SAT-based Techniques for Approximate Circuit Design

Università della Svizzera italiana

Student: Matteo Alberici **Advisor:** Prof. Laura Pozzi **Co-advisor:** Dr. Ilaria Scarabottolo

1. Approximate Computing

Approximate computing provides gains in design area and energy consumption by loosening requirements on total accuracy. The application efficiency is improved by leveraging its error resilience. Approximate circuits outperform traditional ones in terms of speed and design area at the cost of a slight loss in computational accuracy. According to a set of constraints, an approximate circuit is the realization of a logic function that slightly deviates from the original specification. The resulting accuracy is evaluated by measuring the error between the exact circuit against its approximate version. A significant challenge resides in automatically synthesizing approximate circuits.

3. Blif2GV Translator

The Blif2GV Translator offers a clear graphical view of a circuit. It takes in input a blif file and outputs the corresponding file in GV format. A circuit object has a set of inputs and a set of outputs, and is composed of sub-circuits, each having a set of inputs, an operator, and a single output. The steps performed during the conversion are the following:

1. Parsing blif file's header to get the circuit's inputs and outputs

2. Parsing blif file's body to get the sub-circuits:

- `.subckt` lines are parsed in "one shot"
- `.names` lines require truth table identification

3. Simplifying the obtained circuit

4. Writing the corresponding GV file

In point **3.** we perform some operations in order to get a less complex circuit but with the same semantical meaning.

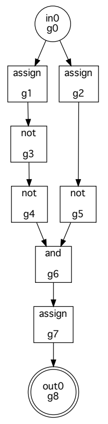


Figure 3: a more complex circuit with respect to the one seen in section 2, but with the same semantical meaning: gate 6 is an AND gate that takes in input the result of two consecutive NOT gates from the left branch and the value of a single NOT gate from the right one; thus, the AND gate will always evaluate to false.

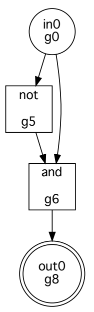


Figure 4: the circuit resulting after performing the simplification operations. All the ASSIGN gates are removed since they only rename the received input and do not affect the semantical meaning of the circuit. Furthermore, the redundant NOT gates, meaning the sequences of NOT gates that do not alter the value received at the beginning, are entirely removed.

2. Blif and GV Formats

The Berkeley Logic Interchange Format (BLIF) represents a logic circuit in textual form. A circuit is a combinational and sequential network of Boolean functions which can be viewed as a directed graph. Each node of such graph is composed of a set of inputs, a Boolean operator, and a single output. The file's body is composed of lines that declare logic gates, which are defined in two ways; let us see an example of an AND gate:

- `.subckt $and g0 g1 g2`

-
  ```
  .names g0 g1 g2
  11 1
  ```

A Graphviz Dot (GV) file provides the characteristics of a graph and allows it to be displayed graphically. Its body can be broken up in two sections:

- Nodes declaration: each line provides the declaration of a single gate

  `<node_name> [label="<node_label>" <style>]`

- Edges assignment: each line defines a link from a source to a destination node

  `<source_node> -> <destination_node>`

In order to depict a circuit graphically, we use the dot tool, which takes a GV file in input and produces a new file containing its corresponding graphical representation.

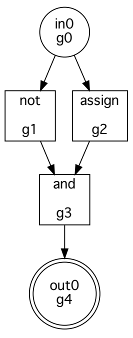


Figure 1: This circuit holds a NOT gate negating the value in input on the left branch, and an ASSIGN gate renaming the input on the right. These two values are forwarded to an AND gate, which returns the Boolean value true only if both the values in input are true. The circuit will always return false since the AND gate receives the initial value and its negated value.

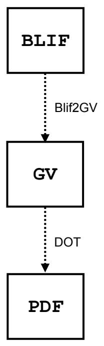


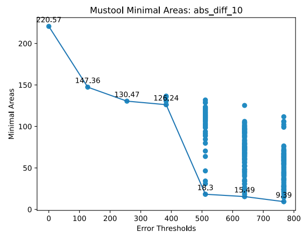
Figure 2: processing and converting a blif file into its GV version and finally displaying it graphically. The first step consists in taking a blif file and parsing it by using the Blif2GV Translator. The second step consists in taking in input the newly-generated GV file and creating its graphical representation by using the DOT tool.

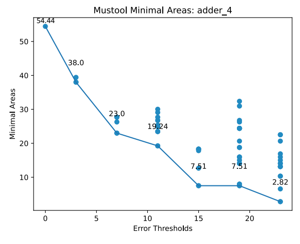
4. SAT for Approximate Synthesis

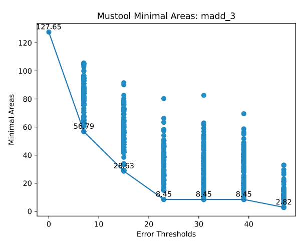
The Boolean satisfiability problem (SAT) aims at establishing whether there exists an assignment of the inputs satisfying a Boolean formula. A circuit can be expressed as a Boolean formula. Given an unsatisfiable formula ϕ, an unsatisfiable subset of clauses U of ϕ is a Minimal Unsatisfiable Subset (MUS) if every proper subset of U is satisfiable (i.e., MUSes contain the smallest number of original clauses required to still be unsatisfiable). By properly formulating the problem, MUSes can be interpreted as approximate circuits, where some constraints (i.e., gates) of the original formula have been removed.

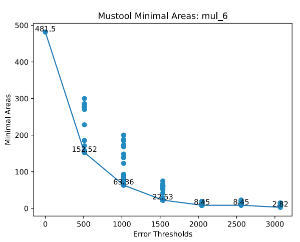
5. MUST Results Analysis

Given a set of benchmarks, we wish to find all MUSes for each among them, along with their areas. More precisely, we aim at obtaining the MUS representing the smallest approximate circuit through the toolchain with respect to an error threshold (ET) and a timeout value. The process of finding and enumerating the MUSes is carried out by the MUST toolchain. After having run the toolchain on a benchmark, we use the yosys tool in order to evaluate the area of each MUS and find the lowest one, hence identifying the smallest and most efficient approximate circuit.









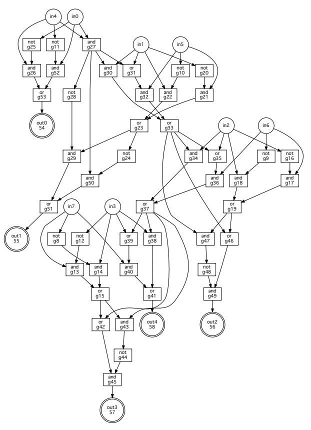


Figure 5: `adder_4` exact circuit

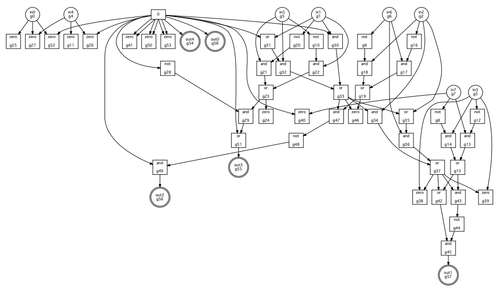


Figure 6: `adder_4` approximate circuit (ET = 23)