Some Proofs of Transforms

Earlin Lutz

TR 89-1041
September 1989

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

Some Proofs of Transforms

Earlin Lutz *

November 8, 1989

Abstract

Three simple examples of *data refinement* – replacement of an abstract program fragment and its variables by a more concrete fragment and variables – are presented in the Polya transform notation. Correctness of each transformation is derived using the formulations of Prinz-Gries, Morris, and Chen/Udding, which are formally equivalent but require different proof strategies. This allows comparison of the three formulations based on ease of use.

Programs that are written using abstract data structures must be replaced by more concrete versions that use realizable data structures. Prins and Gries [PG85], Morris [Mor87], and Chen and Udding [CU89] have proposed statements of when this replacement or transformation process is formally correct. More recently, Chen and Udding proved that the various definitions are equivalent. Because of this equivalence, choice among the methods is a matter of convenience for the particular problem at hand, rather than of correctness.

This note presents three examples of program transformation, using the notation of the Polya programming language. For each example, a correctness proof is carried out with the Prins/Gries, Chen/Udding, and Morris methods.

The first example shows a statement with a history-dependent precondition. This illustrates that all methods will extract resource bounds from transform invariants.

The second example shows a nondeterministic mapping from concrete to abstract.

The third example replaces one nondeterministic statement by another that uses the same variables but is more deterministic.

Brief remarks are made to highlight the relative ease of use of the methods.

*Supported by a grant from the Unisys Corporation and by the NSF and DARPA under grant ASC-88-00465

1 Statement of Proof Obligations

This section states the formal proof obligations as defined by Morris, Prins/Gries, and Chen/Udding.

The notation used is:

Sa	=	an abstract statement
Sc	=	a concrete statement
Ra	=	a predicate containing only abstract variables
Rc	=	a predicate containing only concrete variables
I	=	a predicate between concrete and abstract variables

1.1 Morris

The Morris condition for replacement of an abstract statement Sa by a concrete statement Sc is

$$(\exists\ a :: I \wedge\ wp(Sa,Ra)) \Rightarrow wp(Sc,(\exists\ a :: I \wedge\ Ra)\ )$$

holds for all postconditions Ra.

This is read 'if there is some abstract state that couples to the current concrete one and satisfies $wp(Sa,Ra)$, then the concrete statement is certain to terminate in a state for which there exists some abstract value that couples to the (new) concrete values and satisfies Ra.'

Since the right side of the implication does not involve the abstract statement, there is the possibility of establishing correctness by reaching a **true** result from that side alone, without ever considering the abstract statement or its results. In practice, the quantified Ra prevents this. Ra must be removed by (a) expanding the wp and (b) manipulating to a point where Ra makes the same contribution to both sides and no longer affects the implication.

1.2 Prins/Gries

If a pattern and replacement do not assign to shared variables, the Prins/Gries refinement condition for replacing a statement is

$$I \Rightarrow wp(Sc,\neg wp(Sa,\neg I))$$

given that I holds between initial abstract and concrete values, and also that termination of Sa is assured.

This statement explicitly considers executing both statements and comparing results. It says that for each possible concrete execution and result there must be some possible abstract execution and result that leads back to the coupling.

If the pattern and replacement both assign to some shared variable, this condition must be modified so that (a) execution of one statement does not change values seen by the subsequent execution of the other, and (b) the two results can both appear in the same predicate after the two statements. To achieve this, we require that the shared variable be renamed in one of the statements. If the shared variables are named x in one statement and y in the other, the proof condition then includes the requirement to reestablish $x = y$, given that it holds at the start of execution:

$$(I \wedge \ x = y) \Rightarrow wp(Sc, \neg wp(Sa, \neg(I \wedge \ x = y)))$$

1.3 Chen/Udding

Chen and Udding show that the Morris and Prins/Gries forms are equivalent to each other and to each of three additional forms.

One of their new forms is a sort of mirror image of the Morris form: its postcondition is concrete (rather than abstract), it has a universal quantifier over concrete values (rather than existential over abstracts), and quantification is on the left (rather than the right) of the implication. Like the Morris form, it is (in an overall sense) quantified with respect to all postconditions.

The other two forms given by Chen/Udding use abstract and concrete postconditions more equally, and are very similar to one another. They avoid the internal quantification over values, but the overall quantification over postconditions involves a *pair* of postconditions, one abstract and one concrete. We will give examples of one of these.

The condition to be proven is formally stated as:

$$[I \Rightarrow (Ra = Rc)] \Rightarrow [I \wedge \ wp(Sa, Ra) \Rightarrow wp(Sc, Rc)]$$

for any Ra independent of Sc and Rc independent of Sa.

1.4 A note on use of invariants

A completely mechanical application of these formulas would call for writing down the full formula with the invariant expanded to the data structures at hand. However, the proof process is much simplified if we note that all of the formulas become immediately true if the invariant fails to hold. In each case, a false invariant falsifies the left side of an implication, making the formula as a whole true. Therefore, the effective proof obligation is only to show that the formula holds when the invariant is assumed true. This same reasoning applies to the $x = y$ and $Ra = Rc$ terms in the Prins/Gries and Chen/Udding forms.

This simplification allows the initial proof obligations to be reduced to verification of only the *right* sides if the formulas under the assumption that all left side terms are true. This shortens the amount of equation copying significantly.

In the Morris form, this step is not so clear, because I appears inside of a quantifier. We have therefore chosen to write out the entire expression in each of the proofs below.

2 A Precondition that cannot be Stated in Abstract Terms

The following transform implements a set in an unordered array with duplicates. [1]

Transform SET_AS_ARRAY(n:*nat*)
var A:*set*(?T) **repr** **var** c:**array**[0..n-1] **of** T
var nc:*nat*

Coupling: $0 \leq nc \leq n \wedge A = A_0(c, nc)$
where $A_0(c, nc) = (\bigcup i : 0 \leq i < nc : c.\{i\})$

□ $\{P\}$	$A := A \cup \{a\}$	**into** c,$nc := (c;\ nc{:}a), nc + 1$
□	$A := \{\}$	**into** $nc := 0$
□	$a \in A$	**into** Search c for a ...
□	$\#A > 0$	**into** $nc > 0$

end transform

What is the precondition P for the insertion statement?

Morris proof We derive P using the Morris proof obligation for the replacement of $A := A \cup \{a\}$:

$$(\exists\ A ::\ \ 0 \leq nc \leq n \wedge\ A = A_0(c, nc)\ \ \wedge\ \ wp(A := A \cup \{a\}\ , Ra)) \Rightarrow wp(c,nc := (c;\ nc{:}a), nc + 1\ ,$$

[1] The notation for the transform is that of the Polya language. The structure

Transform name
abstract vars **repr** *concrete vars*
Coupling: P

□	*abstract statement*	**into** *concrete statement*
□	*abstract expression*	**into** *concrete expression*

end transform

gives a left to right description of how abstract variables, statements, and expressions should be transformed to concrete ones in a program. The coupling P describes the invariant relationship being maintained between the concrete and abstract variables.

$$
\begin{array}{ll}
 & \quad (\exists\ A ::\ \ 0 \le nc \le n \ \wedge\ A = A_0(c, nc)\ \ \wedge\ Ra)\ ) \\
= & \quad \{\ \text{Expand each } wp \text{ by textual substitution.}\ \} \\
 & (\exists\ A ::\ \ 0 \le nc \le n \ \wedge\ A = A_0(c, nc)\ \ \wedge\ Ra^{A}_{A \cup \{a\}}) \\
 & \quad \Rightarrow (\exists\ A ::\ \ 0 \le nc \le n \ \wedge\ A = A_0(c, nc)\ \ \wedge\ Ra)^{\,c, nc}_{(c;\ nc{:}a), nc+1} \\
= & (\exists\ A ::\ \ 0 \le nc \le n \ \wedge\ A = A_0(c, nc)\ \ \wedge\ Ra^{A}_{A \cup \{a\}}) \\
 & \quad \Rightarrow (\exists\ A ::\ \ 0 \le nc+1 \le n \ \wedge\ A = A_0((c;\ nc : a), nc+1)\ \ \wedge\ Ra) \\
= & \quad \{\ \text{One point rule, twice}\ \} \\
 & 0 \le nc \le n\ \ \wedge\ Ra^{A}_{A_0(c, nc) \cup \{a\}} \\
 & \quad \Rightarrow 0 \le nc+1 \le n\ \ \wedge\ Ra^{A}_{A_0((c;\ nc : a), nc+1)} \\
= & \quad \{\ \text{Note } A_0(c, nc)\ \cup \{a\} = A_0((c;\ nc : a), nc+1)\ \} \\
 & 0 \le nc \le n\ \ \wedge\ Ra^{A}_{A_0(c, nc) \cup \{a\}}\ \Rightarrow 0 \le nc+1 \le n\ \ \wedge\ Ra^{A}_{A_0(c, nc) \cup \{a\}} \\
= & \quad \{\ \text{arithmetic and predicate calculus}\ \} \\
 & nc < n
\end{array}
$$

This is the formally derived precondition of the insertion statement. It is a concrete restriction that cannot be expressed in terms of the abstract variables alone, since an abstract set does not have a notion of how many insertions (including duplicates) have been done.

Prins/Gries proof The Prins/Gries form proof is:

$$
\begin{array}{ll}
 & wp(c,nc := (c;\ nc{:}a), nc+1\ , \\
 & \quad \neg wp(A := A \cup \{a\}\ , \\
 & \quad\quad \neg(\ 0 \le nc \le n \ \wedge\ A = A_0(c, nc)\ )\ )) \\
= & \quad \{\ \text{Expand both } wp\text{s by textual substitution}\ \} \\
 & \neg\neg(\ \ 0 \le nc+1 \le n \ \wedge\ A \cup \{a\} = A_0((c;\ nc : a), nc+1)\ ) \\
= & \quad \{\ \text{Remove double negation and expand } A \text{ as defined in coupling}\ \} \\
 & 0 \le nc+1 \le n \ \wedge\ (A_0(c, nc) \cup \{a\}) = A_0((c;\ nc : a), nc+1) \\
= & \quad \{\ \text{Set manipulation}\ \} \\
 & 0 \le nc+1 \le n \\
= & \quad \{\ \text{Invariant provides } 0 < nc\ \} \\
 & nc < n
\end{array}
$$

Chen/Udding proof For the Chen/Udding form, we assume that Ra and Rc are predicates that are equal whenever the concrete and abstract variables satisfy the coupling. Then we must show

$$
\begin{array}{l}
0 \le nc \le n \ \wedge\ A = A_0(c, nc)\ \ \wedge\ wp(A := A \cup \{a\}\ , Ra) \\
\quad \Rightarrow wp(c,nc := (c;\ nc{:}a), nc+1\ , Rc)
\end{array}
$$

$=$ { Expand both *wps* by textual substitution }
$0 \leq nc \leq n \wedge A = A_0(c, nc) \wedge Ra^{A}_{A \cup \{a\}} \Rightarrow nc < n \wedge Rc^{c,nc}_{(c;\ nc{:}a),nc+1}$
$=$ { $Ra^{A}_{A \cup \{a\}}$ and $Rc^{c,nc}_{(c;\ nc{:}a),nc+1}$ are equal because the because the substitute values satisfy the invariant }
$0 \leq nc \leq n \wedge A = A_0(c, nc) \Rightarrow nc < n$
$=$ { Coupling assumed true }
$nc < n$

2.1 Digression: Removing the History-Dependence

This example shows that transform preconditions cannot always be expressed in terms of current values abstract variables.

This section discusses several modifications that can remove the limitation and keep the concrete form properly hidden.

Redefine the abstract structure as a bag If the abstract structure is called a *bag*, i.e. a set with multiple elements, the *size* of the bag is precisely the count that is needed for this precondition.

This solution is nicely tailored to the transform, but would prevent future versions of the transform from reverting to correct *set* behavior.

Remove duplicates If duplicates are removed at each insertion, the concrete size restriction $nc < n$ has abstract translation $\#A < n$.

This might appear to be the obvious solution, but it assumes an equality test for the objects. This might not be possible, for instance, if T is a representation of a solid object. Further, it might simply not be the desired implementation.

Declare and implement an artificial variable The following version of the transform requires the abstract program to declare an artificial counter and implement it along with the set itself. The insertion operation is presented in a way that can only be matched by a simultaneous incrementing of the counter.

Transform SET_AS_ARRAY
var A:*set*(?T) **repr** **var** c:**array**[0..n-1] **of** T
var m:*nat* **var** nc:*nat*

Coupling: $0 \leq nc \leq n \wedge A = A_0(c, nc) \wedge m = nc$

□ {P}	$A,m := A \cup \{a\},m+1$	**into** $c,nc := (c;\ nc{:}a),nc+1$
□	$A,m := \{\},0$	**into** $nc := 0$

□	$a \in A$	**into** Search c for a ...
□	$\#A > 0$	**into** $nc > 0$

end transform

Each of these solutions to the visibility of the concrete variable makes the transform a bit harder to use by calling for more precise properties in the abstract program.

3 A Backwards-nondeterministic Coupling

In this transform, each concrete value represents many abstract ones.

Transform INT_AS_BOOL

	var i:*int*	**repr var** b:*bool*

Coupling: $b = \text{odd}.i$

□	$i := 0$	**into** $b :=$ **false**
□	$i := i + 1$	**into** $b := \neg b$
□	odd.i	**into** b

end transform

Morris proof The Morris proof obligation for replacing $i := 0$ by $b :=$ **false** is

$$
\begin{array}{ll}
 & (\exists\ \mathrm{i} ::\ b = \text{odd}.i\ \wedge\ wp(i := 0\ , Ra)) \ \Rightarrow wp(b := \textbf{false}\ , (\exists\ i ::\ b = \text{odd}.i\ \wedge\ Ra)\ ) \\
= & \quad \{\ wp \text{ for assignment } \} \\
 & (\exists\ \mathrm{i} ::\ b = \text{odd}.i\ \wedge\ Ra_0^i\ ) \ \Rightarrow (\exists\ i ::\ b = \text{odd}.i\ \wedge\ Ra)\ {}^{b}_{\textbf{false}} \\
= & \quad \{\ b \text{ not free in } Ra.\ \} \\
 & (\exists\ \mathrm{i} ::\ b = \text{odd}.i\ \wedge\ Ra_0^i\ ) \ \Rightarrow (\exists\ i ::\ \textbf{false} = \text{odd}.i\ \wedge\ Ra) \\
= & \quad \{\ i \text{ not free in } Ra \text{ after substitution.}\ \} \\
 & (\exists\ \mathrm{i} ::\ b = \text{odd}.i)\ \wedge\ Ra_0^i \ \Rightarrow (\exists\ i ::\ \textbf{false} = \text{odd}.i\ \wedge\ Ra) \\
= & \quad \{\ (\exists\ \mathrm{i} ::\ b = \text{odd}.i) \text{ is true } \} \\
 & Ra_0^i \Rightarrow (\exists\ i ::\ \textbf{false} = \text{odd}.i\ \wedge\ Ra) \\
= & \quad \{\ \text{Instantiate with } i{=}0\ \} \\
 & Ra_0^i \Rightarrow \textbf{false} = \text{odd}.0\ \wedge\ Ra_0^i \\
= & \textbf{true}
\end{array}
$$

Prins/Gries proof The Prins/Gries form for replacing $i := 0$ by $b :=$ **false** is

$$wp(b := \textbf{false}\ , \neg wp(i := 0\ , \neg(b = \text{odd}.i)))$$

$=$ { Expand inner wp by textual replacement $(b = \text{odd}.i)^{i}_{0}$ }
$wp(b := \textbf{false}\ , \neg\neg(b = \text{odd}.0))$
$=$ { Cancel double negation }
$wp(b := \textbf{false}\ , b = \text{odd}.0))$
$=$ { Expand wp }
$\textbf{false} = \text{odd}.0$
$=$ $\textbf{true}$

Chen/Udding proof For the Chen/Udding form, we assume that Ra and Rc are predicates that are equal (i.e. both true or both false) whenever the concrete and abstract variables satisfy the coupling. Then we must show

$b = \text{odd}.i \ \wedge \ wp(i := 0\ , Ra) \ \Rightarrow \ wp(b := \textbf{false}\ , Rc)$
$=$ { Expand both wps by textual substitution }
$b = \text{odd}.i \ \wedge \ Ra^{i}_{0} \Rightarrow Rc^{b}_{\textbf{false}}$
$=$ { $Ra^{i}_{0} = Rc^{b}_{\textbf{false}}$ because the substitute values satisfy the invariant }
$b = \text{odd}.i$
$=$ { Coupling assumed true }
$\textbf{true}$

4 A More Deterministic Chooser

The following replacement converts one nondeterministic statement to another, more deterministic, one. Because there are no variables being replaced, we give the replacement in isolation, i.e. without the **Transform** wrapper:

□ $choose(a,A)$ **into** $choose(a,\{A.min, A.max\})$

For proof purposes, we will consider the more general replacement

$$choose(a,B)$$

where $B \subseteq A$ and $B \neq \emptyset$.

Since there are no represented variables, the proof obligation is really only the procedural refinement condition

$$wp(choose(a,A),Ra) \ \Rightarrow \ wp(choose(a,\{A.min, A.max\}),Ra)$$

The data refinement rules apply, despite absence of concrete variables, with couplings identically **true**. The Morris and Prins/Gries forms both degenerate cleanly into pure procedural refinement when there is no change of variables. It is not clear how the Chen/Udding form is to be handled in this case.

Morris proof Morris treats $choose(a,A)$ as a nondeterministic assignment denoted by a:-$\{a \in A\}$. Its weakest precondition is

$$wp(a\text{:-}\{a \in A\},Ra) = (\forall\, b : b \in A : Ra^a_b)$$

(Remark: Morris omits the expected conjunct $A \neq \emptyset$ from his definition of wp. □)

The proof is then

$$\begin{array}{ll} & wp(choose(a,A),Ra) \;\Rightarrow\; wp(choose(a,B),Ra) \\ = & \quad \{ \text{ Convert both sides to nondeterministic assignment } \} \\ & wp(a\text{:-}\{a \in A\},Ra) \;\Rightarrow\; wp(a\text{:-}\{a \in B\},Ra) \\ = & \quad \{ \text{ Morris' } wp \text{ definition for nondeterministic assignment } \} \\ & (\forall\, b : b \in A : Ra^a_b) \;\Rightarrow\; (\forall\, b : b \in B : Ra^a_b) \\ = & \quad \{\; B \subseteq A \;\} \\ & \textbf{true} \end{array}$$

Prins/Gries proof Because the two statements both assign to a common variable a, the Prins/Gries proof obligation requires that a renaming be applied. We will modify the replacement $choose(a,B)$ to $choose(b,B)$. The postcondition in the proof obligation will require that $b = a$:

$$\begin{array}{ll} & wp(choose(b,B),\, \neg wp(choose(a,A),\neg(b = a))) \\ = & \quad \{ \text{ Apply } wp(choose(b,B),Ra) = B \neq \emptyset \wedge\; (\forall\, u : u \in B : Ra^b_u) \text{ to outer } wp \;\} \\ & B \neq \emptyset \wedge\; (\forall\, v : v \in B :\quad \neg wp(choose(a,A),\neg(v = a))) \\ = & \quad \{ \text{ Similar expansion of inner } wp \;\} \\ & B \neq \emptyset \wedge\; (\forall\, v : v \in B : \neg(\; A \neq \emptyset \;\wedge\; (\forall\, u : u \in A : \neg(v = u))\;)\;) \\ = & \quad \{ \text{ Carry negation inside quantifier } \} \\ & B \neq \emptyset \wedge\; (\forall\, v : v \in B : (\; A \neq \emptyset \vee (\exists\, u : u \in B : v{=}u)\;) \\ = & \quad \{\; A \neq \emptyset,\, B \neq \emptyset \text{ assumed. } \} \\ & \textbf{true} \end{array}$$

5 Discussions and Conclusions

As one would expect from the equivalence of the formulas, the various proof methods reached the same conclusions in each of the examples. However, the different types of reasoning in each form lead to (subjective) differences in ease of use.

The methods are all fairly confusing to work with at first. The Morris method requires both a quantifier and reasoning about 'all possible' postconditions. The

Prins/Griesmethod avoids these in the initial predicate, but the double negation that achieves the same effect is initially confusing. The Chen/Udding method has neither quantification nor double negation, but requires reasoning about two unspecified postconditions.

Length of proofs is hard to measure fairly. In the first two examples the Prins/Gries method proofs are both short and easy to arrive at. Explanations of removing double negation are significantly easier than the ones for removing existential quantification. At least for these examples, the Prins/Gries proofs are practically a mechanical result simplifying *wp* expressions.

The two Chen/Udding proofs are short and involve little symbol manipulation. The step at which *Ra* and *Rc* are argued to be equal after substitution is, at least initially, surprising. They are the crux of the proofs; the preceding steps are aiming at precisely this point at which the two postconditions can be declared equal, allowing the equation to collapse. It might be useful to have special conditions that allow one to be able to start a proof *after* this step, when there is neither postcondition quantification, value quantification, nor double negation.

In the final example, the definitions of *wp* for *choose*(a,A) introduce two quantifications into both the Prins/Gries and Morris forms. In the Morris method, the quantifications of abstract and 'concrete' statements are on opposite sides of an implication and can be removed quite easily. In the Prins/Gries method, the quantifications are nested, and there is further complication due to negation and variable renaming.

Acknowledgements This paper was aided by numerous discussions with David Gries and the Polya research group at Cornell University. Carroll Morris provided useful comments on an earlier version.

References

[CU89] Wei Chen and Jan Tijmen Udding. Towards a calculus of data refinement. In *Lecture Notes in Computer Science, No. 375: Mathematics of Program Construction.* Springer-Verlag, 1989.

[Mor87] J.M. Morris. A theoretical basis for stepwise refinement and the programming calculus. *Science of Computer Programming*, 9:287–306, 1987.

[PG85] J. Prins and D. Gries. A new notion of encapsulation. In *Proceedings of Symposium on Language Issues in Programming Environments.* SIGPLAN, June 1985.