Mar 11, 2014

QCIR-13 Format for QBF

This document defines an input format QCIR for QBF solvers, based on the ISCAS-89 format. Unlike QDIMACS, which is restricted to CNF, QCIR format allows non-CNF formulas (expressed as combinational logic circuits). The meaning of a file in QCIR format is largely self-explanatory, so let us start with an example. Below, we show a formula and its representation in QCIR:

$$\forall v_1.\exists v_2.\exists v_3.\ \underbrace{\underbrace{(v_1 \wedge v_2)}_{g_1} \vee \underbrace{(\neg v_1 \wedge \neg v_2 \wedge v_3)}_{g_2}}_{g_3}$$

```
#QCIR-13
forall(v1)
exists(v2, v3)
output(g3)
g1 = and(v1, v2)
g2 = and(-v1, -v2, v3)
g3 = or(g1, g2)
```

As seen above, a file in QCIR format consists of four parts: (1) format identification on the first line, (2) a quantifier prefix, (3) identification of the circuit output, and (4) gate definitions. In general, a formula in QCIR format has the following form:

```
#QCIR-13
quant(var, ..., var)
⋮
quant(var, ..., var)
output(lit)
var = gate_exp
⋮
var = gate_exp
```

where

quant	::=	`exists` | `forall`
var	::=	(A string of ASCII letters, digits, and underscores, whose first character is not a digit.)
lit	::=	*var* | `-`*var*
gate_exp	::=	*gate_type*`(`*lit*`, ..., `*lit*`)`
gate_type	::=	`and` | `or` | `xor` | `ite`

Additional notes:

1. Any line that begins with the "`#`" character is a comment line.
2. `ite`(x, y, z) is an *if-then-else* gate; it is logically equivalent to $(x \wedge y) \vee (\neg x \wedge z)$. Every `ite` gate must have exactly three inputs.
3. Every `xor` gate must have exactly two inputs. This restriction helps ensure that QCIR representation can be efficiently translated to CNF without introducing new variables. Larger `xor` gates must be represented as a series of binary gates.
4. Gate variables must be defined before they are used in the definition of another gate. For example, if the gate definitions include "`g1 = and(v1, v2)`" and "`g2 = or(g1, v3)`", then the definition of `g1` must come before the definition of `g2`. Note that this requirement ensures that the circuit graph is acyclic.
5. An `and` gate with zero inputs represents the constant `true`.
   An `or` gate with zero inputs represents the constant `false`.
6. Unlike the ISCAS-89 format, only a single `output` line is permitted. The ISCAS-89 `input` lines are replaced by the quantifier bindings.
7. Unlike QDIMACS, consecutive quantifier blocks of the same type are allowed. Also, all non-gate variables must be explicitly quantified.
8. Keywords (e.g., "`and`") are case-insensitive. Variable names are case-sensitive.
9. Whitespace is ignored except in so far as it separates tokens.
10. Commas are treated as if they were whitespace. This simplifies both programs that write files and programs that parse them.
11. BNF grammar of the language syntax (not including comment lines):

$$\begin{aligned}
\textit{qcir-file} &::= \textit{format-id}\ \textit{qblock-stmt}^*\ \textit{output-stmt}\ \textit{gate-stmt}^* \\
\textit{format-id} &::= \texttt{\#QCIR-13} \\
\textit{qblock-stmt} &::= \textit{quant}\texttt{(}\textit{var}^*\texttt{)} \\
\textit{output-stmt} &::= \texttt{output(}\textit{lit}\texttt{)} \\
\textit{gate-stmt} &::= \textit{var}\ \texttt{=}\ \textit{gate\_type}\texttt{(}\textit{lit}^*\texttt{)}
\end{aligned}$$

Appendix: Open QBF. The format can be extended to allow formulas with free variables by redefining *quant* ::= `exists` | `forall` | `free`. Only a single `free` block would be allowed, and it must be the outermost block.