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

\[
\phi ::= a \\
\quad | \neg \phi_1 \\
\quad | \phi_1 \land \phi_2 \\
\quad | \phi_1 \lor \phi_2
\]

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

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

For example, the formula \(a \land \neg(b \lor \neg c)\) 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

\[
C ::= D \\
\quad | D \land C \\
D ::= A \\
\quad | A \lor D \\
A ::= a \\
\quad | \neg a
\]

This language can be represented as the following SML datatype:

```sml
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:

```sml
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:

\[-(\neg \phi) \Rightarrow \phi\]
\[-(\phi_1 \land \phi_2) \Rightarrow \neg \phi_1 \lor \neg \phi_2\]
\[-(\phi_1 \lor \phi_2) \Rightarrow \neg \phi_1 \land \neg \phi_2\]
\[\phi_1 \lor (\phi_2 \land \phi_3) \Rightarrow (\phi_1 \lor \phi_2) \land (\phi_1 \lor \phi_3)\]
\[(\phi_1 \land \phi_2) \lor \phi_3 \Rightarrow (\phi_1 \lor \phi_3) \land (\phi_2 \lor \phi_3)\]

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

```sml
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). Remember to document your code; you will be graded on style as well as correctness!!

Your CM file (hw1.cm) should contain the following:

```cm
Library

structure Prop
structure CNF
structure Convert

is

$/basis.cm

prop.sml
cnf.sml
convert.sml
```

**Submission:** Please submit your solution using your phoenixforge svn repository. You should place your files in a directory called hw1.

**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**

2015-01-06 First version.