A Compositional Transformation for Formal Verification

E. Cerny

Dép. d'informatique et de recherche opérationnelle, Université de Montréal
C.P. 6128, Succ. A, Montréal, Québec, H3C 3J7 Canada

Abstract

The conditions under which a conjunction of two relations \( aRb \) and \( bRc \) with existential abstraction of \( b \) can be transformed into an implication \( aRb \Rightarrow bRc \) with universal abstraction of \( b \) are determined. In algorithmic design verification based on tautology checking and automata equivalence this transformation allows to derive new verification algorithms, and to show under which conditions the breadth-first symbolic reachability algorithm used in proving automata equivalence can be applied when the automata are non-deterministic. Boolean characteristic functions of relations that have efficient representation using Binary Decision Diagrams are used in the derivations.

1 Introduction

Algorithmic formal design verification methods, such as model checking and automata equivalence [1-7], are limited in the size of the problem they can handle, in spite of recent advances [6]. To become applicable to realistic problems, hierarchical abstraction and reduction of the automata description with respect to the properties to be verified [8], as well as incremental (modular) approaches using the system's structure [15] must be further developed. When automata reduction is used, the resulting description may become non-deterministic. Also, non-determinism is useful in specifications, as it allows to hide unnecessary implementation details and makes the specification more compact. The problem is that language equivalence of non-deterministic finite state automata (recognizers) has been shown to be P-Space complete [9], generally implying an exponential time algorithm. The question is then whether there exist some properties of the automata used in hardware design and verification that would allow using efficient algorithms such as the symbolic breadth-first reachability analysis [1-7] even in presence of some forms of non-determinism.

We describe here a transformation and its conditions of validity for composing behavioral descriptions expressed in terms of characteristic functions of relations and sets. It is then shown how this transformation can be used to analyze the effect of non-determinism on automata equivalence algorithms. The importance of the transformation is that it opens new possibilities for analyzing the underlying problem. The transformation used in [15] for deriving an incremental tautology checking method is a special case of the transformation presented here.

The notation is based on Boolean Characteristic Functions (CF) [9-13] which can succinctly express relational, set-theoretic, and Boolean algebra related properties. The actual implementation of any resulting algorithm may use Binary Decision Diagrams (BDD) [17, 18]. In Section 2 we review Characteristic Functions and present the transformation. Section 3 discusses its applications, and Section 4 concludes by outlining possible avenues of research.

2 Compositional transformation

Characteristic Functions (CF) are used to describe relations on Boolean spaces [10-14]: Let \( x \in X = B_2^m \), \( y \in Y = B_2^n \). A relation \( X \times Y \supseteq R \) can be described using a CF \( \Phi(x, y) \): \( X \times Y \rightarrow B_2 \) such that \( \Phi(x, y) = 1 \) if \( xRy \), and 0 otherwise. Multiple valued form was discussed in [12].

\[ \sum_y \Phi(x, y) \] represents the inclusive OR of \( \Phi(x, y) \) over all states \( y \in Y \), it is the \( \Sigma \) form of abstraction of the variable(s) \( y \). It corresponds to the quantifier "there is".

\[ \prod_y \Phi(x, y) \] stands for the AND product of \( \Phi(x, y) \) over all states of \( y \in Y \), it is the \( \Pi \) form of variable abstraction. It corresponds to the quantifier "for all".

Let \( C \) be a combinational circuit implementing the vector function \( y = f(x) \), and let \( C(x, y) \) be its CF. It can be easily shown that the equation \( C(x, y) = 1 \) has a unique solution for \( y \) in terms of \( x \) (the function \( f \)), and that

\[ \sum_y C(x, y) = 1 \]
Realization condition: Let \( S(x, y) \) be the CF of a specification \( S \). \( C(x, y) \) implements \( S(x, y) \), where \( \preceq \) signifies implication (not \( C(x, y) \) + \( S(x, y) \) = 1).

Composition rule: Let two circuits \( C_1 \) and \( C_2 \) be (acyclically) connected using signals \( u \) to form a circuit \( C \), and let \( C_1(u, x) \) and \( C_2(u, y) \) be the respective CFs. The CF \( C(x, y) \) of \( C \) can be obtained as:

\[
C(x, y) = \sum_u (C_1(x, u) C_2(u, y)) \quad (2)
\]

Compositional transformation:

Let \( \Phi_1(a, b) \) and \( \Phi_2(b, c) \) be two CFs defined over the Boolean vector variables \( a \in A = B_2^{\lvert a \rvert} \), \( b \in B = B_2^{\lvert b \rvert} \), \( c \in C \subseteq B_2^{\lvert c \rvert} \), such that \( \Phi_1(a, b) = 1 \) is consistent for all \( a \in A \) (i.e., for each \( a \) there is at least one \( b \) such that \( \Phi_1(a, b) = 1 \)). This condition can be generalized to take into account the constrained case where \( \Phi_1 \) is incompletely defined.

Lemma 1:

\[ \prod_{b} (\Phi_1(a, b) + \Phi_2(b, c)) \leq \sum_{b} (\Phi_1(a, b) \Phi_2(b, c)) \quad (3) \]

Proof: See Appendix.

Definition 1: The states \( b \) are non-essential in the composition \( \Phi_1(a, b) \Phi_2(b, c) \) iff for all \( a \in A \) the states of \( b \) do not distinguish among the states of \( c \) that are associated with the state of \( a \) through the composition.

For instance, if \( \Phi_1(a, b) \) and \( \Phi_2(b, c) \) represent combinational functions \( b = f(a) \), resp. \( c = g(b) \), then \( b \) is non-essential, since in the composition the state of \( a \) uniquely determines the associated state of \( c \). Similarly, in the composition of \( \Phi_1 = a_1 b_1 + a_2 b_2 \) and \( \Phi_2 = b_1 c_1 + b_2 c_2 \), the states of \( b = (b_1, b_2) \) are non-essential, because for \( a_1 = 1 \) the states \( c = (c_1 = 1, c_2 = 0) \) and \( c = (c_1 = 1, c_2 = 1) \) are both associated with the states \( b = (1, 0) \) and \( b = (1, 1) \). In contrast, in the composition of the same \( \Phi_1 \) with \( \Phi_2 = b_1 c_1 + b_2 c_2 \), the states of \( b = (b_1, b_2) \) are essential, since for \( a_1 = 1 \) and \( b_1 = 1 \), \( b_2 = 0 \) only \( c_1 = 1, c_2 = 0 \) is associated with this state, while for \( b_1 = b_2 = 1 \) both \( c_1 = 1, c_2 = 0 \) and \( c_1 = 1, c_2 = 1 \) are associated with \( a_1 = 1 \). For \( a_1 = 0 \) there is a unique state \( b_1 = b_2 = c_1 = c_2 = 0 \). When a non-essential variable is \( \Sigma \) abstracted then no new associations are created (i.e., the \( b \) states can be reconstructed up to isomorphism).

\( \Phi_1(a, b) \) and \( \Phi_2(b, c) \) can be expressed in terms of the canonical products (minterms):

\[
\Phi_1(a, b) = \sum_{a \in A} \sum_{b \in B_a} m_a(a) m_b(b) \quad \text{where} \ m_\xi(x) \text{ is the minterm in } x \text{ corresponding to the state } x = \xi \in X
\]

\( \Phi_1(a, b) \) and \( \Phi_2(b, c) \) can be expressed in terms of the canonical products (minterms):

\[ m_\xi(x) \] is the minterm in \( x \) corresponding to the state \( x = \xi \in X \) (i.e., \( m_\xi(\xi) = 1 \)), and \( B_\xi \subseteq B \) is the set of states of \( b \) associated with the state \( a = \alpha \) in the relation characterized by \( \Phi_1 \). Similarly,

\[ \Phi_2(b, c) = \sum_{b \in B} m_b(b) \sum_{c \in C_b} m_c(c) \]

Lemma 2: \( b \) is non-essential in the composition \( \Phi_1(a, b) \Phi_2(b, c) \) iff for all \( a \in A \):

\[ \left[ \bigcup_{b \in B_a} C_b \right] \cap \left[ \bigcup_{b \in B_a} B_b \right] = \emptyset. \quad (4) \]

Proof: See Appendix.

Lemma 3: When \( \Phi_1(a, b) \) is combinational in \( b \) then the states of \( b \) are non-essential in the composition \( \Phi_1(a, b) \Phi_2(b, c) \).

Proof: Follows from the fact that \( \|B_\xi\| = 1 \) in this case.

Lemma 4:

\[ \prod_{b} (\Phi_1(a, b) + \Phi_2(b, c)) \geq \sum_{b} (\Phi_1(a, b) \Phi_2(b, c)) \quad (5) \]

holds iff \( b \) is non-essential in the composition \( \Phi_1(a, b) \Phi_2(b, c) \).

Proof: See Appendix.

Theorem 1: If \( \Phi_1(a, b) \) is completely specified over all \( a \in B_2^{\lvert a \rvert} \) and \( b \) is non-essential in the composition \( \Phi_1(a, b) \Phi_2(b, c) \) then

\[ \prod_{b} (\Phi_1(a, b) + \Phi_2(b, c)) = \sum_{b} (\Phi_1(a, b) \Phi_2(b, c)) \quad (6) \]

Proof: Follows from Lemmas 1 and 4.

The right-hand side of (6) represents the "normal" composition operation, while the left-hand side is an implication. We shall see next how it can be used to rearrange relation (1) (the verification condition) in the case where both the implementation \( C \) and the specification \( S \) consist of a composition of 2 (or more, in general) modules: Let \( C_1, C_2 \) be the modules of \( C \) \( (S) \) interconnected using variables \( u \) (\( v \)). Condition (1) can be written as:

\[ \sum_u (C_1(x, u) C_2(u, y)) \leq \sum_y (S_1(x, v) S_2(v, y)) \quad (7) \]

Theorem 2:

(a) Expression (7) is equivalent to

\[ \sum_x (C_1(x, u) S_1(x, v)) \leq \sum_y (C_2(u, y) S_2(v, y)) \quad (8) \]

iff the composition of \( S_1 \) and \( S_2 \) satisfies the conditions of Theorem 1.

(b) Expression (7) is equivalent to

\[ \sum_x (C_1(x, u) S_1(x, v)) \leq \sum_y (C_2(u, y) S_2(v, y)) \]
iff, in addition to (a), the composition of $C_2$ and $S_2$ satisfies the conditions of Theorem 1.

Proof: a) Transform the right-hand side of (7) using Theorem 1, rewrite using $a \iff b \iff \neg a \iff b = 1$, rearrange quantifiers/abstractions and rewrite to the $\leq$ form.

b) Transform the right-hand side of (8) using Theorem 1.

The importance of Theorem 2 is that it allows to change the order of composition and abstraction: Instead of first composing the modules of $C$ ($S$) and abstracting the interconnection variables $u$ ($v$), we can first compose $C_1$ and $S_1$ abstracting the inputs $x$, compose $C_2$ with $S_2$, abstract the outputs $y$, and then perform the comparison.

3 Applications

3.1 Combinational circuits

In combinational circuits, the conditions of Theorem 1 are trivially satisfied (Lemma 3). This much stricter form was used in [15] to derive Cross-controllability and Cross-observability relations, and their application in tautology checking. The general form of the conditions in Section 2 was discovered recently and presented at a workshop [16]. Its application is discussed next.

3.2 Equivalence of non-deterministic automata

Let $A_1 = (X, U, Z, g_1, f_1, t_1)$ and $A_2 = (X, V, Z, g_2, f_2, t_2)$ be two (non-deterministic) automata completely defined over the same input alphabet $X$ and having output alphabet $Z$, the transition-output CFs are $A_1(x^i, u^i, u^{i+1}, z^i)$ and $A_2(x^i, v^i, v^{i+1}, z^{i+2})$, with the initial state CFs $I_1(u^0)$ and $I_2(v^0)$. In the following, we shall determine under which conditions the equivalence of $I/O$ sequences of $A_1$ and $A_2$ can be verified using the symbolic reachability algorithms of [1-7]. Assuming that $A_1$ is equivalent to $A_2$ for sequences of length $T$, then $A_1$ is equivalent to $A_2$ for input sequences of length $T$ iff Eq. 10 holds, provided that $A_1$ accepts the same input sequences as $A_2$. The realization condition verifies that the set of sequences $\{(x^0)(x^1) \ldots (x^{T-1})(z^T)(z^T)\}$ of $A_1$ is contained in the set of sequences of $A_2$. If $A_2$ is deterministic then the condition verifies behavioral equivalence of the two automata (i.e., $A_1$ and $A_2$ produce the same output for input sequences of length up to $T$).

Note that $\sum_{z^j} \{ (z^j = z^T) \ldots \}$, $i = 1, 2$, renames $z_i$ to $z$.

The proof of behavioral equivalence of two deterministic automata can be achieved by a breadth-first traversal of the product machine $A_1 \times A_2$ forward in time from the initial state $(t_1, t_2)$. The algorithm can be written as:

Forward:

1. $E_0(u, v) = I_1(u) I_2(v)$; $i := 0$;
2. Do
   if not($\sum_{x, u, v, z} E(u, v) A_i(x, u, u', z_i) A_j(x, v, v', z_j) \leq z_i = z_j$)
      then stop; -- not equivalent
      $E_{i+1}(u', v') = E(u', v')$
      $+ \sum_{x, u, v, z} E(u, v) A_i(x, u, u', z_i) A_j(x, v, v', z_j)$
      $i := i + 1$;
      until $E_{i+1} = E_i$; -- fixed point reached.

Ignoring the fixed-point detection mechanism, and assuming that the equivalence holds for sequences of length $T$, the effect of the algorithm for input sequences of length $T$ can be expressed as in Eq. 11.

Note the abstraction of the input variables for all $t$. This is why the algorithm is of polynomial complexity in the number of machine states. For non-deterministic automata, this abstraction of the input sequences cannot be made in (10), and the verification may attain exponential complexity as implied by the increasing length of the sequences in (11).

Under what useful conditions, other than the full determinism of $A_1$ and $A_2$, can (10) be transformed into the breadth-first traversal (11), and thus maintain the polynomial nature of the algorithm? It can be shown that by applying Theorem 1, (10) can be transformed into (11) with the following conclusion: The breadth-first algorithm (11) is applicable if $A_2$ is deterministic, while $A_1$ can be non-deterministic.

\[
\sum_{x} \sum_{u, v, z} [A_1(x, u, u', z_i)] \sum_{u, v, z} [A_2(x, v, v', z_j)] \leq \sum_{x, u, v, z} \{ (x = u) \sum_{v, z} [A_2(x, v, u, z_j)] \}$
\]

\[
= \sum_{x, u, v, z} [A_1(x, u, v', 1)] A_2(x, v, v', 1) \leq \sum_{x, u, v, z} [A_1(x, u, u', z_i) A_2(x, v, v', z_j) \leq z_i = z_j]
\]

\[
= \sum_{x, u, v, z} [A_1(x, u, v', 1)] A_2(x, v, v', 1) \leq \sum_{x, u, v, z} [A_1(x, u, u', z_i) A_2(x, v, v', z_j) \leq z_i = z_j]
\]
5 Conclusions

We have presented a transformation applicable to the composition of behavioral descriptions, identified its conditions of validity, and discussed various applications. Further research includes:
- Identification of less strict conditions for the applicability of polynomial behavioral equivalence algorithms to non-deterministic automata.
- Development of a mixed algorithm for automata having limited non-determinism encountered in real applications; characterization of permissible forms of non-determinism.
- Analysis of undetectable errors when non-essentiality conditions are not respected.
- Other applications of the transformation, e.g., in theorem proving.

Acknowledgments

The work was partially supported by NSERC Canada grant Nos. MEF0040113 and OGP0003879, and by an equipment loan from the Canadian Microelectronics Corporation.

References


Appendix

Proof of Lemma 1: The relation can be rewritten as
\[ \sum_b (\Phi_1(a, b) \Phi_2(b, c)) + \sum_b (\Phi_1(a, b) \Phi_2(b, c)) = 1 \]
and then as
\[ \sum_b [\Phi_1(a, b) (\Phi_2(b, c) + \Phi_2(b, c))] = 1, \]
But \( \Phi_2(b, c) + \Phi_2(b, c) = 1 \), leaving
\[ \sum_b \Phi_1(a, b) = 1, \] which is true for all \( a \in A \) (it is the consistency condition).

Proof of Lemma 2: For a given value \( a = \alpha \), the states of \( b \) cannot be distinguished only when the sets of states of \( c \)
associated with each value of \( b \) are the same, i.e., when \( C_{\beta_1} = C_{\beta_2} \) for all \( \beta_1, \beta_2 \in B_\alpha \). This implies \([C_{\beta_1} \cap C_{\beta_2}] \cup [C_{\beta_1} \cap C_{\beta_2}] = \emptyset \). Since \( C_{\beta_1} \cap C_{\beta_2} = \emptyset \), then after union over all \( \beta_1, \beta_2 \in B_\alpha \) and rearrangement relation (4) is obtained.

**Proof of Lemma 4:** Relation (5) can be rewritten as

\[
\sum_{b} \left( \sum_{\alpha \in A} \left( m_{\alpha}(a) \sum_{\beta \in B_\alpha} m_{\beta}(b) \sum_{\gamma \in C_{\beta}} m_{\gamma}(c) \right) \right) \sum_{b} \left( \sum_{\alpha \in A} \left( m_{\alpha}(a) \sum_{\beta \in B_\alpha} m_{\beta}(b) \sum_{\gamma \in C_{\beta}} m_{\gamma}(c) \right) \right) = 0
\]

which gives

\[
\sum_{b} \left( \sum_{\alpha \in A} \left( m_{\alpha}(a) \sum_{\beta \in B_\alpha} m_{\beta}(b) \sum_{\gamma \in C_{\beta}} m_{\gamma}(c) \right) \right) \sum_{b} \left( \sum_{\alpha \in A} \left( m_{\alpha}(a) \sum_{\beta \in B_\alpha} m_{\beta}(b) \sum_{\gamma \in C_{\beta}} m_{\gamma}(c) \right) \right) = 0
\]

where \( \overline{C}_B = C - C_{\beta} \). This can be further simplified as

\[
\sum_{b} \left( \sum_{\alpha \in A} \left( m_{\alpha}(a) \sum_{\beta \in B_\alpha} m_{\beta}(b) \sum_{\gamma \in C_{\beta}} m_{\gamma}(c) \right) \right) \sum_{b} \left( \sum_{\alpha \in A} \left( m_{\alpha}(a) \sum_{\beta \in B_\alpha} m_{\beta}(b) \sum_{\gamma \in C_{\beta}} m_{\gamma}(c) \right) \right) = 0
\]

After summation over all states of \( b \) in both parts of the lefthand side, we get

\[
\sum_{\alpha \in A} \left( \sum_{\gamma \in C_{\beta}} m_{\alpha}(a) \right) \sum_{\beta \in B_\alpha} m_{\beta}(b) \sum_{\gamma \in C_{\beta}} m_{\gamma}(c) = 0
\]

and since \( m_{\alpha 1}(a) m_{\alpha 2}(a) = 0 \) for \( \alpha 1 \neq \alpha 2 \), this yields

\[
\sum_{\alpha \in A} \left( \sum_{\beta \in B_\alpha} m_{\beta}(b) \sum_{\gamma \in C_{\beta}} m_{\gamma}(c) \right) = 0
\]

This is true iff the two sets defining the states of \( C \) are disjoint, i.e.,

\[
\left[ \bigcup_{\beta \in B_\alpha} C_{\beta} \right] \cap \left[ \bigcup_{\beta \in B_\alpha} C_{\beta} \right] = \emptyset , \text{ for all } \alpha \in A.
\]

By Lemma 2 this lemma holds.