Referential Opacity In Nondeterministic Data Refinement*

Xiaolei Qian and Allen Goldberg†
Kestrel Institute

August 3, 1992

Abstract

Data refinement is the transformation in a program of one data type to another. With the obvious formalization of nondeterministic data types in equational logic however, many desirable nondeterministic data refinements are impossible to prove correct. Furthermore, it is difficult to have a well-defined notion of refinement. We propose an alternative formalization of nondeterministic data types, in which the requirement of referential transparency applies only to deterministic operators. We show how the above-mentioned problems can be solved with our approach.

Categories and Subject Descriptions: D.2.4[**Software Engineering**]: Program Verification — *Correctness proofs*; D.3.3[**Programming Languages**]: Language Constructs and Features — *Abstract data types*; F.3.2[**Logics and Meanings of Programs**]: Semantics of Programming Languages — *Algebraic approaches to semantics*
General Terms: Languages, Theory, Verification
Additional Key Words and Phrases: Algebraic Specification, Data Refinement, Nondeterminism, Program Transformation, Referential Transparency, Theory Morphism

*This work was supported in part by Rome Laboratories under contract F30602-86-C-0026, and in part by DARPA and Rome Laboratories under contract F30602-91-C-0092.

†Xiaolei Qian is with Computer Science Laboratory, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025; Allen Goldberg is with Kestrel Institute, 3260 Hillview Avenue, Palo Alto, CA 94304.

1 Introduction

Data refinement is the transformation in a program of one data type to another. Data refinement has been approached by formalizing the semantics of abstract data types by initial algebras[6], data type specifications by algebraic theories in equational logic[1], and (correct) data refinements by theory morphisms[5, 10]. Such formalization has the nice property that, assuming abstract data type A is refined to abstract data type B, replacing A by B in program P preserves the correctness of P[4].

Nondeterminism provides a convenient vehicle to avoid specifying all details of an implementation prematurely. The stepwise refinement of a specification to an implementation can be viewed as a process in which nondeterminism is gradually removed by making design decisions[12]. The semantics of nondeterministic data types has been formalized as multi-algebras[7, 9], which essentially avoids nondeterminism by encapsulating it through the medium of set construction. However, the straightforward formulation of nondeterministic data type specifications as algebraic theories in equational logic makes many desirable data refinements impossible to prove correct.

Consider for example the refinement of SET to SEQUENCE[3]. It is desirable to refine equality of SET not to equality of SEQUENCE but to an equivalence relation of SEQUENCE, in which sequences that are permutations of the same set are equivalent. Meanwhile, it is also desirable to refine the nondeterministic *choose* operator of SET, which chooses an element from a nonempty set, to the *head* operator of SEQUENCE, which takes the first element of a nonempty sequence. Obviously these two data refinements cannot coexist in the presence of referential transparency: $x = y \to choose(x) = choose(y)$.

Data refinements are often specified as refinement rules, which are formalized as equations in data type specifications. For example, we would like to have a conditional refinement rule in our specification of SET: $x \neq \{\}$: $choose(x \cup y) \Longrightarrow choose(x)$. Assuming both S and S' are nonempty, we would have $choose(S \cup S') \Longrightarrow choose(S)$ and $choose(S' \cup S) \Longrightarrow choose(S')$. Because of referential transparency, they lead to $choose(S) = choose(S')$, which is clearly undesirable.

Referential transparency and related issues were studied in detail in [11]. It was recognized in [2] that one might have to give up referential transparency in order to adequately deal with nondeterminism. There it was also suggested that a well-defined notion of refinement should be reflexive, transitive, and such that all constructs are monotonic with respect to it. Meseguer observed in [8] that term rewriting should not be formalized in equational logic for applications such as nondeterministic data types, concurrent systems, and object-oriented computation.

The rest of the paper is organized as follows. In Section 2, we propose an alternative formalization of nondeterministic data types in which referential transparency applies only to deterministic operators. We show in Section 3 how various desirable data refinements can be proved correct with our approach, and in Section 4 how a well-defined notion of refinement can be incorporated into the formalism. Section 5 concludes the paper.

2 Nondeterministic Data Types

A *signature* Σ is a pair $\langle S, \Omega\rangle$, where S is a set of *sort symbols* and Ω is a family of finite disjoint sets $\{\Omega_{v,s}\}_{v\in S^*, s\in S}$ of *operator symbols*. Ω is divided into two disjoint families Ω^d and Ω^n. Operator symbols in Ω^d are *deterministic*, while operator symbols in Ω^n are *nondeterministic*. We write $f\colon v \to s$ to denote $v \in S^*$, $s \in S$, and $f \in \Omega_{v,s}$. Let *bool* be the sort symbol for truth values, $\Omega_{v,bool}$ is a set of predicate symbols for $v \in S^*$. For every sort $s \in S$ we assume that there is an (infix) predicate symbol $=_s \in \Omega_{s,s,bool}$. The logical connectives $\vee, \wedge, \neg, \to, \leftrightarrow$ are treated as boolean operator symbols.

For signature $\Sigma = \langle S, \Omega\rangle$, the Σ-*terms* are defined inductively as the well-sorted composition of variables and operator symbols in Ω. A Σ-term t is *deterministic* if all operator symbols in t are from Ω^d. Otherwise t is *nondeterministic*. A Σ-*formula* is a formula built from Σ-terms and quantifiers $\forall$ and $\exists$. A Σ-sentence is a closed Σ-formula.

Let $\Sigma = \langle S, \Omega\rangle$ be a signature. A Σ-*algebra* $\mathcal{A}$ is an S-indexed family of *carrier sets* $A = \{A_s\}_{s\in S}$, a function $f_A\colon A_{v_1} \times \cdots \times A_{v_n} \to A_s$ for every $f \in \Omega^d_{v,s}$, and a function $f_A\colon A_{v_1} \times \cdots \times A_{v_n} \to \mathcal{P}(A_s)$ for every $f \in \Omega^n_{v,s}$, where $v = \langle v_1, \ldots, v_n\rangle$ and $\mathcal{P}(A_s)$ denotes the set of nonempty subsets of A_s.

A *nondeterministic data type* T is a pair $\langle \Sigma, \Phi\rangle$, where $\Sigma = \langle S, \Omega\rangle$ is a signature and Φ is a set of Σ-sentences called *axioms*. For every sort $s \in S$ we assume that the following equality axioms are in Φ:

1. *Reflexivity* $(\forall x)(x =_s x)$
2. *Symmetry* $(\forall x, y)(x =_s y \to y =_s x)$
3. *Transitivity* $(\forall x, y, z)(x =_s y \wedge y =_s z \to x =_s z)$
4. *Monotonicity* For $f \in \Omega^d_{v,s}$ where $v = \langle v_1, \ldots, v_i, \ldots, v_n\rangle$,

$$\begin{aligned}&(\forall x, y, z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_n)\\ &\quad (x =_{v_i} y \to f(z_1, \ldots, z_{i-1}, x, z_{i+1}, \ldots, z_n) =_s f(z_1, \ldots, z_{i-1}, y, z_{i+1}, \ldots, z_n))\end{aligned}$$

The monotonicity axiom schemata applies only to deterministic operators, meaning that we do not enforce referential transparency on nondeterministic operators. A T-*model* is a Σ-algebra that satisfies the axioms of T. A Σ-sentence p is a T-*theorem*, denoted as $T \models p$, if p is a logical consequence of the axioms of T. T-*theory* is the set of T-theorems. As examples, the following are the specifications of two nondeterministic data types: SET and SEQ, both of which take ATOM as a parameter data type with sort *atom* and operator $f(\_)\colon atom \to atom$.

$$
\begin{array}{lll}
\textsc{set}(\textsc{atom}) \stackrel{\text{def}}{=} (& \textbf{sorts} & set \\
 & \textbf{deterministic operators} & \{\}: \to set \\
 & & \_ \circ \_: atom, set \to set \\
 & & \_ \in \_: atom, set \to bool \\
 & & apply^f(\_): set \to set \\
 & \textbf{nondeterministic operators} & choose(\_): set \to atom \\
 & \textbf{axioms} & \neg(a \circ S = \{\}) \\
 & & a \circ (b \circ S) = b \circ (a \circ S) \\
 & & a \circ (a \circ S) = a \circ S \\
 & & \neg(a \in \{\}) \\
 & & a \in (b \circ S) \leftrightarrow a = b \vee a \in S \\
 & & apply^f(\{\}) = \{\} \\
 & & apply^f(a \circ S) = f(a) \circ apply^f(S) \\
 & & \neg(S = \{\}) \to choose(S) \in S)
\end{array}
$$

$$
\begin{array}{lll}
\textsc{seq}(\textsc{atom}) \stackrel{\text{def}}{=} (& \textbf{sorts} & seq \\
 & \textbf{deterministic operators} & [\,]: \to seq \\
 & & \_ \circ \_: atom, seq \to seq \\
 & & head(\_): seq \to atom \\
 & & \_ \in \_: atom, seq \to bool \\
 & & apply^f(\_): seq \to seq \\
 & \textbf{axioms} & \neg(a \circ Q = [\,]) \\
 & & a \circ Q = b \circ Q' \to a = b \wedge Q = Q' \\
 & & head(a \circ Q) = a \\
 & & \neg(a \in [\,]) \\
 & & a \in (b \circ Q) \leftrightarrow a = b \vee a \in Q \\
 & & apply^f([\,]) = [\,] \\
 & & apply^f(a \circ Q) = f(a) \circ apply^f(Q))
\end{array}
$$

The intuition behind the nondeterministic *choose* operator of SET is that it can be referentially opaque. We do not expect that $choose(S)$ computes a specific element of S, nor that $choose(S)$ always computes the same element of S. All we require about *choose* is given by the last axiom of SET.

3 Data Refinement

A *signature morphism* $\sigma: \Sigma \to \Sigma'$, where $\Sigma = \langle S, \Omega \rangle$ and $\Sigma' = \langle S', \Omega' \rangle$, is a pair $\langle \delta, \theta \rangle$ where $\delta: S \to S'$ is a map and θ is a family of maps $\{\theta_{v,s}: \Omega_{v,s} \to \Omega'_{\delta^*(v),\delta(s)}\}_{v \in S^*, s \in S}$ where $\delta^*(\langle v_1, \ldots, v_n \rangle)$ denotes $\langle \delta(v_1), \ldots, \delta(v_n) \rangle$ for $v_1, \ldots, v_n \in S$. We write $\sigma(s)$ for $\delta(s)$, $\sigma(v)$

for $\delta^*(v)$, and $\sigma(f)$ for $\theta_{v,s}(f)$ where $f \in \Omega_{v,s}$. Given a Σ-formula p, $\sigma(p)$ denotes the Σ'-formula resulted from replacing every operator symbol f in p by $\sigma(f)$. An obvious signature morphism from SET to SEQ is:

$$\{set \mapsto seq,$$
$$\{\} \mapsto [\,], \circ_{set} \mapsto \circ_{seq}, \in_{set} \mapsto \in_{seq}, apply^f_{set} \mapsto apply^f_{seq}, =_{set} \mapsto =_{seq}, choose \mapsto head\}$$

Let $T = \langle \Sigma, \Phi \rangle$ and $T' = \langle \Sigma', \Phi' \rangle$ be two nondeterministic data types, and $\sigma: \Sigma \to \Sigma'$ be a signature morphism. We say that σ is a *data refinement* from T to T' if $T' \models \sigma(A)$ for every axiom $A \in \Phi$. Apparently the above signature morphism from SET to SEQ is not a data refinement, because SEQ $\not\models a \circ (a \circ Q) = a \circ Q$.

Suppose T_1, T_2 are two nondeterministic data types with equality predicates $=_1, =_2$ respectively. In refining T_1 to T_2, a critical decision is how to refine equality. There are two possible ways that $=_1$ can be refined through signature morphism σ.

1. If $\sigma(=_1)$ is $=_2$, then we are often forced to require more knowledge from T_1. Moreover, it might add too much detail that prohibits certain efficient implementations. For example, if $\sigma(=_{set})$ is $=_{seq}$, then $\sigma(\circ_{set})$ cannot be $\circ_{seq}$. We might require that there is a total ordering $\leq$ on *atom*, and build into SEQ an (ordered) insert operator $\_ \diamond \_ : atom, seq \to seq$ defined by:

$$\begin{aligned}
&a \diamond [\,] = a \circ [\,] \\
&a \diamond (a \circ Q) = a \circ Q \\
&a < b \to a \diamond (b \circ Q) = a \circ (b \circ Q) \\
&b < a \to a \diamond (b \circ Q) = b \circ (a \diamond Q)
\end{aligned}$$

   Now $\sigma(\circ_{set})$ can be $\diamond$. This implementation represents every set as an ordered sequence with no duplicate atoms. It is more efficient when $=_{set}$ is used more frequently than $\circ_{set}$.

2. If $\sigma(=_1)$ is $\approx$ different from $=_2$, then $\approx$ must be logically weaker than $=_2$, namely for all x, y we have $x =_2 y$ implies $x \approx y$, because $\approx$ must satisfy all the equality axioms. The refinement of $=_1$ is underspecified: every object of T_1 is refined to a group of objects of T_2 not distinguishable under $\approx$. This delay of implementation decisions is essential in keeping more implementation options available. For example, we might build into SEQ a range containment predicate $\_ \sqsubseteq \_ : seq, seq \to bool$ and a range equality predicate $\_ =_r \_ : seq, seq \to bool$ defined by:

$$\begin{aligned}
&[\,] \sqsubseteq Q \\
&a \circ Q \sqsubseteq Q' \leftrightarrow a \in Q' \wedge Q \sqsubseteq Q' \\
&Q =_r Q' \leftrightarrow Q \sqsubseteq Q' \wedge Q' \sqsubseteq Q
\end{aligned}$$

Now $\sigma(=_{set})$ can be $=_r$, and $\sigma(\circ_{set})$ can be $\circ_{seq}$. This implementation represents every set by a group of sequences not distinguishable under $=_r$: these sequences all contain the same set of atoms. It is more efficient when $\circ_{set}$ is used more frequently than $=_{set}$.

Suppose T_1 has a nondeterministic operator $f\colon v \to s$. Another critical decision is how to refine f such that the right amount of nondeterminism is removed from T_1.

1. We might refine f to a nondeterministic operator. For example, we might build into SEQ a nondeterministic operator $choose_{seq}(\_)\colon seq \to atom$ defined by

$$\neg(Q = [\,]) \to choose(Q) \in Q$$

   and refine $choose_{set}$ to it. Since data refinement can be viewed as the gradual removal of nondeterminism, this refinement causes unnecessary delay of implementation decisions.

2. We might refine f to a deterministic operator. For example, if there is a min operator on $atom$, we might build into SEQ a select (minimum) operator $select(\_)\colon seq \to atom$ defined by

$$\begin{aligned} select(a \circ [\,]) &= a \\ select(a \circ (b \circ Q)) &= \min(a, select(b \circ Q)) \end{aligned}$$

   and refine $choose_{set}$ to it. This implementation represents arbitrary selection in a set by the selection of the minimal element of a sequence, which is linear in the size of the sequence. It destroys the flexibility about *choose* in SET by having the implementation of $choose(S)$ computes a definite and specific element of S. Alternatively, we might refine *choose* to *head*, which is a constant-time operation that preserves the flexibility about *choose* in SET.

If referential transparency is enforced on nondeterministic operators however, certain desirable refinement combinations of equality and nondeterministic operators become incorrect. In general, if $\sigma(=_v)$ is $\approx$ weaker than $=_{\sigma(v)}$, $f\colon v \to s$, and $\sigma(f)$ is deterministic, then $\sigma(f)$ has to satisfy the additional requirement that $x \approx y \to \sigma(f)(x) =_{\sigma(s)} \sigma(f)(y)$. For example, with referential transparency enforced on $choose_{set}$, the following signature morphism from SET to SEQ is not a data refinement, since SEQ $\not\models Q =_r Q' \to head(Q) = head(Q')$.

$$\begin{aligned} &\{set \mapsto seq, \\ &\ \{\} \mapsto [\,], \circ_{set} \mapsto \circ_{seq}, \in_{set} \mapsto \in_{seq}, apply^f_{set} \mapsto apply^f_{seq}, =_{set} \mapsto =_r, choose \mapsto head\} \end{aligned}$$

Under our approach where nondeterministic operators are referentially opaque, it is easy to verify that the above signature morphism is indeed a data refinement. It does not require additional knowledge on *atom* such as a total ordering or a min operator, and it provides constant-time implementation for both $\circ_{set}$ and $choose_{set}$. The combination of $=_r$ and *head* operators in SEQ captures the essence of nondeterministic behavior exhibited by $=$ and *choose* operators in SET.

4 Refinement Predicate

To capture the notion of refinement rules in nondeterministic data type refinement, we introduce a *refinement predicate*. Given a nondeterministic data type $T = \langle \Sigma, \Phi \rangle$ where $\Sigma = \langle S, \Omega \rangle$, we extend T with a refinement predicate $\Rightarrow_s \in \Omega_{s,s,bool}$ for $s \in S$. For every sort $s \in S$ we assume that the following refinement axioms are in Φ:

1. *Reflexivity* $(\forall x)(x \Rightarrow_s x)$
2. *Transitivity* $(\forall x, y, z)(x \Rightarrow_s y \wedge y \Rightarrow_s z \rightarrow x \Rightarrow_s z)$
3. *Monotonicity* For $f \in \Omega_{v,s}$ where $v = \langle v_1, \ldots, v_i, \ldots, v_n \rangle$,

$$\begin{aligned}&(\forall x, y, z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_n)\\ &\quad (x \Rightarrow_{v_i} y \rightarrow f(z_1, \ldots, z_{i-1}, x, z_{i+1}, \ldots, z_n) \Rightarrow_s f(z_1, \ldots, z_{i-1}, y, z_{i+1}, \ldots, z_n))\end{aligned}$$

4. *Equality* For deterministic Σ-terms t, t',

$$t \Rightarrow_s t' \rightarrow t =_s t'$$

Compared with the equality axioms of Section 2, the monotonicity axiom schemata applies to nondeterministic as well as deterministic operators. The last axiom says that the refinement predicate is stronger than the equality predicate in nondeterministic data types: there are nondeterministic Σ-terms t, t' such that $t \Rightarrow_s t'$ but $\neg(t =_s t')$. In the special case of deterministic data types, the last axiom implies that the refinement predicate is equivalent to the equality predicate, which corresponds to rewriting in equational logic.

A conditional refinement rule $\phi\colon t \Longrightarrow t'$ in the refinement of nondeterministic data type $T = \langle \Sigma, \Phi \rangle$, where ϕ is a Σ-formula and t, t' are Σ-terms, is expressed by a (universally quantified) Σ-sentence p of the form $\phi \rightarrow t \Rightarrow t'$. The refinement of T by this rule is a nondeterministic data type $T' = \langle \Sigma, \Phi' \rangle$ where $\Phi' = \Phi \cup \{p\}$. Conditional refinement rules can be used both to reduce nondeterminism and to obtain optimal implementation of nondeterministic data types. As an example, we can refine SET by the following conditional refinement rules:

$$\begin{aligned}&\neg(S = \{\})\colon \mathit{choose}(S \cup S') \Longrightarrow \mathit{choose}(S)\\ &\neg(S = \{\})\colon \mathit{choose}(\mathit{apply}^f(S)) \Longrightarrow f(\mathit{choose}(S))\end{aligned}$$

The resulting nondeterministic data type SET$'$ contains the following two axioms in addition to axioms of SET:

$$\begin{aligned}&\neg(S = \{\}) \rightarrow \mathit{choose}(S \cup S') \Rightarrow \mathit{choose}(S)\\ &\neg(S = \{\}) \rightarrow \mathit{choose}(\mathit{apply}^f(S)) \Rightarrow f(\mathit{choose}(S))\end{aligned}$$

Compared with SET, SET$'$ has a reduced degree of nondeterminism and is more efficient in computation. The refinement predicate is stronger than the equality predicate, because from $S = S'$ we cannot infer $choose(S) = choose(S')$ (*choose* is referentially opaque), but from $S \Rightarrow S'$ we can infer $choose(S) \Rightarrow choose(S')$. Notice that such a refinement is not semantically sound if nondeterministic operators are referentially transparent. As an example, suppose *choose* is referentially transparent, and we have in SET a union operator $\_ \cup \_ : set, set \rightarrow set$ such that

$$\{\} \cup S' = S'$$
$$(a \circ S) \cup S' = a \circ (S \cup S')$$

¿From $S \cup S' = S' \cup S$, we infer by the monotonicity axiom $choose(S \cup S') = choose(S' \cup S)$. Applying the first conditional refinement rule we get $\neg(S = \{\}) \wedge \neg(S' = \{\}) \rightarrow choose(S) = choose(S')$, which leads to the collapse of the carrier set A_{atom} to a single element in any SET$'$-model, since from $\neg(a \circ \{\} = \{\})$ and $\neg(b \circ \{\} = \{\})$, we infer $choose(a \circ \{\}) = choose(b \circ \{\})$ and hence $a = b$.

5 Conclusion

The refinement of nondeterministic data types is a process which gradually removes nondeterminism by making design decisions that lead to an efficient implementation. However, with the specification of nondeterministic data types formalized in equational logic, many desirable refinements become impossible to prove correct. Nondeterminism often has to be removed entirely in one refinement step, rather than gradually through many refinement steps. This problem is caused by the requirement of referential transparency imposed on nondeterministic operators.

We proposed an alternative formalization of nondeterministic data types, which imposes the requirement of referential transparency only on deterministic operators. With this approach, we can easily show the correctness of many desirable refinements. Moreover, a well-defined notion of refinement can be built into nondeterministic data types as a refinement operator. Using such an operator, the gradual removal of nondeterminism can be formulated as conditional refinement rules, whose stepwise application leads to efficient implementation.

Acknowledgement

We thank Lee Blaine for our collaboration on the DTRE program transformation system, which has been a source of inspiration for the ideas presented here. The first author also thanks Peter Ladkin and José Meseguer for helpful discussions.

References

[1] Bidoit, M., Kreowski, H.-J., Lescanne, P., Orejas, F., Sannella, D., "Algebraic System Specification and Development"; *Lecture Notes in Computer Science* **501**, Springer-Verlag, 1991.

[2] Bird, R., Meetens, L., Wile, D., "A Common Basis for Algorithmic Specification and Development"; *Technical Report* CS-N8702, Center for Mathematics and Computer Science, Amsterdam, April 1987, 89-100.

[3] Blaine, L., Goldberg, A., "DTRE — A Semi-Automatic Transformation System"; *Constructing Programs from Specifications*, B. Möller (editor), North-Holland, 1991, 165-204.

[4] Broy, M., Möller, B., Pepper, P., "Algebraic Implementations Preserve Program Correctness"; *Science of Computer Programming* **7**, 1986, 35-53.

[5] Goguen, J.A., Thatcher, J.W., Wagner, E.G., "An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types"; *Current Trends in Programming Methodology, Vol.4: Data Structuring*, R.T. Yeh (Editor), Prentice-Hall, 1978, 80-149.

[6] Guttag, J.V., Horning, J.J., "The Algebraic Specification of Abstract Data Types"; *Acta Informatica* **10**, 1978, 27-52.

[7] Hesselink, W., "A Mathematical Approach to Nondeterminism in Data Types"; *ACM Transactions on Programming Languages and Systems* **10**:1, January 1988, 87-117.

[8] Meseguer, J., "Conditional Rewriting Logic as a Unified Model of Concurrency"; *Theoretical Computer Science* **96**, 1992, 73-155.

[9] Nipkow, T., "Non-deterministic Data Types: Models and Implementations"; *Acta Informatica* **22**, 1986, 629-661.

[10] Sannella, D., Tarlecki, A., "Toward Formal Development of Programs from Algebraic Specifications: Implementations revisited"; *Acta Informatica* **25**, 1988, 233-281.

[11] Søndergaard, H., Sestoft, P., "Referential Transparency, Definiteness and Unfoldability"; *Acta Informatica* **27**, 1990, 505-517.

[12] Turski, W., Maibaum, T., *The Specification of Computer Programs*, Addison-Wesley, 1987.