CMSC 22610 Winter 2011	Implementation of Computer Languages	Homework 1 Due January 11

Consider the language of *propositional formulae* formed from variables ($a, b, c, \ldots$), negation ($\neg$), conjunction ($\wedge$), and disjunction ($\vee$), according to the following abstract syntax:

$$\begin{array}{rcl} \phi & ::= & a \\ & | & \neg\,\phi_1 \\ & | & \phi_1 \wedge \phi_2 \\ & | & \phi_1 \vee \phi_2 \end{array}$$

We can represent propositional formulae in SML using the following datatype:

```
datatype prop
  = Var of string
  | Not of prop
  | And of prop * prop
  | Or of prop * prop
```

For example, the formula $a \wedge \neg\,(b \vee \neg\,c)$ is represented as the SML value

```
And(Var "a", Not(Or(Var "b", Not(Var "c"))))
```

We define the language of *conjunctive normal forms* (CNF) as

$$\begin{array}{rcl} C & ::= & D \\ & | & D \wedge C \\ D & ::= & A \\ & | & A \vee D \\ A & ::= & a \\ & | & \neg\,a \end{array}$$

This language can be represented as the following SML datatype:

```
datatype conjunct = And of disjunct list
and disjunct = Or of atom list
and atom = Var of string
  | Not of string
```

Because we have used the same constructor names, we must put the `prop` and `conjunct` types in separate modules:

```
structure Prop =
  struct
    datatype prop = ...
  end
```

```
structure CNF =
  struct
    datatype conjunct = ...
  end
```

One can convert an arbitrary formula to CNF by repeated application of the following rewrite rules:

$$\begin{aligned}
\neg(\neg\phi) &\Rightarrow \phi \\
\neg(\phi_1 \wedge \phi_2) &\Rightarrow \neg\phi_1 \vee \neg\phi_2 \\
\neg(\phi_1 \vee \phi_2) &\Rightarrow \neg\phi_1 \wedge \neg\phi_2 \\
\phi_1 \vee (\phi_2 \wedge \phi_3) &\Rightarrow (\phi_1 \vee \phi_2) \wedge (\phi_1 \vee \phi_3) \\
(\phi_1 \wedge \phi_2) \vee \phi_3 &\Rightarrow (\phi_1 \vee \phi_3) \wedge (\phi_2 \vee \phi_3)
\end{aligned}$$

Your assignment is to write an SML function (`toCNF`) that converts propositional formulae to their equivalent CNF. It should have the following signature:

```
val toCNF : Prop.prop -> CNF.conjunct
```

Your solution should consist of four files: `prop.sml` (holding the module `Prop`), `cnf.sml` (holding the module `CNF`), `convert.sml` (holding the `Convert` module, which contains the `toCNF` function), and `hw1.cm` (containing the CM specification). Please ensure that your name appears in a comment at the beginning of each file.

The CM specification in `hw1.cm` should be as follows:

```
Library
  structure Prop
  structure CNF
  structure Convert
is
  $/basis.cm
  prop.sml
  cnf.sml
  convert.sml
```

Submission: Put your solution in a directory named *xxx*`-hw1`, where "*xxx*" is your login ID. Create a tar file from the directory and email it to the TA (`sepopley@cs.uchicago.edu`) by **10:30am, January 11**.

Hint: One approach to this problem is to stage it as two steps: first you push the negations to the leaves, which results in a "simple" formula formed from conjunction, disjunction, and atoms. Then convert the simple formula into CNF.

History

2011-01-04 File created.