Circuit Vertification

Luxue Wen*

September, 2025

1 Modeling the Problem

1.1 Parameters for constructing BDD

- *inputs*: name of input variables.
- *gates*: dictionary of logic gates.
- *outputs*: name of output variables.
- *bdd*: BDD instance.
- *node_bdd*: BDD nodes of each signal.
- $nodes1, nodes2$: final BDDs of two circuits.

1.2 Intermediate Variables for constructing BDD

- $gate\_name, gate\_type, gate\_inputs$: information including name, type and inputs of logic gates.
- *remaining*: copy of logic gates that have not been mapping to BDD nodes.

1.3 Construction Function

1.3.1 Original Version

For initialization,

$$remaing = gates,$$
$$node\_bdd = inputs$$

While remaining $\neq \emptyset$,

$$f_{gate}(x_1, x_2, \ldots, x_k) = \begin{cases} x_1 \wedge x_2 \wedge \cdots \wedge x_k, & gate\_type = AND \\ x_1 \vee x_2 \vee \cdots \vee x_k, & gate\_type = OR \\ \neg x_1, & gate\_type = NOT \\ \neg(x_1 \wedge x_2 \wedge \cdots \wedge x_k), & gate\_type = NAND \\ \neg(x_1 \vee x_2 \vee \cdots \vee x_k), & gate\_type = NOR \\ x_1 \oplus x_2, & gate\_type = XOR \\ \neg(x_1 \oplus x_2), & gate\_type = XNOR \end{cases}$$

Only constructing BDD nodes when their inputs are ready,

$$\text{available} = \{x_i \in \text{gate\_inputs} \mid x_i \in \text{node\_bdd}\}$$

*Student Number: 2271796 Email: l.wen@student.tue.nl

$$node\_bdd = f_{gate}(x_1, ..., x_k),\ with\{x_1, ..., x_k\} \subseteq available$$

After constructing, deleting the gate from remaining,

$$remaining \leftarrow remaining \setminus gate\_inputs$$

Repeating until $remaining = \emptyset$.

1.3.2 Optimized Version

- Using iterative algorithm to construct BDD instead of creating the remaining dictionary.
- Introducing cache in order to avoid repeated construction.

$$\text{node\_bdd}(x) = \begin{cases} x, & x \in inputs \\ cache[x], & x \in dom(cahce) \\ \bigcup\limits_{x \in \text{inputs}(s)} \text{node\_bdd}(x), & \text{otherwise} \end{cases}$$

$$dom(cache) = \{k |\ \exists v,\ (k, v) \in cache\}$$

1.4 Equivalence Checking

Checking the equivalence of all the outputs of two BDDs that are constructed from a pair of bench files,

$$\bigwedge_{\substack{o_1 \in outputs_1 \\ o_2 \in outputs_2}} \big(\text{node\_bdd}_1(o_1) \equiv \text{node\_bdd}_2(o_1)\big)$$

2 Usage of dd.autoref.BDD library

- The program first parses the .bench files to extract inputs, outputs, and gate definitions.
- Then, it constructs BDD nodes for each logic gate and progressively combines them to obtain the output BDDs of the circuits.
- Finally, by comparing the BDDs of corresponding outputs, the program determines whether the two circuits are logically equivalent.

3 Results Part

Table 1: equivalence checking of two bench files and comparing runtime

Pair	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Equivalence	Yes	Yes	No	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	/
runtime(ms)	3	2	14	850	500	0.00	461	9	187	23	130	255	64	overrun
runtime_opt(ms)	2	1	8	500	250	0.00	210	8	31	9	94	128	17	overrun