FmlaChain: Tautologies based on Iterated Equivalences or Implications of Boolean Formulas.

Sam Buss
Department of Mathematics
University of California, San Diego
San Diego, California, USA
sbuss@ucsd.edu

***Abstract*—We describe a software package that generates instances of the Boolean formula implication chain and equivalence chain principles expressed as sets of unsatisfiable CNFs.**

***Index Terms*—Boolean formulas, formula chains, iterated equivalence, iterated implication, satisfiability, CNF, software**

I. Introduction

The Boolean formula equivalence chain tautologies (FmlaEquivChain) and the Boolean formula implication chain tautologies (FmlaImplyChain) were described by Buss and Ramyaa [1]. These tautologies, based on a suggestion of Krajíček [2], are of interest as potentially giving an exponential separation between depth d and depth $d{+}1$ Frege proofs. Surprisingly, [1] showed that if the depth d is held fixed, then the equivalence chain tautologies (expressed as unsatisfiable CNFs) have polynomial size resolution refutations. This remains open for the formula implication chain tautologies, however.

A Boolean formula chain means a sequence $T_1, T_2, \ldots, T_n$ of Boolean formulas. Each formula T_i has constant depth d; the root node (the principal connective) is an $\vee$ gate; each gate has fanin f; and gates alternate between $\vee$ and $\wedge$. The inputs to the formula T_i are represented by variables $x_{i,p}$, with $x_{i,p}$ giving the value of the p-th input to T_i. The formula T_1 will be "obviously true"; see Figure 1. Likewise, T_n will be "obviously false"; see Figure 2.

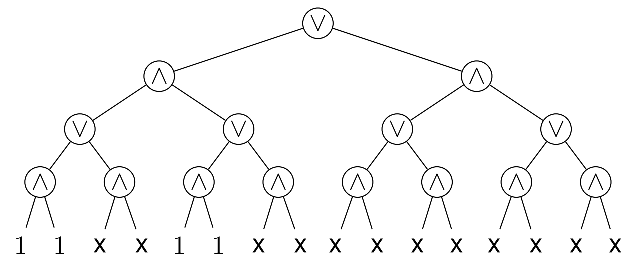

Fig. 1. This is an "obviously" true Boolean formula. The four inputs of 1 (True) are enough to force the formula to have value true. They cause two of the bottom-most AND gates to have value true; this induces the first two OR gates at the third level to have value true. That further causes the first AND gate at the second level have value true, and that induces the value true for the OR at the root of the formula. The general property is that any path from the root gate to the leaves that always selects the leftmost child of any OR gate always reaches an input value of 1. The "x" input values "don't care" values.

For space reasons, the formula is shown with all gates as having fanin $f = 2$. However, it is permitted that the fanin f is any constant ≥ 2.

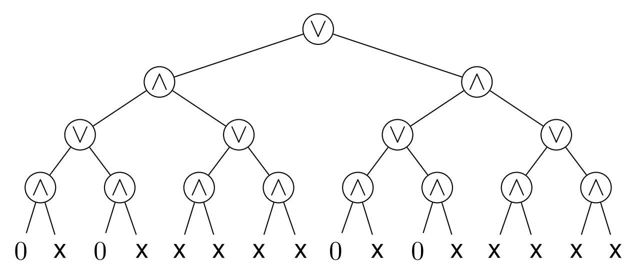

Fig. 2. The obviously false formula is constructed dually to the obviously true formula of Figure 1. Any path from the root of the formula to a leaf that always selects the leftmost child of AND gates reaches a leaf labelled with 0 (False).

Each T_{i+1} is obtained by interchanging inputs to gates in T_i. Figure 3 shows one such interchange, also called a "swap". The interchange is accomplished by interchanging the values of the inputs to T_i to obtain the values of the inputs T_{i+1}. In the "equivalence chain" formulation, called FmlaEquivChain, the interchanged input values have the same true/false value in T_{i+1} as in T_i. In the "implication chain" formulation, called FmlaImplyChain, we have only that the interchanged input values of T_{i+1} are implied by their source input value in T_i.

In both the implication and equivalence chain setting, if T_i evaulates to true, then so does T_{i+1}. It is impossible for this to hold for all i, since T_0 evaluates to true and T_n evaluates to false.

The construction of the principles FmlaEquivChain and FmlaImplyChain is based directly on [1]. However, it differs in allowing many swaps to be carried out during the transition from T_i to T_{i+1}. (Unlike [1], who allowed only one swap at a time.) Every input to every gate in T_i potentially gets swapped when forming T_{i+1}. For this, the swaps are applied in phases. In each phase, the swaps are applied to gates at a fixed depth $d' < d$ below the root, starting with depth $d' = d{-}1$ and ending with $d' = 0$.

II. Variables and Clauses

The FmlaEquivChain and FmlaImplyChain principles are specified with parameters d, f and n, representing the depth of

Supported in part by Simons Foundation grant 578919.

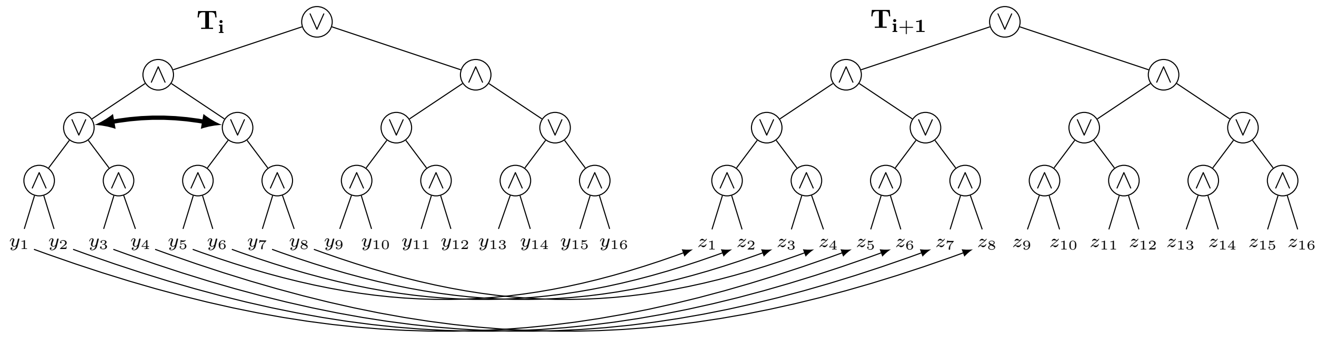

Fig. 3. The formula T_{i+1} is obtained from T_i by swapping the two subformulas as shown. This swap is accomplished by moving the values of variables. A swap variable $g := s_{i,G,\ell,k}$ indicates the swapping of the ℓ-th and k-th input to the gate G in T_i when forming T_{i+1}, (Pictured with $\ell = 1$ and $k = 2$.) In this case, the FmlaImplyChain CNF includes the clauses $g \wedge y_p \to z_q$ where y_p and y_q are the k-th inputs in T_i to the to swapped subformulas (so z_q is the input in T_{i+1} in the same position as y_q). The FmlaEquivChain CNF, for the equivalence chain tautology, also contains the clause $g \wedge z_q \to y_p$.

This figure is overly simplified, showing a single swap. In actuality, there are d rounds of swaps needed to transition from T_i to T_{i+1}, with many swaps at possible at a single level. Since the figure shows a swap at level $v = 2$, the variables y_p should actually be $x_{i,1,p}$ and the variables z_p should actually be $x_{i,2,p}$.

formulas T_i, the uniform fanin of the gates in the T_i's, and the number of formulas T_i. There are f^d many inputs to each T_i, the propositional variables $x_{i,p}$, for $p \leq f^d$, give the values of the inputs. Each T_{i+1} is obtained by performing d rounds of swaps, starting at the bottom level $s = 1$ (the level closest to the inputs), and ending at the root gate of T_i with $s = d$. The inputs to T_i after s rounds of swaps (at the bottom s levels of the formula) are denoted $x_{i,s,p}$. We identify the variables $x_{i,p}$ and $x_{i,0,p}$; these are the input values before any swaps have occurred. We also identify the variables $x_{i,d,p}$ and $x_{i+1,p}$; that is, the values of the inputs to T_{i+1} are same as the values of the inputs that are the result of carrying out all d levels of swaps on T_i.

In addition to the variables $x_{i,p}$ and $x_{i,s,p}$, there are variables $s_{i,G,\ell,k}$ where $i < n$, G is a gate in the formula T_i, and $\ell \leq k \leq f$ denote the ℓ-th and k-inputs to G. If this variable is true, it indicates the ℓ-th and k-th inputs to G in T_i are to be swapped. It is permitted that $\ell = k$, in which case, that input is not subject to a swap. The CNF will contain clauses that ensure that, for any i, G, ℓ, there is exactly one k such that $g_{i,G,\ell,k}$ or $g_{i,G,k,\ell}$ holds.

The clauses in the formula chain CNFs are:

(a) For each $x_{0,p}$ an input to T_0 that is in the leftmost child of each of its $\vee$ ancestors, the unit clause $x_{0,p}$. These ensure that T_0 evaluates to true.
(b) (Optional.) For all other $x_{0,p}$'s (don't care inputs to T_0), the unit clause $\overline{x_{0,p}}$.
(c) For each $x_{n,p}$ an input to T_n that is in the leftmost child of each of its $\wedge$ ancestors, the unit clause $\overline{x_{n,p}}$. These ensure that T_n evaluates to false.
(d) (Optional.) For all other $x_{n,p}$'s (don't care inputs to T_n), the unit clause $x_{n,p}$.
(e) For each $i < n$, each gate G in T_i, and each $i \leq f$, clauses that ensure that there is exactly only value $k \leq f$ such that $s_{i,G,\ell,k}$ or $s_{i,G,k,\ell}$ (the latter if $k < \ell$) holds.
(f) If $i < n$, G is a gate at level s in T_i, $\ell \leq k \leq f$, and $x_{i,s-1,p}$ and $x_{i,s-1,q}$ are the ℓ-th and k-th inputs to the subformula below G, the clause
$$s_{i,G,\ell,k} \wedge x_{i,s-1,p} \to x_{i,s,q}.$$
(g) For the equivalence chain principle only, under the same conditions as (f), the clause
$$s_{i,G,\ell,k} \wedge x_{i,s,q} \to x_{i,s-1,p}.$$

III. Usage

The program FmlaChain is run with the following parameters:

```
% FmlaChain <d> <f> <n> [-equiv] [-imply]
    [-dontcares] [-no-dontcares]
    [-dontcares-init] [-no-dontcares-init]
    [-dontcares-end] [-no-dontcares-end]
```

The integer values `<d>`, `<f>`, and `<n>` specify the formulas' depth, the common gate fanin, and the number of formulas.

Exactly one of the arguments "`-equiv`" or "`-imply`" must be specified. The most common usage is:

```
% FmlaChain <d> <f> <n> -equiv
```

or

```
% FmlaChain <d> <f> <n> -imply
```

The options "`-dontcares`" and "`-no-dontcares`" specify whether or not to include the unit clauses setting the values of the don't-care inputs to T_0 and T_n. The "`-equiv`" option by default includes the don't-care unit clauses, whereas the '`-imply`" option by default does not include these unit clauses. The "`-init`" and "`-end`" versions of these options select the settings separately for the don't care inputs of T_0 and T_n.

IV. Acknowledgements

Thanks to Armin Biere and Marijn Heule for encouragement and feedback, and to Armin Biere for a bug report.

References

[1] S. Buss and R. Ramyaa, “Short refutations for the equivalence-chain principle for constant-depth formulas,” *Mathematical Logic Quarterly*, vol. 64, no. 6, pp. 503–513, 2018.

[2] J. Krajíček, “A form of feasible interpolation for constant depth Frege systems,” *Journal of Symbolic Logic*, vol. 72, no. 2, pp. 774–784, 2010.