Proof of the Decidability of the Uniform Word Problem for Monads Assisted by Elf

Wolfgang Gehrke*

Research Institute for Symbolic Computation
Johannes Kepler University
A – 4040 Linz, AUSTRIA
Wolfgang.Gehrke@risc.uni-linz.ac.at

Abstract. We present a full proof of a canonical system for adjunctions as already suggested in [Cur93]. Termination can be shown in a similar style to [HL86]. Confluence is shown by checking all critical pairs. This is done firstly in the Larch Prover [GG91] and secondly more correctly in the programming language Elf [Pfe89]. Exploiting theorems from category theory [BW85] this system can be used to solve the uniform word problem for monads. The resulting decision procedure is finally implemented in Elf.

1 Introduction

A general problem in category theory is to check the commutativity of certain diagrams where diagrams are nothing but a compact encoding and visualization of equations involving morphisms. In [FS90] one can even find a suitable graphical language. Checking the commutativity means to check the equality of morphisms in a given category. To support this process one can solve the uniform word problem for this category which is of course not always possible having the halting problem in mind.

Starting from the basic definition of a category the most simple notions are functor and natural transformation (cf. the standard text book [ML71], [AHS90] as a recent book, or [Pie91] as an introduction for computer scientists). All these concepts can be characterized in terms of typed equations for morphisms. Canonical systems can be easily found [Geh94] for these basic notions.

Adjunctions (or adjoint functors) are one of the next categorical notions where finding a canonical system becomes more complicated. In [Cur93] a canonical system is given without proof but with an unpublished reference. As a prerequisite the full proof is worked out in this article. Termination is shown with a technique similar to [HL86]. The test for confluence is done with the Larch Prover [GG91] but additionally within the programming language Elf [Pfe89]. Only Elf can represent typed rewriting in an easy way.

Recently the concept of monads which comes from category theory as the categorical generalization of a monoid (cf. [ML71] or [BW85]) became important for computer science, too. In [Mog89] monads are used to structure the semantics of programming languages. Later on this was exploited to structure functional programs as in [Wad93]. Therefore it was natural to ask whether the equational theory of monads is decidable.

The main contributions of this paper consist of:

- a full proof of canonicity of a rewriting system for adjunctions,
- a solution of the uniform word problem for monads using this canonical system,
- the demonstration of the applicability of Elf as a tool for typed rewriting and checking of critical pairs.

Finally we conclude and suggest future work.

* sponsored by the Austrian Science Foundation (FWF), ESPRIT BRP 6471 MEDLAR II

2 Definitions

In this section several equivalent definitions of an adjunction and the definition of a monad are given. For other basic categorical notions see [ML71] or [Pie91]. We assume throughout that the morphisms between two objects form a set.

Definition 1 (Adjunction). Let C and C' be categories and $L : C \to C'$ and $R : C' \to C$ be functors. L and R are *adjoint* $L \dashv R$ if there is a bijection $\zeta_{X,Y} : Hom_{C'}(L(X), Y) \to Hom_C(X, R(Y))$ for all objects $X \in Obj(C)$ and $Y \in Obj(C')$ where the following law is satisfied:

$$\textbf{(Adj)} \qquad \zeta(f \circ' g \circ' L(h)) = R(f) \circ \zeta(g) \circ h$$

assuming $f \in Hom_{C'}(A', B'),\ g \in Hom_{C'}(L(B), A'),\ h \in Hom_C(A, B)$.

Remark. This law can also be split into two equations which are equivalent assuming $f \in Hom_{C'}(A', B'),\ g \in Hom_{C'}(L(B), A'),\ h \in Hom_C(A, B)$:

$$\zeta(f \circ' g) = R(f) \circ \zeta(g)$$
$$\zeta(g \circ' L(h)) = \zeta(g) \circ h$$

Furthermore, since $\zeta_{X,Y}$ is a bijection the following equations are implicitly obtained assuming $f \in Hom_{C'}(L(X), Y),\ g \in Hom_C(X, R(Y))$.

$$\zeta^{-1}(\zeta(f)) = f$$
$$\zeta(\zeta^{-1}(g)) = g$$

where $\zeta^{-1}_{X,Y} : Hom_C(X, R(Y)) \to Hom_{C'}(L(X), Y)$ describes the inverse of ζ.

For describing an adjunction the following notation is in use:

$$\frac{A \to R(B)}{L(A) \to B}$$

Note 2. The constructors representing an adjunction on the level of morphisms are $(id, id', L, R, \zeta, \zeta^{-1}, \circ, \circ')$ in this definition. With the following lemma the number of constructors can be reduced by introducing a new constant $\varepsilon_Z := \zeta^{-1}(id_{R(Z)})$.

Lemma 3. *An adjunction can be completely characterized by $(id, id', \varepsilon, L, \zeta, \circ, \circ')$ on the level of morphisms assuming it is known how R acts on objects.*

Proof cf. [Cur93]. The missing components can be expressed as:

$$R(f) := \zeta(f \circ' \varepsilon_{A'})$$
$$\zeta^{-1}(g) := \varepsilon_{B'} \circ' L(g)$$

where $f \in Hom_{C'}(A', B')$ and $g \in Hom_C(A, R(B'))$. □

Remark. In the literature one often can find another characterization of an adjunction with the help of one natural transformation and a unique morphism closing a certain diagram or with the help of two natural transformations.

Lemma 4. *The following characterizations of an adjunction are equivalent with the definition above when given the categories C, C' and the functors L, R:*

1. *There is a natural transformation* $\eta : I_C \rightarrow (R \circ L)$ *called* unit *such that for each object* $X \in Obj(C)$ *and each morphism* $f \in Hom_C(X, R(Y))$ *there is a unique morphism* $\bar{f} \in Hom_{C'}(L(X), Y)$ *such that*

$$R(\bar{f}) \circ \eta_X = f \qquad \begin{tikzcd} X \arrow[r, "\eta_X"] \arrow[dr, "f"'] & R(L(X)) \arrow[d, "R(\bar{f})"] \\ & R(Y) \end{tikzcd}$$

2. *There is a natural transformation* $\epsilon : (L \circ R) \rightarrow I_{C'}$ *called* counit *such that for each object* $Y \in Obj(C')$ *and each morphism* $g \in Hom_{C'}(L(X), Y)$ *there is a unique morphism* $\tilde{g} \in Hom_C(X, R(Y))$ *such that*

$$\epsilon_Y \circ' L(\tilde{g}) = g \qquad \begin{tikzcd} L(X) \arrow[r, "L(\tilde{g})"] \arrow[dr, "g"'] & L(R(Y)) \arrow[d, "\epsilon_Y"] \\ & Y \end{tikzcd}$$

3. *There are two natural transformations* $\eta : I_C \rightarrow (R \circ L)$ *(the unit) and* $\epsilon : (L \circ R) \rightarrow I_{C'}$ *(the counit).*

Proof. This can be verified by:
i) $\eta_X = \zeta(id_{L(X)})$ and $\bar{f} = \zeta^{-1}(f)$
ii) $\epsilon_X = \varepsilon_X$ and $\tilde{g} = \zeta(g)$
iii) $\eta_X = \zeta(id_{L(X)})$ and $\epsilon_X = \varepsilon_X$.
ii) is a complete characterization as shown in lemma 3. i) is dual in the sense that ζ^{-1} is used instead of ζ.
iii) can be seen from:

$$\zeta(f) = R(f) \circ \eta_A$$
$$\zeta^{-1}(g) = \epsilon_{B'} \circ' L(g)$$

where $f \in Hom_{C'}(L(A), B)$ and $g \in Hom_C(A, R(B'))$. □

Remark. For rewriting we will use only the presentation with $(id, id', \varepsilon, L, \zeta, \circ, \circ')$.

Definition 5 (Monad). Let C be a category. A *monad* $M = (T, \eta, \mu)$ on C is an endofunctor $T : C \rightarrow C$ with two natural transformations $\eta : I_C \rightarrow T$ and $\mu : (T \circ T) \rightarrow T$ where the following laws are satisfied:

(Mon1)	$\mu_A \circ T(\mu_A) = \mu_A \circ \mu_{T(A)}$
(Mon2)	$\mu_A \circ T(\eta_A) = id_{T(A)}$
(Mon3)	$\mu_A \circ \eta_{T(A)} = id_{T(A)}$

$$\begin{tikzcd} T^3(A) \arrow[r, "T(\mu_A)"] \arrow[d, "\mu_{T(A)}"'] & T^2(A) \arrow[d, "\mu_A"] \\ T^2(A) \arrow[r, "\mu_A"'] & T(A) \end{tikzcd} \qquad \begin{tikzcd} T(A) \arrow[r, "T(\eta_A)"] \arrow[d, "\eta_{T(A)}"'] \arrow[dr, "id_{T(A)}"] & T^2(A) \arrow[d, "\mu_A"] \\ T^2(A) \arrow[r, "\mu_A"'] & T(A) \end{tikzcd}$$

associativity *unit laws*

Fig. 1. monad laws

The laws for a monad can be nicely visualized by diagrams cf. figure 1 which commute.

3 Rewriting for an Adjunction

In this section a canonical system for an adjunction is presented together with the proof of termination and confluence.

Note 6. Assuming it is known how $R : C' \to C$ acts on objects there is the following rewriting system (ADJ) for an adjunction $L \dashv R$ with $L : C \to C'$ (cf. [Cur93]):

$$\begin{aligned}
id_B \circ f_{A \to B} &\longrightarrow f && (1)\\
f_{A \to B} \circ id_A &\longrightarrow f && (2)\\
(f_{A3 \to A4} \circ g_{A2 \to A3}) \circ h_{A1 \to A2} &\longrightarrow f \circ (g \circ h) && (3)\\
id'_{B'} \circ' f'_{A' \to B'} &\longrightarrow f' && (4)\\
f'_{A' \to B'} \circ' id'_{A'} &\longrightarrow f' && (5)\\
(f'_{A3' \to A4'} \circ' g'_{A2' \to A3'}) \circ' h'_{A1' \to A2'} &\longrightarrow f' \circ' (g' \circ' h') && (6)\\
L(id_A) &\longrightarrow id'_{L(A)} && (7)\\
L(f_{A2 \to A3}) \circ' L(g_{A1 \to A2}) &\longrightarrow L(f \circ g) && (8)\\
L(f_{A3 \to A4}) \circ' (L(g_{A2 \to A3}) \circ' h'_{A1' \to L(A2)}) &\longrightarrow L(f \circ g) \circ' h' && (9)\\
\zeta(\varepsilon_{B'} \circ' L(f_{A \to R(B')})) &\longrightarrow f && (10)\\
\zeta(\varepsilon_{A'}) &\longrightarrow id_{R(A')} && (11)\\
\varepsilon_{B'} \circ' L(\zeta(f'_{L(A) \to B'})) &\longrightarrow f' && (12)\\
\varepsilon_{B'} \circ' (L(\zeta(f'_{L(A) \to B'})) \circ' g'_{A' \to L(A)}) &\longrightarrow f' \circ' g' && (13)\\
\zeta(f'_{L(B) \to B'}) \circ g_{A \to B} &\longrightarrow \zeta(f' \circ' L(g)) && (14)
\end{aligned}$$

Proposition 7. *ADJ is a canonical rewriting system for adjunctions.*

Remark. The rules 1–3 and 4–6 express that C and C' are categories, resp. Rules 7–9 express that L is a functor. The way of orienting these rules is important. The remaining equations are oriented consequences of the characterization of adjunctions due to lemma 3. Therefore ADJ is adequate. From rule 11 one can see that it is indeed necessary to know how R acts on objects. It remains to show that this is a canonical system, i.e. it is terminating and confluent.

Proof. First termination is shown in a similar way to the proof of the termination of system SUBST in [HL86]. The termination of the rules 1–13 can be seen from the following polynomial interpretation [Lan79]:

$$\begin{aligned}
\mathcal{I}(id) = \mathcal{I}(id') = \mathcal{I}(\varepsilon) &= 1\\
\mathcal{I}(L(f)) &= 2 * \mathcal{I}(f)\\
\mathcal{I}(\zeta(f')) &= \mathcal{I}(f') + 1\\
\mathcal{I}(f \circ g) &= \mathcal{I}(f) * \mathcal{I}(g) + \mathcal{I}(f)\\
\mathcal{I}(f' \circ' g') &= \mathcal{I}(f') * \mathcal{I}(g') + \mathcal{I}(f')
\end{aligned}$$

where the typing is not important for the interpretation.

Problem 8. How can one prove termination of all rules where rule 14 seems to be oriented in the “wrong” way?

Solution 9. The used method is as follows: Try to define a function from terms into natural numbers such that the number of applications of rule 14 is bounded by this function for all possible derivations starting from the term given.

Now we proceed with the proof by defining two functions B_R and B_ζ (R stands for rule 14, B_R will be the desired function to bound R-redexes, B_ζ bounds the potential number of ζ):

$$\begin{aligned}
B_R(id) = B_R(id') = B_R(\varepsilon) &= 0 \\
B_R(L(f)) &= B_R(f) \\
B_R(\zeta(f')) &= B_R(f') \\
B_R(f \circ g) &= B_R(f) + B_R(g) + B_\zeta(f) \\
B_R(f' \circ' g') &= B_R(f') + B_R(g') + B_\zeta(f') \\
B_\zeta(id) = B_\zeta(id') = B_\zeta(\varepsilon) &= 0 \\
B_\zeta(L(f)) &= B_\zeta(f) \\
B_\zeta(\zeta(f')) &= B_\zeta(f') + 1 \\
B_\zeta(f \circ g) &= B_\zeta(f) + B_\zeta(g) \\
B_\zeta(f' \circ' g') &= B_\zeta(f') + B_\zeta(g')
\end{aligned}$$

Using this definition one can see the following:

1. B_R and B_ζ have the subterm property ($\forall s \in \mathit{Subterm}(t) : B(s) \leq B(t)$) which is obvious from the definition.
2. B_ζ is compatible with the structure of terms ($\forall C[\_] : B_\zeta(s) \geq B_\zeta(t) \Rightarrow B_\zeta(C[s]) \geq B_\zeta(C[t])$) which follows by induction over the structure of terms for the context $C[\_]$ (but one can see that B_ζ just counts the number of ζs).
3. $s \in \mathit{Graph}(t) \Rightarrow B_\zeta(s) \leq B_\zeta(t)$ which can be seen from inspecting all rules and using 2.
4. $s \in \mathit{Graph}(t) \Rightarrow [B_R(s) \leq B_R(t) \Rightarrow B_R(C[s]) \leq B_R(C[t])]$ follows by induction over terms forming the context $C[\_]$ using 3. The interesting case is ($\circ^{(\prime)}$ stands for $\circ$ and $\circ'$):
$$B_R(s \circ^{(\prime)} u) = B_R(s) + B_R(u) + B_\zeta(s) \leq B_R(t) + B_R(u) + B_\zeta(t) = B_R(t \circ^{(\prime)} u)$$
5. $s \in \mathit{Graph}(t)$ implies $B_R(s) < B_R(t)$ if R is used, otherwise $B_R(s) \leq B_R(t)$ holds which can be checked for all rules using 4. The interesting case is:
$$\begin{aligned}
B_R(\zeta(f') \circ g) &= B_R(\zeta(f')) + B_R(g) + B_\zeta(\zeta(f')) = \\
B_R(f') + B_R(g) + B_\zeta(f') + 1 > \quad & B_R(f') + B_R(g) + B_\zeta(f') \\
&= B_R(f') + B_R(L(g)) + B_\zeta(f') \\
&= B_R(\zeta(f' \circ' L(g)))
\end{aligned}$$

This completes the proof of termination. Since termination is known the check for confluence is equivalent to the check of local confluence which can be done by checking all critical pairs cf. [KB70].

In the appendix A a trace of this check with the Larch Prover [GG91] can be found. First the termination of the first 13 rules is shown with the polynomial interpretation $\mathcal{I}$. Then the last rule is added and all rules are oriented from left to right. The last check is for local confluence.

But this check is not completely correct since it does not take account of the types of morphisms. Thus this example was implemented again in the programming language Elf where parts of a trace are included in the appendix B as well. All critical pairs had to be generated by backtracking and afterwards they had to be turned into appropriate queries by hand to check convergence. $\square$

Example 1. The system SUBST in [HL86] is connected with the adjunction $\_ \times A \dashv A \Rightarrow \_$. Here we give a more detailed description of the functors and the characterization of this adjunction. The graphical representation of this adjunctions looks as follows:

$$\frac{C \to (A \Rightarrow B)}{C \times A \to B}$$

In the case of this adjunction we have (when A is an object of the given category $\mathcal{C}$):

functors	$L := \_ \times A$	$R := A \Rightarrow \_$
objects	$B \mapsto B \times A$	$B \mapsto A \Rightarrow B$
morphisms	$F \mapsto < F \circ fst, snd >$	$F \mapsto \Lambda(F \circ app)$

In this example we have $\zeta := \Lambda$ and $\varepsilon := app$. $< \_, \_ >$ is pairing with the projections fst and snd, Λ is currying with the evaluation app.

4 Application of the Canonical System to Monads

In this section the canonical system for adjunctions will be exploited to give a decision procedure for the universal word problem for monads.

Note 10. Let C be a category. A monad $M = (T, \eta, \mu)$ on C can be characterized by the following equational specification:

$$id_B \circ f_{A \to B} = f \tag{1}$$
$$f_{A \to B} \circ id_A = f \tag{2}$$
$$(f_{A3 \to A4} \circ g_{A2 \to A3}) \circ h_{A1 \to A2} = f \circ (g \circ h) \tag{3}$$
$$T(id_A) = id_{T(A)} \tag{4}$$
$$T(f_{A2 \to A3} \circ g_{A1 \to A2}) = T(f) \circ T(g) \tag{5}$$
$$T(f_{A \to B}) \circ \eta_A = \eta_B \circ f \tag{6}$$
$$\mu_B \circ T(T(f_{A \to B})) = T(f) \circ \mu_A \tag{7}$$
$$\mu_A \circ T(\mu_A) = \mu_A \circ \mu_{T(A)} \tag{8}$$
$$\mu_A \circ T(\eta_A) = id_{T(A)} \tag{9}$$
$$\mu_A \circ \eta_{T(A)} = id_{T(A)} \tag{10}$$

Remark. The equations 1–3 express that C is a category, 4–5 describe the endofunctor T, 6 states that η is a natural transformation $I_C \to T$, 7 states that μ is a natural transformation $(T \circ T) \to T$, 8–10 are the monad laws.

Problem 11. We wish to decide the universal word problem for monads. Several attempts to orient the equations of the monad failed. The reason seems to be that T acts as an endofunctor and can be iterated since one can define:

$$T^1(f) := T(f)$$
$$T^{n+1}(f) := T(T^n(f))$$

Solution 12. One solution (might not be the only possible one) can be found with the help of the following two theorems from category theory (cf. [BW85]) which allow to translate the problem to the level of an adjunction where a canonical system was given above.

Theorem 13. *Let $D \leftarrow C : L \dashv R : D \to C$ be an adjunction with the unit $\eta : I_C \to (R \circ L)$ and counit $\epsilon : (L \circ R) \to I_D$. Then $(R \circ L, \eta, R(\epsilon_L))$ is a monad on C.*

Proof cf. [BW85]. The proof is done by verification of the monad laws. □

Theorem 14. *Let $M = (T, \eta, \mu)$ be a monad on C. Then there is a category D and an adjunction $D \leftarrow C$: $L \dashv R : D \to C$ such that:*

1. *$T = (R \circ L)$,*
2. *η is the unit of the adjunction,*

3. $R(\epsilon_L) = \mu$ where ϵ is the counit of the adjunction.

Proof cf. [BW85]. The proof is due to Kleisli with the following construction of the category D (also called "Kleisli category"):

objects	same objects as C
morphisms	$Hom_D(A, B) = Hom_C(A, T(B))$
identity of object A	η_A
composition of $f \in Hom_D(A2, A3)$ and $g \in Hom_D(A1, A2)$	$(f \circ_D g) := \mu_{A3} \circ_C T(f) \circ_C g$

This construction works due to the fact that

$$Hom_D(L(A), L(B)) \cong Hom_C(A, R(L(B))) = Hom_C(A, T(B))$$

has to hold. The functors can be defined as follows:

functor	L	R
objects	$A \mapsto A$	$A \mapsto T(A)$
morphisms	$f \in Hom_C(A, B) \mapsto T(f) \circ_C \eta_A$	$f \in Hom_D(A, B) \mapsto \mu_B \circ_C T(f)$

□

Remark. These two theorems state that adjunctions and monads are closely related. This fact can be exploited for solving the universal word problem for monads.

Corollary 15. *The universal word problem for monads is decidable.*

Proof. Let $M = (T, \eta, \mu)$ be a monad on the category C. We have the following equivalences:

1. An equation E for the monad M is expressed by the constructors $(id, \eta, \mu, T, \circ)$ for morphisms.
2. The monad $M = (T, \eta, \mu)$ is equivalent to an adjunction $L \dashv R$ with unit η and counit ϵ according to theorems 13 and 14. Thus E can be transformed into an equation with the constructors $(id, id', \eta, \epsilon, L, R, \circ, \circ')$.
3. As we saw from the equivalent representations of an adjunction this can be expressed equivalently with the constructors $(id, id', \varepsilon, L, \zeta\circ, \circ')$, where finally the canonical system can be applied.

The process of translations which preserve the validity of E is summarized in the following table:

original	first step	second step
$T(f_{A \to B})$	$R(L(f))$	$\zeta(L(f) \circ' \varepsilon_A)$
η_A	η_A	$\zeta(id'_{L(A)})$
μ_A	$R(\epsilon_L)$	$\zeta(\varepsilon_{L(A)} \circ' \varepsilon_{L(R(L(A)))})$

□

Remark. The process of this translation was implemented in the language Elf together with the rewriting system for adjunctions. An outline with some comments can be found in the appendix B.

5 Conclusions and Future Work

The uniform word problem of monads was shown to be decidable by translation into the word problem for a corresponding adjunction which is known to be decidable. This was achieved with the known correspondence between monads and adjunctions (cf. [BW85]). For future work we have two directions in mind — a theoretical and a practical.

On the theoretical side it is still not clear what can be done when adding more equations to a monad. This is the case for a "computational co-monad" (a monad in the dual category with additional equations [BG92]). We conjecture that also this notion is decidable. But the correspondence with an adjunction does not lead to an obvious solution. Furthermore also the notion of a monad with the monadic laws as used in functional programming (cf. [Wad93]) is still to be investigated. Such results could be used to reason about monadic functional programs. But this requires the treatment of higher order rewriting.

On the practical side the programming language Elf was shown to be a useful tool to implement typed rewriting. Unfortunately the current implementation does not support a procedure similar to Knuth-Bendix completion [KB70]. Even critical pairs have to be transformed by hand into a query to check convergence. Thus the incorporation of such procedures seems to be highly desirable especially taking recent results for higher order rewriting into account [Nip91].

Acknowledgements

My thanks to Jochen Pfalzgraf as my advisor for constantly encouraging me in this topic. Furthermore I especially like to thank Frank Pfenning for providing insights into his programming language Elf.

References

[AHS90] J. Adámek, H. Herrlich, and G.E. Strecker. *Abstract and Concrete Categories.* Wiley-Interscience Series in Pure and Applied Mathematics. John Wiley & Sons, Inc., 1990.

[BG92] S. Brookes and S. Geva. Computational Comonads and Intensional Semantics. In M.P. Fourman and P.T. Johnstone and A.M. Pitts, editor, *Categories in Computer Science*, number 177 in London Mathematical Society Lecture Notes, pages 1–44. Cambridge University Press, 1992.

[BW85] M. Barr and C. Wells. *Toposes, Triples and Theories.* Number 278 in Grundlehren der mathematischen Wissenschaften. Springer-Verlag, 1985.

[Cur93] P.-L. Curien. *Categorical Combinators, Sequential Algorithms, and Functional Programming.* Progress in Theoretical Computer Science. Birkhäuser, second edition, 1993.

[FS90] P.J. Freyd and A. Scedrov. *Categories, Allegories.* Elsevier Science Publishers, 1990.

[Geh94] W. Gehrke. Rewriting Techniques applied to Basic Category Theory. Technical Report 94–56, Research Institute for Symbolic Computation, Linz, Austria, 1994.

[GG91] S.J. Garland and J.V. Guttag. *A Guide to LP, The Larch Prover.* MIT, 1991.

[HL86] T. Hardin and A. Laville. Proof of Termination of the Rewriting System SUBST on CCL. *Theoretical Computer Science*, 46:306–312, 1986.

[KB70] D.E. Knuth and P.B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[Lan79] D.S. Lankford. On proving term rewriting systems are Noetherian. Technical report, Louisiana Technical University, Ruston, LA, 1979.

[ML71] S. Mac Lane. *Categories for the Working Mathematician.* Number 5 in Graduate Texts in Mathematics. Springer-Verlag, 1971.

[Mog89] E. Moggi. Computational Lambda-calculus and Monads. In *Fourth Annual Symposium on Logic in Computer Science*, pages 14–23. IEEE, June 1989.

[Nip91] T. Nipkow. Higher-Order Critical Pairs. In *Sixth Annual Symposium on Logic in Computer Science*, pages 342–349. IEEE, July 1991.

[Pfe89] F. Pfenning. Elf: a language for verified meta-programming. In *Fourth Annual Symposium on Logic in Computer Science*, pages 313–322. IEEE, June 1989.

[Pie91] B.C. Pierce. *Basic Category Theory for Computer Scientists.* Foundations of Computing. MIT Press, 1991.

[Wad93] P. Wadler. Monads for functional programming. In M. Broy, editor, *Program Design Calculi*, volume 118 of *NATO ASI Series F: Computer and System Sciences*, pages 233–264. Springer-Verlag, 1993.

A LP Traces

This is the complete trace of a session with the Larch Prover. It shows the termination of the first 13 rules for an adjunction and the local confluence of all 14 rules. Types are not taken into consideration.

```
Larch Prover (17 March 1993) logging on 4 September 1994 17:04:01 to
`/usr0/wgehrke/data/decstation/rew/adjunct/lp/adj.lplog´.
LP2: execute adj
LP2.1: declare sort M, M´
LP2.2: declare variables f, g, h: M
LP2.3: declare variables f´, g´, h´: M´
LP2.4: declare operators
        id: -> M        % identity1
        id´: -> M´      % identity2
        +: M,M -> M     % composition1
        *: M´,M´ -> M´  % composition2
        l: M -> M´      % one functor
        z: M´ -> M      % one part of iso
        ep´: -> M´      % z^-1(id1)
..
LP2.5: set ordering polynomial
The ordering-method is now `polynomial´.
LP2.6: register polynomial     id      1
LP2.7: register polynomial     id´     1
LP2.8: register polynomial     ep´     1
LP2.9: register polynomial     l       2 * x
LP2.10: register polynomial     z       x + 1
LP2.11: register polynomial     +       x * y + x
LP2.12: register polynomial     *       x * y + x
LP2.13: assert
%  category1
        id + f == f
        f + id == f
        (f + g) + h == f + (g + h)
%  category2
        id´ * f´ == f´
        f´ * id´ == f´
        (f´ * g´) * h´ == f´ * (g´ * h´)
%  functor
        l(id) == id´
        l(f) * l(g) == l(f + g)
        l(f) * (l(g) * h´) == l(f + g) * h´
%  adjunction
        z(ep´ * l(f)) == f
        z(ep´) == id
        ep´ * l(z(f´)) == f´
        ep´ * (l(z(f´)) * g´) == f´ * g´
..
Added 13 equations named user.1, ..., user.13 to the system.
The system now contains 13 rewrite rules.
LP2.14: display
Rewrite rules:
```

```
user.1:  id + f -> f
user.2:  f + id -> f
user.3:  (f + g) + h -> f + (g + h)
user.4:  id´ * f´ -> f´
user.5:  f´ * id´ -> f´
user.6:  (f´ * g´) * h´ -> f´ * (g´ * h´)
user.7:  l(id) -> id´
user.8:  l(f) * l(g) -> l(f + g)
user.9:  l(f) * (l(g) * h´) -> l(f + g) * h´
user.10: z(ep´ * l(f)) -> f
user.11: z(ep´) -> id
user.12: ep´ * l(z(f´)) -> f´
user.13: ep´ * (l(z(f´)) * g´) -> f´ * g´
LP2.15: unorder
Changed 13 rewrite rules into equations.
LP2.16: set order left-to-right
The ordering-method is now `left-to-right´.
LP2.17: assert
        z(f´) + g == z(f´ * l(g))
..
Added 1 equation named user.14 to the system.
The system now contains 14 rewrite rules.
LP2.18: complete
The system is not guaranteed to terminate.  If it does terminate, then it is
complete.
End of input from file `/usr0/wgehrke/data/decstation/rew/adjunct/lp/adj.lp´.
LP3: quit
```

B Elf Program

Some parts of the Elf program are presented to illustrate the suitability for typed rewriting. Firstly the dynamic part of the program is presented which generates critical pairs and does rewriting. The decision procedure for monads follows. No static parts of the program are shown but the definitions can be deduced from the usage in the dynamic parts. Critical pairs were generated from the query "`?-cps X Y.`" by backtracking and could be checked to converge by "`?- {...} X == Y.`" where all occurring free variables in X and Y had to be quantified by hand.

B.1 Rewriting and Critical Pairs

```
%%% rewriting for adjunctions
drule: rrule -> type.
adj1 : drule (allm [F] srule (id (B : obj adjL) * F) F).
adj2 : drule (allm [F] srule (F * id (A : obj adjL)) F).
adj3 : drule (allm [F : mor (O3 : obj adjL) (O4 : obj adjL)]
                 allm [G : mor (O2 : obj adjL) (O3 : obj adjL)]
                 allm [H : mor (O1 : obj adjL) (O2 : obj adjL)]
                 srule ((F * G) * H) (F * (G * H))).
adj4 : drule (allm [F´] srule (id (B´ : obj adjR) * F´) F´).
adj5 : drule (allm [F´] srule (F´ * id (A´ : obj adjR)) F´).
adj6 : drule (allm [F´ : mor (O3´ : obj adjR) (O4´ : obj adjR)]
```

```
                allm [G´ : mor (O2´ : obj adjR) (O3´ : obj adjR)]
                allm [H´ : mor (O1´ : obj adjR) (O2´ : obj adjR)]
                srule ((F´ * G´) * H´) (F´ * (G´ * H´))).
adj7 : drule (srule (L (id A)) (id (L´ A))).
adj8 : drule (allm [F : mor O2 O3]
                allm [G : mor O1 O2]
                srule ((L F) * (L G)) (L (F * G))).
adj9 : drule (allm [F : mor O2 O3]
                allm [G : mor O1 O2]
                allm [H´: mor O1´ (L´ O1)]
                srule ((L F) * (L G) * H´) (L (F * G) * H´)).
adj10: drule (allm [F : mor A (R´ B´)]
                srule (zeta ((eps B´)* (L F))) F).
adj11: drule (srule (zeta (eps A´)) (id (R´ A´))).
adj12: drule (allm [F´ : mor (L´ A) B´]
                srule ((eps B´) * (L (zeta F´))) F´).
adj13: drule (allm [F´ : mor (L´ A) B´]
                allm [G´ : mor A´ (L´ A)]
                srule ((eps B´) * ((L (zeta F´)) * G´))(F´ * G´)).
adj14: drule (allm [F´ : mor (L´ B) B´]
                allm [G : mor A B]
                srule ((zeta F´) * G) (zeta (F´ * L(G)))).
%%% auxiliary
%%  nonvariable subterm
nvar : mor A B -> type.
nv_i : nvar (id Z).
nv_e : nvar (eps A´).
nvZ  : nvar (zeta H).
nvL  : nvar (L H).
nvR  : nvar (R H´).
nv*  : nvar (X * Y).
%%  unifying subterm for overlapping rules
unis´: mor A B -> rrule -> mor A B -> type.
unisM: unis´ Subterm (allm Rule2) R2´
        <- unis´ Subterm (Rule2 Z) R2´.
unisU: unis´ Subterm (srule Subterm R2) R2.
unis : mor O1 O2 -> mor O1 O2 -> rrule -> mor O3 O4 -> type.
unis0: unis Subterm´ Subterm Rule2 R2´
        <- nvar Subterm´
        <- unis´ Subterm Rule2 R2´.
unis1: unis (F´ * G´) (F * G) Rule2 (R2´ * G)
        <- unis F´ F Rule2 R2´.
unis2: unis (F´ * G´) (F * G) Rule2 (F * R2´)
        <- unis G´ G Rule2 R2´.
unis3: unis (L F´) (L F) Rule2 (L R2´)
        <- unis F´ F Rule2 R2´.
unis4: unis (R F´) (R F) Rule2 (R R2´)
        <- unis F´ F Rule2 R2´.
unis5: unis (zeta F´) (zeta F) Rule2 (zeta R2´)
        <- unis F´ F Rule2 R2´.
%%% critical pairs
```

```
cp´  : rrule -> rrule -> rrule -> mor A B -> mor A B -> type.
CP1  : cp´ (allm Rule1´) (allm Rule1) Rule2 X Y
        <- {z} cp´ (Rule1´ z) (Rule1 Z) Rule2 X Y.
CP0  : cp´ (srule L1´ R1´) (srule L1 R1) Rule2 R1 R2´
        <- unis L1´ L1 Rule2 R2´.
cp   : rrule -> rrule -> mor A B -> mor A B -> type.
CP   : cp Rule1 Rule2 X Y
        <- cp´ Rule1 Rule1 Rule2 X Y.
cps  : mor A B -> mor A B -> type.
CPS  : cps X Y
        <- drule Rule1
        <- drule Rule2
        <- cp Rule1 Rule2 X Y.
%%% rewrite relation
rew  : mor A B -> mor A B -> type.
step : rew F F´´
        <- drule Rule
        <- unis F F Rule F´
        <- rew F´ F´´.
fini : rew F F.
%%  dynamic equality of morphisms
==   : mor A B -> mor A B -> type. %name == EQ
%infix none 8 ==
moreq: F == G
        <- rew F H
        <- rew G H.
```

B.2 The Decision Procedure for Monads

```
%%% decision procedure for monads
%%  translation of morphisms
%   first step
tran1: mor A B -> mor A B -> type.
t1I  : tran1 (id A) (id A).
t1E  : tran1 (eta A) (eta A).
t1M  : tran1 (mu A) (R (eps (L´ A))).
t1T  : tran1 (T F) (R (L F´))
        <- tran1 F F´.
t1*  : tran1 (F * G) (F´ * G´)
        <- tran1 F F´
        <- tran1 G G´.
t1   : tran1 F F.
%   second step
tran2: mor A B -> mor A B -> type.
t2I  : tran2 (id A) (id A).
t2eta: tran2 (eta A) (zeta (id (L´ A))).
t2eps: tran2 (eps A´) (eps A´).
t2L  : tran2 (L F) (L F´)
        <- tran2 F F´.
t2R  : tran2 (R (F´ : mor A´ B´)) (zeta (F´´ * (eps A´)))
        <- tran2 F´ F´´.
```

```
t2*  : tran2 (F * G) (F´ * G´)
        <- tran2 F F´
        <- tran2 G G´.
t2   : tran2 F F.
trans: mor A B -> mor A B -> type.
TRANS: trans F F´´
        <- tran1 F F´
        <- tran2 F´ F´´.
%%  dynamic equality for monads
===  : mor A B -> mor A B -> type. %name === EQU
%infix none 8 ===
moneq: F === G
        <- trans F F´
        <- trans G G´
        <- F´ == G´.
```

This article was processed using the LaTeX macro package with LLNCS style