A Technique for Analyzing the Effects of Changes in Formal Specifications

D. Richard Kuhn

National Institute of Standards and Technology
Technology Administration
U.S. Dept. of Commerce
Gaithersburg, Maryland 20899 USA

Reprinted from *The Computer Journal (British Computer Society)* , Vol. 35, No. 6 (December, 1992)

ABSTRACT

Formal specifications are increasingly used in modeling software systems. An important aspect of a model is its value as an analytical tool to investigate the effect of changes. This paper defines the notion of *predicate differences* and shows how predicate differences may be used to analyze the effects of changes in formal specifications. Predicate differences have both theoretical and practical applications. As a theoretical tool, predicate differences may be used to define a meaning for the "size" of a change to a formal specification. Practical applications include analyzing the effect of design changes on a previously verified design; defining an affinity function for reusable software components; computing slices of formal specifications, similar to program slices; investigating the conditions under which invalid assumptions will render a system non-secure; and formalizing the database inference problem.

1. Introduction

Formal specifications are increasingly used in verifying that a design meets critical requirements, such as safety or security. In addition to design verification, formal models are useful as analytical tools, to answer questions about how the system will behave in various circumstances. A model should also be useful to investigate the effect of changes to design or requirements. For example, suppose a design P is stated formally, then shown to meet the requirements specification S through a formal proof that $P \Rightarrow S$. The design may be changed from P to P', so that verifying the new design requires showing $P' \Rightarrow S$. Depending on the formulas involved, changing the value of a variable x may or may not affect the truth of the implication. In general, the values of other terms will determine whether a change in the value of x will change the implication $P \Rightarrow S$. This paper defines the notion of *predicate differences* and shows how predicate differences may be used to analyze the effects of changes in formal specifications. This paper extends the work described in [Kuhn, 1991].

Predicate differences might be used in formal specification language tools to compute "predicate slices" from formal specifications, similar to the program slices defined by Weiser [1984]. A program slice selects all lines from a program that may directly or indirectly affect the value of a particular variable at a particular point. Computing the predicate difference for a substitution in a formal specification gives the conditions under which the change makes a difference, in effect a "slice" through the specification.

The changes that will be considered in this paper are those that are made by replacing some variable x with an expression e in a predicate formula or subformula. This is denoted P^x_e. (The notation P^x_e represents predicate P with every free occurence of variable x replaced by expression e, with suitable renaming to prevent variable capture. The symbols &, |, ¬, ⇒ represent *and, or, not, implies,* respectively. The exclusive or operation is denoted by $\oplus$.) In some cases, additional terms may be added to the formula.

For example, suppose an invariant is $A \& B \& C \& D \Rightarrow S$, and it is changed to $G \& B \& C \& D \Rightarrow S$. The desired new invariant $G \& B \& C \& D \Rightarrow S$ is given by $(A \& B \& C \& D)^A_G \Rightarrow S$. When the invariant is a Boolean formula, the effect of such a change can be determined using the Boolean difference. The predicate difference, introduced in Section 3, can be used to determine the effects of changes in predicate calculus formulas. It will be helpful in discussing the predicate difference to first review the properties of the Boolean difference.

2. Boolean Difference

The Boolean difference [Reed, 1954; Akers, 1959], can be used to calculate the dependency of a Boolean function on a literal x_i of that function. The Boolean difference of x_i with respect to F, dF/dx_i, gives the conditions under which the value of F will change if the value of x_i changes. Boolean differences have been used in digital circuit testing [Marinos, 1971], [Reed, 1973] and in computer security access control [Trueblood and Sengupta, 1986]. The Boolean difference has been generalized to multi-valued logic for VLSI circuit testing [Bell et. al, 1972], [Lu and Lee, 1984], and [Whitney and Muzio, 1988].

For a function $F = f(x_1, ..., x_i, ..., x_n)$, the Boolean difference of F with respect to x_i is

$$dF/dx_i = f(x_1, ..., x_i, ..., x_n) \oplus f(x_1, ..., \neg x_i, ..., x_n).$$

This is equivalent to

$$dF/dx_i = f(x_1, ..., 0, ..., x_n) \oplus f(x_1, ..., 1, ..., x_n),$$

which follows from the fact that x_i must be either 0 or 1. The difference dF/dx_i is an expression that does not contain x_i.

A useful property of the Boolean difference is that

$$dF/dx_i = \begin{cases} 1 & \textit{if F is unconditionally dependent on } x_i \\ 0 & \textit{if F is unconditionally independent of } x_i \\ F' & \textit{an expression not containing } x_i\textit{, otherwise} \end{cases}$$

To see intuitively why dF/dx_i gives the conditions under which a change in the value of x_i will change the value of F, consider that F will change if either (a) it is initially true and changing the value of x_i makes it false: $F \ \& \ \neg(F^{x}_{\neg x})$, or (b) it is initially false and changing the value of x_i makes it true: $\neg F \ \& \ (F^{x}_{\neg x})$. Note that the disjunction of (a) and (b) is, by definition, the exclusive OR.

The Boolean difference of a function $F = f(F_1, ..., F_n)$, with respect to one of its component functions F_i is

$$dF/dF_i = f(F_1, ..., F_i, ..., F_n) \oplus f(F_1, ..., \neg F_i, ..., F_n).$$

The *partial Boolean difference* gives the effect on the truth value of a Boolean formula of a component of the formula, through a particular term. For a formula $F = f(F_1, ..., F_n)$, the partial Boolean difference of F with respect to F_i with respect to a variable x_j of F_i, is

$$dF/d(x_j \mid F_i) = dF/dF_i \ \& \ dF_i/dx_j$$

3. Predicate Difference

The Boolean difference suggests a similar *predicate difference* in predicate calculus. The properties of the predicate difference are similar to those of the Boolean difference. However, the Boolean difference with respect to a term gives the conditions under which a change in the value of the term will change the value of the Boolean function. A Boolean term can change only from x to $\neg x$. The change to a predicate depends on the expression substituted for x. Thus a predicate difference is with respect to a particular change x/e (the substitution of expression e for free variable x), rather than simply with respect to x. Note also that the predicate difference with respect to a change x/e may still contain x.

Definition 1. *Independence*: P is *independent* of x/e when P has the same truth value as P^x_e, i.e. $P \equiv P^x_e$.

Definition 2. *Dependence*: If P is not independent of x/e, then P is *dependent* on the value of x/e.

Definition 3. *Predicate difference*: The *predicate difference* for a predicate P with respect to variable substitution x/e, denoted dP^x_e, is $P \oplus P^x_e$.

The following lemmas establish some properties of the predicate difference that are used in the remainder of the paper.

Lemma 1. dP^x_e = 0 iff P is independent of the value of x/e.

Proof.

Assume ("if" direction) $P \equiv P^x_e$ {definition of independence}

Then $(P \Leftrightarrow P)$

$\equiv (P \oplus P = 0)$ {definition of $\oplus$}

$\equiv (P \oplus P^x_e = 0)$ {Definition 3}

Assume $(P \oplus P^x_e = 0)$ ("only if" direction)

$\equiv (\neg(P \Leftrightarrow P^x_e) = 0)$ {definition of $\oplus$}

$\equiv (P \Leftrightarrow P^x_e)$

□

Definition 4. *Unconditional Dependence*: P is *unconditionally dependent* on x/e if P has the opposite truth value of P^x_e, i.e. $(P \Leftrightarrow \neg P^x_e)$ & $(P^x_e \Leftrightarrow \neg P)$

Lemma 2. dP^x_e = 1 iff P is unconditionally dependent on the value of x/e.

Proof.

$(P \Leftrightarrow \neg P^x_e)$ & $(P^x_e \Leftrightarrow \neg P)$

$\equiv \neg(P \Leftrightarrow P^x_e)$

$\equiv (P \oplus P^x_e) = 1$

□

If dP^x_e is not 0 and not 1, then the resulting formula can be solved for 1 to determine the conditions under which P^x_e will be dependent on x. Note that if e is a Boolean term and $e = \neg x$ in a propositional formula, the predicate difference is equivalent to the Boolean difference.

The predicate difference can also be derived using predicate transforms. The predicate transform $wp(''x := e'', R)$ gives the minimal conditions under which an assignment in a program will result in the condition specified by R [Dijkstra, 1976; Gries, 1987]. That is, if $Q \Rightarrow wp(''x := e'', R)$, then execution of $x := e$ in a state in which Q holds will result in a state in which R holds. The transform $wp(''x := e'', R)$ is R^x_e. The predicate difference gives the conditions under which a substitution will change the value of a predicate, say R; that is, the conditions under which $\neg(R \Leftrightarrow wp(''x := e'', R))$. But, $\neg(R \Leftrightarrow wp(''x := e'', R)) \equiv \neg(R \Leftrightarrow R^x_e) \equiv R \oplus R^x_e \equiv dR^x_e$.

4. Partial Predicate Difference

The predicate difference of a predicate formula $F = f(F_1,...,F_n)$, consisting of component formulas connected by &, |, or $\Rightarrow$ with respect to one of its component formulas F_i is

$$dF/dF_i = f(F_1, ..., F_i, ..., F_n) \oplus f(F_1, ..., \neg F_i, ..., F_n).$$

Definition 5. *Partial Predicate difference*: the *partial predicate difference* gives the effect on a formula of a component of the formula, through a change in a particular term. For a formula

$F = f(F_{1,} ..., F_n)$,

the partial predicate difference of F with respect to F_i with respect to a change in a variable x_j/e of F_i, is

$$dF/d(F_i)_e^{x_j} = dF^{F_i}_{\neg F_i} \ \& \ dF_{i\,e}^{\ x_j}$$

5. The Size of Changes to Predicates

How "big" is a change? A metric for changes to a predicate can be defined by using the predicate difference to define an ordering relation: $x/e \le z/f$ if $dP_e^x \Rightarrow dP_f^z$ (x may equal z and e may equal f). Also define $x/e < z/f$ if $dP_e^x \Rightarrow dP_f^z$ but not $dP_f^z \Rightarrow dP_e^x$. The ordering $x/e \le z/f$ expresses the fact that the change x/e is "smaller" than z/f. The smallest change x/e is no change at all, where $dP_e^x = 0$, as shown in Lemma 1.

5.1. Example

Given a predicate $(a \mid b)$, does a/c represent a bigger or smaller change than $a/(a \mid c)$? The predicate difference $d(a \mid b)_c^a$ is $\neg a \ \& \ \neg b \ \& \ c \mid a \ \& \ \neg b \ \& \ \neg c$, and $d(a \mid b)_{a \mid c}^a$ is $\neg a \ \& \ \neg b \ \& \ c$. So $d(a \mid b)_{a \mid c}^a \Rightarrow d(a \mid b)_c^a$, i.e., a/c is a bigger change than $a/(a \mid c)$. Although the substitution $a/(a \mid c)$ is a greater *textual* change than a/c, the predicate that results from $a/(a \mid c)$ simply enlarges the number of states (since $a \mid b \Rightarrow a \mid b \mid c$), but a/c changes the predicate to define a different set of states.

□

If $dP_e^x \Rightarrow dP_f^z$ then it can be said that P_e^x differs less from P than does P_f^z. This idea can be generalized to define a meaning for how much two predicates Q and R differ from a third, P. To compare how two predicates Q and R differ from P, the differences $P \oplus Q$ and $P \oplus R$ can be computed. (We do not necessarily know what substitutions x/e, if any, will make P_e^x equal to Q or R.) If $P \oplus Q \Rightarrow P \oplus R$ then Q differs less from P than R, otherwise R differs less than Q (unless $Q \equiv R$). This view makes intuitive sense by noting that, if $P \ne Q \ne R$, then $P \oplus Q \Rightarrow P \oplus R$ is equivalent to $(P \Rightarrow Q \Rightarrow R) \mid (R \Rightarrow Q \Rightarrow P)$. That is, Q is "closer" to P in both sides of the disjunction.

5.2. Application

A possible application for the size metric is for the definition of affinity functions for software components. Briefly, an affinity function estimates the degree of similarity between two components [Gibbs, et al., 1990]. The affinity function for object oriented programs described in [Gibbs et al., 1990] is based on counts of the methods that are common among classes, a purely syntactic feature.

Using the notions developed in this section, a semantics-based affinity function can be defined using the formal specifications of components. Suppose a requirement R is to be implemented. Three components are available, A, B, and C, whose specifications are S_A, S_B, and S_C, respectively. The differences $S_A \oplus R$, $S_B \oplus R$, $S_C \oplus R$, can be computed to determine which of the specifications S_i is closest to the requirement specification R.

6. The Relationship Between Predicate Differences and Boolean Differences

The Boolean difference can be viewed as an "upper bound" on the result of changes to an individual variable in a component formula. In terms of the size metric of the previous section, the change from a variable in a component formula is never larger than the change that results from negating the entire component formula.

The predicate difference of a predicate formula $F = f(F_{1,} \cdots, F_n)$, consisting of component formulas connected by &, |, or $\Rightarrow$, with respect to one of its component formulas F_i is

$$dF^{F_i}_{\neg F_i} = f(F_{1,} \cdots, F_{i,} \cdots, F_n) \oplus f(F_{1,} \cdots, \neg F_{i,} \cdots, F_n)$$

which is equivalent to the Boolean difference of F with respect to F_i. Theorem 1 shows the relationship between this Boolean difference with respect to a component formula and the partial predicate difference with respect to a variable of the component formula.

Theorem 1:

The partial predicate difference of a formula $dF/d(F_i)^x_e$ implies the predicate difference $dF^{F_i}_{\neg F_i}$, where F_i is a component formula of F, and x is some variable in F_i:

$dF/d(F_i)^x_e \Rightarrow dF^{F_i}_{\neg F_i}$.

Proof:

First, determine an expression for $dF^{F_i}_{\neg F_i}$. Without loss of generality, assume that F is converted to conjunctive normal form. Then F_i may appear in one or more conjuncts of the converted formula. With the conjuncts containing F_i appearing first, followed by conjuncts not containing F_i, the formula appears as follows: (G,H etc. are component subformulas not containing F_i.)

$(F_i \mid G_1 \mid G_2 \mid \cdots) \quad \& \ (F_i \mid H_1 \mid H_2 \mid \cdots) \quad \& \ (F_i \mid J_1 \mid J_2 \mid \cdots) \cdots \quad \& \ (K_1 \mid K_2 \mid \cdots)$
$\& \ (L_1 \mid L_2 \mid \cdots) \cdots$.

Let the conjuncts not containing F_i be abbreviated by R. Then the predicate difference with respect to F_i is

$(F_i \mid G_1 \mid G_2 \mid \cdots) \;\&\; (F_i \mid H_1 \mid H_2 \mid \cdots) \;\&\; (F_i \mid J_1 \mid J_2 \mid \cdots) \cdots \;\&\; R$

$\quad \oplus (\neg F_i \mid G_1 \mid G_2 \mid \cdots) \;\&\; (\neg F_i \mid H_1 \mid H_2 \mid \cdots) \;\&\; (\neg F_i \mid J_1 \mid J_2 \mid \cdots) \cdots \;\&\; R.$

Factoring F_i out of the formula up to R gives

$(F_i \mid ((G_1 \mid G_2 \mid \cdots) \;\&\; (H_1 \mid H_2 \mid \cdots) \;\&\; (J_1 \mid J_2 \mid \cdots) \cdots)) \;\&\; R$

$\quad \oplus (\neg F_i \mid ((G_1 \mid G_2 \mid \cdots) \;\&\; (H_1 \mid H_2 \mid \cdots) \;\&\; (J_1 \mid J_2 \mid \cdots) \cdots)) \;\&\; R.$

This is equal to

$(F_i \oplus \neg F_i) \;\&\; \neg((G_1 \mid G_2 \mid \cdots) \;\&\; (H_1 \mid H_2 \mid \cdots) \;\&\; (J_1 \mid J_2 \mid \cdots) \cdots) \;\&\; R.$

And since $X \oplus \neg X$ is 1, this is equal to

$\quad \neg((G_1 \mid G_2 \mid \cdots) \;\&\; (H_1 \mid H_2 \mid \cdots) \;\&\; (J_1 \mid J_2 \mid \cdots) \cdots) \;\&\; R.$

Computing $dF/d(F_i)^x_e$ in the same way shows that

$dF/d(F_i)^x_e \equiv (F_i \oplus (F_i)^x_e) \;\&\; \neg((G_1 \mid G_2 \mid \cdots) \;\&\; (H_1 \mid H_2 \mid \cdots) \;\&\; (J_1 \mid J_2 \mid \cdots) \cdots) \;\&\; R,$

so

$dF/d(F_i)^x_e \Rightarrow dF^{F_i}_{\neg F_i}.$

□

Corollary

Applying modus tollens gives the following corollary:

If the Boolean difference is 0, then the predicate difference is 0 as well:

$(dF^{F_i}_{\neg F_i}) = 0 \Rightarrow (dF/d(F_i)^{x_j}_{e^j}) = 0.$

The next section discusses how predicate differences can be used to evaluate the effects of changes in specifications.

7. Analyzing the Effects of Changes

Let I be an invariant $P \Rightarrow S$. To determine the effect on the invariant I of changing variable x in P to e, the partial predicate difference dI/dP^x_e can be computed. This gives the conditions under which the invariant will change value, in other words, the conditions under which it becomes false, since it was true before the change.

After analyzing the effect of the change, if a conjunct M is added to the antecedent, it is necessary to show that the new antecedent maintains the invariant. There are then two ways to proceed with showing that the modified antecedent $(P^x_e) \;\&\; M$ maintains the invariant. The first is to show directly that $P^x_e \;\&\; M \Rightarrow S$. The second is to show that the modification guarantees that the invariant will not change value by showing that the conditions under which it becomes false do not occur, i.e., the antecedent $P^x_e \;\&\; M$ implies the negation of the partial predicate difference dI/dP^x_e, i.e.: $P^x_e \;\&\; M \Rightarrow \neg\left[dI/dP^x_e\right]$. Proving this is equivalent to proving $P^x_e \;\&\; M \Rightarrow S$ directly. This result is proved formally below.

Theorem 2. $\left[(P \Rightarrow S)\ \&\ \left[M \Rightarrow \neg\left[dI/dP_e^x\right]\right]\right] \Leftrightarrow [(P \Rightarrow S)\ \&\ (P_e^x\ \&\ M \Rightarrow S)]$

Proof

$(P \Rightarrow S)\ \&\ \left[M \Rightarrow \neg\left[dI/dP_e^x\right]\right]$

$\equiv (P \Rightarrow S)\ \&\ (M \Rightarrow \neg(P_e^x\ \&\ \neg S))$ {Theorem 3 (below)}

$\equiv (P \Rightarrow S)\ \&\ (M \Rightarrow (\neg P_e^x \mid S))$

$\equiv (P \Rightarrow S)\ \&\ (M \Rightarrow P_e^x \Rightarrow S)$

$\equiv (P \Rightarrow S)\ \&\ (M\ \&\ P_e^x \Rightarrow S)$

□

Thus, if the modification term M implies the negation of the predicate difference, the invariant will be preserved.

If the invariant I has already been shown and we wish to modify P to P_e^x, we can compute the conditions under which the value of the invariant will change using the following result:

Theorem 3. Let I be an invariant $P \Rightarrow S$. Then I is dependent on the value assigned to x in P under the conditions given by $\neg P\ \&\ P_e^x\ \&\ \neg S\ \equiv\ P_e^x\ \&\ \neg S$.

Proof

$dI/dP_e^x = (P \oplus P_e^x)\ \&\ \neg S$ {Definition 3 and 5}

$\equiv (P\ \&\ \neg P_e^x \mid \neg P\ \&\ P_e^x)\ \&\ \neg S$ {Definition of $\oplus$}

$\equiv \neg P\ \&\ P_e^x\ \&\ \neg S$ {assumed: $(P \Rightarrow S) \equiv \neg(P\ \&\ \neg S)$}

$\equiv P_e^x\ \&\ \neg S$ {$(P \Rightarrow S) \Rightarrow (\neg P\ \&\ \neg S \equiv \neg S)$}

□

The form $\neg P\ \&\ P_e^x\ \&\ \neg S$ may be more useful if we expect the change x/e to maintain the invariant, because showing either $P_e^x\ \&\ \neg P = 0$ or $P_e^x\ \&\ \neg S = 0$ is sufficient to show that $P_e^x \Rightarrow S$. If $P_e^x\ \&\ \neg P$, is easier to calculate, and the result is 0, then there is no need to compute the predicate difference. Note that by Lemma 1, the invariant is independent of the change if $P_e^x\ \&\ \neg S = 0$, which is equivalent to $P_e^x \Rightarrow S$.

These results can be used to evaluate the effects of changes to parts of large formulas. Suppose a specification $P \Rightarrow S$ is given and $P = A\ \&\ B\ \&\ C\ \&\ D$. A change will be made to C. An additional condition M will be added but the modified antecedent must still meet the requirement S. The new value of P will be $P' = A\ \&\ B\ \&\ C'\ \&\ D\ \&\ M$, and it must be shown that $P' \Rightarrow S$. If $M \Rightarrow \neg dP_{\neg C}^{C}$, then we do not have to compute dP/dC_e^x, because Theorem 1 shows that $M \Rightarrow \neg dP_{\neg C}^{C}$ $\Rightarrow \neg dP/dC_e^x$. This, plus Theorem 2, shows that $P' \Rightarrow S$.

These analysis techniques may be used for a variety of applications. Some examples are considered in the next section.

8. Applications

This section presents some problems to which predicate differences can be applied. The applications presented here are from the field of computer security, but the technique could be used with formal specifications for other types of applications as well.

8.1. Change Analysis Example

Consider a system which uses a token to control access to a network. To gain access, a user must have both a valid token and the right password. The system maintains the following state invariants (among others) as security requirements.

A user is authorized only if the token is authorized:

$(u\_auth \Rightarrow t\_auth)$

A token is authorized only if its password is active (non-zero):

$(t\_auth \Rightarrow pw \neq 0)$

We wish to ensure that the following state transition invariant holds:

A token can be activated (i.e., its password changed from zero to non-zero) only by the security officer:

$(pw' \neq 0 \ \& \ pw = 0 \Rightarrow s\_auth)$

The password changing function is

```
chgpasswd(input_val)
{
/* if security officer, then change password to input value */
        if (s_auth) pw := input_val
}
```

This *chgpasswd* function is modeled by

$(s\_auth \Rightarrow pw' = inval) \ \& \ (\neg s\_auth \Rightarrow pw' = pw)$

A proof is done to show that the state invariants plus the effect of the *chgpasswd* function ensure the state transition invariant (the function must also maintain the invariants, but the proof is omitted for brevity).

$(u\_auth \Rightarrow t\_auth)$ &

$(t\_auth \Rightarrow pw \neq 0)$ &

$(s\_auth \Rightarrow pw' = inval)$ &

$(\neg s\_auth \Rightarrow pw' = pw)$

$\Rightarrow (pw' \neq 0 \;\&\; pw = 0 \Rightarrow s\_auth)$

Suppose that the design is to be changed to allow either the user or the security officer to change passwords, rather than requiring the security officer to do so. The *chgpasswd* function specification then becomes:

$((s\_auth \mid u\_auth) \Rightarrow pw' = inval) \;\&\; (\neg(s\_auth \mid u\_auth) \Rightarrow pw' = pw)$

After making the change to the specification, a new proof must be conducted. If the proof fails, the specification must be analyzed manually to determine why, then appropriate changes made. The conditions under which the change will affect the state transition invariant can be calculated using the predicate difference. As it turns out, the predicate difference is 0, so the change will not affect the invariant. By Theorem 3, the predicate difference is

$(u\_auth \Rightarrow t\_auth)$ &

$(t\_auth \Rightarrow pw \neq 0)$ &

$(u\_auth \mid s\_auth \Rightarrow pw' = inval)$ &

$(\neg(u\_auth \mid s\_auth) \Rightarrow pw' = pw)$ &

$\neg(pw' \neq 0 \;\&\; pw = 0 \Rightarrow s\_auth) \equiv 0$

Depending on the problem, the predicate difference may be either more or less effort to calculate than a new proof. The advantage in computing the predicate difference is in determining the conditions under which a change will render non-secure a system that was previously shown secure.

8.2. Analyzing the Effect of Security Flaws

One important problem in security evaluations is to determine the effect of violations of assumptions. In general, violations of assumptions will affect the security of the system under some conditions, but not make the system non-secure all the time. The predicate difference for a hypothesized violation of assumptions gives the conditions under which the security invariant does not hold.

In a state machine model a proof is given that transitions T_i imply the security invariant S, i.e., $(T_1 \Rightarrow S) \;\&\; (T_2 \Rightarrow S) \;\&\; \cdots \;\&\; (T_n \Rightarrow S)$. A violation of assumptions in a transition, such as the failure of a variable to maintain a specific value, can be modeled by letting an expression e represent the potential new value of a variable x, then computing the predicate difference $d(T \Rightarrow S)^x_e$. The predicate difference gives the conditions under which the invariant will change truth value, that is, the conditions under which the system would not be secure.

8.3. Formalizing the Database Inference Problem

An important topic in computer security and privacy is *database inference,* which concerns the question of how database systems can prevent the inference of confidential information from responses to queries. Although the system may never explicitly release the confidential fact, a user may be able to infer the secret by putting together other facts that are not protected. This problem can be modeled by letting S represent the secret information and F represent the facts that the database has revealed so far. If a new query is given, changing F to F', and $F' \Rightarrow S$, then the secret has effectively been disclosed and security has failed.

The database inference problem can be modeled using predicate differences. Assume that the system has not revealed the secret S, i.e. $\neg(F \Rightarrow S)$ is true. If x is a variable in the predicate F that defines the current set of facts, and a query will give an expression e as the new value of x, then the system is secure if and only if the expression $\neg(F \Rightarrow S)$ is independent of the substitution of e for x. Let I be the invariant $F \,\&\, \neg S$, which is equivalent to $\neg(F \Rightarrow S)$. Then if dI/dF_e^x = 0, the substitution of e for x will not affect the invariant; i.e., secrecy will not be compromised.

9. Conclusions and Future Directions

Predicate differences can be an effective analytical tool for evaluating the effect of changes to formal specifications. They may also be useful in re-verifying specifications after modification; determining if a change will cause a previously secure system to become non-secure; and as a metric for changes to predicates.

Examples presented in this paper were based on real applications, but additional experience is needed to explore the technique. Integrating the calculation of predicate differences into a tool for a formal specification language would make it possible to compute "predicate slices" of the formal specification. Tools to compute predicate slices for popular specification languages such as Z and InaJo could be useful in evolving and maintaining system specifications.

10. Acknowledgements

I am grateful to John Cherniavsky, Jim Lyle, Bill Majurski, and Charles Nicholas for many helpful comments on this work.

11. References

[Akers, 1959] S.B. Akers . "On a Theory of Boolean Functions," *SIAM Journal* Vol. 7, No. 4.

[Bell et. al, 1972] N. Bell, E.W. Page, and M.G. Thomason, "Extension of the Boolean Difference Concept to Multi-valued Logic Systems," *Proceedings of the 1972 Symposium on the Theory and Applications of Multiple-Valued Logic Design*

[Dijkstra, 1976] E.W. Dijkstra, *A Discipline of Programming,* Prentice Hall, Englewood Cliffs, NJ, 1976.

[Gibbs, et al., 1990] S. Gibbs, D. Tsichritzis, E. Casais. "Class Management for Software Communities," CACM, Vol. 33, No. 9, (Sept. 1990).

[Gries, 1987] D. Gries. *The Science of Programming,* Springer Verlag, New York, 1987.

[Kuhn, 1991] D.R. Kuhn. Predicate Differences and the Analysis of Dependencies in Formal Specifications, *14th National Computer Security Conference,* Washington, D.C., October, 1991.

[Lu and Lee, 1984] H. Lu and S.C. Lee, "Fault Detection in M-Logic Circuits Using the M-Difference," *Proceedings of the International Symposium on Multiple Valued Logic,* 1984."

[Marinos, 1971] P.N. Marinos. "Derivation of minimal complete sets of test-input sequences using Boolean differences," *IEEE Transactions on Computers,* Vol. C-20, No. 1.

[Muller, 1954] D.E. Muller. "Application of Boolean Algebra to Switching Circuit Design and Error Detection," *Transactions of the Institute of Radio Engineers,* Vol. EC-3.

[Reed, 1954] I.S. Reed. "A Class of Multiple-error Correcting Codes and the Decoding Scheme," *Transactions of the Institute of Radio Engineers,* Vol. IT-4.

[Reed, 1973] I.S. Reed. "Boolean Difference Calculus and Fault Finding," *SIAM Journal of Applied Mathematics,* Vol. 24, No. 1.

[Trueblood and Sengupta, 1986] R.P. Trueblood and A. Sengupta. "Dynamic Analysis of the Effects Access Rule Modifications Have Upon Security," *IEEE Transactions on Software Engineering,* Vol. SE-12, No. 8 (Aug., 1986).

[Weiser, 1984] M. Weiser, "Program Slicing," *IEEE Transactions on Software Engineering,* Vol. SE-10, No. 7 (July, 1984).

[Whitney and Muzio, 1988] M. Whitney and J. Muzio. "Decisive Differences and Partial Differences for Stuck-at Fault Detection in MVL Circuits," 18th Intl. Symposium on Multiple-Valued Logic, Palma de Mallorca, Spain, May 24-26, 1988.