15-414: Bug Catching: Automated Program Verification

Lecture Notes on Propositional Encodings*

Matt Fredrikson

Carnegie Mellon University

Lecture 15
Thursday, March 27, 2022

1 Introduction

In the last lecture, we learned algorithms to solve propositional formulas and that SAT solvers are able to solve very large formulas with millions of variables and clauses. However, in order to use existing SAT solvers, we must first encode the problem we want to solve into CNF. In this lecture, we will learn how to encode problems into the language accepted by SAT solvers, i.e. formulas in Conjunctive Normal Form (CNF).

Learning Goals.

After this lecture, you should learn that:

- Formulas can be converted in linear time to CNF using the Tseitin encoding.
- There are multiple ways to encode values from finite domains as propositional constraints, with tradeoffs that depend on the size of the encoded domain.
- Consistency and arc-consistency are desirable properties for propositional encodings when using SAT solvers that employ Boolean Constraint Propagation.

*Based on lectures notes written by Ruben Martins

2 Tseitin Encoding

Given a propositional formula, one can use De Morgan's laws and distributive law to convert it to CNF. However, in some cases, converting a formula to CNF can have an exponential explosion on the size of the formula.

Suppose we have the following formula φ,

$$\varphi = (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n)$$

and want to convert φ to CNF. If we apply De Morgan's laws and distribute law then we will obtain a formula φ' such that:

$$\varphi' = (x_1 \vee x_2 \vee \ldots \vee x_n) \wedge (y_1 \vee x_2 \vee \ldots \vee x_n) \wedge (y_1 \vee y_2 \vee \ldots \vee y_n)$$

.

Note that φ' has an exponential number of clauses, namely 2^n clauses. Can we avoid this exponential blowup on the size of the formula? Yes, with the Tseitin encoding we can transform any propositional formula into an *equisatisfiable* CNF formula.

Definition 1 (Equisatisfiable)**.** Two formulas φ and ϕ are *equisatisfiable* if φ is satisfiable iff ϕ is satisfiable.

Note that equisatisfiability is weaker than equivalence but useful if all we want to do is to determine the satisfiability of a formula.

The key idea behind the *Tseitin Encoding* is to introduce fresh variables to encode subformulas and to encode the meaning of these fresh variables with clauses. This procedure avoids duplicating whole subformulas and can transform a propositional formula into CNF with a linear increase in the size of the formula.

Example 2. Consider the formula $\phi = (x \wedge \neg y) \vee (z \vee (x \wedge \neg w))$. This formula can be viewed as a tree as depicted in Figure 1. The terminal nodes denote the atoms of the formula and the intermediate nodes denote fresh variables that encode each subformula.

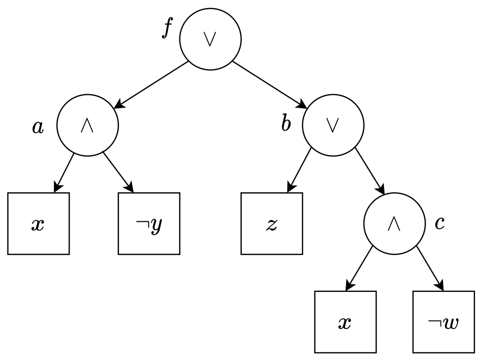

Figure 1: Tree representation of a propositional formula

For each fresh variable f, a, b, c, we introduce clauses that represent their equivalence with the respective subformula. In particular, we add the following clauses:

- $f \leftrightarrow (a \vee b) \equiv (\neg f \vee a \vee b) \wedge (\neg a \vee f) \wedge (\neg b \vee f)$
- $a \leftrightarrow (x \wedge \neg y) \equiv (\neg a \vee x) \wedge (\neg a \vee \neg y) \wedge (\neg x \vee y \vee a)$
- $b \leftrightarrow (z \vee c) \equiv (\neg b \vee z \vee c) \wedge (\neg z \vee b) \wedge (\neg c \vee b)$
- $c \leftrightarrow (x \wedge \neg w) \equiv (\neg a \vee x) \wedge (\neg a \vee \neg w) \wedge (\neg x \vee w \vee a)$

Since we want the formula to hold, we additionally need to add the unit clause (f). Note that by adding this unit clause, unit propagation would simplify the first three clauses to $(a \vee b)$.

Let's take a closer look at the previous formula $\varphi = (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n)$. Recall that this formula would require an exponential number of clauses if we would use De Morgan's laws and distribute law. If instead, we use the Tseitin Encoding we can have an equisatisfiable formula φ'' in CNF composed by the following clauses:

- $w_1 \leftrightarrow (x_1 \wedge y_1) \equiv (\neg w_1 \vee x_1) \wedge (\neg w_1 \vee y_1) \wedge (w_1 \vee \neg x_1 \vee \neg y_1)$
- $\ldots$
- $w_n \leftrightarrow (x_n \wedge y_n) \equiv (\neg w_n \vee x_n) \wedge (\neg w_n \vee y_n) \wedge (w_n \vee \neg x_n \vee \neg y_n)$
- $(w_1 \vee w_2 \vee \ldots \vee w_n)$

This would result in a formula φ'' with $3n + 1$ clauses and with n auxiliary variables.

3 Finite Domains

Many real-world problems require the encoding of finite domains to propositional logic. In this section, we will present two different ways of encoding integer domains in propositional logic by using *unary* and *binary* representations of these finite domains. The intuition behind these representations is that an *unary* representation considers a Boolean variable for each possible value, while a *binary* representation considers the binary representation of an integer.

Example 3. Suppose we want to encode the domain of an integer variable $\mathcal{X} = \{1, 2, 3\}$.

Unary representation

Consider the auxiliary variables x_1, x_2, x_3. We want to encode the meaning that x_i is *true* iff $X = i$. To encode this property we need to encode that:

1. At least one of these variables must occur:
   $(x_1 \vee x_2 \vee x_3)$
2. At most one of these variables must occur:
   $(\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3)$

Binary representation

Consider the binary representation of integers and the auxiliary variables b_1, b_0. We want to encode the following property:

- If $X = 1$ then $b_0 = 0 \wedge b_1 = 0$
- If $X = 2$ then $b_0 = 0 \wedge b_1 = 1$
- If $X = 3$ then $b_0 = 1 \wedge b_1 = 0$

In this case, the meaning of each variable can be used to implicitly encode the possible values of X. The only information we need to encode is possible integer values that are *not part of the domain* of X. In this case, $X = 4$ is not part of the domain but can be encoded using these two variables, therefore we need to disallow this value from occurring by adding the clause $(\neg b_0 \vee \neg b_1)$.

3.1 Properties of representations

The main advantage of the binary representation is that only requires a logarithmic number of auxiliary variables to encode the finite domain. In contrast, we need a linear number of auxiliary variables for the unary encoding, so it may seem like the lesser choice in most cases. However, when encoding problems using a binary encoding, it can be cumbersome to express constraints that relate to different numbers since each number is represented by a conjunction of variables instead of a single variable. Moreover, unit propagation is able to infer more information when using a unary encoding than when using binary encoding.

These considerations are illustrated by two general properties. The first, called consistency, says that whenever an assignment to the propositional variables of the encoding is not compatible with the domain being encoded, unit propagation should result in immediate conflict. For example, if a unary encoding is in use and two encoding variables are set to *true*, then any solver that employs BCP will detect the conflict before proceeding further.

Definition 4 (Consistent Encoding)**.** An encoding is *consistent* if, when given a partial propositional assignment that is not compatible with any solution to the domain, unit propagation leads to a conflict.

The second useful property is known as arc-consistency, which expands on consistency by requiring that a partial assignment will result in unit propagation that discards inconsistent assignments to the remaining encoding variables. For example, with a unary encoding, if one variable is assigned *true* then the remaining should be implied *false* by unit propagation.

Definition 5 (Arc-Consistent Encoding)**.** An encoding is *arc-consistent* if it is consistent, and additionally unit propagation on a partial assignment discards inconsistent values for the encoding variables.

While both of the encodings discussed in this section are arc-consistent, this property is especially useful for the unary encoding: while it requires more variables to encode, whenever any of the variables is decided *true*, arc-consistency means that the remaining encoding variables need not be decided, and do not expand the search space for the solver.

In practice, the size of the domain is usually the decider between choosing one or other encoding. For small domains, unary encoding is usually preferred while for large domains the binary encoding is usually the best choice.

4 Encoding Graph Coloring as a SAT problem

Suppose that we want to encode the graph coloring problem to SAT, i.e. we want to ask the question, given a graph if there exists a k-coloring such that no two nodes that are connected have the same color.

When encoding a problem to SAT, we start by defining the meaning of the *variables* that we will use in our formula. In this case, we can use an unary encoding and consider 3 variables per color for each node. Let's denote A^y, A^b, A^r Boolean variables that are true if A is colored yellow (y), blue (b), or red (r), respectively. Similarly, we can define variables B^y, B^b, B^r, C^y, C^b, C^r, D^y, D^b, D^r, E^y, E^b, E^r, for the remaining nodes. Given these variables, we can now encode the problem by adding the following clauses:

- If two nodes are connected then they do not have the same color:
  $(\neg A^y \vee \neg E^y) \wedge (\neg A^b \vee \neg E^b) \wedge (\neg A^r \vee \neg E^r)$
  $(\neg A^y \vee \neg C^y) \wedge (\neg A^b \vee \neg C^b) \wedge (\neg A^r \vee \neg C^r)$
  $(\neg C^y \vee \neg B^y) \wedge (\neg C^b \vee \neg B^b) \wedge (\neg C^r \vee \neg B^r)$
  $(\neg C^y \vee \neg D^y) \wedge (\neg C^b \vee \neg D^b) \wedge (\neg C^r \vee \neg D^r)$
  $(\neg B^y \vee \neg E^y) \wedge (\neg B^b \vee \neg E^b) \wedge (\neg B^r \vee \neg E^r)$
  $(\neg D^y \vee \neg E^y) \wedge (\neg D^b \vee \neg E^b) \wedge (\neg D^r \vee \neg E^r)$

- Each node has at-least-one color:
  $(A^y \vee A^b \vee A^r)$
  $(B^y \vee B^b \vee B^r)$
  $(C^y \vee C^b \vee C^r)$
  $(D^y \vee D^b \vee D^r)$
  $(E^y \vee E^b \vee E^r)$

- Each node has at-most-one color:
  $(\neg A^y \vee \neg A^b) \wedge (\neg A^y \vee \neg A^r) \wedge (\neg A^r \vee \neg A^b)$
  $(\neg B^y \vee \neg B^b) \wedge (\neg B^y \vee \neg B^r) \wedge (\neg B^r \vee \neg B^b)$
  $(\neg C^y \vee \neg C^b) \wedge (\neg C^y \vee \neg C^r) \wedge (\neg C^r \vee \neg C^b)$
  $(\neg D^y \vee \neg D^b) \wedge (\neg D^y \vee \neg D^r) \wedge (\neg D^r \vee \neg D^b)$
  $(\neg E^y \vee \neg E^b) \wedge (\neg E^y \vee \neg E^r) \wedge (\neg E^r \vee \neg E^b)$

A SAT solver can solve this formula and return the interpretation $I = \{A^y, B^r, C^b, D^y, E^b\}$ (for simplicity omit the variables assigned to false from the interpretation). If we decode this interpretation to the original problem, we obtain the coloring presented in Figure 2.

5 Summary

- Using the **Tseitin encoding** we can convert any propositional formula into an **equisatisfiable** CNF formula with a linear increase in the size of formula.
- Integer numbers can represented in **unary** or **binary**.
- Problems such as **graph coloring** can be easily encoded to CNF.

References

15-414: Bug Catching: Automated Program Verification

Lecture Notes on SAT Encodings

Ruben Martins

Carnegie Mellon University
Lecture 13
Tuesday, March 23, 2021

1 Introduction

In the last lecture, we learned algorithms to solve propositional formulas and that SAT solvers are able to solve very large formulas with millions of variables and clauses. However, in order to use existing SAT solvers, we must first encode the problem we want to solve into CNF. In this lecture, we will learn how to encode problems into the language accepted by SAT solvers, i.e. formulas in Conjunctive Normal Form (CNF).

2 Tseitin Encoding

Given a propositional formula, one can use De Morgan's laws and distributive law to convert it to CNF. However, in some cases, converting a formula to CNF can have an exponential explosion on the size of the formula.

Suppose we have the following formula φ,

$$\varphi = (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n)$$

and want to convert φ to CNF. If we apply De Morgan's laws and distribute law then we will obtain a formula φ' such that:

$$\varphi' = (x_1 \vee x_2 \vee \ldots \vee x_n) \wedge (y_1 \vee x_2 \vee \ldots \vee x_n) \wedge (y_1 \vee y_2 \vee \ldots \vee y_n)$$

.

Note that φ' has an exponential number of clauses, namely 2^n clauses. Can we avoid this exponential blowup on the size of the formula? Yes, with the Tseitin encoding we can transform any propositional formula into an *equisatisfiable* CNF formula.

Figure 2: 3-coloring of a graph.