Constrained Interpolation for Guided Logic Synthesis

Ana Petkovska*, David Novo*, Alan Mishchenko† and Paolo Ienne*

* Ecole Polytechnique Fédérale de Lausanne (EPFL)
School of Computer and Communication Sciences
CH–1015 Lausanne, Switzerland
{ana.petkovska, david.novobruna, paolo.ienne}@epfl.ch

† University of California, Berkeley
Department of EECS
alanmi@eecs.berkeley.edu

***Abstract*—Craig interpolation is a known method for expressing a target function f as a function of a given set of base functions G. The found interpolant represents the dependency function h, such that $f = h(G)$. Generally, the set G contains enough base functions to enable the existence of multiple dependency functions whose quality mainly depends on the base functions selected for reconstruction. The interpolation is not an optimization problem and thus, often, it selects some random base functions and, particularly, omits others potentially required for an optimal implementation of the target function. Mainly, it is impossible to impose that the interpolant uses a specific base function.**

In this paper, we propose a method that forces a specific base function g_i as a primary input of a dependency function. Such dependency function is build as a Shannon expansion of two constrained Craig interpolants for the assignments of the primary inputs for which g_i evaluates to 0 and 1, respectively. We also introduce a method that iteratively imposes, one by one, a predefined set of base functions. In each iteration, we force a base function by generating a new dependency function used as a target function for the next iteration.

We show that, unlike the standard Craig interpolation method, our carving method succeeds, with very high probability, to impose the desired base functions. It recomposes single-output logic circuits as their delay- or area-optimised implementations regardless of the input implementation. The proposed methods can be efficiently employed for rewriting circuits in some synthesis-based algorithms.

I. Introduction

In recent years, we have seen significant progress in SAT solvers and researchers have looked for new applications of these powerful tools. One such application is the generation of Craig interpolants that was integrated in variety of logic synthesis algorithms [1–4]. With this paper we propose a *carving interpolation method* that overcomes some limitations of the standard Craig interpolation method used for reimplementing a target function with a given set of base functions.

Namely, some synthesis-based *Engineering Change Order (ECO)* algorithms [4], [5] use Craig interpolation to derive logic circuits, called *patches*, that make a corrupted implementation functionally equivalent to the wanted one. To maximise the reuse of logic from the old implementation, the patch is built as an interpolant that uses as base functions a set of already implemented functions. The proposed carving method gives better control over the selection of the functions to be included in the logic cone representing a patch. This is especially useful because the standard interpolation method relies on the structure of the proof, which in turn is strongly biased by the heuristics used by the SAT solver. These heuristics are agnostic of the purposes of the ECO engine and therefore generally do not give the best resulting circuit structure. However, the carving method can overcome the deficiencies of these heuristics and result in more compact patches.

On the other hand, new attempts to synthesise logic circuits by performing global restructuring may also profit from our carving technique. Some logic optimisation heuristics [6], [7] reconstruct the input circuit structure by making use of small single-output circuits, called *bricks*, that are found to be useful by a particular utility function. These algorithms construct the output circuit structure gradually using sets of bricks. The standard interpolation method is incompatible with these heuristics, since they require the target function to be recomposed with some specific or all bricks from a set. However, the proposed carving technique surpass this limitation. Additionally, the original heuristics produce a functionally correct circuit at the end of the algorithm, while the carving technique produces one every time when a base function is forced to be included in the circuit. This feature, besides offering a palette of implementations for the input circuit, can also assist in tuning the bricks' utility function.

We compare our carving technique with the standard Craig interpolation method, which can also produce a new circuit structure containing a set of desired base functions. Our results show that our carving technique is able to successfully include a desired base function 99% of the times, while the Craig interpolation method has a failure rate up to 59%. We also study the runtime of our techniques.

The rest of the paper is organised as follows. First, we motivate in Section II our work with an example. Next, we give background on satisfiability problem, functional dependency, Craig interpolation, the standard interpolation method, and Shannon expansion in Section III. In Section IV, we describe in detail the proposed carving methods. Finally, we present the experimental results in Section V and conclude in Section VI.

II. Motivating Example

Assume we have already implemented the the sum function of a 2-bit adder

$$s_1 = a_1 \oplus b_1 \oplus (a_0 \cdot b_0)$$

using the base functions $g_1 = a_1 \oplus b_1$ and $g_2 = a_0 \cdot b_0$. Next, we want to recompose the carry function of a 4-bit adder

$$c_1 = (a_1 \cdot b_1) + (a_0 \cdot b_0 \cdot (a_1 + b_1)).$$

and as additional base functions we have only $g_3 = a_1$ and $g_4 = b_1$. We could now use a SAT-solver and the Craig interpolation method to rewrite the target function c_1 using the given set of base functions $G = \{g_1, g_2, g_3, g_4\}$. Unfortunately, the standard interpolation method might return arbitrarily any of the three dependency functions

$$h_1 = (g_3 \cdot g_4) + (g_2 \cdot (g_3 + g_4)),$$
$$h_2 = (g_3 \cdot g_4) + (g_2 \cdot (g_3 \oplus g_4)), \text{ or}$$
$$h_3 = (g_3 \cdot g_4) + (g_2 \cdot g_1).$$

However, having already implemented s_1 as above, we might be interested in h_3, which is the area-optimal solution and avoids reimplementing existing logic.

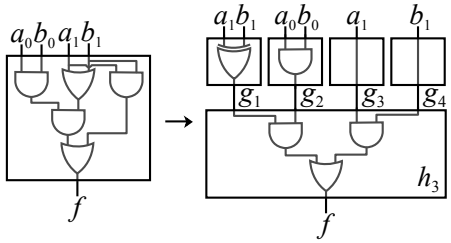


Fig. 1: Representation of the target function c_1 from Section II as a function h_3 of the set of base functions G. A dependency function h_i exist if and only if the function c_1 functionally depends on the set of base functions G.

Our carving method obtains exactly h_3 by imposing g_1 as a primary input. Situations qualitatively similar to this one can arise in a variety of logic synthesis situations such as techniques for restructuring complex circuits into well studied architectures. This example could also represent an ECO problem, if the function s_1 is implemented flawlessly, while the function c_1 has to be rectified.

III. Background Information

In this section, we give the definitions of satisfiability problem, functional dependency, Craig interpolation, the standard interpolation method, and Shannon expansion.

A. Satisfiability Problem

J.-H. R. Jiang et al. [3] have shown that one can check efficiently if a function can be expressed by a set of base functions by formulating a *satisfiability (SAT)* problem. Furthermore, *SAT solvers* play a main role in the construction of interpolants. In this subsection, we define a SAT problem and the associated terminology.

A *literal*, l, is either a variable $l = v$ or its negation $l = \overline{v}$. A disjunction (OR, $+$) of literals forms a *clause*, $c = l_1 + \ldots + l_k$. A *propositional formula* or a *Boolean formula* is a logic expression defined over variables that take values in the set $\{0, 1\}$. To solve a SAT problem, the propositional formula is converted into its *Conjunctive Normal Form (CNF)* as a conjunction (AND, $\cdot$) of clauses, $F = c_1 \cdot \ldots \cdot c_k$ [8].

Definition 1: A *satisfiability (SAT) problem* is a decision problem that receives a propositional formula in CNF form and decides that the formula is *satisfiable* (*SAT*) if there is an assignment of the variables from the formula for which the CNF evaluates to 1. Otherwise, the propositional formula is *unsatisfiable* (*UNSAT*).

SAT solvers return a *satisfying assignment* when the problem is SAT. Otherwise, modern SAT solvers, such as MiniSat [9], can compute a *proof of unsatisfiability*, also called a *refutation proof*. To define refutation proof, we need to define first the resolution principle.

Definition 2: Let $c_1 = x + R_1$ and $c_2 = \overline{x} + R_2$ be any two clauses, such that if there is a literal x in c_1, then its complement, $\overline{x}$, is literal in c_2. The *resolution principle* says that the *resolvent* of the clauses c_1 and c_2 is the disjunction $R_1 + R_2$, given that $R_1 + R_2$ is non-tautological. The literal x is called a *pivot variable*.

Definition 3: A *refutation proof* Π of a set of clauses C is a directed acyclic graph (V_Π, E_Π), where E_Π is set of edges connecting the vertices with their predecessor vertices, and V_Π, the set of vertices, is a set of clauses such that

- for every vertex $c \in V_\Pi$, c is either a *root clause*, such that $c \in C$, or c is an *intermediate clause* and represents the resolvent of its two predecessors c_1 and c_2, and
- the unique *leaf vertex* is an empty clause.

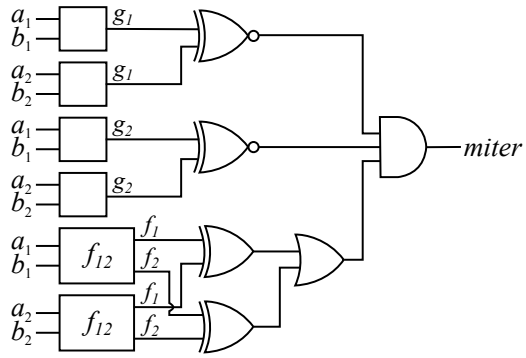


Fig. 2: The circuit constructed for checking if the function f_{12}, with two outputs f_1 and f_2, and two inputs a and b, functionally depends on the set of base functions $G = \{g_1, g_2\}$. Functional dependency exist if and only if the miter evaluates to 0 for any two assignments of the primary inputs, that is, the miter is UNSAT.

For example, consider the UNSAT set of clauses of Figure 3: the refutation proof asserts that the last two clauses can only be satisfied if $b = 0$ and, therefore, can be replaced with their resolvent $(a + c)$. Next, the clauses $(\overline{a} + d)$ and $(a + c)$ can be replaced by the resolvent $(c + d)$, and so on until we obtain the leaf clause 0, which proves that the original set of clauses is UNSAT. In Section III-C, we show how the refutation proof is used for building an interpolant.

B. Functional Dependency

The check for functional dependency ensures that a given function f can be expressed by a given set of base functions G. Additionally, the necessary and sufficient condition for existence of functional dependency helps defining the construction of the circuits from which the interpolant is built.

Definition 4: A function $f(X)$ *functionally depends* on a set of Boolean functions $G = \{g_1(X), \ldots, g_n(X)\}$, defined over the same variable vector $X = (x_1, \ldots, x_m)$, if there exists a Boolean function h such that $f(X) = h(g_1(X), \ldots, g_n(X))$ [3].
The functions f, g_i, and h are called *target function*, *base function*, and *dependency function*, respectively.

As an example, consider the target function c_1, the variable vector $X = (a_0, b_0, a_1, b_1)$, and the set of base functions $G = \{g_1, g_2, g_3, g_4\}$ given in Section II. Since the target function c_1 can be rewritten as any of the dependency functions h_1, h_2, or h_3, it follows that f functionally depends on the set of base functions G. Figure 1 shows the target function c_1 from Section II with one of its dependency functions h_3.

A base function $g_i \in G$ is an *essential* base function, if f functionally does not depend on G when g_i is removed from the set G. Otherwise, g_i is an *auxiliary* base function. For the motivating example from Section II, given c_1 as target function and the set $G = \{g_1, \ldots, g_4\}$, the base functions g_2, g_3 and g_4 are essential, while g_1 is an auxiliary base function.

Next, we give the necessary and sufficient condition for functional dependency that we use to construct the circuit for checking if a function f functionally depends on a set of base functions G.

Theorem 1: Let $f(X)$ be a target function and let G be a set of Boolean functions $G = \{g_1(X), \ldots, g_n(X)\}$, both defined over the variable vector X. For any two assignments P and Q of X, when $f(P) \neq f(Q)$, then the set G contains at least one base function $g_i(X)$, for $i = 1, \ldots, n$, such that $g_i(P) \neq g_i(Q)$, if and only if, the function f functionally depends on the set G [10].

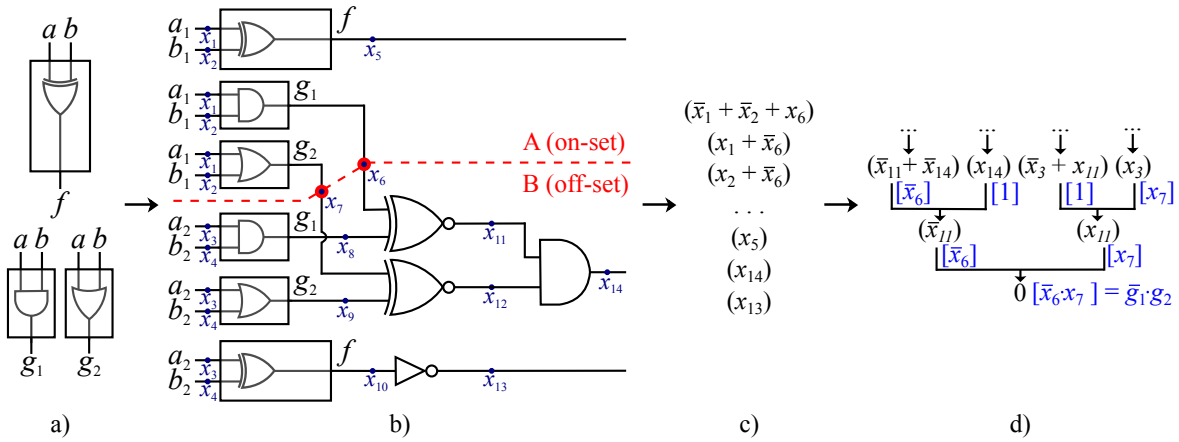


Fig. 4: The process for computing a single-output dependency function $h(g_1, g_2)$, for the target function $f(a, b)$ and the set of base functions $G = \{g_1, g_2\}$ shown with Figure 4a. Figure 4b presents the DLN used for deriving the interpolant. The variable vector $X = (x_1, \ldots, x_{14})$ correspond to the introduced CNF variables for each signal. The dashed line partitions the gates whose clauses belong to the A and B, respectively. The dots on this line depict the outputs of the base functions whose CNF variables are common variables for the on-set A and the off-set B. Thus, these outputs are candidates for the support set of the dependency function. Figure 4c shows the same DLN represented as CNF clauses, which are given to a SAT solver. Figure 4d shows the final clauses of the refutation proof from which the dependency function $h(g_1, g_2) = \overline{g}_1 \cdot g_2$ is derived.

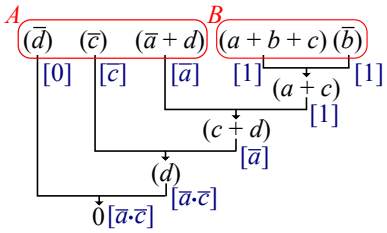


Fig. 3: Computing an interpolant from the refutation proof of $A = (\overline{d}) \cdot (\overline{c}) \cdot (\overline{a} + d)$ and $B = (a + b + c) \cdot (\overline{b})$ using McMillan's algorithm described with Definition 5. The intermediate clauses are derived using the resolution principle defined in Section III-A. In brackets, we give the Boolean formulas $p(c)$ assigned to each clause and the one assigned to the leaf clause, $p(0) = \overline{a} \cdot \overline{c}$, is the interpolant.

Following Theorem 1, to check if a function f functional depends on a set of base functions G, we construct a circuit called *miter*, which we can transform to CNF clauses and give to a SAT solver. The miter evaluates to 1 if and only if two assignments for the primary inputs P and Q exist, for which each $g_i \in G$ evaluates to the same value and at least one output of the function f evaluates to a different value. The output that evaluates to a different value cannot be represented as a function of G, and there is no functional dependency. Otherwise, if the miter evaluates to 0 for all possible assignments of the primary inputs, then f functionally depends on G. As an example, Figure 2 shows a miter constructed for a multiple-output target circuit.

C. Interpolant

In this subsection, we present the Craig interpolation theorem, which was first proved by W. Craig [11], and define the method for constructing interpolants proposed by McMillan [12].

Theorem 2: Let (A, B) be a pair of sets of clauses, such that $A \cdot B$ is unsatisfiable. Then, there exist a formula P such that:

- A implies P,
- $P \cdot B$ is unsatisfiable, and
- P refers only to the common variables of A and B [11].

The formula P represents an *interpolant* of A and B. Given the pair (A, B) and their refutation proof, a procedure called *interpolation system* constructs an interpolant in linear time and space in the size of the proof [12], [13] . Next, we explain the construction of the interpolant through McMillan's system [12].

Definition 5: Let (A, B) be a pair of clause sets and let Π be their refutation proof. To all clauses c from Π, we assign a Boolean formula $p(c)$, such that

- if c is a root clause, and
  - if $c \in A$, then $p(c)$ is the disjunction of c's global literals, whose variable appear in both A and B, or
  - else, if $c \notin A$, then $p(c) = 1$;
- else, if c is an intermediate clause, then let c_1 and c_2 be the predecessors of c, and let x be their pivot variable. Then,
  - if $x \in B$, then $p(c) = p(c_1) \cdot p(c_2)$, and
  - if $x \notin B$, then $p(c) = p(c_1) + p(c_2)$.

The Π-interpolant of (A, B) is the Boolean formula assigned to the leaf clause $p(0)$ [12].

The Boolean circuit representing the interpolant is constructed by substituting the intermediate vertices and the leaf with gates corresponding to the executed operation between their predecessors. Figure 3 shows how the interpolant for $A = (\overline{d}) \cdot (\overline{c}) \cdot (\overline{a} + d)$ and $B = (a+b+c) \cdot (\overline{b})$ is constructed by following McMillan's algorithm.

D. The Standard Interpolation Method

J.-H. R. Jiang et al. [3] suggested using interpolation for re-expressing a target function with some dependency function over other base functions (also used by C.-H. Lin et al. [14]). Figure 4b shows the *Dependency Logic Network (DLN)* required for computing the dependency function h of the target function f in terms of the set of base functions $G = \{g_1, g_2\}$. The DLN has similar property as the miter shown in Figure 2, and can be used for the functional dependency check of a single-output function. Thus, if f functionally depends on the set of base functions G, then we can represent the DLN as a set of CNF clauses partitioned into two sets A and B as suggested by J.-H. R. Jiang et al. [3]. The Tseitin Transformation [15]

converts a circuit from combinational logic to a set of CNF clauses by introducing new variables for each primary input and for each gate. The SAT solver receives as input the CNF clauses, constrains the CNF variables assigned to the two copies of the primary inputs of the target function and produces a refutation proof from which the interpolant is computed. Since the CNF variables of the outputs of one copy of the base functions belong both to the on-set A and to the off-set B, these outputs represent candidate inputs for the support set of the interpolant and, implicatively, for the dependency function. Figure 4 shows the process for computing the dependency function with this *standard interpolation method*.

E. Shannon Expansion

Shannon expansion [16] is a fundamental theorem used for simplification and optimisation of logic circuits.

Theorem 3: Any Boolean function f defined over a variable vector $X = (x_1, \ldots, x_n)$, can be written in the form

$$f = \overline{x}_i \cdot f_{\overline{x}_i} + x_i \cdot f_{x_i},$$

where $f_{\overline{x}_i} = f(x_1, \ldots, 0, \ldots, x_n)$ and $f_{x_i} = f(x_1, \ldots, 1, \ldots, x_n)$.

The expressions $f_{\overline{x}_i}$ and f_{x_i} represent, respectively, a *negative* and *positive cofactor* of f with respect to the *control variable* x_i.

IV. The Carving Interpolation Method

In this section, first, we explain the deficiency of the standard interpolation method and we present in detail the carving interpolation method proposed in this paper.

A. Deficiency of the Standard Interpolation Method

A base function belongs to the support set of the dependency function if and only if its clauses are part of the refutation proof. All essential base functions belong to the support set, because their clauses are required for building a refutation proof. However, whether an auxiliary base function belongs to the support set depends on the selection of the SAT solver. Since the SAT solver does not consider the importance of a base function while building the refutation proof, the standard interpolation method often omits base functions that enable a specific implementation of the target function.

On the other hand, some techniques require some specific or all selected base functions to be used as primary inputs of the dependency function. For instance, A. K. Verma et al. [7] propose an algorithm that optimises a circuit by iteratively generating and selecting a set of base functions. After the first iteration, it uses the dependency function to generate a new set of base functions. Thus, if a base function is disconnected from the dependency function, then it is discarded, although it was previously selected as an adequate and desirable base function for reconstructing the circuit. Therefore, this behaviour of the standard interpolation method hinders its applicability to such logic synthesis techniques.

B. Carving Out a Base Function

In this subsection, we describe in detail the *carving interpolation method* for imposing the use of a single base function by constructing a dependency function as a Shannon expansion of two constrained Craig interpolants.

Assume we have a target function that functionally depends on a set of base functions from which we want to impose the use of a selected base function. For the carving interpolation method, we construct the same DLN as for the standard interpolation method and represent it as a set of CNF clauses.

Assume the CNF clauses are expressed in terms of the variable vector $X = (x_1, \ldots, x_n)$. We denote this set of CNF clauses as $C(x_1, \ldots, x_n)$. The unsatisfiability of the SAT problem defined with the DLN signify that the set of CNF clauses is UNSAT for any assignment of X. From this, it follows that both $C_{\overline{x}_i} = C(x_1, \ldots, 0, \ldots, x_n)$ and $C_{x_i} = C(x_1, \ldots, 1, \ldots, x_n)$, in which we have assigned the variable x_i to 0 and 1, respectively, are also UNSAT for any assignments of X. A CNF variable is assigned to a constant by extending the existing set of CNF clauses with a single-variable clause x_i or $\overline{x}_i$, which makes an assumption that the variable x_i evaluates to 1 or 0, respectively.

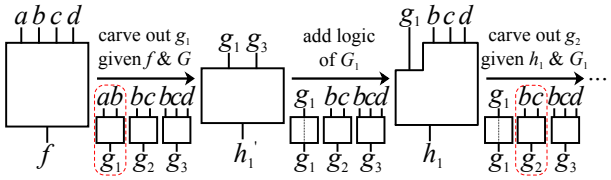


Fig. 5: The process for imposing the first base function g_1 from the set of base functions $G = \{g_1, g_2, g_3\}$. After a base function is imposed, in the set G, it is substituted with an identity function. For imposing the second function g_2, the carving method is given the last computed interpolant h_1 and the modified set of base functions G_1.

Using the refutation proofs for $C_{\overline{x}_i}$ and C_{x_i}, we can construct two constrained interpolants $I_{\overline{x}_i}$ and I_{x_i}, respectively. These two interpolants represent the cofactors of a feasible interpolant I for the satisfiability problem expressed with the set of clauses $C(x_1, \ldots, x_n)$, with respect to x_i. Thus, using the formula for Shannon expansion, we can generate the interpolant I as

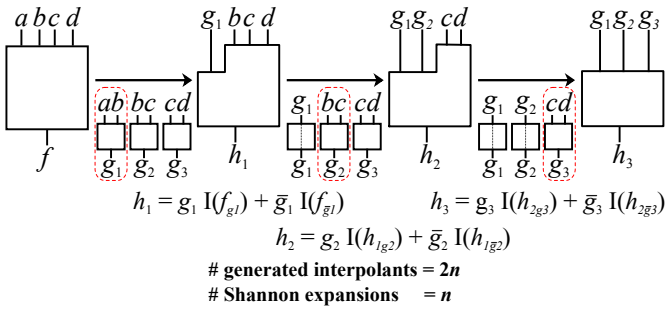
$$I = \overline{x}_i \cdot I_{\overline{x}_i} + x_i \cdot I_{x_i}.$$

In order to impose a selected base function g_i, we perform the Shannon expansion with respect to the CNF variable x_i assigned to the output of the function g_i. When we assign the variable x_i to 0, we evaluate all assignments for which the function g_i evaluates to 0. At the same time, for the assignments for which g_i evaluates to 1, there is a conflict with the assumed value of the output of g_i and the problem is UNSAT. Similarly, the dual case applies for the assignments for which g_i evaluates to 1. Thus, the two interpolants, built for the assignments for which the selected base function g_i evaluates to 0 and 1, respectively, represent the negative and the positive cofactors of the dependency function with respect to the function g_i, and they can be used to obtain the final dependency function with Shannon expansion.

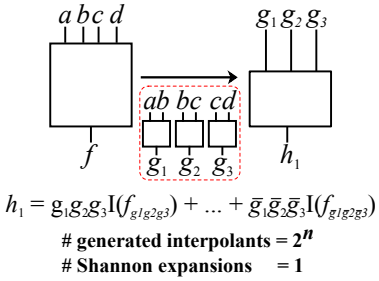
C. Carving Out a Set of Base Functions

Given a set of base functions $G = \{g_1, \ldots, g_n\}$, such that the target function f functionally depends on G, the base functions are iteratively carved out one by one. For carving out the first base function g_1, we use the function f and the set G. As a result we receive a dependency function h_1' that as primary inputs has the imposed base function g_1 and a subset of the remaining base functions from G. To retain g_1 as a primary input, after carving it out, we modify G to G_1 by substituting g_1 with an *identity function*, which propagates the input as an output. To be able to impose the remaining base functions, we construct a function h_1 by adding the logic of the non-imposed base functions to the function h_1'. In the next iterations, for imposing the base function g_i, where $i = 2, \ldots, n$, we use the dependency function h_{i-1} and the modified set of base functions G_{i-1}. Figure 5 shows the first iteration for imposing a set of base functions $G = \{g_1, g_2, g_3\}$ given a target function f.

Carving a set of base functions by our method is usually much slower than by the standard interpolation method, because our method



(a) One-by-one carving of the base functions.



(b) Simultaneous carving multiple base functions.

Fig. 6: For carving out n base functions successively, we need to compute $2n$ interpolants. On the other hand, to carve them out simultaneously, we have to compute 2^n interpolants.

builds two interpolants per base function, while the standard one generates only one interpolant per layer. Accordingly, since we know that the standard interpolation method always uses the essential base functions, we propose an optimised carving method that improves the runtime by generating a single interpolant for all essential base functions. Moreover, it increases the success rate for imposing the set of base functions, since it prioritise the auxiliary base functions.

The optimised carving method starts with partitioning the set G into two subsets, G_e and G_a, that contain the essential and auxiliary functions, respectively. Then, we carve out the base functions from the set G_a, which might be omitted by the standard interpolation method, as explained in Section IV-A. Assuming that the set G_a contains i base functions, once all functions are carved out, we have constructed the function h_i and the set G was modified to G_i. In the end, the final dependency function is build with the standard interpolation method that reconstructs the function h_i as a function from the set G_i. For example, for the target function f and the set G from Figure 5, instead of first imposing the base function g_1 as shown for the basic carving method, we first impose the function $g_2 \in G_a$ and obtain the function h_1. Next, instead of imposing the functions from $G_e = \{g_1, g_3\}$, we generate the final dependency function with the standard interpolation method given the function h_1 as a target function and the set $G_1 = \{g_1, g_{2id}, g_3\}$, where g_{2id} is the identity function introduced for g_2.

The Shannon expansion can also be performed on multiple variables. Thus, in one iteration, multiple base functions can be simultaneously carved out of the target function. However, as Figure 6 shows, due to the nature of the Shannon expansion, the number of interpolants required for simultaneous carving out multiple base functions grows exponentially with the number of base functions carved out simultaneously, opposed to the linear growth of the one-by-one single-function carving.

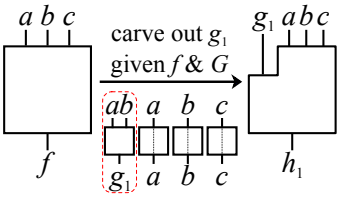


Fig. 7: The setup for imposing the base function g_1 using the carving method. The base functions are always expressed as functions of the primary inputs, thus the selected base function is accompanied with the essential identity functions of the target function's primary inputs a, b, and c. After deriving the dependency function h_1, we test if g_1 is used as a primary input.

V. Experimental Results

In this section, we introduce our experimental setup and we present the experimental results which compare the carving methods with the standard interpolation method.

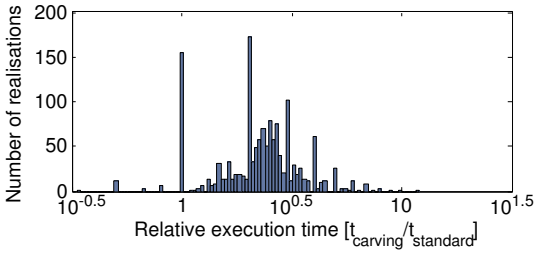
A. Experimental Setup

We implemented both the basic and the optimised carving interpolation methods described in this paper as commands in ABC [17]. *ABC* is an open-source software system for synthesis, technology mapping, and formal verification of sequential Boolean logic circuits used in synchronous hardware design. ABC relies heavily on the *And-Inverter Graph* (AIG) data structure, which represents a multi-level logic network composed of two-input AND gates and inverters. AIGs enable short runtimes and high-quality results for synthesis, mapping and verification due to their simplicity and flexibility. Another advantage is that the modern SAT solver MiniSat is integrated into ABC, and it provides the proof of unsatisfiability for UNSAT problems.

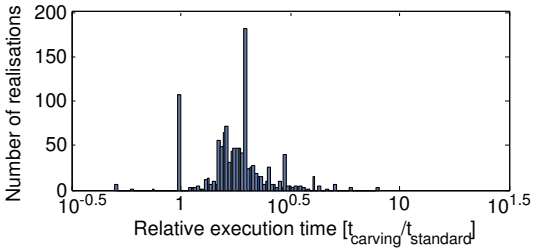
Although the carving technique is general and not tied to any purpose or logic synthesis heuristic, we have developed an experimental setup somehow inspired from Iterative Layering proposed by A. K. Verma et al. [7]. Such heuristic restructures a circuit by gradually imposing a set of preselected base functions. Reimplementing the whole algorithm is not our purpose here, but we want to explore interpolation as the means to transform progressively the original circuit in the optimised one (it is worth noticing that the original authors of Iterative Layering do not use SAT solvers and follow a different strategy altogether). In our exeriments, we thus use an oracle that gradually provides the base functions for composing the optimal circuit structure.

To ABC, we provide an input implementation and a final implementation of the circuit we want to optimise. The final implementation, which is either a wanted known implementation of the circuit or the input implementation optimised in ABC, serves as a reference goal from which the oracle computes the set of base functions for reconstructing the circuit. In our case, this set consists of the logic functions of non-overlapping k-input cuts, which cover the whole circuit. With this setup, we show that we can recompose any input implementation to any final implementation as long as we have the adequate base functions. Two scenarios are presented: in the first one, we carve out only one base function while, in the second one, we carve out the whole layer as described in Section IV-C.

For our experiments, we use the large combinational MCNC benchmarks and a set of arithmetic circuits, which contains adders, zero-leading detectors, multipliers and majority functions. When the benchmark is a multiple-output circuit, we process each output separately, since the interpolation methods can process only a single

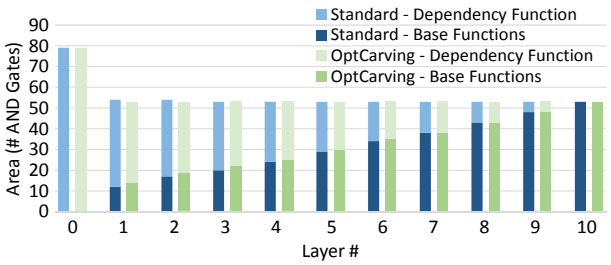


(a) Arithmetic benchmarks.

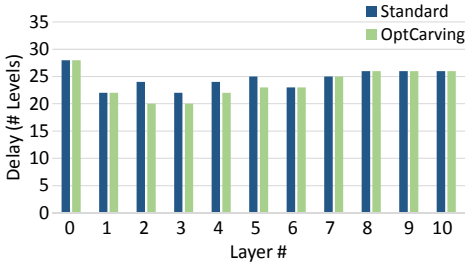


(b) Area-optimised MCNC benchmarks.

Fig. 8: The relative execution time for generating a dependency function for a set composed of a single base function and the essential identity functions.



(a) Area comparison of the standard interpolation and the optimised carving method.



(b) Delay comparison of the standard interpolation and the optimised carving method.

Fig. 9: Area and the delay of the MSB of a 14-bit adder, reconstructed using 10 layers. The area is expressed in terms of number of AND gates, while the delay is expressed as number of levels of the AIG. For each layer, Figure 9a shows the cumulative area of the base functions and the area of the dependency function built with the standard method and with the optimised carving method, respectively. Figure 9b compares the delay of the same circuits. In layers 2 and 3, the optimised carving method offers a solution that has lower delay and same area than the one returned by the final reference circuit from layer 10, which was obtained by optimising the input circuit from layer 0 in ABC.

output at a time. The results are described in detail in the following subsections, while Table I summarises the failure rates of the interpolation methods.

B. Imposing a Single Base Function

For the following experiment, the oracle provides the complete set of base functions G and, for each $g_i \in G$, we would like to construct a dependency function that has the function g_i in its support set.

In order to create a dependency function, the target function f should functionally depend on the given set of base functions. Thus, for imposing a single base function g_i, first, we form a set G_i which contains the function g_i and all identity functions of the primary inputs of the target function. Next, we remove the added identity functions that are auxiliary given the function f and the set G_i. Thus, G_i contains only the identity functions of the primary inputs that are required for achieving functional dependency and the function g_i. If a removed identity function represents a primary input that is also a primary input of g_i, both, the standard and the carving method, have to use g_i to recreate the target function. Otherwise, if the identity functions of all primary inputs of g_i are given, then several dependency functions exist—some that use the function g_i, and other that reimplement its logic with the identity functions given with G_i. Figure 7 shows the setup for imposing the base function g_1 with the carving method.

In our experiment, when it is possible to return a dependency function that omits the selected base function, we count the number of missed opportunities to use the base function for the two interpolation methods. For the arithmetic circuits, the standard interpolation method fails to use the selected base function in 55.71% of the cases, while the carving method always imposes it. For the MCNC benchmarks, when the input structure of the circuits is optimised for area, we observe 59.12% and 0.15% failure rate for the standard method and the carving method, respectively. On the other hand, when the input structure is optimised for delay, we obtain 56.40% and 0.15% failure rate, respectively.

Regarding the runtime of the two methods, as expected, the carving method takes on average twice the time required by the standard interpolation method, since the dependency function is derived from two interpolants instead of one. Figure 8 shows the distribution of the relative execution time for the arithmetic circuits, as well as for the area-optimised MCNC benchmarks.

C. Imposing a Set of Base Functions

For the following experiment, the oracle provides the complete set of base functions partitioned into subsets, called layers, on which the input circuit functionally depends. Each layer contains base functions that have only outputs of the previously composed layers as primary inputs. An identity function is introduced for each base function propagated through the layer. Figure 10 presents one possible solution for the cuts and layers for the most significant bit of a 4-bit adder.

In Section IV-C, we presented two carving methods—the basic carving method that imposes all functions from a given set successively, and the optimised carving method that, first, imposes the auxiliary functions and then generates the final dependency function using the standard interpolation method presented in Section III-D. To compare them, for each layer received from the oracle, we generate one dependency function using each method. On the other hand, for the same layers, we derive only one dependency function per layer with the standard interpolation method. Figure 9 displays the area and delay of the recomposed circuit of the MSB of a 14-bit adder after the dependency function is built for each of the 10 layers. For each layer,

TABLE I: A summary of the experimental results for the failure rates of the interpolation methods.

	Imposing a Single Base Function		Imposing a Set of Base Functions					
	Number of disconnected imposed base functions		Number of layers with at least one omitted base function			Number of disconnected base functions among all layers		
				Carving			Carving	
	Standard	Carving	Standard	Basic	Optimised	Standard	Basic	Optimised
Arithmetic Circuits	55.71%	0.00%	74.59%	0.26%	0.07%	64.02%	0.17%	0.04%
Large MCNC (area-optimised)	59.12%	0.15%	81.82%	83.33%	54.55%	34.00%	26.55%	19.85%
Large MCNC (delay-optimised)	56.40%	0.15%	92.89%	83.76%	76.14%	77.56%	34.30%	30.77%

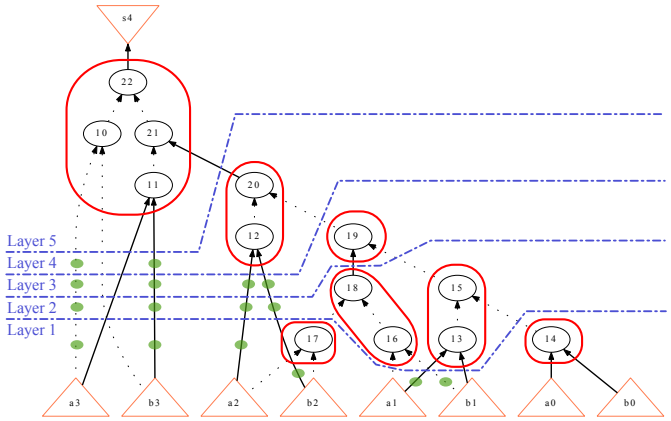


Fig. 10: Non-overlapping 3-input cuts and the formed layers for the most significant bit of a 4-bit adder. The rounded sets represent the 3-input cuts whose logic functions are the base functions. The dashed lines divide the cuts in layers. The small ovals show the signals for which an identity function is introduced because they are primary inputs to base functions from a higher layer.

we have verified that the resultant implementation is functionally equivalent to the target function given as input.

If all the base functions from a layer are essential base functions, then all of them are always used by the standard interpolation method and there is no need to use the carving method. However, if there is at least one auxiliary base function which might thus be omitted, we count as failure each layer from which at least one base function was not included in the support set of the built dependency function.

For the arithmetic circuits, at least one base function from the layer is omitted for 74.59% of the layers when the dependency function is constructed by the standard interpolation method. In contrast, the basic and the optimised carving method fail to use at least one base function for only 0.26% and 0.07% of the layers, respectively. Although we expected similar results for the MCNC benchmarks, when the input structure of the circuits is optimised for area, we obtain 81.82% failure rate for the standard method, while for the base and the optimised carving method, the failure rates are 83.33% and 54.55%, respectively. For the same benchmarks, when the input structure is optimised for delay, the failure rates are 92.89% for the standard interpolation method, and 83.76% and 76.14% for the standard and optimised carving method, respectively.

These failure rates arise because we consider a failed layer as soon as a single base function is disconnected. For instance, for a given layer, even if the standard method has omitted five functions, while the carving method has omitted only one, for both methods we consider a 100% failure rate. Thus, we also analysed the number of

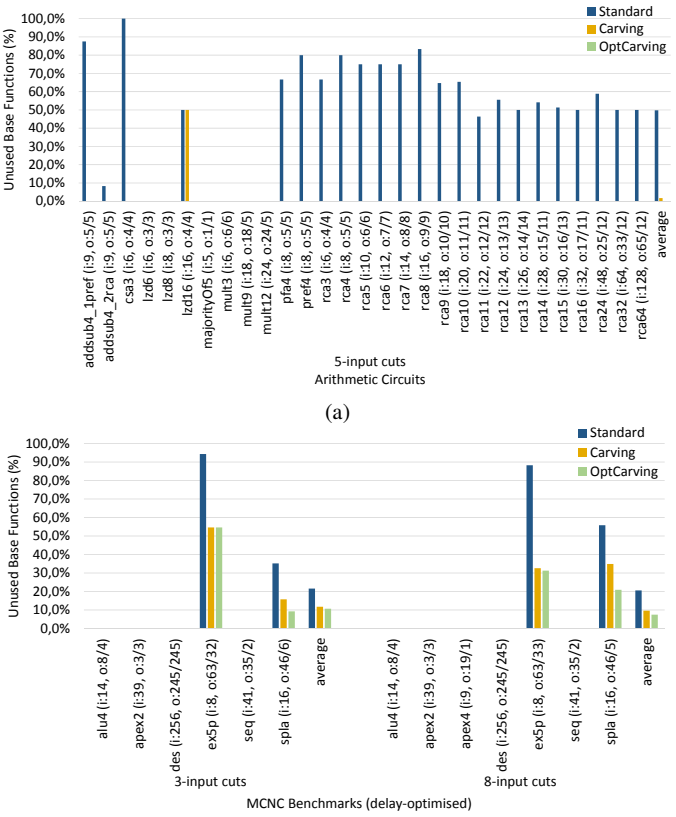
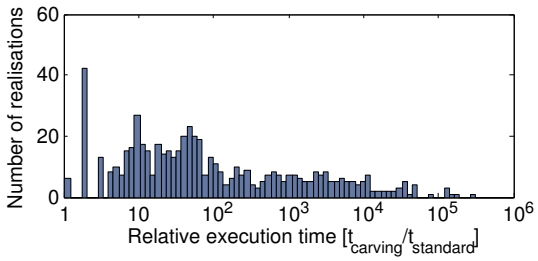
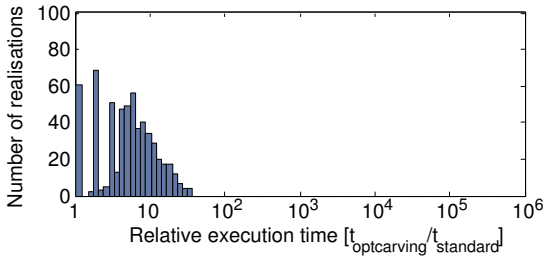


Fig. 11: The percentage of disconnected base functions among all layers for the three methods. For the presented benchmarks, the base functions are generated using 5-input cuts for the arithmetic circuits, and using 3- and 8-input cuts for the large combinatorial delay-optimised MCNC benchmarks. From the MCNC benchmarks, we were able to process at least one output within the given timeout only for the shown benchmark circuits. The formatting of the benchmarks shows their name, the number of primary inputs and outputs, as well as the number of processed outputs for which we report results. For example, the last arithmetic circuit, rca64, has 128 primary inputs and 65 primary outputs, but we processed only the first 12 outputs.

disconnected base functions among all layers. Most drastic difference is observed for the large delay-optimised MCNC benchmarks, for which the standard interpolation method omits 77,56%, while the standard carving and the optimised carving methods omit 34,30% and 30,77% of the base functions among all layers, respectively. The results for the other benchmarks are presented in Table I. When



(a) The basic carving method is almost always slower than the standard interpolation method, since it computes two interpolants per base function instead of one for the whole set.



(b) The optimised carving method improves over the basic carving method since it computes one interpolants for the essential base functions.

Fig. 12: Comparison of the execution time of the carving methods with the the standard interpolation method for the arithmetic circuits when generating a dependency function for a set of base functions.

analyzing the failures in further detail, Figure 11 shows that the carving methods fail more often in circuits on which the standard method fails as well, and they usually have lower failure rates than the standard method.

The basic carving interpolation method builds two interpolants for each base function of the layer, while the standard interpolation method constructs one interpolant per layer. Thus, the runtime of the first, in most of the cases, is significantly higher than the one of the second. Due to the extensive runtime, we fail to execute the algorithm for all the outputs for 28% of the larger MCNC benchmarks. On the other hand, for the smaller MCNC benchmarks and the arithmetic circuits, we fail to complete the algorithm for only 2% and 4% of the circuits, respectively. Figure 12 shows the distribution of the relative execution time for the arithmetic circuits.

Regarding the optimised carving method, as expected, although it spends some time on dividing the base functions on auxiliary and essential, it is generally much faster than the basic carving method. Also, as expected it succeeds more in imposing the base functions, since it gives a priority to the ones that might be omitted.

VI. Conclusions

With this paper, we present a new technique to enable the restructuring of an input circuit while imposing a given subcircuit or base function. Our results show that the proposed carving technique is able to successfully include the desired base functions most of the times (more than 99% success rate when forcing a single base function), while the reference technique, based on Craig interpolation, fails far more often. For instance, the success rate of the standard interpolation method for the large area-optimised MCNC benchmarks barely reaches 40%. Our results also show that our carving technique is significantly slower than the standard interpolation method when forcing a whole set of base functions, but takes only about twice when forcing a single base function. This is mostly because the carving technique requires more SAT solver calls than the reference technique. For imposing a set of base functions, we also propose an optimised carving method that represents a hybrid of the basic carving method and the standard interpolation method. This method offers the best failure rate and significantly improves the runtime required for carving, but is still slower than the standard interpolation method. Thus, heuristics for global circuit restructuring, as well as synthesis-based ECO algorithms can benefit from our carving technique to optimise circuits of limited size.

References

[1] H.-P. Lin, J.-H. R. Jiang, and R.-R. Lee, "To SAT or not to SAT: Ashenhurst decomposition in a large scale," in *Proceedings of the International Conference on Computer Aided Design*, San Jose, Calif., Nov. 2008, pp. 32–7.

[2] R.-R. Lee, J.-H. R. Jiang, and W.-L. Hung, "Bi-decomposing large boolean functions via interpolation and satisfiability solving," in *Proceedings of the 45th Design Automation Conference*, Anaheim, Calif., Jun. 2008, pp. 636–41.

[3] J.-H. R. Jiang, C.-C. Lee, A. Mishchenko, and C.-Y. R. Huang, "To SAT or not to SAT: Scalable exploration of functional dependency," *IEEE Transactions on Computers*, vol. C-59, no. 4, pp. 457–67, Apr. 2010.

[4] K.-F. Tang, C.-A. Wu, P.-K. Huang, and C.-Y. R. Huang, "Interpolation-based incremental ECO synthesis for multi-error logic rectification," in *Proceedings of the 48th Design Automation Conference*, San Diego, Calif., Jun. 2011, pp. 146–51.

[5] B.-H. Wu, C.-J. Yang, C.-Y. R. Huang, and J.-H. R. Jiang, "A robust functional ECO engine by SAT proof minimization and interpolation techniques," in *Proceedings of the International Conference on Computer Aided Design*, San Jose, Calif., Nov. 2010, pp. 729–34.

[6] A. K. Verma, P. Brisk, and P. Ienne, "Progressive Decomposition: A heuristic to structure arithmetic circuits," in *Proceedings of the 44th Design Automation Conference*, San Diego, Calif., Jun. 2007, pp. 404–9.

[7] ——, "Iterative Layering: Optimizing arithmetic circuits by structuring the information flow," in *Proceedings of the International Conference on Computer Aided Design*, San Jose, Calif., Nov. 2009, pp. 797–804.

[8] C. P. Gomes, H. Kautz, A. Sabharwal, and B. Selman, "Satisfiability solvers," in *Handbook of Knowledge Representation*, ser. Foundations of Artificial Intelligence, F. van Harmelen, V. Lifschitz, and B. Porter, Eds. Elsevier, Jan. 2008, vol. 3, pp. 89–134.

[9] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, ser. Lecture Notes in Computer Science, vol. 2919. Springer, May 2003, pp. 502–18.

[10] J.-H. R. Jiang and R. K. Brayton, "Functional dependency for verification reduction," in *Proceedings of the International Conference on Computer Aided Verification*, Boston, Mass., Jul. 2004, pp. 268–80.

[11] W. Craig, "Linear reasoning. A new form of the Herbrand-Gentzen theorem," *The Journal of Symbolic Logic*, vol. 22, no. 3, pp. 250–68, Sep. 1957.

[12] K. L. McMillan, "Interpolation and SAT-based model checking," in *Proceedings of the International Conference on Computer Aided Verification*, ser. Lecture Notes in Computer Science, vol. 2725. Springer, Jul. 2003, pp. 1–13.

[13] P. Pudlák, "Lower bounds for resolution and cutting plane proofs and monotone computations," *The Journal of Symbolic Logic*, vol. 62, no. 3, pp. 981–98, Sep. 1997.

[14] C.-H. Lin, C.-Y. Wang, and Y.-C. Chen, "Dependent-latch identification in reachable state space," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. CAD-28, no. 8, pp. 1113–26, Aug. 2009.

[15] G. S. Tseitin, "On the complexity of derivation in propositional calculus," in *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970*, ser. Symbolic Computation, J. Siekmann and G. Wrightson, Eds. Berlin: Springer, 1983, pp. 466–83.

[16] C. E. Shannon, "The synthesis of two-terminal switching circuits," *The Bell System Technical Journal*, vol. 28, no. 1, pp. 59–98, Jan. 1949.

[17] *ABC: A System for Sequential Synthesis and Verification*, Berkeley Logic Synthesis and Verification Group, Berkeley, Calif., Mar. 2014, release 40301, `http://www.eecs.berkeley.edu/~alanmi/abc/`.