J. Symbolic Computation (1994) **17**, 393–408

Injectivity of Composite Functions

KIM S. LARSEN[†] AND MICHAEL I. SCHWARTZBACH

Computer Science Department, Aarhus University, Ny Munkegade, 8000 Aarhus C, Denmark

(Received 27 May 1994)

The problem of deciding injectivity of functions is addressed. The functions under consideration are compositions of more basic functions for which information about injectivity properties is available. We present an algorithm which will often be able to prove that such a composite function is injective. This algorithm constructs a set of propositional Horn clause axioms from the function specification and the available information about the basic functions. The existence of a proof of injectivity is then reduced to the problem of propositional Horn clause deduction. Dowling and Gallier have designed several very fast algorithms for this problem, the efficiency of which our algorithm inherits. The proof of correctness of the algorithm amounts to showing soundness and completeness of the generated Horn clause axioms.

1. Introduction

Many fundamental mathematical concepts have significant impact on computer science, and the concept of injectivity is no exception. For a computer scientist, the most natural question to ask in connection with this concept is: "Given a description of a function f, is f injective"? It is well known that this problem is undecidable in general, and this fact might be what has kept researchers from further explorations. However, just as some NP-complete problems have extremely fast approximations algorithms, there might exist natural classes of functions for which injectivity can be decided.

This paper is a first step in that direction. Our starting point is a collection of basic or built-in functions, each of which has some information attached, reflecting its properties of injectivity. We present an algorithm which takes as input a term f composed of the basic functions. There are two possible outputs from the algorithm: "proved injective" and "no proof". If the answer is "no proof", then the composite function f could still be injective—our algorithm just failed to provide a proof. However, our algorithm is complete in the sense that if the answer is "no proof", then there exists a non-injective function to which the same information could have been attached. Thus, we have exhausted the potential of our approach.

Our algorithm works as follows. From the term f and the information attached to function symbols, a set of propositional Horn clause axioms is generated. The result of evaluating one query on these axioms gives us the answer to whether a proof of injectivity

[†] Present address: Department of Mathematics and Computer Science, Odense University, Campusvej 55, 5230 Odense M, Denmark.

0747–7171/94/050393 + 16 $08.00/0

can be found or not, given the information available. The size of the axioms will be proportional to the size of the function description, $|f|$. Dowling & Gallier (1984) have designed a linear-time algorithm for evaluating propositional Horn clause queries, so via this reduction, we obtain an $O(|f|)$ algorithm.

There are many possible applications for an "injectivity tester". In the following, we discuss a few of these.

In mathematical tools like MATHEMATICA Wolfram (1988) and in functional programming languages like MIRANDA Turner (1985), ML Milner *et al.* (1990), or SCHEME Abelson *et al.* (1989), sets are of great importance. Either because *set* is a type in the language, or because users tend to write applications involving sets. In either case, lists will probably be used to implement the set type. So, sets are lists with the invariant that there are no duplicate elements.

Consider the functional program fragment *map* f l, where f is a function and l is a set (implemented as a list). This is a frequently occurring construction in functional languages with the interpretation that if l is the list $[a_1, \ldots, a_n]$, then *map* f l is $[f(a_1), \ldots, f(a_n)]$. Obviously, if f is not injective, then the list $[f(a_1), \ldots, f(a_n)]$ might contain duplicates. In order to maintain the set invariant, the list will then have to be sorted to check for duplicates and to remove these if necessary.

This is where our "injectivity tester" comes in. If it can be determined that f is injective, then this sorting can be avoided, and the complexity would be brought down from $O(n \log n)$ to $O(n)$. Our algorithm could be a part of the implementation of languages supporting a set type. For other languages, it could be a library routine, which could be used to improve efficiency of user-defined set manipulations.

In languages supporting lazy evaluation, lists can be (potentially) infinite. This implies that the set invariant cannot be maintained as described above by first computing the whole list and then sorting to remove duplicates. Instead, insertion sort has to be used. Dynamically, the value $f(a_i)$ has to be compared with the set of values $\{f(a_1), \ldots, f(a_{i-1})\}$, and it should only be output if it has not already appeared. Under these circumstances, knowing that f is injective would reduce the complexity dramatically from $O(n^2)$ to $O(n)$, where n is the length of the prefix of the infinite list which is printed.

There are also possible applications in the area of databases. In the relational model, a relation is a set of tuples, but again, the implementation is often in the form of lists without duplicate tuples. In the relational model, a few unary operators are singled out, but in real database query languages, more powerful mechanisms for constructing complex unary queries are usually supplied. Clearly, the prospect of an "injectivity tester" is once again to avoid sorting. This is particularly interesting in connection with relations since these are often huge, and must be stored on secondary memory.

2. The Framework

This section contains a precise statement of the problem.

2.1. AXIOMS FOR INJECTIVITY

Properties about the injectivity of the basic functions must be provided. A full equational theory would be an obvious choice, but injectivity is not decidable in this framework. Also, a particular function could be very easy to implement and yet very difficult to axiomatize. Furthermore, such specifications need not be very modular; if we introduce a new function, then we may have to relate it to all the existing ones. We would prefer to specify information for each function purely locally.

A different and natural approach addresses all the concerns. The most simplistic solution would be to attach one bit of information to each function, stating whether it was injective in all its arguments. In this case, a composite function would be injective if it was composed of only injective basic functions. We can refine this technique considerably. For each k-ary basic function we specify a set of propositional Horn clause axioms, the propositions of which are among the integers $\{1, \dots, k\}$. Clauses such as

$$1 \leftarrow 2 \qquad \text{and} \qquad 2 \leftarrow 1$$

state that the resulting value, together with the second argument, completely determines the first argument, and vice versa. Both are true of the $+$ operation on integers, for example: if we know that $x + y = 87$ and $y = 40$, then we can uniquely determine x (as $x = 47$). The *append* function satisfies the same axioms. Such specifications are local to the function and they are easy to construct and verify. Another example is the *cons* function which satisfies the axioms $1 \leftarrow$ and $2 \leftarrow$, since it is in fact injective in both arguments. The *sublist* function, taking a list and two integer indices, satisfies the axioms $3 \leftarrow 1, 2$ and $2 \leftarrow 1, 3$. All functions satisfy trivial axioms such as $i \leftarrow i$.

Note that compositions of non-injective functions may be injective. An example is the function defined by

$$(a, b, x) \mapsto (\mathit{append}(a, b), \mathit{reverse}(b), x + \mathit{length}(a))$$

which is injective even though the functions *append*, *length*, and $+$ are not. The argument is as follows. Assume that the value of $(\mathit{append}(a, b), \mathit{reverse}(b), x + \mathit{length}(a))$ is known. We prove that then the values of a, b, and x can be determined. As $\mathit{reverse}(b)$ is injective, we can determine b. Let us denote this by $b \leftarrow \mathit{reverse}(b)$. From b and $\mathit{append}(a, b)$, we can determine a, i.e., $a \leftarrow b, \mathit{append}(a, b)$. Finally, $\mathit{length}(a) \leftarrow a$ and therefore, $x \leftarrow \mathit{length}(a), x + \mathit{length}(a)$.

As another example consider

$$(x, y) \mapsto (2(x + y), (x + y)/x)$$

Here, $x + y \leftarrow 2(x + y)$ which gives us x as $x \leftarrow x + y, (x + y)/x$. Finally, $y \leftarrow x, x + y$.

2.2. (Σ, Δ)-ALGEBRAS

Working in a homogeneous algebraic framework, we can formalize these ideas. Standard definitions can be found in great detail in Wirsing (1989), for example. However, our application is slightly non-standard. We use a fixed set of *variables* $X = \{x_1, \dots, x_n\}$.

DEFINITION 2.1. *A* signature *is a ranked set*

$$\Sigma = \bigcup_{k \in \mathbb{N}} \Sigma_k$$

where Σ_k contains the functions of arity k, and $X \subseteq \Sigma_0$.

DEFINITION 2.2. *The* terms *over the signature* Σ, $\mathrm{Term}(\Sigma)$*, are defined inductively to be the least set such that*

$$\sigma \in \Sigma_0 \Rightarrow \sigma \in \mathrm{Term}(\Sigma)$$

$$\begin{array}{l} \sigma \in \Sigma_k,\ k > 0,\ \textit{and}\, t_1, \ldots, t_k \in \mathrm{Term}(\Sigma) \\ \Downarrow \\ \sigma(t_1, \ldots, t_k) \in \mathrm{Term}(\Sigma) \end{array}$$

Henceforth, a term is an element of Term(Σ).

Such terms will be used to denote function expressions.

Definition 2.3. *A* function clause *is of the form* $\lfloor d_0 \leftarrow d_1, \ldots, d_m \rfloor$*, where* $m \in I\!\!N$ *and* $d_0, d_1, \ldots, d_m \in I\!\!N \setminus \{0\}$.

Function clauses will be attached to functions to specify their properties of injectivity. If σ is a function of arity k, then the clause $\lfloor d_0 \leftarrow d_1, \ldots, d_m \rfloor$ should be interpreted: "from the result of an application of σ to k arguments *and* the d_1th to the d_mth argument we can uniquely determine the d_0th argument". (The floor symbols $\lfloor$ and $\rfloor$ are only used as delimiters).

Definition 2.4. *A pair* (Σ, Δ) *is a* specification *if* Σ *is a signature and* Δ *maps function symbols to sets of function clauses such that*

$$\sigma \in \Sigma_k \ \wedge\ \lfloor d_0 \leftarrow d_1, \ldots, d_m \rfloor \in \Delta(\sigma) \ \Rightarrow\ \{d_0, d_1, \ldots, d_m\} \subseteq \{1, \ldots, k\}$$

We use a fixed specification (Σ, Δ) *throughout the paper.*

Definition 2.5. *If* D *is a set, then* $\bar{v}$ *denotes the tuple* $\langle v_1, \ldots, v_p \rangle$*, where* $v_1, \ldots, v_p \in D$*. If* $i \in \{1, \ldots, p\}$ *then* $\bar{v}.i$ *denotes the value* v_i*. If* $p = n$ *(the cardinality of the fixed set of variables* X*) and* $a \in X$*, then* $a = x_i$ *for some* $x_i \in X$ *and we let* $\bar{v}.a$ *denote* v_i*.*

Our models will be algebras over (Σ, Δ). Such algebras should, of course, be faithful to the information in (Σ, Δ), i.e., for each function symbol in Σ, we will have a function in our algebra with the correct arity (as specified by Σ) such that the properties of injectivity promised in Δ are actually fulfilled.

Definition 2.6. M *is a* (Σ, Δ)*-algebra if it provides a carrier domain* dom_M *and for each* $\sigma \in \Sigma_k \setminus X$ *a function* σ^M *such that*

$$\sigma^M\colon\ dom_M^k \rightarrow dom_M$$

for each $\lfloor d_0 \leftarrow d_1, \ldots, d_m \rfloor \in \Delta(\sigma)$ *we have for all* $\bar{v}, \bar{w} \in dom_M^k$:

$$\begin{array}{c} \langle \bar{v}.d_1, \ldots, \bar{v}.d_m, \sigma^M(\bar{v}) \rangle \ =\ \langle \bar{w}.d_1, \ldots, \bar{w}.d_m, \sigma^M(\bar{w}) \rangle \\ \Downarrow \\ \bar{v}.d_0 \ =\ \bar{w}.d_0 \end{array}$$

Let Alg(Σ, Δ) *be the set of all* (Σ, Δ)*-algebras. As the specification* (Σ, Δ) *is fixed, an* algebra*, henceforth, denotes a member of* Alg(Σ, Δ).

A term t can in a given algebra be interpreted as a function of arity n.

Definition 2.7. *Let M be an algebra and t a term. The function $t^M: dom_M^n \to dom_M$ is obtained from t as follows:*

if $t = x_i \in X$, then $t^M(\bar{v}) = \bar{v}.i$
if $t \in \mathrm{Term}(\Sigma) \backslash X$, then $t = \sigma(t_1, \dots, t_k)$ for some $\sigma \in \Sigma_k$ and

$$t^M(\bar{v}) = \sigma^M(t_1^M(\bar{v}), \dots, t_k^M(\bar{v}))$$

This simply interprets t as a function of the n variables x_i.

2.3. the decision problem

We are interested in injectivity in the usual mathematical sense. That is, when is a term, interpreted as a function in an algebra, injective?

Definition 2.8. *Let M be an algebra. A term t is called* semantically injective w.r.t. M *if $t^M: dom_M^n \to dom_M$ is an injective function in the usual mathematical sense, i.e.,*

$$\forall \bar{v}, \bar{w} \in dom_M^n: \; \bar{v} \neq \bar{w} \Rightarrow t^M(\bar{v}) \neq t^M(\bar{w})$$

We use $Sem(t)$ to denote that t is semantically injective w.r.t. any algebra.

We shall construct an algorithm to decide $\mathrm{Sem}(t)$. If this property holds, then the answer is "proved injective"; this is safe, since we are in an algebra, and t is injective in *all* algebras. Otherwise, the answer is "no proof"; this is the best we can do, since all we know is that in *some* algebra, t is not injective.

3. The Algorithm

It is not at all obvious how to decide a "semantic" property like $\mathrm{Sem}(t)$. Therefore, we find a more "syntactic" property to check instead. Of course, we then have to show that these two properties are equivalent.

As seen from definition 2.2, terms are built from variables and constants using function symbols to combine terms. We want to obtain complete information, as a set of Horn clause axioms, as to which variable can be retrieved from a term. That is, if we know the value of $t^M(\bar{v})$, for some algebra M and values $\bar{v}$, which v_i's can we then retrieve? If all v_i's can be retrieved, then t^M has a *left inverse* and, hence, is injective.

Definition 3.1. *If t is a term, then $Sub(t)$ is the set of subterms of t defined recursively by*

$$Sub(t) = \{t\} \cup \bigcup_i Sub(t_i)$$

where $t = \sigma(t_1, \dots, t_k)$.

Definition 3.2. *Let t be a term. Then a t-clause is of the form*

$$\lfloor s_0 \leftarrow s_1, \dots, s_k \rfloor$$

where $s_i \in Sub(t)$.

Definition 3.3. *The* denotation $[\![t]\!]$ *of a term* t *is the least finite set of* t*-clauses containing*

The main clause $\lfloor t \leftarrow \rfloor$.
The clauses in $\tilde{\Delta}$*, defined by*

$$\bigcup_{s \in \mathrm{Sub}(t)} \bigcup_{\delta \in \Delta(\sigma)} \{\lfloor s_{d_0} \leftarrow s_{d_1}, \ldots, s_{d_m}, s \rfloor \mid s = \sigma(s_1, \ldots, s_k), \delta = \lfloor d_0 \leftarrow d_1, \ldots, d_m \rfloor\}$$

For each $\sigma \in \Sigma_k \backslash X$ *and* $\sigma(s_1, \ldots, s_k) \in Sub(t)$*, a functionality clause*

$$\lfloor \sigma(s_1, \ldots, s_k) \leftarrow s_1, \ldots, s_k \rfloor$$

As we shall later see, the denotation contains all *the information pertinent to* t.

The desired syntactic property can now be expressed by deductions in the denotation.

Definition 3.4. *A term* s *can be* deduced *in* $[\![t]\!]$*, written* $[\![t]\!] \vdash s$*, if* $\lfloor s \leftarrow s_1, \ldots, s_k \rfloor \in [\![t]\!]$ *and* $\forall i \in \{1, \ldots, k\}: [\![t]\!] \vdash s_i$*. Deductions can conveniently be represented as* proofs *of the form*

$$\cfrac{\cfrac{\vdots}{s_1} \cdots \cfrac{\vdots}{s_k}}{s}$$

i.e., finite trees where each line represents an application of a clause.

Definition 3.5. *A term* t *is* syntactically injective*, written* $Syn(t)$*, if*

$$\forall x_i \in X: [\![t]\!] \vdash x_i$$

As we shall see, this property exactly captures injectivity.

Taking this alternative definition on faith, we can present an algorithm.

We first compute the denotation. It is a set of Horn clause axioms with subterms as propositions. Continuing the example from section 2.1, we list the denotation for

$$exp = (append(a, b), reverse(b), x + length(a))$$

$[\![exp]\!]$ is:

$exp \leftarrow$	main clause
$append(a, b) \leftarrow exp$	property of $(\cdot, \cdot, \cdot)$
$reverse(b) \leftarrow exp$	property of $(\cdot, \cdot, \cdot)$
$x + length(a) \leftarrow exp$	property of $(\cdot, \cdot, \cdot)$
$a \leftarrow b, append(a, b)$	property of *append*
$b \leftarrow a, append(a, b)$	property of *append*
$append(a, b) \leftarrow a, b$	functionality
$b \leftarrow reverse(b)$	property of *reverse*
$reverse(b) \leftarrow b$	functionality
$x \leftarrow length(a), x + length(a)$	property of $+$
$length(a) \leftarrow x, x + length(a)$	property of $+$
$x + length(a) \leftarrow x, length(a)$	functionality
$length(a) \leftarrow a$	functionality

Of course, we will not represent the subterms directly. Instead, by traversing the parse tree of t, we enumerate all such subterms and use these numbers for proposition symbols. We have $O(|t|)$ subterms of t. The enumeration can be done in linear time, and the same expenditure can provide a table connecting the number of each subterm to those of its immediate subterms.

Apart from the main clause, the denotation contains $\tilde{\Delta}$-clauses and functionality clauses. Let $|\sigma|$ denote the total number of propositions in $\Delta(\sigma)$. Then each subterm of t contributes at most $\max_{\sigma\in\Sigma}|\sigma|$ propositions to the $\tilde{\Delta}$-part of the denotation. As each subterm appears as a proposition at most twice in a functionality clause, this part of the denotation contains at most $2|t|$ propositions.

Thus, the denotation has size $O(|t|\max_{\sigma\in\Sigma}|\sigma|)$, and it can clearly be computed in this time too. We are left with verifying the deductions $[\![t]\!] \vdash x_i$. Continuing the example from above, the longest derivation is the one deducing x. It follows here (for clarity, we use the subterms instead of numbers):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathit{exp}}{\mathit{reverse}(b)}}{b} \qquad \dfrac{\mathit{exp}}{\mathit{append}(a,b)}}{a}}{\mathit{length}(a)} \qquad \dfrac{\mathit{exp}}{x + \mathit{length}(a)}}{x}$$

An algorithm in Dowling & Gallier (1984) can verify $x_1 \wedge x_2 \wedge \cdots \wedge x_n$ in a single computation in time $O(|[\![t]\!]|)$. In conclusion, Syn(t) can be decided in time $O(|t|\max_{\sigma\in\Sigma}|\sigma|)$. For the usual case of a fixed (Σ, Δ), the algorithm runs in time $O(|t|)$, i.e., the algorithm is linear in the size of the input.

4. Proof of Correctness

Since we have given a correct algorithm for deciding Syn(t), we need only show that Syn(t) is equivalent to Sem(t). This obligation is decomposed into two parts: soundness and completeness.

4.1. SOUNDNESS

We set out to prove that syntactic injectivity implies semantic injectivity.

DEFINITION 4.1. *A t-clause* $\lfloor s_0 \leftarrow s_1, \ldots, s_k \rfloor$ *is* sound *if for all algebras M and $\bar{v}, \bar{w} \in dom_M^n$ we have*

$$\begin{array}{rcl} & \langle s_1^M(\bar{v}), \ldots, s_k^M(\bar{v}), t^M(\bar{v})\rangle = \langle s_1^M(\bar{w}), \ldots, s_k^M(\bar{w}), t^M(\bar{w})\rangle \\ \Downarrow & \\ & s_0^M(\bar{v}) = s_0^M(\bar{w}) \end{array}$$

This is in line with the definition of function clauses.

LEMMA 4.1. *All clauses in $[\![t]\!]$ are sound.*

PROOF. For all algebras, M, we have

Soundness of the main clause states that

$$\langle t^M(\bar{v})\rangle = \langle t^M(\bar{w})\rangle \Rightarrow t^M(\bar{v}) = t^M(\bar{w})$$

Soundness of the $\tilde{\Delta}$-clauses states that

$$\begin{aligned} &\langle s^M_{d_1}(\bar{v}),\ldots,s^M_{d_m}(\bar{v}),s^M(\bar{v}),t^M(\bar{v})\rangle \\ =\ &\langle s^M_{d_1}(\bar{w}),\ldots,s^M_{d_m}(\bar{w}),s^M(\bar{w}),t^M(\bar{w})\rangle \end{aligned}$$

implies $s^M_{d_0}(\bar{v}) = s^M_{d_0}(\bar{w})$.

By definition 2.7, it follows that $s^M(\bar{v}) = \sigma^M(s^M_1(\bar{v}),\ldots,s^M_k(\bar{v}))$ and $s^M(\bar{w}) = \sigma^M(s^M_1(\bar{w}),\ldots,s^M_k(\bar{w}))$. Since M is an algebra, it follows from definition 2.6 that $s^M_{d_0}(\bar{v}) = s^M_{d_0}(\bar{w})$.

The functionality clauses are sound since each σ^M is a function in the algebra.

□

We show that the deduced terms can be retrieved semantically.

LEMMA 4.2. *Let t and s be terms. Then for all algebras, M, we have*

$$\begin{aligned} &[\![t]\!] \vdash s \\ \Downarrow\ & \\ &\forall \bar{v},\bar{w} \in dom^n_M\colon\ t^M(\bar{v}) = t^M(\bar{w}) \Rightarrow s^M(\bar{v}) = s^M(\bar{w}) \end{aligned}$$

PROOF. We proceed by induction in the size of a proof of the form

$$\frac{\dfrac{\vdots}{s_1}\cdots\dfrac{\vdots}{s_k}}{s_0}$$

By hypothesis, the result holds for the s_i's since they have smaller proofs. We have used the clause $\lfloor s_0 \leftarrow s_1,\ldots,s_k\rfloor$, which is already proved sound in lemma 4.1. But then

$$\begin{aligned} &t^M(\bar{v}) = t^M(\bar{w}) \\ \Downarrow\ & \\ &\forall i \in \{1,\ldots,k\}\colon\ s^M_i(\bar{v}) = s^M_i(\bar{w}),\ \text{by the induction hypothesis} \\ \Downarrow\ & \\ &s^M_0(\bar{v}) = s^M_0(\bar{w}),\ \text{by soundness of the clause} \end{aligned}$$

Notice that the base case is when s_0 is a fact. □

THEOREM 4.1. **(soundness)** *Let t be a term. If t is syntactically injective, then t is also semantically injective, i.e., $Syn(t) \Rightarrow Sem(t)$.*

Proof.

$$\begin{array}{l}
\mathrm{Syn}(t)\\
\Downarrow\\
\forall x_i \in X\colon\ [\![t]\!] \vdash x_i,\ \text{by definition}\\
\Downarrow\\
\forall M \in \mathit{Alg}(\Sigma,\Delta)\ \forall x_i \in X\ \forall \bar{v},\bar{w} \in \mathrm{dom}_M^n\colon\\
\qquad t^M(\bar{v}) = t^M(\bar{w}) \Rightarrow \bar{v}.x_i = \bar{w}.x_i,\ \text{by lemma 4.2}\\
\Downarrow\\
\forall M \in \mathit{Alg}(\Sigma,\Delta)\ \forall \bar{v},\bar{w} \in \mathrm{dom}_M^n\colon\ t^M(\bar{v}) = t^M(\bar{w}) \Rightarrow \bar{v} = \bar{w}\\
\Downarrow\\
\forall M \in \mathit{Alg}(\Sigma,\Delta)\ \forall \bar{v},\bar{w} \in \mathrm{dom}_M^n\colon\ \bar{v} \neq \bar{w} \Rightarrow t^M(\bar{v}) \neq t^M(\bar{w})\\
\Downarrow\\
\mathrm{Sem}(t),\ \text{by definition}
\end{array}$$

□

4.2. completeness

We now endeavor to prove that the other implication holds too, i.e., that semantic injectivity implies syntactic injectivity. Inspired by completeness proofs in logic, the most natural approach is to construct a falsifying model when syntactic injectivity fails. In our case, this means finding two different arguments which yield equal results. In order to obtain these, we introduce some distinct constants.

Definition 4.2. *Let Σ^+ be Σ with the addition of two extra constants, x_i° and $x_i^\bullet$, for each $x_i \in X$. More formally,*

$$\Sigma_0^+ = \Sigma_0 \cup \{x_i^\circ \mid x_i \in X\} \cup \{x_i^\bullet \mid x_i \in X\}$$

and $\Sigma^+ = \Sigma_0^+ \cup \Sigma$.

If $t \in \mathrm{Term}(\Sigma)$, then $t^\circ \in \mathrm{Term}(\Sigma^+)$ is obtained by replacing each x_i with x_i°. We similarly define $t^\bullet$ by replacing each x_i with $x_i^\bullet$.

In the following, the X_i°'s can be thought of as one input to a function t giving rise to one result t°, and the $x_i^\bullet$'s as another input giving rise to another result $t^\bullet$.

We want to generate the term algebra from a very *weak* theory that is designed to be only just strong enough to prove the two results equal. We then show that this theory is too weak to force the arguments to be equal. This would then be our falsifying model.

Definition 4.3. *If $t \in \mathrm{Term}(\Sigma)$, then Mod($t$) is the initial algebra over Σ^+, where the defining equations are:*

The main equation: $t^\circ = t^\bullet$.
The $\tilde{\Delta}$-equations: for each $\lfloor s_0 \leftarrow s_1,\ldots,s_k \rfloor \in \tilde{\Delta}$, we include

$$(s_1^\circ = s_1^\bullet) \wedge \ldots \wedge (s_k^\circ = s_k^\bullet) \Rightarrow s_0^\circ = s_0^\bullet$$

Values are congruence classes of $\mathrm{Term}(\Sigma^+)$ *under the least congruence generated by the defining equations. The class with representative s is denoted $[s]$.*

There is a standard theory of term algebras:

Lemma 4.3. *Mod(t) is completely axiomatized by the theory Eq(t) obtained by adding to the defining equations:*

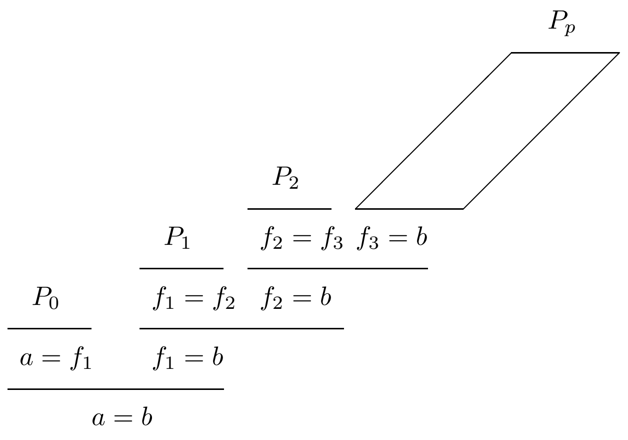
$$\begin{array}{ll}
\textit{reflexivity:} & a = a \\
\textit{symmetry:} & a = b \Rightarrow b = a \\
\textit{transitivity:} & (a = b) \wedge (b = c) \Rightarrow a = c \\
\textit{substitutivity:} & \qquad (a_1 = b_1) \wedge \cdots \wedge (a_k = b_k) \\
 & \Downarrow \\
 & \qquad \sigma(a_1, \ldots, a_k) = \sigma(b_1, \ldots, b_k)
\end{array}$$

i.e., in Mod(t) we have $[t_1] = [t_2]$ if and only if Eq(t) $\vdash t_1 = t_2$.

Proof. Immediate from definitions; for details see Wirsing (1989). □

The following result will facilitate the analysis of transitivity inferences in later proofs.

Lemma 4.4. *Any proof in Eq(t) concluded with a transitivity inference has a normal form of the following kind:*



where the concluding inferences in the P_i's are not *transitivity. If such a concluding inference is symmetry, then the inference immediately above is neither symmetry nor transitivity. Let R_i be the last inference in P_i that is not symmetry. We call R_0 the* left-hand *inference and $R_1, \ldots, R_p$ the* right-hand *inferences.*

Proof. By applying the transformation,

$$\frac{\dfrac{b = c \qquad c = a}{b = a}}{a = b} \quad \to \quad \frac{\dfrac{c = a}{a = c} \qquad \dfrac{b = c}{c = b}}{a = b}$$

we eliminate the symmetry inferences below transitivity inferences. Next, sequences of symmetry inferences are replaced by a single or none. Finally, by applying the transformation,

$$\dfrac{\dfrac{a=d \qquad d=c}{a=c} \qquad c=b}{a=b} \qquad \to \qquad \dfrac{a=d \qquad \dfrac{d=c \qquad c=b}{d=b}}{a=b}$$

we move transitivity inferences to the right. □

The theory Mod(t) was designed to be weak. In the following, we show that it is so weak that only structurally equivalent terms can be proved equal.

Definition 4.4. *Two terms in Mod(t) are called* structurally equivalent *if, when every x_i° and $x_i^\bullet$ are replaced by x_i, they become identical.*

Proposition 4.1. *If $a = b$ can be proved in Mod(t), then a and b are structurally equivalent.*

Proof. By induction in the size of a proof of $a = b$ in Eq(t). The base cases are reflexitivity and the main equation. The remaining case, symmetry, transitivity, substitutivity, and $\tilde{\Delta}$-equations are handled inductively. All cases are trivial. □

Next, we show that we are justified in calling Mod(t) an algebra.

Lemma 4.5. *Mod(t) is a (Σ, Δ)-algebra, when $\sigma \in \Sigma_k \backslash X$ is interpreted as the function $\sigma^{\mathrm{Mod}(t)}\colon ([r_1], \dots, [r_k]) \mapsto [\sigma(r_1, \dots, r_k)]$.*

Proof. Substitutivity ensures that $\sigma^{\mathrm{Mod}(t)}$ is in fact a function. We must further show that it satisfies the requirements given by $\Delta(\sigma)$. Look at the definition of any such requirement. It is vacuously satisfied unless we have an equality of the form

$$\sigma^{\mathrm{Mod}(t)}([a_1], \dots, [a_k]) = \sigma^{\mathrm{Mod}(t)}([b_1], \dots, [b_k])$$

By lemma 4.3, we have such an equality if and only if

$$\mathrm{Eq}(t) \vdash \sigma(a_1, \dots, a_k) = \sigma(b_1, \dots, b_k)$$

We proceed by induction in the size of such a proof and look at the last inference performed. From lemma 4.4, we can assume that the proof is in normal form.

$\tilde{\Delta}$-equation or the main equation: The conclusion is of the form $s^\circ = s^\bullet$, where $s \in \mathrm{Sub}(t)$. By definition, all $\Delta(\sigma)$-requirements on such subterms are directly included as defining equations.

Reflexivity: Here $a_i = b_i$ and we are done.

Symmetry: The result follows trivially from the induction hypothesis.

Transitivity: We use the terminology from lemma 4.4. We consider all cases based on the left-hand and the right-hand inferences. Notice that these can only be reflexitivity, substitutivity, the main equation, or a $\tilde{\Delta}$-equation.

– If the left-hand or some right-hand inference is reflexitivity, then that part of the proof looks like:

$$\dfrac{\dfrac{}{f_i = f_i} \qquad \dfrac{P}{f_i = b}}{f_i = b}$$

So, we have a smaller proof of $f_i = b$ and therefore also of $a = b$, and we can apply the induction hypothesis. Similarly, if a symmetry inference follows the reflexitivity inference in question.

– If the left-hand inference is substitutivity, then the last part of the proof is of the form:

$$\dfrac{\dfrac{a_1 = c_1 \; \cdots \; a_k = c_k}{\sigma(a_1,\ldots,a_k) = \sigma(c_1,\ldots,c_k)} \qquad \sigma(c_1,\ldots,c_k) = \sigma(b_1,\ldots,b_k)}{\sigma(a_1,\ldots,a_k) = \sigma(b_1,\ldots,b_k)}$$

If the equality we want to prove is $a_i = b_i$, then we apply the induction hypothesis to $\sigma(c_1,\ldots,c_k) = \sigma(b_1,\ldots,b_k)$ to obtain $c_i = b_i$. The equality $a_i = c_i$ can be found among the premises for the substitutivity inference, so $a_i = b_i$ can be deduced using transitivity. If a symmetry inference follows the substitutivity inference, then we still have the pair-wise equality among the arguments, which is all we need.

– If every right-hand inference is substitutivity, then the argument is much like the above. From proposition 4.1, it follows that the outer-most function symbol must be the same everywhere in the part of the proof which is of the form depicted in lemma 4.4. Additionally, for every equality $\sigma(c_1,\ldots,c_k) = \sigma(d_1,\ldots,d_k)$, we have proofs of equality among arguments in the same position, i.e., $c_i = d_i$. This either follows directly from the substitutivity inferences (possibly followed by symmetry), or it follows from these proofs combined with a number of transitivity inferences.

The last part of the proof looks like:

$$\dfrac{\sigma(a_1,\ldots,a_k) = \sigma(c_1,\ldots,c_k) \qquad \sigma(c_1,\ldots,c_k) = \sigma(b_1,\ldots,b_k)}{\sigma(a_1,\ldots,a_k) = \sigma(b_1,\ldots,b_k)}$$

If $a_i = b_i$ is the desired equality, then we apply the induction hypothesis to $\sigma(a_1,\ldots,a_k) = \sigma(c_1,\ldots,c_k)$ to obtain $a_i = c_i$. As argued above, we have proofs of pair-wise equality among any of the arguments to the right; $c_i = b_i$ in particular. By transitivity, the result follows.

– If the left-hand inference and some right-hand inference both are $\tilde{\Delta}$-clauses or main clauses, then the conclusion of the proof looks like:

$$\dfrac{s^\circ = s^\bullet \qquad s^\bullet = b}{s^\circ = b}$$

Somewhere up to the right, we have a situation as illustrated below:

$$\dfrac{q^\circ = q^\bullet \qquad q^\bullet = b}{q^\circ = b}$$

By proposition 4.1, two terms which are proved equal are structurally equivalent. As all the inferences below the left-hand and right-hand inferences are either symmetry or transitivity, this means that all the terms depicted in lemma 4.4 are structurally equivalent. In particular, this implies that q° and s°

are structurally equivalent. Since they are both all white, they must be identical. Thus, $q^\circ = b$ is really a smaller proof of the equality $s^\circ = b$. The result follows by applying the induction hypothesis to this smaller proof.

Substitutivity: Here all the arguments must be pairwise equal, so $a_i = b_i$, and any $\Delta(\sigma)$-requirement is trivially satisfied.

Since we have covered all cases, the result follows. □

The next result provides the important link between the term algebra and the denotation.

Lemma 4.6. *Model equality implies denotational deduction, i.e.,*

$$\forall s \in \mathit{Sub}(t)\colon \ \ ((\mathit{Eq}(t) \vdash s^\circ = s^\bullet) \text{ or } (\mathit{Eq}(t) \vdash s^\bullet = s^\circ)) \ \Rightarrow\ [\![t]\!] \vdash s$$

Proof. We inductively transform one proof into another. In Eq(t), we have a proof with conclusion $s^\circ = s^\bullet$ or $s^\circ = s^\bullet$. From lemma 4.4, we can assume that this proof is in normal form. We consider the last inference performed:

The main equation:

$$\frac{}{t^\circ = t^\bullet}$$

The result is trivial since $\lfloor t \leftarrow \rfloor \in [\![t]\!]$.
A $\tilde{\Delta}$-equation:

$$\frac{\dfrac{\vdots}{s_1^\circ = s_1^\bullet} \cdots \dfrac{\vdots}{s_k^\circ = s_k^\bullet}}{s^\circ = s^\bullet}$$

By $s_i \in \mathrm{Sub}(t)$ and the induction hypothesis, we have $[\![t]\!] \vdash s_i$. The $\tilde{\Delta}$-equation comes from the Δ-clause $\lfloor s \leftarrow s_1, \ldots, s_k \rfloor$, so we conclude $[\![t]\!] \vdash s$.
Reflexivity: We either have

$$\frac{}{s^\circ = s^\bullet} \quad \text{or} \quad \frac{}{s^\bullet = s^\circ}$$

However, since we used the reflexitivity inference, s° and $s^\bullet$ are actually identical. This means that they contain no x_i's, so both conclusions are actually $s = s$. An easy induction shows that any such $s \in \mathrm{Sub}(t)$ can be derived from $[\![t]\!]$ using only functionality clauses.
Symmetry: Since we have a situation like

$$\frac{\dfrac{\vdots}{s^\bullet = s^\circ}}{s^\circ = s^\bullet} \quad \text{or} \quad \frac{\dfrac{\vdots}{s^\circ = s^\bullet}}{s^\bullet = s^\circ}$$

we can apply the induction hypothesis to obtain the result.
Transitivity: We use the terminology from lemma 4.4. We consider all cases based on the left-hand and the right-hand inferences. Notice that these can only be reflexitivity, substitutivity, the main equation, or $\tilde{\Delta}$-equations. We only treat the case, where the conclusion is $s^\circ = s^\bullet$; the other case is completely symmetric to this one.

– If the left-hand or some right-hand inference is a $\tilde{\Delta}$-equation or the main equation, then somewhere in the proof, we have an inference as shown below:

$$\frac{q^\circ = q^\bullet \qquad q^\bullet = b}{q^\circ = b}$$

This is assuming that there was not a symmetry inference right below the inference in question. If a symmetry inference was used, this part of the proof would look like:

$$\frac{\dfrac{q^\circ = q^\bullet}{q^\bullet = q^\circ} \qquad q^\circ = b}{q^\bullet = b}$$

In either case, the proof of the equation $q^\circ = b$ is smaller than the entire proof. Since the conclusion of the whole proof is $s^\circ = s^\bullet$, it follows from proposition 4.1 that all the terms depicted in lemma 4.4 are structurally equivalent. Since both q° and s° are all white, they must be identical. Thus, $q^\circ = b$ is really a smaller proof of $s^\circ = b$. We can now apply the induction hypothesis to obtain the result.

– If the left-hand or some right-hand inference is reflexitivity, then we can construct a smaller proof of the original equality by exactly the same technique as was used in the reflexitivity case in the transitivity part of the proof of lemma 4.5.

– Now we assume that both the left-hand and all the right-hand inferences are substitutivity. The proof is of the form:

$$\frac{s^\circ = c \qquad c = s^\bullet}{s^\circ = s^\bullet}$$

By proposition 4.1, c is structurally equivalent to s° and $s^\bullet$. If c is identical to one of the two, then the induction hypothesis can be applied to obtain the result. So, assume that c contains both white and black variables. Such a term is from now on called a *mixed* term.

We now discuss which inference rules can be used to obtain an equality, where terms are mixed. Clearly, this rules out the main equation and the $\tilde{\Delta}$-equations. We are left with reflexitivity, symmetry, transitivity, and substitutivity.

Consider a path going up through the proof of $s^\circ = c$, say, until we meet an inference, which is reflexitivity, the main equation, or a $\tilde{\Delta}$-equation. If we found a reflexitivity inference, $q = q$, then q can be derived, as we have already argued, using only functionality clauses. If it is not reflexitivity, then we have something of the form $q^\circ = q^\bullet$ or $q^\bullet = q^\circ$ and the induction hypothesis applies, i.e., $[\![t]\!] \vdash q$. Assume that we have done this for all such paths. All these deductions can now be combined as we go back down the proof tree again through the symmetries, transitivities, and substitutivities.

The terms in the conclusion of a symmetry or a transitivity inference are structurally equivalent to the terms in the premises, so the same deduction can be used. Going down through a substitutivity inference, we apply a functionality clause in order to derive the fact corresponding to the terms in the conclusion of that inference.

Substitutivity:

$$\frac{\dfrac{\vdots}{s_1^\circ = s_1^\bullet} \cdots \dfrac{\vdots}{s_k^\circ = s_k^\bullet}}{s^\circ = s^\bullet}$$

where $s^\circ = \sigma(s_1^\circ, \ldots, s_1^\circ)$ and $s^\bullet = \sigma(s_1^\bullet, \ldots, s_k^\bullet)$. Since $s \in \mathrm{Sub}(t)$ implies that $\mathrm{Sub}(s) \subseteq \mathrm{Sub}(t)$, we can apply the induction hypothesis to conclude that $[\![t]\!] \vdash s_i$. As we further have the functionality clause $\lfloor s \leftarrow s_1, \ldots, s_k \rfloor$, we are done.

Since we have covered all cases, the result follows. □

THEOREM 4.2. **(completeness)** *Let t be a term. If t is semantically injective, then t is also syntactically injective, i.e.,*

$$\mathit{Sem}(t) \;\Rightarrow\; \mathit{Syn}(t)$$

PROOF. Assume $\neg\mathrm{Syn}(t)$. Then $\mathrm{Mod}(t)$ is a falsifying model:

$$\begin{array}{l} \quad \neg\mathrm{Syn}(t) \\ \Downarrow \\ \quad \exists x_i \in X\colon\ [\![t]\!] \not\vdash x_i, \text{ by definition} \\ \Downarrow \\ \quad \exists x_i \in X\colon\ \mathrm{Eq}(t) \not\vdash x_i^\circ = x_i^\bullet, \text{ by lemma 4.6} \\ \Downarrow \\ \quad \exists x_i \in X\colon\ [x_i^\circ] \neq [x_i^\bullet], \text{ by completeness of } \mathrm{Eq}(t) \\ \Downarrow \\ \quad \langle [x_1^\circ], \ldots, [x_n^\circ] \rangle \neq \langle [x_1^\bullet], \ldots, [x_n^\bullet] \rangle \end{array}$$

But in $\mathrm{Mod}(t)$ we have $[t^\circ] = [t^\bullet]$, which is the same as

$$t^{\mathrm{Mod}(t)}(\langle [x_1^\circ], \ldots, [x_n^\circ] \rangle) = t^{\mathrm{Mod}(t)}(\langle [x_1^\bullet], \ldots, [x_n^\bullet] \rangle)$$

Hence, the interpretation of t is non-injective in $\mathrm{Mod}(t)$. The existence of such a model implies $\neg\mathrm{Sem}(t)$. □

5. Conclusion

This work represents one small step towards the goal of determining injectivity of functions. This is a goal which can never be reached completely, but as we have demonstrated, partial results can be obtained.

The results in this paper are of particular interest in functional languages, mathematical tools, relational databases, and anywhere else, where lists are to be maintained as sets

It is not clear to what extent our technique can cover other standard language constructions. Recursion, or some other form of iteration, would be an interesting study. Notice that if the "injectivity tester" fails to prove injectivity before one iteration over a list, partial injectivity information may have been deduced. This information can be used in the following iteration, making it more likely that injectivity can be proved in that round.

Using recursion instead of a simpler iteration scheme, conditionals are necessary. It would be very useful, if our algorithm could be extended in that direction as well. This seems quite difficult, but one could hope for some conservative result, e.g., sometimes being able to prove injectivity if the two statement parts of a conditional can never produce the same value.

A proposal independent of language constructions would be to give up the completely automatic approach. Quite often it is obvious to the programmer that some user-defined function is injective or partially injective. So, we could let the programmer have the option of attaching this information to the function declaration in the same way as type information is often provided by the user. We believe that such a semi-automatic approach could be very promising.

ACKNOWLEDGEMENTS

We would like to thank the anonymous referees for some very useful suggestions.

References

Abelson, H. *et al.* (1989). Revised report on the algorithmic language Scheme.

Dowling, W. F., Gallier, J. H. (1984). Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *J. Logic Programming* **1**(3), 267-284.

Milner, R., Tofte, M., Harper, R. (1990). *The Definition of Standard ML.* MIT Press.

Turner, D. A. (1985). Miranda: a non-strict functional language with polymorphic types. In: *Proc. Conference on Functional Programming Languages and Computer Architecture. Lecture Notes in Computer Science* **201**, 1-16.

Wirsing, M. (1989). Algebraic specification. In: *Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics*, 675-788. Elsevier Science Publishers.

Wolfram, S. (1988). *Mathematica.* Addison-Wesley.