A QBF Instance Generator for Chen Formulae: Implementation and Results

Noel Arteche Echeverría

Supervisor:
Montserrat Hermo

eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

FACULTY OF COMPUTER SCIENCE
UNIVERSITY OF THE BASQUE COUNTRY

Contents

1 Abstract

In this document we present an implementation for the so-called Chen Formulas of type 1 and 2. These are two classes of QBF formulas presented by Hubie Chen in his article *Proof Complexity Modulo the Polynomial Hierarchy: Understanding Alternation as a Source of Hardness*. We propose a computer program for generating instances of these classes and checking their running times on popular QBF solvers.

2 Introduction

In his 2016 article *Proof Complexity Modulo the Polynomial Hierarchy: Understanding Alternation as a Source of Hardness*, Hubie Chen showed a few exponential lower bounds on certain proof systems for the problem of deciding the unsatisfiability of quantified Boolean formulas.

The first set of formulas, now named *Chen Formulas of Type 1*, is a class of formulas defined for every positive natural number n such that the resulting formula is always unsatisfiable. Surprisingly, though, these formulae have proofs of linear size in some proof systems while, for others, these formulas constitute an example of exponential lower bound on the system.

The second set of formulas, the *Chen Formulas of Type 2*, are a more subtle set of QBF instances defined by an arithmetic charaterizing property.

In this document, we define both classes of formulae, we review the basic properties shown in Chen's article and propose a Python program that can generate the instances of any size and send them as input on popular QBF solvers available at the time of writing this report.

3 Chen Formulae of Type 1

3.1 Definition and basic properties

The Type 1 formulae, as defined by Chen in section 6 of his article, are the following:

Definition 1 *For every positive natural number n and every $i \in \{0\} \cup [n]$, let X_i be the set of variables $\{x_{i,j,k} \mid j,k \in \{0,1\}\}$. Analogously, we have the set $X'_i = \{x_{i,j,k} \mid j,k \in \{0,1\}\}$, except for $i = 0$, when X'_i is not defined. Additionally, we have, for every positive i, a variable y_i. With these variables, we define $\vec{P}_n$ to be the quantifier prefix $\exists X_0 \exists X'_1 \forall y_1 \exists X_1 \ldots \exists X'_n \forall y_n \exists X_n$. Now, we define the following sets of clauses:*

- $B = \{(\neg x_{0,j,k}) \mid j,k \in \{0,1\}\} \cup \{(x_{n,j,0} \vee x_{n,j,1}) \mid j \in \{0,1\}\}$
- *For every $i \in [n]$ and every $j \in \{0,1\}$, $H_{i,j} = \{(\neg x'_{i,0,k} \vee \neg x'_{i,1,l} \vee x_{i-1,j,0} \vee x_{i-1,j,1}) \mid k,l \in \{0,1\}\}$*
- *For every $i \in [n]$, $T_i = \{(\neg x_{i,0,k} \vee y_i \vee x'_{i,0,k} \mid k \in \{0,1\}\} \cup \{(\neg x_{1,i,k} \vee \neg y_i \vee x'_{i,1,k}) \mid k \in \{0,1\}\}$*

For every positive natural number n, a QBF instance Φ is a Chen Formula of Type 1 *when $\Phi = \vec{P}_n : \phi$, where $\vec{P}_n$ is the prefix vector defined before and ϕ is the Boolean formula obtained from the conjunction of all the clauses in B, $H_{i,j}$ and T_i for every $i \in [n]$ and every $j \in \{0,1\}$.*

The fact that the sentences Φ_n are false can be understood following Chen's intuitive argument on his article. The interest of these formulae is in the essential property that the proof size needed to show their unsatisfiability varies drastically from one proof system to another. This is in shown in Proposition 6.1 and Theorem 6.6 in the original article. Here, we reproduce them in the following theorems:

Theorem 1 *The sentences $\{\Phi_n\}_{n \geq 1}$ have QU-resolution proofs of size linear in n.*

Theorem 2 *Treelike relaxing QU-res requires proof of size $\Omega(2^n)$ on the sentences $\{\Phi_n\}_{n \geq 1}$.*

3.2 Implementation

With the above context in mind, we easily understand the interest on being able to construct instances of these sentences and check whether available QBF solvers deal with them in linear time or in exponential time.

In order to do so, we have implemented a Python program that can generate instances of both Type 1 and Type 2 formulae (Type 2 is described later in this document). We now present this program.

3.2.1 Structure

The Python project consists of the following packages and modules:

- ***main.py***: the main script, where the menu is created and presented to the user and the options are displayed.
- ***tests.py***: testing functions for repeated tests, called from the main module.
- ***/tools***: a Python package with two modules: *system_tools*, used from some operations performed on the operating system; and *QBF*, a class encapsulating the structure of a QBF instance. A QBF instance consists of a quantifier prefix and a formula. As in classic SAT solvers, the formula is represented (written in Conjunctive Normal Form) as a list of lists, where each sublist represents a clause and literals are represented using either positive or negative integers. Additionally, the QBF class has some methods for transforming a QBF instance stored in an object into a string

encoded in the QDIMACS standard, which can then be printed onto a file. Later in the document we discuss the possibility of encoding a QBF instance as a circuit instead of as a formula in CNF as well as how the QCIR standard is used for this purpose. For the moment, we will use QDIMACS, as Definition 1 defines formulas in classic CNF.

- ***/generators***: this package contains two modules, one for each class of Chen Formulae. In this section we will be discussing the generator for Type 1 formulae.

3.2.2 The generator for Type 1

The generator for Type 1 is quite simple, as Definition 1 already provides a syntactic description of the formulas. Thus, the generator consists of four functions. The main one, `generate_ChenType1(n)`, receives a positive natural number n as input and is in charge of generating the formula. This is the method that can be called from outside.

In order to construct the formula, we create an object of the class `QBF` to store the formula and then call `generate_B_clauses(phi, n)`, `generate_H_clauses(phi, n)` and `generate_T_clauses(phi, n)` to add the corresponding clauses to the formula. Once it is ready, we return the complete formula object.

The main particularity of this generator is related to the way the formulas are represented. As mentioned above, we use objects that encapsulate CNF formulas. Following QDIMACS' tradition of using integers for variables, we encode clauses as lists of integers. However, Definition 1 refers to variables of the form $x_{i,j,k}$, $x'_{i,j,k}$ and y_i. It is clear that there is a need for a function that given a certain variable for one of those types can return a unique positive integer that represents it. In other words, we need a bijection between the set of variables and a subset of $\mathbb{N}$. Since it is straightforward to verify that an instance Φ_n has $4(n+1)+4n+n = 9n+4$ variables, we need a bijection from the set of variables V to the set $\{1,\dots,9n+4\} \subset \mathbb{N}$. For this generator, we will use the following bijection, where jk_{10} is the number jk in base 10, since $j,k \in \{0,1\}$. The result of using this mapping can be seen on the table on the right:

$$f(v_{i,j,k}) = \begin{cases} jk_{10}+1 & \text{if } i=0 \\ 9i+4 & \text{if } v=y \\ 9(i-1)+jk_{10}+5 & \text{if } v=x \\ 9(i-1)+jk_{10}+9 & \text{if } v=x' \end{cases}$$

In Python, the function receives a tuple that can be of the form `(i, None, 'y')` for the variables y_i and `(i, jk, 'p')` or `(i, jk, 'np')` for the rest of them.

Naturally, since f is a bijection there is an inverse function that converts natural numbers to variables encoded as tuples, but, for the time being, it has not been implemented, as it is of no use in this generator.

$x_{0,0,0}$	1
$x_{0,0,1}$	2
$x_{0,1,0}$	3
$x_{0,1,1}$	4
$x_{1,0,0}$	5
$x_{1,0,1}$	6
$x_{1,1,0}$	7
$x_{1,1,1}$	8
$x'_{1,0,0}$	9
$x'_{1,0,1}$	10
$x'_{1,1,0}$	11
$x'_{1,1,1}$	12
y_1	13
$x_{2,0,0}$	14
$\dots$	
$x'_{2,0,0}$	18
$\dots$	
y_2	22
$\vdots$	

3.3 Experimental results

References