QBDef
Cheatsheet and language summary

Noel Arteche

July 2020

QBDef is a command-line tool written in Python that reads definitions of parameterised quantified Boolean formula (QBF) families in a domain-specific language and outputs instances of these formulas in the QCIR and QDIMACS formats.

This cheatsheet contains examples of the syntax used by the language, followed by short explanations on format of each of the sections. At the end, twp full examples are provided.

Syntax of the language

Name

```
name: My QBF Definition;
```

The name can be any piece of text (including whitespaces and unicode characters like Greek letters, math symbols and emojis) not containing `;`.

Format

```
format: CNF;
format: circuit-prenex;
format: circuit-nonprenex;
```

QBDef supports formulas written in Prenex Conjunctive Normal Form (PCNF), prenex circuits and (experimentally) non-prenex circuits. These are specified as: `CNF`, `circuit-prenex`, `circuit-nonprenex`.

Parameters

```
parameters: {
    p : int;
    n : int, `n >= 2`;
    m : other, `len(m) == n`;
}
```

The parameters field is **optional**: it can be completely removed if the definition is not parameterised.

Identifiers can be any Unicode character except `, ; : ( ) { }`. **A parameter identifier cannot start with a capital letter**.

The type tag has no effects for execution of the tool. The flag must be one of the following: `int`, `str`, `float`, `list`, `bool`, `other`.

Constraints imposed to the parameters (like $n \geq 2$) must be enclosed in backticks and written in Python syntax, like `` `n >= 2` `` (see the section on embedded Python).

Variables

```
variables: {
    x(i)        where i in 1..n;
    x(0);
    y(i, j)     where i, j in 1..n;
    z(i, j, k)  where i, j in 1..n, k in 0..`n**2 + 7`;
    w;
}
```

Variable identifiers **cannot start with a capital letter** and can be formed by any Unicode character (including Greek letters or emojis) except `, ; : ( ) { }`.

If a variable has subindices, like `x(i)` to represent x_i, we must specify the range of the subindex. This is done like in the example. Index identifiers follow the same rules as variable identifiers.

In the range limits like `1..n` or `` 1..`n + 2` ``, anything that is not a natural number literal or a reference to an identifier must be enclosed in backticks.

Blocks

```
blocks: {
    define block A := x, y;
    define block B := x, -y, A;

    define blocks grouped in G {
        C(i) := x(j);
    } where i, j in 1..n, `i != j`;

    block A quantified with E;
    block B operated with AND;
    all blocks in G operated with OR;

    ...
}
```

A ***block*** is a sequence of *bricks*, which are literals (input variables that may be negated) or references to other blocks (also possibly negated). A block can then be assigned a single *attribute*, i.e. a *quantifier* ($\forall$ or $\exists$, written with `A` or `E` or with the unicode symbols for $\forall/\exists$) or a *logical operator*. Currently the supported logical operators and their syntax (with unicode) are:

- *Conjunction*: $\wedge$ or `AND`
- *Disjunction*: $\vee$ or `OR`
- *Exclusive disjunction*: $\oplus$ or `XOR`
- *Implication*: $\rightarrow$, $\Rightarrow$ or `=>`
- *Double implication*: $\leftrightarrow$, $\Leftrightarrow$ or `<=>`
- *If-then-else*: `if-then-else` or `ITE`

A brick can be negated inside a block definition with $\neg$ or `-`.

Blocks and grouping identifiers **cannot start with an lower-case letter** and can contain any unicode symbol (including Greek letters and emojis) excluding whitespace and `, ; : ( ) { }`.

When defining multiple blocks in the same definition environment, blocks can be grouped together. A *grouping* is is just a collection of blocks, so that they can be operated together more easily.

Output

For prenex formulas, of the form $\Phi = Q : F$, where Q is the quantifer prefix and F the propositional matrix, we have something like:

```
blocks: {
    ...
    define block Phi := Q, F;
}

output block: Phi;
```

We assume that block `Q` represents the prefix and block `F` represents the matrix.

Non-prenex formulas are allowed only experimentally.

Comments

Comments must always be multiline: `/* ... */`.

Examples

A QBF without parameters

We define the formula

$$\Phi = \forall x \exists y \exists z : (x \vee y) \wedge z$$

```
name: Example1;
format: circuit-prenex;

variables: {
    x;
    y;
    z;
}

blocks: {
    define block A_x := x;
    define block E_yz := y, z;
    define block Q := A_xy, E_xy;

    block A_x  quantified with A;
    block E_yz quantified with E;

    define block B1 := x, y;
    define block B2 := B1, z;

    block B1 operated with OR;
```

```
    block B2 operated with AND;

    define block Phi := Q, B2;
}

output block: Phi;
```

A parameterised QBF

Let $n \in \mathbb{N}^*$. We will consider the formula family containing QBF over variables $x_1, \ldots, x_n, y_1, \ldots, y_n$ of the form

$$\Phi(n) = \exists x_1 \forall y_1 \exists x_2 \forall y_2 \ldots \exists x_n \forall y_n : \varphi(x_1, \ldots, x_n, y_1, \ldots, y_n)$$

where φ is the matrix is given by

$$\varphi(x_1, \ldots, x_n, y_1, \ldots, y_n) = (\bigvee_{i=1}^{n} \neg x_i) \wedge \bigwedge_{i=1}^{n} (x_i \vee y_i)$$

For instance, in $n = 2$, the QBF is:

$$\Phi(2) = \exists x_1 \forall y_1 \exists x_2 \forall y_2 : (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee y_1) \wedge (x_2 \vee y_2)$$

```
name: The more difficult example;
format: CNF;

parameters: {
    n : int, `n >= 1`;
}

variables: {
    x(i)    where i in 1..n;
    y(i)    where i in 1..n;
}

blocks: {

    /* Quantifer prefix */

    define blocks grouped in QX {
        QX(i) := x(i);
    } where i in 1..n;

    define blocks grouped in QY {
```

```
        QY(i) := y(i);
    } where i in 1..n;

    all blocks in QX quantified with E;
    all blocks in QY quantified with A;

    define blocks grouped in QXY {
        QXY(i) := QX(i), QY(i);
    } where i in 1..n;

    define block Q := all blocks in QXY;

    /* Matrix */

    define blocks {
        X := -x(i);
    } where i in 1..n;

    define blocks grouped in XY {
        XY(i) := x(i), y(i);
    } where i in 1..n;

    block X operated with OR;
    all blocks in XY operated with OR;

    define blocks {
        F := X, XY(i);
    } where i in 1..n;

    block F operated with AND;

    define block Phi := Q, F;
}

output block: Phi;
```

Embedded Python

When writing conditions and range limits in the definitions, we can use Python expressions enclosed in backticks (in fact, anything other than a reference to a variable or a natural number literal must be enclosed in backticks). This means that it is possible to use any built-in Python data structure and function as a

parameter.

Executing QBDef

QBDef is a command-line tool working on Python 3. The source code and information on how to install and run the tool can be found in the GitHub repository of this project:

`https://github.com/alephnoell/QBDef`

On Linux, the tool can be run by executing the `QBDef.py` script on a terminal:

```
python3 QBDef.py definition_file values_file [-internal] [-verbose]
                                 [-QDIMACS {file.qdimacs | [-stdIO]}]
                                 [-QCIR {file.QCIR | [-stdIO]}]
                                 [-non-prenex-QCIR {file.QCIR | [-stdIO]}]
```

If the definition has no parameters, then the second parameter must be an empty `.txt` file. If there are values to be assigned, then the second parameter is a file with contents like:

```
value: n = 10;
value: my_list = `[1, 2, 3]`;
```

The list is enclosed in backticks because it is a Python expression.