

$$\begin{aligned} p &= p_0 * p_1 * r, \\ q &= q_0 * q_1 * r. \end{aligned}$$

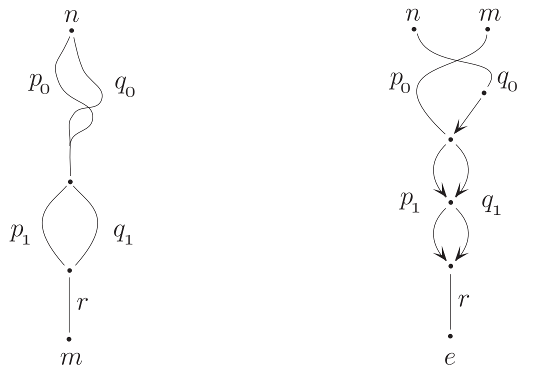
We say that the above paths p_0, q_0, p_1, q_1, r are a *diamond factorization* of p, q.

13.4. Tail Factorization. Let $p : n \rightsquigarrow e$ and $q : m \rightsquigarrow e$. Then there exist paths p_0, q_0, p_1, q_1, r, with $p_1 \parallel q_1$ and

$$\begin{aligned} p &= p_0 * p_1 * r, \\ q &= q_0 * q_1 * r. \end{aligned}$$

We say that the above paths p_0, q_0, p_1, q_1, r are the (unique) *tail factorization* of p, q if r and $p_1 * r$ (and hence also $q_1 * r$) are of maximal length. Observe that $q_0 * q_1$ is nonempty if q is not a tail part of p. Furthermore, note that q_0 is nonempty if $m \notin p$ and $m \neq e$. We call the last reference of q_0 the *entrance reference* of q to p.

The above factorizations can be visualized as follows.



13.5. Proposition. *Let $p, q : n \rightsquigarrow m$, $p \neq q$ be paths such that p and q intersect (apart from n, m) only in data nodes. Furthermore, suppose p, q are not root cyclic. Then*

$$p \precsim q \quad \Rightarrow \quad p \textit{ is marked.}$$

Proof. Let p_0, q_0, p_1, q_1, r be a diamond factorization of p, q. Then $p_1 \diamond q_1$. Say n_0 be the first node of p_1 (and of q_1).

Case 1. p_1 and q_1 are not empty. Observe that r is a data path.

Case 1a. $n_0 = n$. Then p_0, q_0 are empty (otherwise p or q would have been root cyclic). So by assumption $p_1 \precsim q_1$, and hence p_1 is marked. By remark 13.1 (ii) also $p_1 * r$ is marked.

Case 1b. $n_0 \neq n$. Since n_0 is a data node, it is simple, and again $p_1 \precsim q_1$. Consequently, p_1 and therefore $p_0 * p_1 * r$ are marked.

Case 2. p_1 is empty, and q_1 is nonempty. Note that $q_1 : n_0 \rightsquigarrow n_0$. Since q is not root cyclic it follows that $n \neq n_0$. So p_0 and q_0 are not empty. Say b is the last reference of p_0, i.e $p_0 = p_0' * b$. Then one has $b \sim b * q_1$ which implies that b is marked. Therefore also $p_0' * b * r = p$ is marked.

Case 3. q_1 is empty, and p_1 not. Similar to case 2. □

13.6. Lemma. *Let $p * a : n \rightsquigarrow n$ be a cycle. Furthermore, let $b \in acc(n)$. Suppose prootb $\notin p$. Then one has the following.*

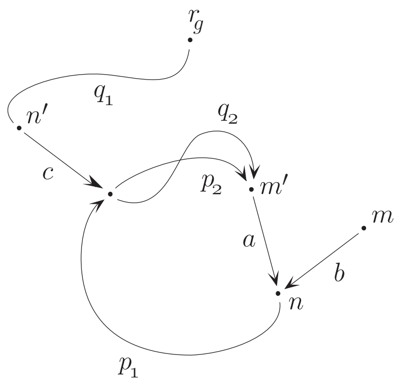
(i) *b and $b * p * a$ are marked.*

(ii) *If use is saturated then a is marked.*

Proof. Say $m = \mathrm{r}(b)$.

(i) Set $q = b * p * a$. Observe that $b, q : m \rightsquigarrow n$ and that $b \sim q$. Furthermore, q is not root cyclic. Then b and q are marked by proposition 13.5.

(ii) Say $m' = \mathrm{r}(a)$. If m' is a function node we are done, since $b * p * a$ is marked. Assume m' is a data node. Moreover suppose, towards a contradiction, that $use(a) \neq \otimes$. We will derive that every path leading to m' is marked. This contradicts the assumption, by saturation. Indeed, let $q : r_g \rightsquigarrow m'$. Say $c = (n', j)$ is the entrance reference of q to p. Write $q = q_1 * c * q_2$.



By (i) one has $use(c) = \otimes$. Write $p = p_1 * p_2$ with $p_1 : n \rightsquigarrow \mathrm{d}(c)$ and $p_2 : \mathrm{d}(c) \rightsquigarrow m'$. Now consider that path $p_1 * q_2 * a$ and the reference b. Again by (i) one has $b * p_1 * q_2 * a$ is marked, and hence $c * q_2 * a$ is marked (by remark 13.1 (ii)). By assumption a is not marked so $c * q_2$ is marked. □

13.7. Proposition. *Let $p * a : n \rightsquigarrow n$, and let $b \in acc(n)$. If $\mathrm{r}(a) \neq \mathrm{r}(b)$ then $use(a) = use(b) = \otimes$.*

Proof. Say $n_a = \mathrm{r}(a)$ and $n_b = \mathrm{r}(b)$. By course-of-values induction on the length of p. Suppose $|p| = k$, and the statement holds for paths of smaller length. If $n_b \notin p$ then we are done by lemma 13.6 (this covers in particular the case $|p| = 0$). Assume $n_b \in p$. Write $p = p_1 * p_2$ with $p_1 : n \rightsquigarrow n_b$ and $p_2 : n_b \rightsquigarrow n_a$. Since $n_a \neq n_b$, p_2 is not empty and hence $|p_1| < k$. Consider the path $p_1 * b$ and the reference a. By induction hypothesis $use(a) = use(b) = \otimes$. □

13.8. Corollary. *Let $p * a : n \rightsquigarrow n$, and let $b \in acc(n)$. Suppose $a \neq b$. If $\mathrm{r}(a) \neq \mathrm{r}(b)$ or $\mathrm{r}(a), \mathrm{r}(b)$ are data nodes then $use(a) = use(b) = \otimes$.*

13.9. Proposition. *Let p be a cycle, say $p : n \rightsquigarrow n$, and let $q : m \rightsquigarrow n$ such that $p \neq q$. Suppose p_1, q_1, p_2, q_2, r is the tail factorization of p, q. Furthermore, suppose r is a data path, and q_2 (and hence also p_2) is a simple path. If $q_1 * q_2$ is not empty then p and q are marked.*

Proof. By a case distinction.

Case 1. q_2 is not empty. Write $q_2 = q_2' * a$ and $p_2 = p_2' * b$. Since a and b start with the same simple node both references are marked, showing that p and q are marked.

Case 2. q_2 is empty and q_1 is not empty. Suppose p_1 is not empty. Write $q_1 = q_1' * a$ and $p_1 = p_1' * b$. Observe that a and b start with different nodes. By lemma 13.6 $use(a) = use(b) = \otimes$, and hence p and q are marked. Now suppose p_1 is empty, and hence $p = r$ and $q = q_1 * r$. Say c is the entrance reference of q_1 to p. Since p is a data path, both p and q are marked by corollary 13.8. □

The actual correctness proof of markings is split into two parts. First we will show that for extending reductions the corresponding labeling is a marking for reduct. Thereafter the same will done for projections.

In the sequel, let $g \in \mathcal{G}$, and let $\Delta = \langle R, \mu \rangle$ be a $\mathcal{R}$-redex in g. Say $g \xrightarrow{\Delta} h$. Moreover, let use_g be a saturated marking for g, and use_R a marking for R.

The proofs are structured as follows. We distinguish various situations of critical path combinations $(p, a) \wedge (p', a')$ in h, according to the position of a, a' and to the form of p, p' respectively. It will be shown that such critical paths are *well marked*, i.e. in any situation $p * a$ is marked (according to the constructed use_h) if $p * a \underset{\sim}{\trianglelefteq} p' * a'$, and $p' * a'$ is marked in case $p' * a' \underset{\sim}{\trianglelefteq} p * a$.

In the drawings, dotted lines refer to paths and references in the original graph g.

Extending reductions

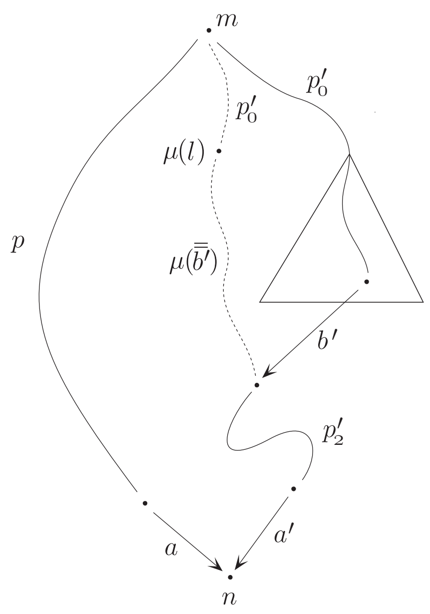
Suppose R is extending. Set $use_h = use_g^{\Delta}$.

13.10. Proposition. *Let $(p, a) \wedge (p', a')$ be a critical path combination, such that a, a' are both old. Then these paths are well marked.*

Proof. Distinguish cases as to the form of p, p'. Say m is the joining node of p, p'.

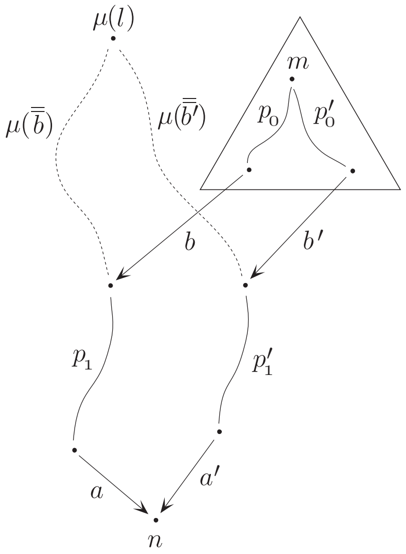
Case 1. p, p' are old. Then we are done.

Case 2. p, p' are new. Suppose $p * a \underset{\sim}{\trianglelefteq} p' * a'$; the reverse case is treated similarly. Let b, b' be the border references on p, p' respectively.



Write $p = p_0 * b * p_1$ and $p' = p'_0 * b' * p'_1$. Note that $b \underset{\sim}{\trianglelefteq} b'$ in $R \mid r$. Since R is consistent with the argument classification, one has $\overline{b} \underset{\sim}{\trianglelefteq} \overline{b'}$, and therefore $\mu(\overline{\overline{b}}) \underset{\sim}{\trianglelefteq} \mu(\overline{\overline{b'}})$ in g. Observe that $\mu(\overline{\overline{b}}) * p_1 * a \neq \mu(\overline{\overline{b'}}) * p'_1 * a'$, since $a \neq a'$. By proposition 13.5 the path $\mu(\overline{\overline{b}}) * p_1 * a$ is marked. If $\mu(\overline{\overline{b}})$ is marked then $use(b) = \otimes$. Otherwise $p_1 * a$. So in both cases $b * p_1 * a$ is marked showing that $p * a$ is marked.

Case 3. Otherwise. Suppose, without loss of generality, p is old and p' is new. Let b' be the border reference on p'.



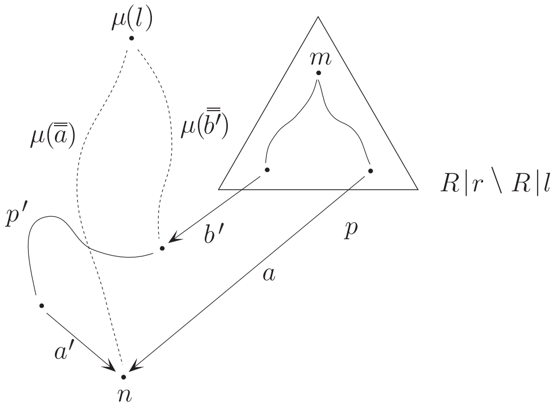
Write $p' = p'_0 * p'_1 * b' * p'_2$ with $p'_0 : m \rightsquigarrow r_R$, $p'_1 : r_R \rightsquigarrow \mathrm{r}(b')$ and $p'_2 : \mathrm{d}(b') \rightsquigarrow a'$. Then in g one has $p'_0 : m \rightsquigarrow \mu(l)$. Consider in g the path $p'_0 * \mu(\overline{\overline{b'}}) * p'_2$. Since $a \neq a'$ one has $p * a \neq p'_0 * \mu(\overline{\overline{b'}}) * p'_2 * a'$. If $p * a \trianglelefteq\!\!\!\!\sim\; p' * a'$ then $p * a$ is marked by proposition 13.5. If $p' * a' \trianglelefteq\!\!\!\!\sim\; p * a$ then the same proposition shows that $p'_0 * \mu(\overline{\overline{b'}}) * p'_2 * a'$ is marked in g. Hence $p' * a'$ is marked in h. $\square$

The situation in which the pair a, a' contains a new reference is more involved.

13.11. Proposition. *Let $(p, a) \wedge (p', a')$ be a critical path combination, such that a or a' is new. Then these paths are well marked.*

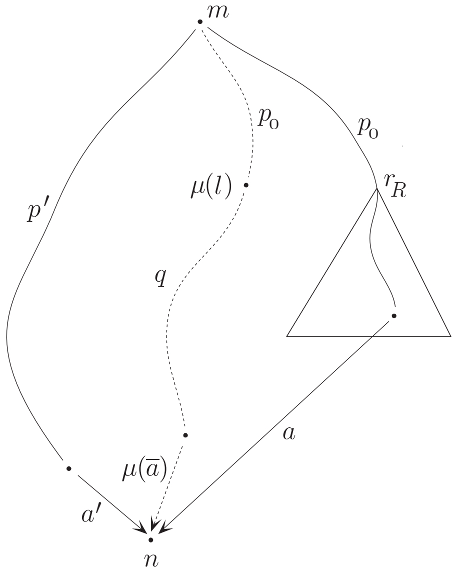
Proof. Suppose, without loss of generality, that a is new. If a' is also new and $\mathrm{d}_R(a) = \mathrm{d}_R(a')$ then we are done since $R|r$ is well marked. If a' is a redirected reference then we only have to consider the case that $\mathrm{d}(a) = r_R$. But then r_R is on a cycle, so $use(R) = \otimes$, so a' is marked in h. Now assume a is a border reference. We proceed by a case distinction. Say m is the joining node of the critical path combination.

Case 1. p' is new. Let b' be the border reference of $p' * a'$.



If $p * a \precsim p' * a'$, then $a \precsim b'$ in $R \mid r$, and, by consistency of R with the argument classification, $\overline{a} \precsim \overline{b'}$. Then one has $\mu(\overline{\overline{a}}) \precsim \mu(\overline{\overline{b'}})$, and therefore also $\mu(\overline{\overline{a}}) \precsim \mu(\overline{\overline{b'}}) * p' * a'$ in g. Now observe that $\mu(\overline{\overline{b'}}) * p' * a', \mu(\overline{\overline{a}}) : \mu(l) \rightsquigarrow n$. By absence of stacked references one has $\mu(\overline{\overline{b'}}) * p' * a' \neq \mu(\overline{\overline{a}})$. Then $\mu(\overline{\overline{a}})$ is marked by proposition 13.5, and therefore $use(a) = \otimes$. If $p' * a' \precsim p * a$, one similarly concludes that $\mu(\overline{\overline{b'}}) * p' * a'$ is marked. Note that if $\mu(\overline{\overline{b'}})$ is marked then $use(b') = \otimes$. Hence also $b' * p' * a'$ is marked. (Note that case 1 applies if a' is old as well as if a' is new; in the latter case one immediately has $b' \neq a$.)

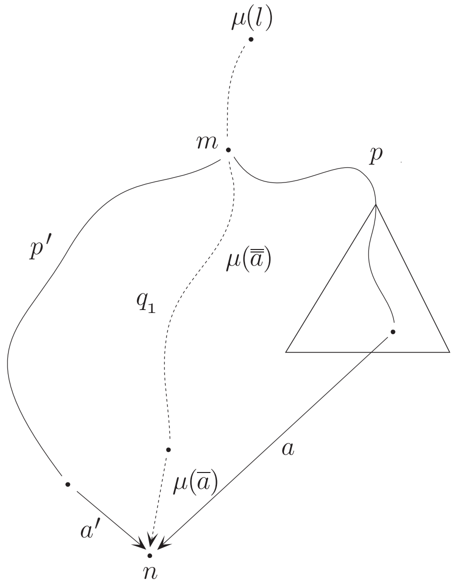
Case 2. p' is old.



Say $p_0 : m \rightsquigarrow r_R$ is the initial part of p. Then $p_0 : m \rightsquigarrow \mu(l)$ is a path in g. Moreover, $p_0 * \mu(\overline{\overline{a}}) : m \rightsquigarrow n$ is a path in g. If m and n coincide then $p' * a'$ is cyclic. Hence by proposition 13.9 and the fact that $\mu(l) \notin p' * a'$, both $\mu(\overline{\overline{a}})$ and $p' * a'$ are marked in g, so $p * a$ and $p' * a'$ are marked in h. Now assume $m \neq n$.

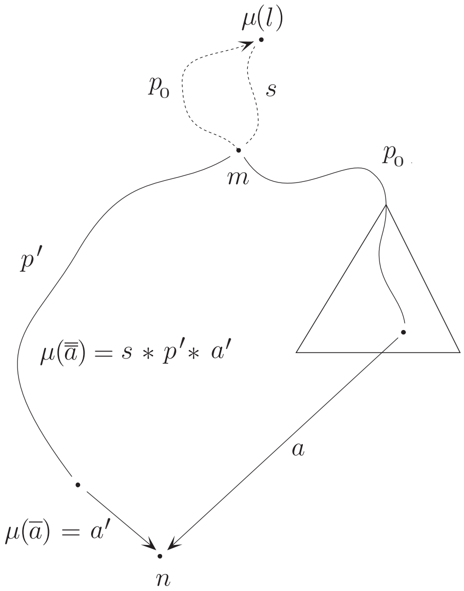
Case 2a. $p' * a'$ is not a final part of $\mu(\overline{\overline{a}})$. If $m \notin \mu(\overline{\overline{a}})$ set $q_1 = p_0 * \mu(\overline{\overline{a}})$. Otherwise let q_1 be the tail part of $p_0 * \mu(\overline{\overline{a}})$ starting with the last occurrence of m. Note that

$q_1 : m \rightsquigarrow n$ and that q_1 is not root cyclic. The case $m \in \mu(\overline{\overline{a}})$ is depicted below.



Observe that that q_1 is not empty (since $m \neq n$), and that $p' * a' \neq q_1$. If $p * a \precsim p' * a'$ in h then also $q_1 \precsim p' * a'$ (observe that m is simple in case $m \in \mu(\overline{\overline{a}})$). Hence q_1 is marked by proposition 13.5. Since $\mu(l) \in \mu(\overline{\overline{a}})$ is a function node it follows that $\mu(\overline{\overline{a}})$ is marked, so $use(a) = \otimes$. If $p' * a' \precsim p * a$ then $p' * a'$ is marked by proposition 13.5.

Case 2b. $\mu(\overline{\overline{a}}) = s * p' * a'$ for some path s.



First note that $\mu(\overline{\overline{a}})$ contains a cycle if $n \in s$. Since $\mu(l)$ is not on this cycle, $\mu(\overline{\overline{a}})$ is marked by proposition 13.9. Assume $n \notin s$.

Claim. In g every path leading to m is marked. Hence by saturation the first reference of $p' * a'$ is marked. Consequently, $p' * a'$ and $\mu(\overline{\overline{a}})$ are marked, and $use(a) = \otimes$, so $p * a$ is marked.

Proof of the claim. Let $t : r_g \rightsquigarrow m$ be a path in g. Suppose s_1, t_1, s_2, t_2, r is the tail factorization of s, t. If s_1 is not empty then t_2 is simple, since $\mu(l) \notin t_2$. By proposition 13.9 t is marked. If s_1 is empty reasoning becomes slightly more delicate. Note that $t_2 * r \parallel s$. We distinguish two cases.

Case 1. In g there exists a path $q : r_g \rightsquigarrow m$ containing a reference $c = (n', i)$ such that $c \notin s$, $n' \neq \mu(l)$ and either $\mathrm{d}(c) \in s$ or $\mathrm{d}(c) = m$. Observe that q, s can be written as $q = q' * c * r'$ and $s = s' * c' * r'$ respectively, with $c \neq c'$. By proposition 13.9 the path $p_0 * t_2 * r$ is marked (consider either $c * r'$ or $c' * r'$). Hence t is marked since t_2 starts with the function node $\mu(l)$.

Case 2. Otherwise. Say $[\overline{\overline{a}}] = j$; write $s = (\mu(l), j) * s'$. Now observe that t can be written as $t = t' * (\mu(l), i) * s'$ for some i. Since m is present in h, there is a border reference c in h such that either $\mathrm{d}(c) \in s$ or $\mathrm{d}(c) = m$. Note that $c \neq a$. Say $[\overline{\overline{c}}] = k$. Then $\mu(\overline{\overline{c}}) = (\mu(l), k) * s''$ for some initial part s'' of s'. By absence of stacked references one has $j \neq k$. Moreover a and c are stack suspect. Hence by lemma 12.7 (applied twice) one has $i \precsim j$ and $i \precsim k$. Since either $i \neq j$ or $i \neq k$ the first reference $(\mu(l), i)$ of $t_2 * r$ is marked, so t is marked. □

This completes the proof of correctness of $use_h = use_g^{\Delta}$.

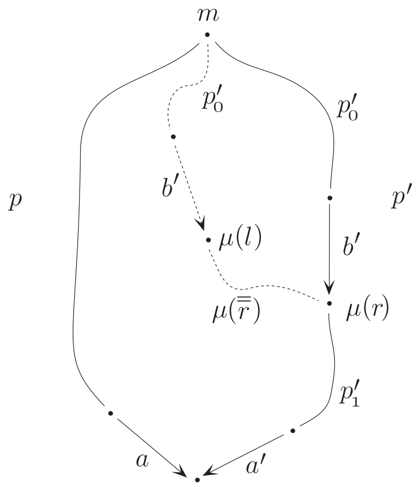
Projecting reductions

Suppose R is projecting. Set $use_h = use_g^{\Delta}$. Although projections look simpler than extending reductions, the analysis requires a more complex case distinction.

First of all, if $\mu(l) = \mu(r)$, then the redirection is trivial and we are done immediately. Assume, for the rest of this proof, that $\mu(l) \neq \mu(r)$. Since moreover the pattern of R does not contain function nodes apart from l, the path $\mu(\overline{\overline{r}})$ is not root cyclic.

13.12. Proposition. *Let $(p, a) \wedge (p', a')$ be a critical path combination, such that a, a' are both old. Then these paths are well marked.*

Proof. Say m is the joining node of the critical path combination. If both paths are old and $p * a \precsim p' * a'$ then trivially $p * a$ is marked. Now suppose p is old, and p' is new. Write $p' = p'_0 * b' * p'_1$ with $p'_1 : \mu(r) \rightsquigarrow \mathrm{r}(a')$.



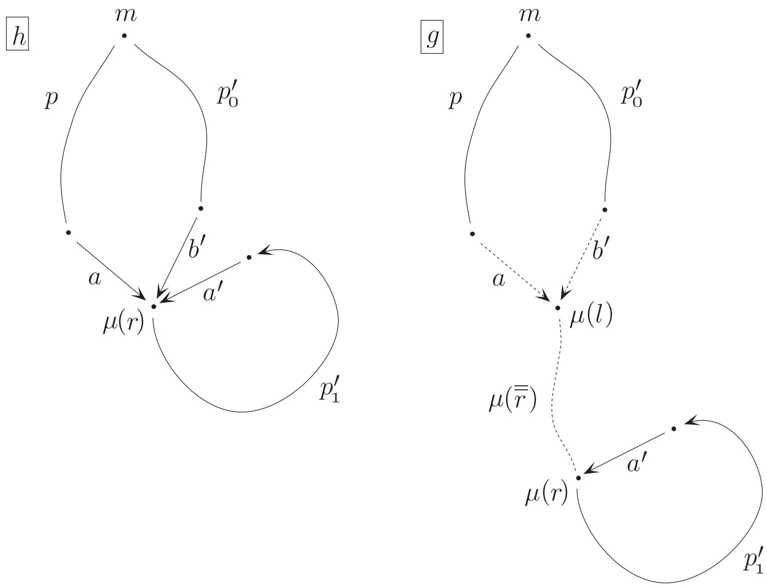
Case 1. $p * a \precsim p' * a'$ in h. Then in g one has $p * a \precsim p'_0 * b' * \mu(\overline{\overline{r}}) * p'_1 * a'$. Hence $p * a$ is marked by proposition 13.5.

Case 2. $p' * a' \precsim p * a$ in h. Then $p'_0 * b' * \mu(\overline{\overline{r}}) * p'_1 * a'$ is marked, again by proposition 13.5. If $p'_1 * a'$ is marked we are done. Otherwise $\mu(\overline{\overline{r}})$ is marked in g since it starts with a function node and therefore b' is marked in h, showing that $p' * a'$ is marked. □

13.13. Proposition. *Let $(p, a) \wedge (p', a')$ be a critical path combination, where $a \neq a'$, $\mathrm{d}_h(a) = \mathrm{d}_h(a')$, and a or a' is new. Then these paths are well marked.*

Proof. Say m is the joining node of $(p, a) \wedge (p', a')$.

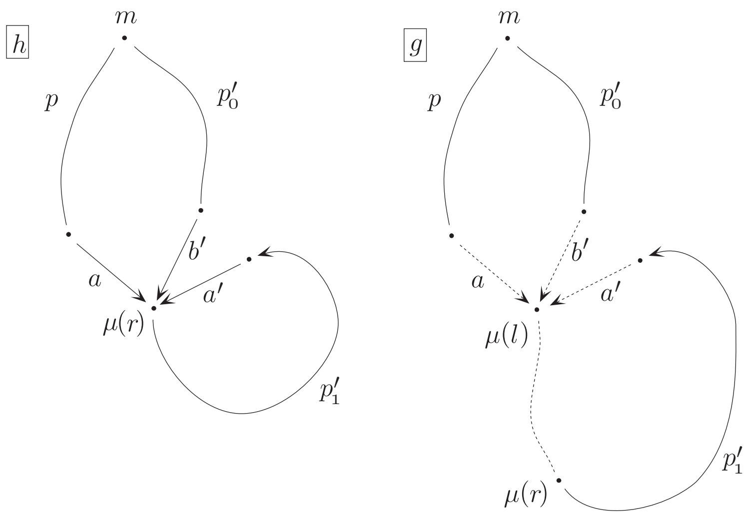
Case 1. a is new, a' is old, p is old and p' is new. Write $p' = p'_0 * b' * p'_1$ where b' is the new reference on p'. Consider in g the paths $\mu(\overline{\overline{r}})$ and $\mu(\overline{\overline{r}}) * p'_1 * a'$. This looks as follows.



Since $\mu(l) \notin p'_1$ the path $\mu(\overline{\overline{r}})$ contains a reference not appearing on the cycle $p'_1 * a'$. By proposition 13.9 both of these paths are marked. Since $\mu(\overline{\overline{r}})$ is marked, also $p * a$ is marked in h. Moreover because $\mu(\overline{\overline{r}}) * p'_1 * a'$ is marked it follows that $(p'_0 *)\ b' * p'_1 * a'$ is marked in h.

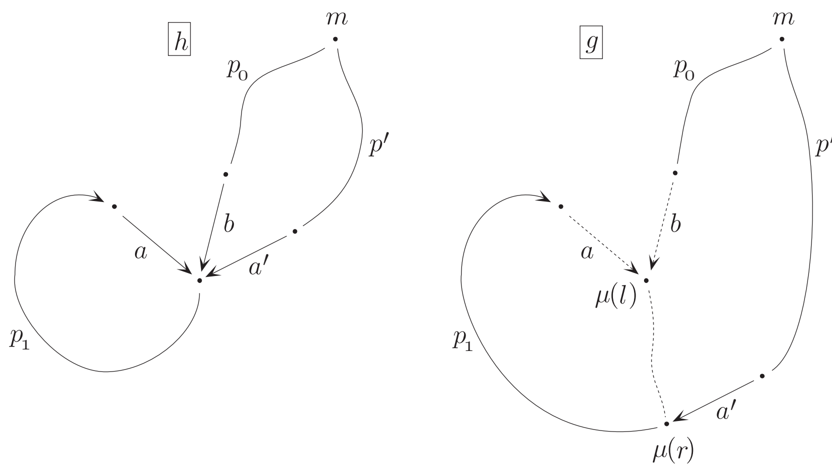
Case 2. a, a' are both new, and one of p, p' is new. Assume, without loss of generality, that p is old and p' is new. Again write $p' = p'_0 * b' * p'_1$. The situation in h

is the same as in case 1. In g, however, one has the following.



By proposition 13.7 both $p * a$ and $\mu(\overline{\overline{r}}) * p_1' * a'$ are marked in g, since a and a' start in different nodes. Hence $p * a$ and $p' * a'$ are marked in h.

Case 3. a is new, a' is old, p is new and p' is old. Write $p = p_0 * b * p_1$ with b new. The situation is as follows.

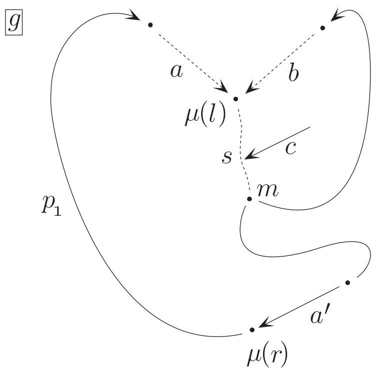


As to $p * a$, observe that a and b start in distinct nodes since p is acyclic. Therefore a is marked by proposition 13.7.

As to $p' * a'$, if $m = \mu(r)$ then $p' * a'$ is marked by proposition 13.9 (consider $\mu(\overline{\overline{r}})$ and $p' * a'$). Now suppose $m \neq \mu(r)$.

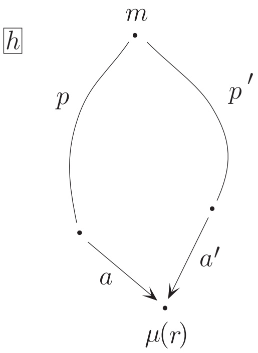
Case a. $p' * a'$ is not a tail part of $\mu(\overline{\overline{r}})$. Then we are done by proposition 13.9 (consider $p' * a'$ and $p_1 * a * \mu(\overline{\overline{r}})$).

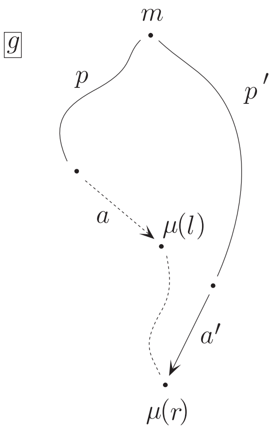
Case b. $p' * a'$ is a tail part of $\mu(\overline{\overline{r}})$. Write $\mu(\overline{\overline{r}}) = s * p' * a'$.



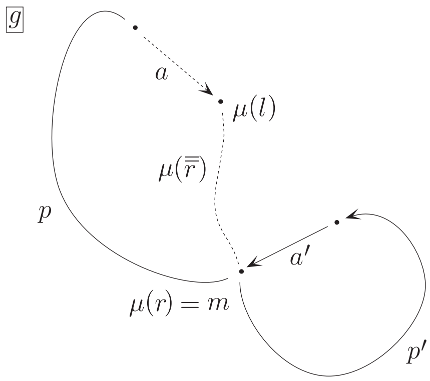
Since m is present in h there exists a reference c to s, starting in a node not occurring on s. Note that c is old since $m \neq \mu(r)$. This shows that s is marked (consider the cycle $s * p' * a' * p_1 * a$ and use proposition 13.7). From this we can deduce, as in the proof of proposition 13.11 (case 2b) but easier, that any rooted path to data node m is marked. Hence by saturation the first reference of $p' * a'$ is marked and we are done.

Case 4. a is new, a' is old, and p, p' are old.





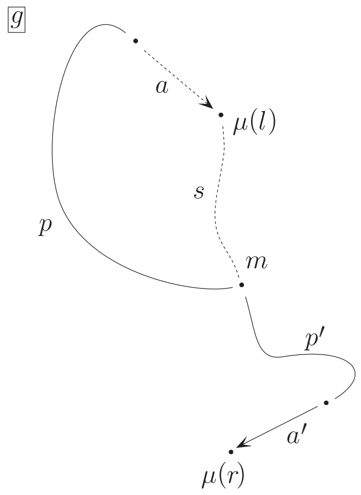
If $m = \mu(r)$ then the following situation appears in g.



Now both $\mu(\overline{\overline{r}})$ and $p' * a'$ are marked by proposition 13.9. Hence in h the paths $p * a$ and $p' * a'$ are marked. Now suppose $m \neq \mu(r)$.

Case a. $p' * a'$ is not a tail part of $\mu(\overline{\overline{r}})$. Let q be the unique subpath of $p * a * \mu(\overline{\overline{r}})$ such that $q : m \rightsquigarrow \mu(r)$ and q is not root cyclic. If $p * a \lesssim\!\!\!\!\lhd\, p' * a'$ in h then $q \lesssim\!\!\!\!\lhd\, p' * a'$ in g (observe that m is simple if it occurs on $\mu(\overline{\overline{r}})$). Hence by proposition 13.5 (note that q and $p' * a'$ only intersect in data nodes) q is marked in g, so $p * a$ is marked in h. In case $p' * a' \lesssim\!\!\!\!\lhd\, p * a$ a similar argument shows that $p' * a'$ is marked.

Case b. $p' * a'$ is a tail part of $\mu(\overline{\overline{r}})$.



First note that m is reachable from r_h, so there exists a reference c to a node of s (not equal to $\mu(l)$) starting in a node not occurring on s. Hence s is marked by proposition 13.7, so a is marked in h, so $p * a$ is marked. As to $p' * a'$, as before one can show that the first reference of $p' * a'$ is marked by saturation.

Case 5. a and a' are new, and p, p' are old. Then also in g one has $p * a \lesssim\!\!\!\!\lhd\, p' * a'$ or $p' * a' \lesssim\!\!\!\!\lhd\, p * a$ respectively, so $p * a$ (or $p' * a'$ respectively) is marked. □

Thus it is shown that use_g^{Δ} is a marking for h.

14. Typing reducts

In this section we will complete the proof that uniqueness typing is preserved during reduction.

In the proofs of this section, let $\mathfrak{T}$ be Curry safe for $\mathcal{F}$. Let $g \in \mathcal{G}$, and let $\Delta = \langle R, \mu \rangle$ be a redex in g. Say $g \xrightarrow{\Delta} h$. Let $\mathcal{U}_g$ be a $\mathcal{F}, \mathcal{A}$-uniqueness typing for g according to use_g.

The key result is the following.

14.1. Proposition. *Suppose use_g is saturated. Then there exists a marking use_h for h and a $\mathcal{F}, \mathcal{A}$-uniqueness typing $\mathcal{U}_h$ according to use_h such that $\mathcal{U}_g(r_g) = \mathcal{U}_h(r_h)$.*

The proof will be split into two parts. We start with some results on coercions.

14.2. Lemma. *Let $u, v \in \mathbb{M}$. Then*

(i) $\sigma \leq^u \tau,\ \tau \leq^v \rho \ \Rightarrow\ \sigma \leq^{u+v} \rho$.

(ii) $\sigma \leq \tau,\ \tau \leq^v \rho \ \Rightarrow\ \sigma \leq^v \rho$.

PROOF. Straightforward. $\square$

14.3. LEMMA. *$\sigma \leq \tau$, $[\sigma] = \times \;\Rightarrow\; \sigma \leq^{\otimes} \tau$.*

PROOF. Simple. $\square$

Typings for extending reductions

14.4. DEFINITION. Using corollary 9.6, determine $\mathcal{U}$ and *use* for $R \mid r$ such that

$$\mathcal{U}(r) \leq^{use(R)} \mathcal{U}_g(\mu(l)) \tag{1}$$

and for any $n \in (R \mid l) \cap (R \mid r)$

$$\mathcal{U}_g(\mu(n)) \leq \mathcal{U}(n), \tag{2}$$

$$[\mathcal{U}(n)] \neq \times \;\Rightarrow\; \mu(\overline{\overline{n}}) \text{ is not marked by } use_g. \tag{3}$$

(i) Set $use_h = use^{\Delta}$ (see definition 12.4).

(ii) Define $\mathcal{U}_h = \mathcal{U}^{\Delta}$ as follows.

$$\begin{array}{rcll} \mathcal{U}^{\Delta}(n) & = & \mathcal{U}_g(n) & \text{if } n \in g, \\ & = & \mathcal{U}(n) & \text{if } n \in (R \mid r) \backslash (R \mid l). \end{array}$$

Note that use_h is a marking for h, by theorem 12.8.

14.5. LEMMA. *Let $\sigma \in \widehat{\mathbb{T}}$, and $u \in \mathbb{M}$.*

(i) *$\mathcal{U}_g(\mu(l)) \leq^{u} \sigma \;\Rightarrow\; \mathcal{U}(r) \leq^{u+use(R)} \sigma$.*

(ii) *For each border reference a of R*

$$\mathcal{U}(\mathrm{d}(a)) \leq^{u} \sigma \;\Rightarrow\; \mathcal{U}_g(\mu(\mathrm{d}(a)) \leq^{u+use(\mu(\overline{\overline{a}}))} \sigma.$$

PROOF. (i) Note that

$$\begin{array}{rcll} \mathcal{U}(r) & \leq^{use(R)} & \mathcal{U}_g(\mu(l)), & \text{by (1)} \\ & \leq^{u} & \sigma, & \text{by assumption.} \end{array}$$

Hence by lemma 14.2 (i) we are done.

(ii) Observe that

$$\begin{array}{rcll} \mathcal{U}_g(\mu(\mathrm{d}(a)) & \leq & \mathcal{U}(\mathrm{d}(a)), & \text{by (2)} \\ & \leq^{u} & \sigma, & \text{by assumption.} \end{array}$$

In case $[\mathcal{U}(\mathrm{d}(a))] = \times$ the result follows from the lemmas 14.3 and 14.2 (ii). If, on the other hand, $[\mathcal{U}(\mathrm{d}(a))] \neq \times$ then $use_g(\mu(\overline{\overline{a}})) = \odot$ by (3), so we are done by lemma 14.2 (ii). $\square$

14.6. LEMMA. *$\mathcal{U}_h$ is a uniqueness typing according to use_h.*

PROOF. Since the node types are taken from $\mathcal{U}_g$ and $\mathcal{U}$ we only have to check that the coercions induced by the new use_h are valid. If $use_h(a)$ is equal to $use_g(a)$ or $use(a)$ this is simple. In other cases (redirected and border references) the correctness of $\mathcal{U}_h$ follows from lemma 14.5. $\square$

Finally note that $\mathcal{U}_h(r_h) = \mathcal{U}_g(r_g)$ since $r_g = r_h$: the root does not take part in the rewriting process. This completes the proof of proposition 14.1 in the extension case.

Typings for projections

14.7. DEFINITION. Using corollary 9.6, determine $\mathcal{U}$ and *use* for $R \mid r$ such that

$$\mathcal{U}(r) \leq^{use(R)} \mathcal{U}_g(\mu(l)) \tag{1}$$

and for any $n \in (R \mid l) \cap (R \mid r)$

$$\mathcal{U}_g(\mu(n)) \leq \mathcal{U}(n), \tag{2}$$

$$[\mathcal{U}(n)] \neq \times \;\Rightarrow\; \mu(\overline{\overline{n}}) \text{ is not marked by } use_g. \tag{3}$$

(i) Set $use_h = use^{\Delta}$ (see definition 12.10).
(ii) Define $\mathcal{U}_h = \mathcal{U}^{\Delta} = \mathcal{U}_g \upharpoonright N_h$.

First note that use_h is a marking function for h, by theorem 12.11.

14.8. LEMMA. *Let $\sigma \in \widehat{\mathbb{T}}$, and $u \in \mathbb{M}$. Then*

$$\mathcal{U}_g(\mu(l)) \leq^{u} \sigma \;\Rightarrow\; \mathcal{U}_g(\mu(r)) \leq^{u+use(\mu(\overline{\overline{r}}))} \sigma.$$

PROOF. Analogous to the proof of lemma 14.5 (ii). □

14.9. LEMMA. *$\mathcal{U}_h$ is a uniqueness typing for h according to use_h.*

PROOF. By the fact that $\mathcal{U}_g$ is a uniqueness typing for g, and lemma 14.8 (showing that the modifications made to use_g are harmless). □

The subject reduction property

Finally we can state and prove the main result.

14.10. SUBJECT REDUCTION THEOREM. *Suppose $\mathfrak{T} = \langle \mathcal{G}, \mathcal{R} \rangle$ is Curry safe for $\mathcal{F}$. Then for any $g \in \mathcal{G}$*

$$\mathcal{F}, \mathcal{A} \vdash g : \sigma, \quad g \underset{\mathcal{R}}{\rightarrow} h \;\Rightarrow\; \mathcal{F}, \mathcal{A} \vdash h : \sigma.$$

PROOF. Say $\mathcal{U}_g$ is a $\mathcal{F}, \mathcal{A}$-uniqueness typing for g according to use_g. By the Saturation Theorem 11.10 there exists a saturated marking function use'_g and a typing $\mathcal{U}'_g$ according to use'_g such that $\mathcal{U}'_g(r_g) = \mathcal{U}_g(r_g)$. Now proposition 14.1 applies and we are done. □

15. Applications

In this section we will give some examples that show how the two problems mentioned in the introduction can be solved by using uniqueness types.

Consider the following list reversing function which can be implemented as a 'destructive' function if the given uniqueness type is used.

$\mathbf{Rev} : \overset{\bullet}{\mathrm{List}}(\overset{\times}{\alpha}) \rightarrowtail \overset{\bullet}{\mathrm{List}}(\overset{\times}{\alpha})$	$\mathbf{Rev}\ (l)$	$\rightarrow$	$\mathbf{H}(l, \mathbf{Nil})$
$\mathbf{H} : (\overset{\bullet}{\mathrm{List}}(\overset{\times}{\alpha}), \overset{\bullet}{\mathrm{List}}(\overset{\times}{\alpha})) \rightarrowtail \overset{\bullet}{\mathrm{List}}(\overset{\times}{\alpha})$	$\mathbf{H}\ (\mathbf{Nil}, r)$	$\rightarrow$	r
	$\mathbf{H}\ (\mathbf{Cons}(h, t), r)$	$\rightarrow$	$\mathbf{H}(t, \mathbf{Cons}(h, r))$

Note that not only the space of the obsolete **Cons** can be re-used, but also parts of its contents. In fact it even suffices to redirect the reference to t such that it points to r.

The second example shows how a predefined function **WriteChar** can be typed, having as input an object of type File and a character that should be written to the given file. The output consists of the modified file which is also unique, so it can be used for further writing.

$$\boxed{\textbf{WriteChar} : (\overset{\bullet}{\text{File}}, \overset{\times}{\text{Char}}) \rightarrowtail \overset{\bullet}{\text{File}}}$$

A more elaborated example, presenting an efficient quicksort algorithm that performs the sorting *in situ* on the data structure, can be found in Smetsers *et al.* [1993]. This algorithm is based on the same idea as used in the list reversal example.

16. Future research

Theoretical research is performed on type derivation in the uniqueness type system. If one allows uniqueness attribute variables then it is possible to formulate a notion of *principal uniqueness type* consisting of a type scheme and a collection of 'coercion constraints' on the occurring attribute variables. Furthermore, the relation between reduction strategies and the symbol classification can be concretized further.

A variant of the type system described here is currently being incorporated in the language *Concurrent Clean*.

References

Abramsky S., D. Gabbay and T. Maibaum, *Handbook of Logic in Computer Science*, volume I, Oxford University Press, 1992.

Achten P.M., J.H.G. van Groningen and M.J. Plasmeijer, High Level Specification of I/O in Functional Languages, in: *Proc. of International Workshop on Functional Languages*, Glasgow, UK, Springer Verlag, 1993.

Bakel S. van, S. Smetsers and S. Brock, Partial Type Assignment in Left-Linear Term Rewriting Systems, in: J.C. Raoult, editor, *Proc. of 17th Colloqium on Trees and Algebra in Programming (CAAP'92)*, pages 300–322, Rennes, France, Springer Verlag, LNCS 581, 1992.

Barendregt H.P., Lambda Calculi with Types, in: Abramsky *et al.* [1992], 1992.

Barendregt H.P., M.C.J.D. van Eekelen, J.R.W. Glauert, J.R. Kennaway, M.J. Plasmeijer and M.R. Sleep, Towards an Intermediate Language based on Graph Rewriting, in: *Proc. of Parallel Architectures and Languages Europe (PARLE)*, pages 159–175, Eindhoven, The Netherlands, Springer Verlag, LNCS 259 II, 1987.

Barendregt H.P., M.C.J.D. van Eekelen, J.R.W. Glauert, J.R. Kennaway, M.J. Plasmeijer and M.R. Sleep, Term Graph Reduction, in: *Proc. of Parallel Architectures and Languages Europe (PARLE)*, pages 141–158, Eindhoven, The Netherlands, Springer Verlag, LNCS 259 II, 1987.

Barendsen Erik and Sjaak Smetsers, Graph Rewriting and Copying, Technical Report 92-20, University of Nijmegen, 1992.

Guzmán J.C. and P. Hudak, Single-Threaded Polymorphic Lambda Calculus, in: *Proc. of 5th IEEE Symp. on Logic in Computer Science*, pages 333–343, Philadelphia, PA, IEEE Computer Society Press, 1990.

Jacobs B.P.F., Conventional and Linear Formulas in a Logic of Coalgebras, Typescript, University of Utrecht, 1993.

Nöcker E.G.J.M.H., J.E.W. Smetsers, M.C.J.D. van Eekelen and M.J. Plasmeijer, Concurrent Clean, in: *Proc. of Parallel Architectures and Languages Europe (PARLE'91)*, pages 202–219, Eindhoven, The Netherlands, Springer Verlag, LNCS 505, 1991.

Robinson J.A., A Machine-Oriented Logic Based on the Resolution Principle, *Journal of the ACM*, 12(1), 1965.

Sastry William, A.V.S. Clinger and Zena Ariola, Order-of-Evaluation Analysis for Destructive Updates in Strict Functional Languages with Flat Aggregates, in: *Proc. Conference on Functional Languages and Computer Architecture (FPCA '93)*, pages 266–276, Copenhagen, Denmark, ACM Press, 1993.

Smetsers Sjaak, Erik Barendsen, Marko van Eekelen and Rinus Plasmeijer, Guaranteeing Safe Destructive Updates through a Type System with Uniqueness Information for Graphs, Technical report, University of Nijmegen, 1993.

Wadler P., Linear types can change the world!, in: *Proc. of Working Conference on Programming Concepts and Methods*, pages 385–407, Israel, North Holland, 1990.

Wadsworth C.P., *Semantics and Pragmatics of the Lambda Calculus*, PhD thesis, Oxford University, 1971.