02140 Languages and Parsing: Mandatory Exercise Set 2

Michael R. Hansen and Martin Fränzle

October 28, 2004

Report Requirements

The report for this mandatory assignment must be handed in no later than **November 12, at 13.00** (hard deadline). Your report must be put into the box named 02140, which is placed in just inside the west entrance of Building 322.

The report should comprise the following:

- Your names with signatures and student numbers. Group size should be 2 to 3.
- Concerning Exercise 1:
  - The course homepage contains examples. You must test your programs on these examples and present the tests in the report.
  - For the questions 2–5 you must give the programs you have constructed. Give brief explanations when neccessary. The programs will be evaluated according to correctness and elegance; but not according to efficiency. Thus aim at easily understandable programs.
- Concerning Exercise 2:
  - Test examples are available on the course homepage. You must test your programs on these examples and present the tests in the report.
- Concerning Exercise 3:
  - The proofs will be evaluated according to correctness, precision and elegance. Thus, aim at presenting the proof in a brief and understandable manner.

Exercise 1: Tautology Checker for Propositional Logic

A formula in propositional logic is either an atom given by its name which is a string, or a composite proposition built from atoms using the operators for negation ($\neg$), conjunction ($\wedge$), and disjunction ($\vee$).

For example,

$$\mathrm{P} \vee (\mathrm{Q} \wedge \neg \mathrm{R})$$

is a formula of propositional logic, where P, Q and R are the atoms.

For a given *assignment of thruth values* to the atoms, a formula is either true or false according to the well-known truth tables for the propositional operators ($\neg$, $\wedge$ and $\vee$).

A formula ϕ is called *satisfiable* if there is a value assignment which makes it true, and ϕ is called a *tautology* if it is true for any assignment of truth values to its atoms. If ϕ is a taotology, then ϕ is also called a *valid* formula.

The formula in the example above is satisfiable, since the value assigment

$$[\mathrm{P} \mapsto \mathrm{true}, \mathrm{Q} \mapsto \mathrm{false}, \mathrm{R} \mapsto \mathrm{false}]$$

makes it true, while it is not valid as the assignment

$$[\mathrm{P} \mapsto \mathrm{false}, \mathrm{Q} \mapsto \mathrm{false}, \mathrm{R} \mapsto \mathrm{false}]$$

makes is false.

1. Give examples of formulas containing at least three different atoms which are
   - not satisfiable
   - satisfiable and not valid (do not use the above example)
   - valid
2. Define an SML datatype to represent formulas in propositional logic.
3. A formula is in *negation normal form* if the negation operator only appears as applied directly to atoms. Write an SML function transforming a formula into an equivalent formula in negation normal form, using the de Morgan laws:

$$\begin{array}{rcl} \neg(p \wedge q) & \Leftrightarrow & (\neg p) \vee (\neg q) \\ \neg(p \vee q) & \Leftrightarrow & (\neg p) \wedge (\neg q) \end{array}$$

   and the law: $\neg(\neg p) \;\Leftrightarrow\; p$.
4. A *literal* is an atom or its negation. A formula is in *conjunctive normal form* if it is a conjunction of formulas, where each conjunct (i.e. formula in the conjunction) is a disjunction of literals. Write an SML function which transforms a formula into an equivalent formula in conjunctive normal form, using the above result and the laws:

$$\begin{array}{rcl} p \vee (q \wedge r) & \Leftrightarrow & (p \vee q) \wedge (p \vee r) \\ (p \wedge q) \vee r & \Leftrightarrow & (p \vee r) \wedge (q \vee r) \end{array}$$

   Argue *informally* that the function value is in conjunctive normal form, under the assumption that the argument is in negation normal form.
5. A disjunction of literals is a tautology exactly when it contains both A and $\neg A$, for some atom A. A conjunction is a tautology precisely when each conjunct is a tautology. Write a tautology checker in SML, i.e. an SML function which determines whether a formula is a tautology or not.
6. Discuss briefly how you could decide the satisfiability of a formula in propositional logic.

Exercise 2: Scanning and Parsing Formulas

You shall make a scanner and parser for formulas of propositional logic, and combine the scanner and parser with your program from Exercise 1 in order to check whether formulas are tautologies.

Concrete syntax for formulas

The concrete syntax of formulas is described as follows:

Atoms: An atom is an non-empty string where the first symbol is a letter and the remaining symbols are letters or digits, for example:

```
P    p    x1   x87kdkj   v1   SweetJohn
```

Propositional formulas: For the propositional part we use `!` for negation, | for disjunction, `&` for conjunction, => for implication and <=> for biimplication. Brackets may be put around a formula. An example of a formula is:

```
  ! SweetJohn
& !( SweetPeter <=> SweetPaul)
& (SweetPaul => SweetJohn)
=> SweetPeter
```

The precedence rules for the logical connectives are:

!	negation	highest
&	conjunction	
|	disjunction	
=>, <=>	implication, equivalence	lowest

Furthermore, conjunction and disjunction associate to the left, while implication and equivalence associate to the right.

Format of the file

The scanner and parser should read its input from a named file, where the file contains a sequence of formulas separated by ';'. An example of the contents of a file containing a sequence of 3 formulas is:

```
!( (P & !Q ) | (R & Q) );

  ! SweetJohn
& !( SweetPeter <=> SweetPaul)
& (SweetPaul => SweetJohn)
=> SweetPeter
; ( !p => !q) => (q => p)
```

Your shall make a program which can read a sequence of $n, n \geq 0$ well-formed formulas $\phi_1; \phi_2; \cdots; \phi_n$ from a named file and as result it should give a list of n truth values $[b_1, b_2, \ldots, b_n]$, where b_i is true iff ϕ_i is a tautology.

1. Construct input specifications to 'mosmllex' and 'mosmlyacc'. (Hint: exploit that implication and equivalence can be expressed in terms of negation, disjunction and conjunction.)
2. Test your program on the example files available on the homepage. Document the result. Does it match your expectations?

Exercise 3: Contextfree grammars and DFAs

In this exercise we shall consider contextfree grammars of a simple form called *simple right-linear grammars*. Each productions of such a grammar has one of the forms

$$S \to aT \qquad\qquad \text{or} \qquad\qquad S \to \epsilon$$

where S and T are non-terminals and a is a terminal symbol.

From every DFA A we can construct a simple right-linear grammar accepting the same language as A. Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA, where $Q \cap \Sigma = \emptyset$. The simple right-linear grammar constructed from A is defined by:

$$G(A) = (Q, \Sigma, P, q_0)$$

where

$$P = \begin{array}{l} \{q \to aq' \mid \delta(q,a) = q',\ \text{for } q, q' \in Q \text{ and } a \in \Sigma\} \\ \cup\ \{q \to \epsilon \mid q \in F\} \end{array}$$

1. Give $G(A)$, where A is the automaton in Exampe 2.4 in HMU, and show $001100 \in G(A)$.
2. Prove, for an arbitrary DFA $A = (Q, \Sigma, \delta, q_0, F)$ where $Q \cap \Sigma = \emptyset$, that

   $$\hat{\delta}(q, w) = q' \qquad \text{iff} \qquad q =>^*_{G(A)} wq'$$

   for any $q, q' \in Q$ and $w \in \Sigma^*$.
3. Prove $L(A) = L(G(A))$.