A Simple CNF Translator

T. Tsuchiya
Osaka University
t-tutiya@ics.es.osaka-u.ac.jp

September 10, 2001

This tool can be used to construct from a given Boolean formula a Boolean formula in CNF that is satisfiable iff so is the given formula. (The output CNF formula is in DIMACS format.)

1 Translation Rules

This tool is based on the algorithm proposed by Plaisted and Greenbaum [1]. Given a Boolean formula A, the algorithm generates another formula $L_A \wedge A^+$ where L_A is a new Boolean variable and A^+ is obtained by recursively applying the following rules.

$$\begin{aligned}
(A \wedge B)^+ &= (L_{A\wedge B} \rightarrow L_A \wedge L_B) \wedge A^+ \wedge B^+ \\
(A \vee B)^+ &= (L_{A\vee B} \rightarrow L_A \vee L_B) \wedge A^+ \wedge B^+ \\
(\neg A)^+ &= A^- \\
(A \equiv B)^+ &= (L_{A\equiv B} \rightarrow (L_A \equiv L_B)) \wedge A^+ \wedge B^+ \wedge A^- \wedge B^- \\
(A \rightarrow B)^+ &= (L_{A\rightarrow B} \rightarrow (L_A \rightarrow L_B)) \wedge A^- \wedge B^+ \\
(A \wedge B)^- &= ((L_A \wedge L_B) \rightarrow L_{A\wedge B}) \wedge A^- \wedge B^- \\
(A \vee B)^- &= ((L_A \vee L_B) \rightarrow L_{A\vee B}) \wedge A^- \wedge B^- \\
(\neg A)^- &= A^+ \\
(A \equiv B)^- &= ((L_A \equiv L_B) \rightarrow L_{A\equiv B}) \wedge A^- \wedge B^- \wedge A^+ \wedge B^+ \\
(A \rightarrow B)^- &= ((L_A \rightarrow L_B) \rightarrow L_{A\rightarrow B}) \wedge A^+ \wedge B^-
\end{aligned}$$

The implementation produces one or more clauses when each rule is applied. To see how this works, it is convenient to translate the above rules into equivalent ones as shown below.

$$\begin{aligned}
(A \wedge B)^+ &= (\neg L_{A\wedge B} \vee L_A) \wedge (\neg L_{A\wedge B} \vee L_B) \wedge A^+ \wedge B^+ \\
(A \vee B)^+ &= (\neg L_{A\vee B} \vee L_A \vee L_B) \wedge A^+ \wedge B^+ \\
(\neg A)^+ &= A^- \\
(A \equiv B)^+ &= (\neg L_{A\equiv B} \vee \neg L_A \vee L_B) \wedge (\neg L_{A\equiv B} \vee L_A \vee \neg L_B) \wedge A^+ \wedge B^+ \wedge A^- \wedge B^- \\
(A \rightarrow B)^+ &= (\neg L_{A\rightarrow B} \vee \neg L_A \vee L_B) \wedge A^- \wedge B^+ \\
(A \wedge B)^- &= (\neg L_A \vee \neg L_B \vee L_{A\wedge B}) \wedge A^- \wedge B^- \\
(A \vee B)^- &= (\neg L_A \vee L_B \vee L_{A\vee B}) \wedge (L_A \vee \neg L_B \vee L_{A\vee B}) \wedge (\neg L_A \vee \neg L_B \vee L_{A\vee B}) \wedge A^- \wedge B^- \\
(\neg A)^- &= A^+ \\
(A \equiv B)^- &= (L_A \vee L_B \vee L_{A\equiv B}) \wedge (\neg L_A \vee \neg L_B \vee L_{A\equiv B}) \wedge A^- \wedge B^- \wedge A^+ \wedge B^+ \\
(A \rightarrow B)^- &= (\neg L_A \vee \neg L_B \vee L_{A\rightarrow B}) \wedge (L_A \vee \neg L_B \vee L_{A\rightarrow B}) \wedge (L_A \vee L_B \vee L_{A\rightarrow B}) \wedge A^+ \wedge B^-
\end{aligned}$$

In addition, the following rules are necessary (although the original paper does not mention it explicitly).

$$\begin{aligned} Q^+ &= True \\ Q^- &= True \\ \neg Q^+ &= True \\ \neg Q^- &= True \\ L_Q &= Q \\ L_{\neg Q} &= \neg Q \end{aligned}$$

where Q is a Boolean variable that occurs in the given formula.

2 Syntax

This section describes the syntax of the input in detail.

2.1 Lexical conventions

An atom in the syntax described below may be any sequence of characters in the set {`A-Z,a-z,0-9,_,-`}, beginning with an alphabetic character. All characters in a name are significant, and case is significant. Whitespace characters are space, tab and newline. An atom identifies a variable.

2.2 Expressions

An expression is constructed from variables and a collection of Boolean connectives. The syntax of expressions is as follows.

```
expr ::
        atom                    ;; a variable identifier
        | "!" expr              ;; logical not
        | expr1 "&" expr2       ;; logical and
        | expr1 "|" expr2       ;; logical or
        | expr1 "->" expr2      ;; logical implication
        | expr1 "=" expr2       ;; logical equivalence
```

The order of parsing precedence from high to low is

```
        !
        &
        |
        ->
        =
```

Operators of equal precedence associate to the left. Parentheses may be used to group expressions.

References

[1] D.A. Plaisted and S. Greenbaum, "A structure-preserving clause form translation," *J. Symbolic Computation*, 2, 293-304, 1986.