CORRECTNESS OF PROGRAM TRANSFORMATIONS VIA THE WEAKEST PRE-CONDITION FORMALISM OF DIJKSTRA

Dennis Kibler

TECHNICAL REPORT #90
July 1, 1976

Correctness of Program Transformations via the Weakest Pre-condition Formalism of Dijkstra

Dennis Kibler

July 1,1976

TR #90

Abstract: Dijkstra's weakest pre-condition formalism for proving correctness of programs is modified and extended to show the validity of several source-to-source transformations. Examples of the method developed include transformations involving goto elimination, loop fusion and splitting, distribution over conditionals, commutativity of statements, and removal of the empty statement.

Software errors are taking an increasing percentage of the computer dollar. Empirical evidence suggests that half of all programming errors are simple clerical ones (Boehm 1), e.g. exceeding array bounds, changing data types, using the wrong format, etc. Also the amount of software produced per programmer, measured in the number of lines of code per day, appears to be independent of the level of language used. One approach that both increases the productivity of a programmer and decreases the number of errors, is to allow the programmer to use an abstract very high level language which protects him from clerical details. This abstract program is then either automatically translated into a lower language form or, more realistically, the translation is guided by the programmer in an interactive way. Moreover transformations might be suggested by the system which could massage the program into one which required less storage, executed faster, or was more pleasing to the eye. A transformational system of this sort is being developed by Standish et al (2,3).

A programmer manipulating his work through a series of transformations needs to be assured that the changes made do not invalidate his work, i.e. do not introduce any new errors. The entire transformational system rests on the fact that a correct program will be transformed into another correct program which performs the same task. Not only must a transformation map a program into another syntactically valid program, but the image program must be semantically equivalent to the original program. We will assure the syntactic validity of a transformation and concentrate on the problem of showing that it preserves correctness. To this aim we adopt the framework of Dijkstra (4).

Before introducing Dijkstra´s ideas perhaps it would be useful to discuss what we mean by proving a program correct. Immediately we are struck with the fact that when we prove a program is correct, we are not guaranteeing that it will will execute properly. We are, however, increasing our belief that the program will run properly. For example, if one uses Floyd´s assertion method then one is implicitly assuming the correctness of i) predicate calculus, ii) machine implementation of instructions, and iii) the compiler. But this is not a new story. In mathematical theories one always starts with axioms and definitions and builds from there. Since our belief in the validity of predicate calculus and of machine execution is great, a proof of correctness gives us confidence in a program. If a program that had been proven to be correct did not run, one would suspect a compiler error.

Dijkstra's idea for defining semantics is to associate with each program construct or mechanism S a predicate transformer, written wp(S,R), which maps an arbitrary predicate R into the predicate wp(S,R) and wp(S,R) is the weakest pre-condition such that its validity followed by the executions of S guarantees the validity of R. Alternatively, one could define wp(S,R) in the Hoare formalinm by demanding that it satisfy the following two contraints:

i) $\{wp(S,R)\}\ S\ \{R\}$ and

ii) if $\{P\}\ S\ \{R\}$ then $P \Rightarrow wp(S,R)$.

Dijkstra would say that if the input predicate implied wp(program, output predicate), then the program is semantically valid. We wish to define the semantic validity of a transformation. A _block_ is a sequence of statements with a single entrance and exit. Two blocks S and S' are _semantically equivalent_ iff wp(S,R)=wp(S',R) for all predicates R and the blocks have the same entrance and exit points. Notice that if a program is semantically valid and it contains a block S which is semantically equivalent to a block S', then the new program formed by replacing S by S' is also semantically valid. Actually the new program program is valid as long as $wp(S,R) \Rightarrow wp(S',R)$. A transformation T _preserves correctness_ if for all predicates R, $wp(S,R) \Rightarrow wp(T(S),R)$. A transformation is _equivalence preserving_ (and so may be applied in either direction) if for all predicates R, wp(S,R)= wp(T(S),R).

Before we can prove that a particular transformation is semantically valid, the semantics of the particular programming language must be defined. In figure 1. we give the semantics of some simple program constructs. Also listed in this figure are some general laws or properties about the weakest pre-condition which will aid us in verifying the correctness of transformations. The validity of these properties is established by Dijkstra in (4).

Example 1. Simplifying Conditionals

Consider the transformation of

if B then (if A then S1 else S2) else S2

into

if $B \wedge A$ then S1 else S2.

The computation of the weakest pre-conditions proceeds as follows, with the numbers referring to figure 1.

$$\mathrm{wp}(\text{if } B \text{ then (if } A \text{ then } S1 \text{ else } S2) \text{ else } S2, R)$$

$$\Updownarrow \text{(by 5a)}$$

$$B \wedge \mathrm{wp}(\text{if } A \text{ then } S1 \text{ else } S2, R) \vee \sim B \wedge \mathrm{wp}(S2,R)$$

$$\Updownarrow \text{(by 5a)}$$

$$B \wedge (A \wedge \mathrm{wp}(S1,R) \vee \sim A \wedge \mathrm{wp}(S2,R)) \vee \sim B \wedge \mathrm{wp}(S2,R)$$

$$\Updownarrow \text{(by distribution)}$$

$$B \wedge A \wedge \mathrm{wp}(S1,R) \vee B \wedge \sim A \wedge \mathrm{wp}(S2,R) \vee \sim B \wedge \mathrm{wp}(S2,R)$$

$$\Updownarrow \text{(since } B \wedge \sim A \vee \sim B = \sim(A \wedge B))$$

$$B \wedge A \wedge \mathrm{wp}(S1,R) \vee \sim(B \wedge A) \wedge \mathrm{wp}(S2,R)$$

$\Updownarrow$ (by 5a)

wp(if $B \wedge A$ then S1 else S2,R).

This computation has shown that the transformation is an equivalence and that no enabling conditions are necessary. If the two predicates are not equal then any predicate which implies their equality will be a sufficient enabling condition. Hence the weakest pre-condition calculation will also suggest enabling conditions.

Example 2. Commutativity of Statements

The transformation of S1;S2 into S2;S1 requires that wp(S1,wp(S2,R))=wp(S2,wp(S1,R)) for all R. In general there is no way to prove this so it becomes the enabling condition. To discuss a particular example of this transformation let Sub(T,X1,Y1,X2,Y2,...) denote T where X1 has been replaced by Y1, X2 has been replaced by Y2, etc. Now if S1 is X:=X+1 and S2 is X:=X+3 then

wp(S1;S2,R)

$\Updownarrow$ (by 2)

wp(S1,wp(S2,R))

$\Updownarrow$ (by 1)

wp(S1,Sub(R,X,X+1))

$\Updownarrow$ (by 1)

Sub(Sub(R,X,X+1),X,X+3)

$\Updownarrow$ (by simple manipulation)

Sub(R,X,X+4)

Similarly one checks that wp(S2;S1,R) is Sub(R,X,X+4) so the predicates are equal and the statements may be commuted. If however S1 were X:=6*Y and S2 were Y:=6*X then wp(S1;S2,R)=Sub(R,Y,36*Y,X,6*Y) while wp(S2;S1,R)=Sub(R,X,36*X,Y,6*X). Since these expressions are not equal the statements may not be permuted.

Example 3. Generation of Enabling Conditions

One transformation listed in (3) maps

while B do empty into empty.

Let us compute the weakest pre-condition of each statement. From figure 1. we easily see that wp("empty",R)=R. The computation of the while form is somewhat more complicated in general, but in this instance is fairly simple. By 6 $H(\emptyset,R)=\sim B\wedge R$. Again by 6 we have that $H(1,R)=B\wedge H(\emptyset,R)\vee\sim B\wedge H(\emptyset,R)\vee H(\emptyset,R)$. But this reduces to $H(1,R)=\sim B\wedge R$. By a simple inductive argument we find that $H(k,R)=\sim B\wedge R$. Hence the weakest pre-condition of the while form above is $\sim B\wedge R$. In order that these two predicates be equal we require the enabling condition $\sim B$. But note that the original left hand side is really an infinite loop unless B is true. Hence by a straightforward application of Dijkstra´s methods we discover an error in the catalogue and the correcting enabling condition.

Example 4. Movement over Conditionals

Consider the transformation of

if B then C; if not B then D

into

if B then C else D.

Let us now apply the rules contained in figure 1. to calculate the weakest pre-condition of each program fragment.

$$wp(\text{ if } B \text{ then } C; \text{if not } B \text{ then } D; R)$$

$$\Updownarrow \text{(by 2)}$$

$$wp(\text{if } B \text{ then } C, wp(\text{if not } B \text{ then } D, R))$$

$$\Updownarrow \text{(by 5b)}$$

$$wp(\text{if } B \text{ then } C, \sim B \wedge wp(D,R) \vee B \wedge R)$$

$$\Updownarrow \text{(by several simplifications)}$$

$$B \wedge wp(C, B \wedge R) \vee B \wedge wp(C, \sim B \wedge wp(D,R)) \vee \sim B \wedge wp(D,R)$$

If $B \wedge wp(C,R)=wp(C,B \wedge R)$ then we say that B is invariant with respect to C. We assert that if B is invariant with respect to C then so is not B (see appendix for proof). In order to carry on the calculation we assume that B is invariant with respect to C. Hence $B \wedge wp(C,B \wedge R)$ simplifies to $B \wedge wp(C,R)$ and $B \wedge wp(C, \sim B \wedge wp(D,R)$ simplifies to $wp(C, B \wedge \sim B \wedge wp(D,R))$ which is F. Now the weakest pre-condition of the entire first program fragment reduces to $B \wedge wp(C,R) \vee \sim B \wedge wp(D,R)$ which is exactly the weakest pre-condition of the second fragment. Hence under the enabling condition of the invariance of B with respect to C, the two program fragments are equivalent and the transformation is valid.

Transformations such as

while false do S

```
                into
                empty
                 or
        S ; if B then C else D
                into
        if B then (S;C) else (S;D)
```

can be shown to be correct by analogous manipulations of the weakest pre-condition.

Example 5. Loop Splitting

We wish to show the equivalence of the fragments

```
        for I:=1 to n+m do S(I)
                  and
        for I:=1 to n do S(I);
      for I:= n+1 to n+m do S(I)
```

We define the semantics of "for I:=1 to n do S(I)" to be the semantics of the composition "S(1);S(2);...S(n)". With this interpretation the verification that the above program fragments have the same weakest pre-conditions is a triviality. By enlarging this interpretation we could treat arbitrary fixed increments in a loop.

Example 6. Goto Elimination

We wish to prove that the following transformation is valid

```
    Go to A                Go to B
       .                      .
       .                      .
    A: Go to B     =>      empty
       .                      .
       .                      .
```

B:S B:S .

The defintion of a semantically valid transformation contained two constraints. One of these we have ignored since it has always been satisfied up to now. We demanded that the code pieces each have only one entrance and one exit and that these points were the same for each of the two fragments. In the current case this means that an enabling condition is that the label A never be the destination of a goto other than the one under consideration. Since wp(go to,R)=R , the fragments are easily seen to be equivalent.

Example 7. Transforming the While Statement

Let S1 be the construct "while B do S" and let S2 be the construct "if B do S; while B do S". To show the equivalence of these two constructs we must show that wp(S1,R)=wp(S2,R) for all R. Handling the while statement is somewhat complicated so the proof will be divided into two cases.

Case 1. Assume B is not initially true.

By 6 we have that $wp(S1,R)=H(k,R)$ for some k where $H(0,R)=R\wedge\sim B$ and for k greater than zero, $H(k,R)=B\wedge wp(S1,R)\vee\sim B\wedge H(k-1,R)\vee H(0,R)$. In the case that B is false this simplifies to $H(k,R)=\sim B\wedge H(k-1,R)\vee H(0,R)$. If we let k be one we see that $H(1,R)=H(0,R)$. By a simple inductive argument we have $H(k,R)=H(0,R)$ for all k so $wp(S1,R)=\sim B\wedge R$. Using this fact we simplify wp(S2,R) to wp(if B then S,$\sim B\wedge R$) which easily reduces (by 5b) to $\sim B \wedge R$. Hence in the case that B is false, S1 and S2 are equivalent program fragments.

Case 2. Now we assume that B is initially true.

It suffices to show that B wp(S1,R)=B wp(S2,R).

$$B \wedge wp(S2,R)$$

$$\Updownarrow \text{(by 2)}$$

$$B \wedge wp(\text{if } B \text{ do } S, wp(\text{while } B \text{ do } S,R))$$

$$\Updownarrow \text{(by 6)}$$

$$B \wedge wp(\text{if } B \text{ do } S, H(k,R) \text{ for some } k)$$

$$\Updownarrow \text{(by property 5)}$$

$$B \wedge wp(\text{if } B \text{ do } S, H(s,R)) \text{ for some } s$$

$$\Updownarrow \text{(since } H(\emptyset,R) \wedge B=F)$$

$$B \wedge (wp(\text{if } B \text{ do } S, H(s,R)) \text{ for some } s, \vee H(\emptyset,R))$$

$$\Updownarrow \text{(by definition of } H(k,R))$$

$$B\ H(s+1,R) \text{ for some } s$$

$$\Updownarrow \text{(since } H(\emptyset,R) \wedge B=F)$$

$$B \wedge H(s,R) \text{ for some } s$$

$$\Updownarrow \text{(since } H(\emptyset,R) \wedge B=F)$$

$$B \wedge wp(S1,R).$$

This completes the proof for the second case, so the two program fragments are equivalent.

Conclusions.

In this paper we have seen many examples of verifying the correctness of transformations via calculation of the weakest pre-condition. After doing a few calculations of this sort, the method seems clear and straightforward. The proofs are short and even suggest the enabling condition when the calculated pre-conditions are not equal. This treatment can probably be extended to include array variables, and with more difficulty, arbitrary structures. The extension to include procedures and especially recursive procedures may be completely unwieldy.

Acknowledgement: I would like to thank Tim Winkler for directing me to the ideas of Dijkstra as presented in *A Discipline of Programming*(4).

Figure 1.

General Properties of Program Mechanisms

Property 1. $wp(S,F)=F$.

Property 2. if $Q \Rightarrow R$ then $wp(S,Q) \Rightarrow wp(S,R)$.

Property 3. $wp(S,Q) \wedge wp(S,R) = wp(S,Q \wedge R)$.

Property 4. $wp(S,Q) \vee wp(S,R) = wp(S,Q \vee R)$.

Property 5. If for each non-negative integer r there is a predicate C(r) and for each r, C(r) implies C(r+1) then

$wp(S, \text{for some } r, C(r)) = wp(S,C(s))$ for some s.

Semantic Definitions of some Program Mechanisms

1. Assignment: $wp(x:=y,R)=sub(x,y,R)$.

2. Concatenation: $wp(S1;S2,R)=wp(S1,wp(S2,R))$.

3. Empty statement: $wp(empty,R)=R$.

4. Goto: $wp(goto,R)=R$.

5. Conditionals: a) $wp(\text{if } B \text{ then } S1 \text{ else } S2,R)=$

$$B \wedge wp(S1,R) \vee \sim B \wedge wp(S2,R).$$

b) $wp(\text{if } B \text{ then } S,R)=B \wedge wp(S,R) \vee \sim B \wedge R$.

6. Iteration: $wp(\text{while } B \text{ do } S,R)=H(k,R)$ for some k, where

$H(\emptyset,R)= R \wedge \sim B$

$H(k,R)=wp(\text{if } B \text{ then } S,H(k-1,R))) \vee H(\emptyset,R)$

$=B \wedge wp(S,H(k-1,R)) \vee \sim B \wedge H(k-1,R) \vee H(\emptyset,R)$.

(note $H(k,R)$ implies $H(k+1,R)$).

Appendix

Lemma: If B is invariant with respect to C then B is also invariant with respect to C.

Proof: We are given that $B \wedge wp(C,R) = wp(C, B \wedge R)$ and would like to show that $\sim B \wedge wp(C,R) = wp(C, \sim B \wedge R)$. The proof reduces to two computations. First we calculate $\sim B \wedge wp(C,R)$.

$$\sim B \wedge wp(C,R)$$

$$\Updownarrow \text{(since } B \vee \sim B = T\text{)}$$

$$\sim B \wedge wp(C,(B \vee \sim B) \wedge R)$$

$$\Updownarrow \text{(by property 4)}$$

$$\sim B \wedge wp(C,B\ R) \vee \sim B \wedge wp(C, \sim B \wedge R)$$

$$\Updownarrow \text{(by invariance of B)}$$

$$\sim B \wedge B \wedge wp(C,R) \vee \sim B \wedge wp(C, \sim B \wedge R)$$

$$\Updownarrow \text{(immediately)}$$

$$\sim B \wedge wp(C, \sim B \wedge R).$$

Now we simplify $wp(C, \sim B \wedge R)$.

$$wp(C, \sim B \wedge R)$$

$$\Updownarrow \text{(since } B \vee \sim B = T\text{)}$$

$$(B \vee \sim B) \wedge wp(C, \sim B \wedge R)$$

$$\Updownarrow \text{(by invariance of B)}$$

$$\sim B \wedge wp(C, \sim B \wedge R) \vee wp(C, B \wedge \sim B \wedge R)$$

$$\Updownarrow \text{(since } wp(S,F) = F\text{)}$$

$$\sim B \wedge wp(C, \sim B \wedge R).$$

These simplifications show that B is invariant with respect to C.

Bibliography

1.Boehm,B.W.,Software and its Impact:A Quantitative Assessment,P.4946,Rand Corp.,Santa Monica, Ca.,Dec.1972.

2.Standish,T.,Harriman,D.,Kibler,D.,and Neighbors,J., The Irvine Program Transformation Catalogue Dept. of Information and Computer Science,Univ., Irvine.Ca.,(Jan.1976).

3.Standish,T.,et al, Improving and Refining Programs by Program Manipulation,Dept. of Information and Computer Science, Univ. of Ca. at Irvine,Ca. (Feb.1976).

4.Dijkstra,E.W.,A Discipline of Programming,Prentice-Hall, Englewood Cliffs, N.J.,(1976).

5.Dahl,O.-J.,Dijkstra,E.W., and Hoare,C.A.R., Structured Programming, Academic Press, London, (1972).

6.Manna,S.,Mathematical Theory of Computation,McGraw-Hill Inc.,(1974).