Transformers for Symbolic Computation and Formal Deduction*

William M. Farmer and Martin v. Mohrenschildt

Department of Computing and Software
McMaster University
1280 Main Street West
Hamilton, Ontario L8S 4L7
Canada

wmfarmer@mcmaster.ca
mohrens@mcmaster.ca

14 June 2000

Abstract. A *transformer* is a function that maps expressions to expressions. Many transformational operators—such as expression evaluators and simplifiers, rewrite rules, rules of inference, and decision procedures—can be represented by transformers. Computations and deductions can be formed by applying sound transformers in sequence. This paper introduces machinery for defining sound transformers in the context of an axiomatic theory in a formal logic. The paper is intended to be a first step in a development of an integrated framework for symbolic computation and formal deduction.

1 Introduction

Mechanized mathematics is the study of how the computer can be used to support, improve, and automate the mathematical reasoning process. The field is divided into two quite separated camps: *computer algebra* and *theorem proving*. Computer algebra focuses on nonbranching symbolic computations over concrete structures implemented by fast, but not necessarily, sound algorithms. Theorem proving focuses on the construction of sound deductions in abstract theories expressed in formal logics. Computer algebra systems are usually restricted to just a few areas of mathematics and are often mathematically unreliable, but they provide strong support for computation and are relatively easy to use. Theorem proving systems tend to be wide in scope and mathematically rigorous, but provide little support for computation and are difficult to use. There

* Presented at the Workshop on the Role of Automated Deduction in Mathematics, CADE-17, Carnegie Mellon University, Pittsburgh, Pennsylvania, June 20–21, 2000.

is no mechanized mathematics system today that offers both the symbolic computation capabilities of computer algebra systems and the formal deduction capabilities of theorem proving systems.

Mechanized mathematics has the potential to revolutionize how mathematics is learned and practiced by students, engineers, scientists, and possibly even mathematicians. This potential can only be achieved by combining the capabilities of both computer algebra and theorem proving systems. We are developing an integrated framework for symbolic computation and formal deduction which is intended to serve as a basis for the implementation of a new kind of mechanized mathematics system.

The framework allows computation and deduction to be freely mixed together. As a result, parts of a deduction can be performed by computations. For example, an assertion that says f' is the derivative of f can be verified by directly computing f's derivative instead of inferring the assertion from axioms and previously proven theorems. As another result, a computation can be directed by a context of assumptions possibly causing the computation to branch. For example, a computation of an integral (antiderivative) of $\lambda\, x \,.\, x^n$ has two possible branches depending on whether $n = -1$ is true or false. One of the branches may be eliminated if $n = -1$ can be proved or disproved from the assumptions placed on n.

One of key ideas in our framework for integrating computation and deduction is the notion of a "transformer" in an axiomatic theory. The purpose of this paper is to define what transformers are, explain what it means for transformers to be sound, and illustrate techniques for creating sound transformers.

The paper is organized as follows. The underlying logic for the framework is briefly discussed in section 2. Transformers are introduced in section 3, computations and deductions formed by applying sound transformers in sequence are defined in section 4, and the notion of a "transformational theory" is presented in section 5. Techniques for defining new sound transformers is the subject of section 6. The papers ends with a short conclusion.

2 STMM

The underlying logic of the integrated framework is a version of von-Neumann-Bernays-Gödel (NBG) set theory [9, 10] called STMM [4, 5]. Unlike traditional set theories (such as Zermelo-Fraenkel (ZF) set theory and NBG), STMM is a well-suited foundation for mechanized mathematics. It allows terms to be undefined, has a definite description operator, provides a sort system for classifying terms by value, and includes lambda-notation with term constructors for function application and function abstraction. In short, it includes both the set-theoretic machinery of NBG set theory and the function-theoretic machinery of LUTINS [1–3], the logic of the IMPS Interactive Mathematical Proof System [6, 7].

For the purposes of this paper, the reader does not need an intimate understanding of STMM. A *language* of STMM contains two kinds of *expressions*: *terms* which may be undefined and *formulas* which denote true or false and are always defined. An *axiomatic theory* of STMM is a pair $(L, \mathit{\Gamma})$ where L is a language of STMM and $\mathit{\Gamma}$ is a set of formulas of L called the *axioms* of T. In the following, "language" and "axiomatic theory" will mean "language of STMM" and "axiomatic theory of STMM", respectively.

Terms and formulas are constructed using the usual logical connectives ($=, \neg, \wedge, \vee, \supset, \equiv, \forall, \exists$, "application", λ) plus a few special connectives ($\mathrm{I}, \mathsf{if}, \downarrow, \simeq$). For terms s and t, $s = t$ means s and t are both defined with the same value, and $s \simeq t$ means $s = t$ or s and t are both undefined. For formulas A and B, $A \supset B$ means A implies B, and $A \equiv B$ means $A \supset B$ and $B \supset A$. I is a definite description operator for forming *definite descriptions* of the form $(\mathrm{I}x : \alpha \mathrel{.} A)$, if is an if-then-else term constructor for forming *conditional terms* of the form $\mathsf{if}(A, s, t)$, and $\downarrow$ is a definedness operator for forming *definedness assertions* of the form $s\downarrow$. For expressions E and E', $E \overset{\alpha}{=} E'$ means E and E' are alpha-equivalent, i.e., they are identical up to a renaming of bound variables. $\overset{\alpha}{=}$ often serves as syntactic identity.

Although we have chosen a highly expressive logic with special machinery for reasoning with undefinedness and functions, the ideas given in the paper will work in less sophisticated logics.

3 Transformers

The "transformer" is the key notion in our framework. It has two meanings. Its *algorithmic* meaning is a function on expressions, and its *axiomatic* meaning is a set of expressions. Many transformational operations—such as expression evaluators and simplifiers, rewrite rules, rules of inference, and decision procedures—can be represented by transformers.

For the rest of the paper, let us fix an axiomatic theory $T = (L, \mathit{\Gamma})$. A *term transformer* of L is a total function on the terms of L, and a *formula transformer* of L is a total function on the formulas of L. Let τ

be a term or formula transformer. τ *moves* a member E of its domain if E is not identical to $\tau(E)$. In the following, let a "transformer" mean a "transformer of L", an "expression" mean a "expression of L", etc.

A transformer can be "sound" in different ways. A term transformer τ is *computationally sound* in T if, for all terms s, $T \models s \simeq \tau(s)$. A formula transformer τ is *computationally sound* in T if, for all formulas A, $T \models A \equiv \tau(A)$. A formula transformer τ is *deductively sound* in T if, for all formulas A, $T \models A \supset \tau(A)$. A formula transformer τ is *reductively sound* in T if, for all formulas A, $T \models \tau(A) \supset A$.

Example 1. Let $T = (L, \Gamma)$ be an axiomatic theory which formalizes real arithmetic in a standard way, and let τ be the term transformer that is defined by the following rules:

1. If $t \overset{\alpha}{=} s/s$, where s is a term of sort $\mathbf{R}$ (the sort of the real numbers), then
$$\tau(t) := \mathsf{if}(s \neq 0, 1, \bot_{\mathbf{R}})$$
where $\bot_{\mathbf{R}}$ is a canonical undefined term of sort $\mathbf{R}$.
2. Otherwise, $\tau(t) := t$.

τ is computationally sound in T since
$$T \models \forall x : \mathbf{R} \,.\, x/x \simeq \mathsf{if}(x \neq 0, 1, \bot_{\mathbf{R}}).$$

It is a sound formalization of an unsound simplification procedure commonly used in computer algebra systems.

4 Computations and Deductions

Let E be an expression and τ be a (term or formula) transformer. Each subexpression occurrence in E can be uniquely represented by a sequence of positive integers called the *path* of the occurrence in E. One subexpression occurrence in E *contains* another subexpression occurrence in E if the path of the former is an initial segment of the path of the latter. Two subexpression occurrences in E are *disjoint* if neither one contains the other.

A *target* of τ in E is an occurrence of a subexpression in E which is moved by τ. A target is *maximal* if its path in E is not an initial segment of the path of another target of τ in E. A set of maximal targets of τ in E is obviously pairwise disjoint. Given expressions E_1 and E_2,
$$E_1 \xrightarrow{\tau} E_2$$

means there is a pairwise disjoint set S of targets of τ in E_1 such that E_2 is obtained from E_1 by simultaneously replacing each target E in S with $\tau(E)$.

$$E_1 \xrightarrow{\max,\tau} E_2$$

means E_2 is obtained from E_1 by simultaneously replacing each maximal target E of τ in E_1 with $\tau(E)$. Of course,

$$E_1 \xrightarrow{\max,\tau} E_2 \text{ implies } E_1 \xrightarrow{\tau} E_2.$$

A *computation* of L from an expression E_0 to an expression E_n using a set Π of transformers of L is a sequence

$$\langle E_0, \tau_1, E_1, \ldots, \tau_n, E_n \rangle$$

such that:

1. $n \geq 1$.
2. $\tau_i \in \Pi$ for all i with $1 \leq i \leq n$.
3. $E_{i-1} \xrightarrow{\tau_i} E_i$ for all i with $1 \leq i \leq n$.

A *deduction* is a computation $\langle E_0, \tau_1, E_1, \ldots, \tau_n, E_n \rangle$ where each E_i is a formula (but each τ_i can be either a term or formula transformer).

A computation $\langle E_0, \tau_1, E_1, \ldots, \tau_n, E_n \rangle$ of L is *computationally sound* in T if each τ_i is computationally sound in T.

A deduction $\langle A_0, \tau_1, A_1, \ldots, \tau_n, A_n \rangle$ is *deductively sound* in T provided, for all i with $1 \leq i \leq n$, if τ_i is a term transformer, then τ_i is computationally sound in T and if τ_i is a formula transformer, then one of the following statements is true:

1. τ_i is computationally sound in T.
2. τ_i is deductively sound in T and applied only to positive[1] subformula occurrences in A_{i-1}.
3. τ_i is reductively sound in T and applied only to negative subformula occurrences in A_{i-1}.

A deduction $\langle A_0, \tau_1, A_1, \ldots, \tau_n, A_n \rangle$ is *reductively sound* in T provided, for all i with $1 \leq i \leq n$, if τ_i is a term transformer, then τ_i is computationally sound in T and if τ_i is a formula transformer, then one of the following statements is true:

[1] A subformula occurrence of A in a formula B is *positive* [*negative*] if the occurrence is in the scope of an even [odd] number of occurrences of $\neg$ in B and the occurrence in not in a definite description or a conditional term.

1. τ_i is computationally sound in T.
2. τ_i is reductively sound in T and applied only to positive subformula occurrences in A_{i-1}.
3. τ_i is deductively sound in T and applied only to negative subformula occurrences in A_{i-1}.

Proposition 1. *Let $T = (L, \Gamma)$ be an axiomatic theory and*

$$C = \langle E_0, \tau_1, E_1, \ldots, \tau_n, E_n \rangle$$

be a computation of L.

1. *If C is a term computation which is computationally sound in T, then $T \models E_0 \simeq E_n$.*
2. *If C is a formula computation which is computationally sound in T, then $T \models E_0 \equiv E_n$.*
3. *If C is a deduction which is deductively sound in T, then $T \models E_0 \supset E_n$.*
4. *If C is a deduction which is reductively sound in T, then $T \models E_n \supset E_0$.*

A *forward proof* of a formula A in T is a deduction from true[2] to A which is deductively sound in T. A *backward proof* of a formula A in T is a deduction from A to true which is reductively sound in T.

Example 2. There are simple transformers for introducing and eliminating theorems (and axioms) into and from deductions. Suppose $T = (L, \Gamma)$ is an axiomatic theory and A is a theorem of T (i.e., $T \models A$). Let τ_1 be the "elimination" formula transformer that is defined by the following rules:

1. If $B \overset{\alpha}{=} A$, then $\tau_1(B) := \mathsf{true}$.
2. Otherwise, $\tau_1(B) := B$.

Let τ_2 be the "introduction" formula transformer defined by the rule $\tau_2(B) := B \wedge A$. τ_1 and τ_2 are both computationally sound in T.

5 Transformational Theories

A *transformational theory* of STMM is a tuple

$$U = (L, \Gamma, \Pi_{\mathsf{ct}}, \Pi_{\mathsf{cf}}, \Pi_{\mathsf{df}}, \Pi_{\mathsf{rf}})$$

where:

[2] true is a canonical true formula of L.

1. L is a language.
2. $\mathit{\Gamma}$ is a set of formulas of L called the *explicit axioms* of U.
3. $\mathit{\Pi}_{\mathsf{ct}}$ is a set of term transformers of L called the *primitive computational term transformers* of U.
4. $\mathit{\Pi}_{\mathsf{cf}}$ is a set of formula transformers of L called the *primitive computational formula transformers* of U.
5. $\mathit{\Pi}_{\mathsf{df}}$ is a set of formula transformers of L called the *primitive deductive formula transformers* of U.
6. $\mathit{\Pi}_{\mathsf{rf}}$ is a set of formula transformers of L called the *primitive reductive formula transformers* of U.

The *axiomatic theory* of U, written $\mathsf{ax\text{-}thy}(U)$, is the axiomatic theory

$$(L, \mathit{\Gamma} \cup \mathit{\Gamma}_{\mathsf{ct}} \cup \mathit{\Gamma}_{\mathsf{cf}} \cup \mathit{\Gamma}_{\mathsf{df}} \cup \mathit{\Gamma}_{\mathsf{rf}})$$

where:

1. $\mathit{\Gamma}_{\mathsf{ct}} = \{s \simeq \tau(s) : \tau \in \mathit{\Pi}_{\mathsf{ct}}$ and s is a term of $L\}$.
2. $\mathit{\Gamma}_{\mathsf{cf}} = \{A \equiv \tau(A) : \tau \in \mathit{\Pi}_{\mathsf{cf}}$ and A is a formula of $L\}$.
3. $\mathit{\Gamma}_{\mathsf{df}} = \{A \supset \tau(A) : \tau \in \mathit{\Pi}_{\mathsf{df}}$ and A is a formula of $L\}$.
4. $\mathit{\Gamma}_{\mathsf{rf}} = \{\tau(A) \supset A : \tau \in \mathit{\Pi}_{\mathsf{rf}}$ and A is a formula of $L\}$.

A formula A of L is an *axiom* or *theorem* of U if A is an axiom or theorem, respectively, of $\mathsf{ax\text{-}thy}(U)$. $U \models A$ means $\mathsf{ax\text{-}thy}(U) \models A$.

Proposition 2. *Let $U = (L, \mathit{\Gamma}, \mathit{\Pi}_{\mathsf{ct}}, \mathit{\Pi}_{\mathsf{cf}}, \mathit{\Pi}_{\mathsf{df}}, \mathit{\Pi}_{\mathsf{rf}})$ be a transformational theory and $T = \mathsf{ax\text{-}thy}(U)$. Then each $\tau \in \mathit{\Pi}_{\mathsf{ct}} \cup \mathit{\Pi}_{\mathsf{cf}}$ is computationally sound in T, each $\tau \in \mathit{\Pi}_{\mathsf{df}}$ is deductively sound in T, and each $\tau \in \mathit{\Pi}_{\mathsf{rf}}$ is reductively sound in T.*

6 Defining New Transformers

An axiomatic theory is "developed" by defining new constants and proving theorems. A transformational theory is developed by defining new constants, proving theorems, and *defining new transformers and proving their soundness.* There are three ways of defining sound transformers. First, sound transformers can be generated automatically from theorems in several ways depending on the syntactic form of the theorem. Second, sound transformers can be constructed from other sound (and in some cases possibly unsound) transformers using constructors that always produce sound transformers. Third, a transformer can be manually defined and then manually proven to be sound.

These ways of defining transformers are illustrated with examples inspired by the *macete mechanism* of IMPS (see [7, 8].)

6.1 Generating Transformers from Theorems

There are several ways that sound transformers can be automatically generated from theorems. We will give two representative examples.

Example 3 (Implication Transformers). Suppose

$$T \models \forall\, x^1_{\alpha_1}, \ldots, x^n_{\alpha_n} \,.\, A' \supset A''.$$

Let τ_1 be the formula transformer of L defined by the following rules:

1. If $B \overset{\alpha}{=} A'\sigma$ where σ is a substitution with domain $\{x^1_{\alpha_1}, \ldots, x^n_{\alpha_n}\}$, then $\tau_1(B) := A''\sigma$.
2. Otherwise, $\tau_1(B) := B$.

Let τ_2 be the formula transformer of L defined by the following rules:

1. If $B \overset{\alpha}{=} A''\sigma$ where σ is a substitution with domain $\{x^1_{\alpha_1}, \ldots, x^n_{\alpha_n}\}$, then $\tau_2(B) := A'\sigma$.
2. Otherwise, $\tau_2(B) := B$.

τ_1 is deductively sound in T, and τ_2 is a reductively sound in T. τ_1 is a forward chaining rule of inference, while τ_2 is a backward chaining rule of inference.

Example 4 (Term Conditional Rewrite Transformers). Suppose

$$T \models \forall\, x^1_{\alpha_1}, \ldots, x^n_{\alpha_n} \,.\, A \supset s' \simeq s''.$$

Let τ be the term transformer of L defined by the following rules:

1. If $t \overset{\alpha}{=} s'\sigma$ where σ is a substitution with domain $\{x^1_{\alpha_1}, \ldots, x^n_{\alpha_n}\}$, then $\tau(t) := \mathsf{if}(A\sigma, s''\sigma, t)$.
2. Otherwise, $\tau(t) := t$.

τ is a forward conditional rewrite rule which is computationally sound in T. The corresponding backward conditional rewrite rule can be defined in a similar way.

6.2 Constructing New Transformers

Sound term or formula transformers can be constructed using the following *transformer constructors*:

1. **Lift** The *term lift* of a transformer τ is the term transformer $\mathsf{t\text{-}lift}(\tau)$ defined by the rule $\mathsf{t\text{-}lift}(\tau)(s) := t$ iff $s \overset{\max,\tau}{\longrightarrow} t$. The *formula lift* of a transformer τ is the formula transformer $\mathsf{f\text{-}lift}(\tau)$ defined in a similar way to $\mathsf{t\text{-}lift}(\tau)$.
2. **Composition** The *composition* of two term or formula transformers τ_1 and τ_2 is the transformer $\tau_1 \circ \tau_2$ defined by the rule $(\tau_1 \circ \tau_2)(E) := \tau_1(\tau_2(E))$. For a transformer τ, τ^n is the transformer $\tau \circ \cdots \circ \tau$ (n times) where $n \geq 1$.
3. **Fixpoint** The *fixpoint* of a term or formula transformer τ is the transformer $\mu\tau$ defined by the rule $\mu\tau(E) := E'$ where $E' \overset{\alpha}{=} \tau^n(E)$ and $E' \overset{\alpha}{=} \tau(E')$ for some $n \geq 1$. ($\mu\tau$ is the identity (term or formula) transformer if, for some expression E, $\tau^n(E)$ is not alpha-equivalent to $\tau^{n+1}(E)$ for all $n \geq 1$.)
4. **Conditional** Let τ_1 and τ_2 be term transformers and A be a formula. The *conditional* of τ_1 and τ_2 with respect to A is the transformer $\mathsf{cond}(A, \tau_1, \tau_2)$ defined by the rule $\mathsf{cond}(A, \tau_1, \tau_2)(s) := \mathsf{if}(A, \tau_1(s), \tau_2(s))$.
5. **Sounder** Let τ, τ_1 and τ_2 be term or formula transformers. The *sounder* of τ with respect to τ_1 and τ_2 is the transformer $\mathsf{sound}(\tau, \tau_1, \tau_2)$ defined by the following rules:
   (a) If $(\tau_1 \circ \tau)(E) \overset{\alpha}{=} \tau_2(E)$, then $\mathsf{sound}(\tau, \tau_1, \tau_2)(E) := \tau(E)$.
   (b) Otherwise, $\mathsf{sound}(\tau, \tau_1, \tau_2)(E) := E$.

Proposition 3. *Let $T = (L, \Gamma)$ be an axiomatic theory, τ, τ_1 and τ_2 be term or formula transformers of L, and A be a formula of L.*

1. *If τ is computationally sound in T, then* $\mathsf{t\text{-}lift}(\tau)$ *and* $\mathsf{f\text{-}lift}(\tau)$ *are computationally sound in T.*
2. *If τ_1 and τ_2 are computationally, deductively, or reductively sound in T, then $\tau_1 \circ \tau_2$, $\mu\tau_1$, and* $\mathsf{cond}(A, \tau_1, \tau_2)$*, are computationally, deductively, or reductively sound in T, respectively.*
3. *If τ_1 and τ_2 are computationally sound in T, then* $\mathsf{sound}(\tau, \tau_1, \tau_2)$ *is computationally sound in T.*

6.3 Manually Defining Transformers

A transformer can be defined by a program (written in some a programming language) that takes an expression as input and returns an expression as output. It is the responsibility of whoever uses the transformer either to verify that the transformer is sound or to only use the transformer in a sound way. The former may require proving that the program is sound. The latter can be done by employing the sound transformer constructor given above. We illustrate this with the following example:

Example 5. Let τ be a term transformer defined by a program that computes an integral (antiderivative) of a function, τ_1 be a term transformer that maps a function to its derivative, and τ_2 be a transformer that maps a term to its canonical representation. Assume τ_1 and τ_2 are known to be computationally sound, but τ may be unsound for some inputs. Since integration and derivative are inverses of each other, the sounder $\mathsf{sound}(\tau, \tau_2 \circ \tau_1, \tau_2)$ will apply τ only when it is sound. This illustrates how a simpler transformer (performing differentiation) is used to check a more complex transformer (performing integration).

7 Conclusion

We have introduced the notion of a transformer, shown how computations and deductions can be formed by applying sound transformers in sequence, demonstrated how transformers can be used to define axiomatic theories, and presented machinery for generating sound transformers from theorems and defining new sound transformers from old sound transformers. The paper is intended to be a first step in the development of an integrated framework for symbolic computation and formal deduction.

In the second step in the development of the framework, we will introduce branching computations. A branch can potentially be introduced in a computation whenever a conditional term appears in the expression being transformed. Branching is controlled with the use *contexts*, which are sets of formulas that serve as local assumptions [11]. Branching computations are more convenient for both humans and software to work with than linear, nonbranching computations.

References

1. W. M. Farmer. A partial functions version of Church's simple theory of types. *Journal of Symbolic Logic*, 55:1269–91, 1990.
2. W. M. Farmer. A simple type theory with partial functions and subtypes. *Annals of Pure and Applied Logic*, 64:211–240, 1993.
3. W. M. Farmer. Theory interpretation in simple type theory. In J. Heering et al., editor, *Higher-Order Algebra, Logic, and Term Rewriting*, volume 816 of *Lecture Notes in Computer Science*, pages 96–123. Springer-Verlag, 1994.
4. W. M. Farmer. A proposal for the development of an interactive mathematics laboratory for mathematics education. In E. Melis, editor, *CADE-17 Workshop on Deduction Systems for Mathematics Education*, pages 20–25, 2000.

5. W. M. Farmer and J. D. Guttman. A set theory with support for partial functions. *Studia Logica*, 66:59–78, 2000.
6. W. M. Farmer, J. D. Guttman, and F. J. Thayer Fábrega. IMPS: An updated system description. In M. McRobbie and J. Slaney, editors, *Automated Deduction—CADE-13*, volume 1104 of *Lecture Notes in Computer Science*, pages 298–302. Springer-Verlag, 1996.
7. W. M. Farmer, J. D. Guttman, and F. J. Thayer. IMPS: An Interactive Mathematical Proof System. *Journal of Automated Reasoning*, 11:213–248, 1993.
8. W. M. Farmer, J. D. Guttman, and F. J. Thayer. Contexts in mathematical reasoning and computation. *Journal of Symbolic Computation*, 19:201–216, 1995.
9. K. Gödel. *The Consistency of the Axiom of Choice and the Generalized Continuum Hypothesis with the Axioms of Set Theory*, volume 3 of *Annals of Mathematical Studies*. Princeton University Press, 1940.
10. E. Mendelson. *Introduction to Mathematical Logic*. Van Nostrand, 1964.
11. L. G. Monk. Inference rules using local contexts. *Journal of Automated Reasoning*, 4:445–462, 1988.