Problem 3: All−NAND representation of a Boolean function

2005 ICCAD/SIGDA CADathlon

1. Introduction

Many synthesis transformations become much more easy to implement when the logic circuit is represented in some sort of standard or normalized way. Here we study the so−called all−NAND representation, i.e., a logic circuit is represented by a binary tree of solely NAND and NOT logic operation nodes. For simplicity we consider only single output circuits.

2. Problem statement

2.1. Brief description

Write a program that accepts a single logic expression in terms of AND, NAND, OR, and NOT operations, and that normalizes it to an all−NAND expression tree.

It is assumed that you program a straightforward conversion from AND/NAND/OR/NOT tree to NAND/NOT tree and remove all occurrences of double inverters.

2.2. Input/output specification

The structure of the combinational circuit is a binary tree. This tree is defined by the input expression. Its precise syntax is defined below. A normalized tree has only NOT (inverter) and 2−input NAND nodes and no occurrences of double inverters.

You may assume that the actual supplied input conforms to the format as specified below. A parser for the input format will be provided.

2.2.1. Input

The input consists of a single logic expression terminated with a semicolon. White−space (blank, tab, newline) has no semantic meaning and may be used in any quantity to enhance the readability. The same holds for comments that start with the hash character '#' and extend to the end of the line.

The Sheffer stroke '|' is the NAND operator. It binds just as strong as the AND−operator. Binding among equal precedence operators proceeds from left to right, so e.g. a | b & c is parsed as (a | b) & c. The AND−operator '&' binds stronger than the OR− operator '+'. All three are commutative and associative. Note that the AND−operator is optional and may thus also be denoted by juxtaposition of Factors. The NOT−operator '!' has the highest precedence. Do not get fooled by the close resemblance of the '|' and '!' operators. Variables are just like C identifiers; they act merely as place−holders and will not appear in the parse tree; in a negative literal, however, the implied NOT−operator is preserved.

The input format is defined by the following Backus Naur Form syntax:

```
Input       : Expression ";" .
Expression  : Term [ "+" Expression ] .
Term        : Factor [ ( [ "&" ] | "|" ) Term ] .
Factor      : Primary | "!" Factor .
Primary     : Literal | "(" Expression ")" .
Literal     : Variable [ "'" ] .
Variable    : C−language identifier .
```

2.2.1.1. Example input

Here is a typical input file for your program.

```
# Example circuit
a + !b' c d + a | b' | c + xyz | (a b + a' c);
```

2.2.2. Output

The output consists of two numbers: the size of the input tree followed on the next line by the size of the normalized tree. The size of a tree is defined as the total number of its operation nodes.

2.2.2.1. Example output

This is the output for the input example given earlier.

```
15
19
```

3. Additional information

Look at the supplied source files; they include useful material. Run the tests and verify their output.

4. References

- No dedicated paper. Any introductory course to logic synthesis will suffice.