Harvey Mudd College
Computer Science 80
Logic for Computer Science
Fall Semester 1999

Project #1 – Propositional Logic: Implementing Resolution
Phase 1: Conversion to CNF
Due 5:00pm, Thursday November 11, 1999

The purpose of this project is to implement the first phase of a resolution refutation theorem prover. In particular, you will implement the conversion to Conjunctive Normal Form. The project is to be implemented in either `rex` or `SML`. I have provided datatype/representation specifications below for both languages.

For this phase you are to implement two functions: `cnf`, and `cnf_list`. The first takes a representation of a formula and converts it to conjunctive normal form, represented as a list of lists of literals. The second does the same, but for a list of formulas, representing a conjunction of those formulas. (Its output is therefore the concatenation of the lists resulting from converting each one individually.)

The definition of `cnf_list` should be written in terms of `cnf`, and should be only a couple of lines long. The `cnf` function does almost all the work. You will find it helpful to implement `cnf` in terms of a group of support functions each of which implements one phase of the conversion.

Rex Specification

If you are using `rex` then formulas are to be represented using strings and lists as follows:

- $\bot$ — `"false"`
- $\top$ — `"true"`
- p (a propositional letter) — `"p"`
- $\neg A$ — `["NOT",`A`]`
- $A \wedge B$ — `[`A`,"AND",`B`]`
- $A \vee B$ — `[`A`,"OR",`B`]`
- $A \Rightarrow B$ — `[`A`,"IMPLIES",`B`]`
- $A \equiv B$ — `[`A`,"EQUIV",`B`]`

You will find it useful during testing to define the following variables.

```
a = "a";
b = "b";
c = "c";
d = "d";
not = "NOT";
and = "AND";
or = "OR";
imp = "IMPLIES";
equ = "EQUIV";
```

and to similarly define variables for any other propositional letters you use in you tests. This way you can write `[a,or,[not,b]]` instead of `["a","or",["not","b"]]`.

SML Specification

If you are using `SML` for your solution, you should use the following `datatype` to store formulas:

```
datatype wff = Bot
             | Top
             | Atom    of string
             | Not     of wff
             | And     of wff * wff
             | Or      of wff * wff
             | Implies of wff * wff
             | Equiv   of wff * wff;
```

The two required functions should have the type:

```
val cnf = fn : wff -> wff list list
val cnf_list = fn : wff list -> wff list list
```

During testing it will save you some typing if you make the following definitions:

```
val a = Atom "a";
val b = Atom "b";
val c = Atom "c";
val d = Atom "d";
```

and to similarly define variables for any other propositional letters you use in you tests. This way you can write `Or(a,Not(b))` instead of `Or(Atom "a",Not(Atom "b"))`.

You'll also find it helpful to issue the following two commands at the beginning of your source code, so that the system will echo back more of a given data structure.

```
Compiler.Control.Print.printDepth := 100;
Compiler.Control.Print.printLength := 100;
```