A non-CNF DIMACS style

Fahiem Bacchus[1]* and Toby Walsh[2]**

[1] Department of Computer Science, University of Toronto,
Toronto, Ontario, Canada
fbacchus@cs.toronto.edu
[2] Cork Constraint Computation Center,
University College Cork, Ireland.
tw@4c.ucc.ie

1 Introduction

In 1992, as part of the Second DIMACS Implementation Challenge on "NP Hard Problems: Maximum Clique, Graph Coloring, and Satisfiability", Michael Trick proposed a simple format for specifying clausal propositional satisfiability problems. More recently this been extended to quantified Boolean formulae, again in clausal form. There was also a DIMACS format for non-clausal formulae but this was restricted to a limited number of connectives and has not been widely used. The ISCAS'85 benchmark circuits also provided a netlist format for non-clausal formulae but this has never been formally documented, and is again restricted to a limited number of connectives.

To encourage development of non-clausal solvers, and to support a non-clausal competition at the annual SAT conference, we propose here an extension of the DIMACS style to arbitrary propositional formulae. We suspect that part of the success of the CNF DIMACS style is that it is essentially just a sequence of numbers, so it is relatively easy to write a parser for it. The extension proposed here also follows this principle.

We have also choosen to limit some of the allowed flexibility of the original DIMACS format. In particular, the original format allowed a clause specification to run over multiple lines, and a new clause to start on the same line on which the previous clause ended. A number of SAT solvers in fact never supported this flexibility, and the recent SAT competitions have also not allowed this flexibility. In line with this we restrict the use of line breaks in the specified format.

2 Problem Semantics

A Non-CNF file specifies a single output boolean circuit with some number of inputs. The circuit is *satisfiable* if and only if there is some setting of the input variables under which the circuit's output is true. The circuit is specified as some collection of interconnecting gates which eventually feed into a single *root* output. Below we describe the format for specifying the circuit.

* Supported by Natural Science and Engineering Research Council of Canada.
** Supported by Science Foundation Ireland.

3 Header

3.1 Comments

The header contains comment lines and the problem line. A comment line begins with a lower case “c”. The rest of the line is then ignored.

```
c This is an example of a comment line
```

All comment lines must come at the start of the file. No comment line can appear after the problem line appears.

3.2 Problem Line

The problem line is of the form:

```
p noncnf VARS
```

VARS is the largest IO (see below) number that appears in the circuit specification.

4 Body

Each line in the body describes one of the gates within the propositional formula. Each line has the form:

```
GATE #_PARAMETERS PARAMETER1 ... PARAMETERk IO0 ... IOn 0
```

- GATE: is the type of gate. This is a positive non-zero integer. The format specification contains a number of predefined gates types, reserves some numbers for new gate types to enter the specification, and leaves some numbers available for application specific (i.e., non-portable) use.
- #_PARAMETERS: this is positive non-zero integer. It specifies the number of parameters to follow. If #_PARAMETERS is -1 this means that there are no parameters (zero is reserved to terminate the gate). When #_PARAMETERS is -1 the next numbers will specify the IOs.
- PARAMETERi: the parameters, if any, are specified by integers. Their interpretation is specific to the gate type.
- IOi: positive and negative non-zero integers specifying the inputs and outputs to the gate. Their interpretation is specific to the gate type, and may be position dependent. A negative integer means that the input/output is negated.
- 0: each gate specification is terminated by zero, “0” and a line break. Line breaks may not appear before the gate specification is complete.

4.1 Root

The root output of the circuit is taken to be the largest IO number that appears in the circuit.

4.2 Gate types

As noted above each gate is specified by a positive non-zero integer type. There are a number of predefined gate types as well as room to add new types. Application specific implementations are also free to define their own non-portable gate types. Of course such non-portable types will probably not be supported by other solvers.

The following connectives (gate types) are currently supported[1]:

GATE	Name	#_PARAMETERS	PARAMETERS	IOs	Comment
1	FALSE	-1	none	IO0 = output	always *false*
2	TRUE	-1	none	IO0 = output	always *true*
3	NOT	-1	none	IO0 = output, IO1 = input	unary negation, $\texttt{IO}0 = \neg\texttt{IO}1$
4	AND	-1	none	IO0 = output IO1...n = inputs	n-ary conjunction, $\texttt{IO}0 = \texttt{IO}1 \wedge \ldots \wedge \texttt{IO}n$
5	NAND	-1	none	IO0 = output IO1...n = inputs	n-ary NAND, $\texttt{IO}0 = \neg(\texttt{IO}1 \wedge \ldots \wedge \texttt{IO}n)$
6	OR	-1	none	IO0 = output IO1...n = inputs	n-ary disjunction, $\texttt{IO}0 = \texttt{IO}1 \vee \ldots \vee \texttt{IO}n$
7	NOR	-1	none	IO0 = output IO1...n = inputs	n-ary NOR, $\texttt{IO}0 = \neg(\texttt{IO}1 \vee \ldots \vee \texttt{IO}n)$
8	XOR	-1	none	IO0 = output IO1...n = inputs	n-ary XOR, $\texttt{IO}0 = \texttt{IO}1 \oplus \ldots \oplus \texttt{IO}n$
9	XNOR	-1	none	IO0 = output IO1...n = inputs	n-ary XNOR, $\texttt{IO}0 = \neg(\texttt{IO}1 \oplus \ldots \oplus \texttt{IO}n)$
10	IMPLIES	-1	none	IO0 = output, IO1 = input, IO2 = input	binary implication, $\texttt{IO}0 = \neg\texttt{IO}1 \vee \texttt{IO}2$
11	IFF	-1	none	IO0 = output IO1...n = inputs	n-ary equivalence, $\texttt{IO}0 = \texttt{IO}1 \leftrightarrow \ldots \leftrightarrow \texttt{IO}n$
12	IFTHENELSE	-1	none	IO0 = output, IO1..3 = input¡	ternary if-then-else, $\texttt{IO}0 = (\texttt{IO}1 \rightarrow \texttt{IO}2) \wedge (\neg\texttt{IO}1 \rightarrow \texttt{IO}3)$
13	ATLEAST	1	k	IO0 = output, IO1..n = input	at least k inputs are *true*, $\texttt{IO}0 = \sum_{i=1}^{n} \texttt{IO}i \geq k$
14	ATMOST	1	k	IO0 = output, IO1..n = input	at most k inputs are *true*, $\texttt{IO}0 = \sum_{i=1}^{n} \texttt{IO}i \leq k$
15	COUNT	1	k	IO0 = output, IO1..n = input	exactly k inputs are *true*, $\texttt{IO}0 = \sum_{i=1}^{n} \texttt{IO}i = k$
$\geq$ 10000					Gates numbered from 10000 can be used for application specifi c gates. The numbers 16–9999 are reserved for future expansion of the standard.

[1] Please email the authors if you wish to see other connectives supported.

5 Valid Benchmarks

A *valid* benchmark is one that is subject to two additional conditions.

1. No `IO` number should be specified as an output of more than one gate.
2. The root output (the highest `IO` number specified) should not be specified as an input to any other gate.

Non-CNF solvers utilizing this format can assume that they their input is a valid benchmark. It is up to the benchmark provider to ensure that the benchmark is valid.