CS 511 Formal Methods, Fall 2018 — Instructor: Assaf Kfoury

# Lecture 8: BDD's and QBF's

*September 27, 2018* — Scribe: Gavin Brown

(These lecture notes are **not** proofread and proof-checked by the instructor.)

## 1 Coursera MOOC

We watched parts of two videos from a Coursera MOOC, which stands for "Massively Open Online Course." The first dealt with the 8 Queens Problem: trying to place 8 queens on a chessboard so that none can capture another. The second dealt with rectangle fitting, a problem similar to bin-packing: given a large rectangle and several smaller rectangles, can we arrange the smaller ones so they fit inside the large one without overlap? Unlike 8 Queens, this problem requires SMT since we need to model equality.

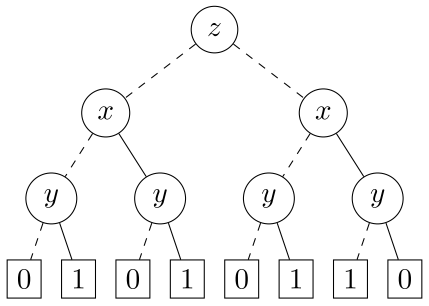
In both cases the course stressed that we work only to specify/model the problem, while the SAT-SMT solver does the solving.

To watch the videos, one must sign up for the (free) course. The course covers roughly what 511 covers in the first half of the semester. You can find the course at the following address: https://www.coursera.org/learn/automated-reasoning-sat?authMode=login

## 2 Binary Decision Diagrams, Handout 13

Binary decision diagrams (BDD's) are a rich topic. For example, Donald Knuth included roughly 170 pages on BDD's in a draft of *The Art of Computer Programming, Vol 4.* This section was last updated in 2008, and there is more that has happened in the last decade.

When we left off in Handout 13, we had introduced binary decision trees (BDT's), which are now purely combinatorial objects. For example, consider the following BDT. Our convention is that dashed lines mean "false" and sit on the left, while solid lines are on the right and mean "true."



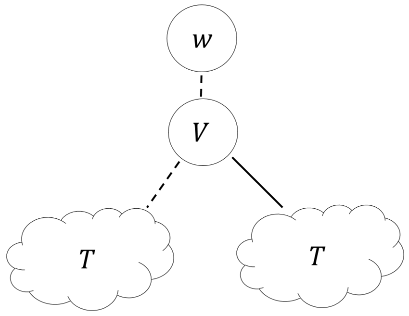
On page 363, LCS provides three valid operations for manipulating BDD's:

**C1:** Remove duplicate terminal nodes.

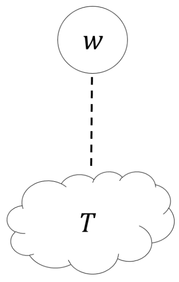
**C2:** Remove redundant destinations.

**C3:** Remove duplicate non-terminals.

The first merges all the terminal 1's and 0's into just two end nodes. After this we no longer had a tree, but instead a directed acyclic graph (DAG). The second rule, C2, allows us to delete a node when both of its outgoing edges end at the same child. The third rule is a generalization of the first, and allows us to delete a node when both its children are identical as combinatorial structures. For example, C3 allows us to take



and turn it into:



After we can no longer apply C1-C3 we are left with a representation that is often much more compact than the full BDT. For example the propositional formula (with Boolean function notation)

$$\phi := (x_1 + x_2) \cdot (x_3 + x_4) \cdot (x_5 + x_6)$$

has 6 variables and thus the BDT has $2^6$ terminal leaves. The ROBDD is much more compact, as you can see on Handout 13 page 19. Note that the order in which we expand the variables matters, as you can see on the next page.

## 3 Quantified Boolean Formulas, Handout 14

We introduce Quantified Boolean Formulas (QBF's) to give a sense of the power of quantifiers without dealing with additional concepts like equality. There are QBF solvers, although the state-of-the-art lags behind that of SAT-SMT solvers. We won't use them in this class, though.

### 3.1 Bound vs. Binding vs. Free Variables

Variables can appear in formulas three different ways. All of the variables are free in the following formula:

$$(x \vee \neg y) \wedge (z \vee y \vee v \vee \neg w).$$

If we introduce a quantifier, $v$ appears as both a binding and bound variable:

$$\exists \underbrace{v}_{\text{binding}} (x \vee \neg y) \wedge (z \vee y \vee \underbrace{v}_{\text{bound}} \vee \neg w).$$

Of course, we can also have a binding variable without a corresponding bound occurence:

$$\forall p \; \exists v \; (x \vee \neg y) \wedge (z \vee y \vee v \vee \neg w).$$

This is equivalent to an unused parameter in a program, a "dummy input." To continue in coding terms: the formula is like a single function definition, and the free variables are globally defined.

### 3.2 Naming and Overlap

We can nest quantifiers and a formula can contain non-overlapping quantifiers. We cannot have overlapping non-nested quantifiers, however, just like we cannot have a programming function in overlapping partially with another larger program.

Note that if $x$ has a bound occurence in $\phi$ is does *not* follow that $x \notin \text{FV}(\phi)$, the free variables. As a counterexample, take

$$\underbrace{x}_{\text{free}} \wedge (\forall \underbrace{x}_{\text{binding}} (\underbrace{x}_{\text{bound}} \vee y)). \tag{1}$$

This is allowed but may cause confusion. If we were coding and had overlapping global and local variable names:

```
x = y + 3
def foo(x, y)
    x
    y
```

it might be better to rename and write:

```
x = y + 3
def foo(x', y)
    x'
    y
```

Similarly we could write (1) as

$$x \wedge (\forall x'(x' \vee y)).$$

This type of name-overlap is especially difficult in first-order proof assistants, where we run into issues of "name-capturing." An example of this can be found in the handout on page 17.