Heinrich Heine Universität Düsseldorf

Software Engineering and Programming Languages
Prof. Dr. Michael Leuschel
David Geleßus, Fabian Vu

Introduction to Logic Programming – WS 2023
Exercise Sheet 5

1 Exercises

There are unit tests for each programming exercise in the ILIAS.

The exercises will be discussed on 21st November 2023.

Exercise 1 (Lecture – Inverted Classroom)
Watch the lecture videos *10 LP FOL Models*[1], and *10 LP FOL Resolution*[2] in the HHU Mediathek. The corresponding slides are uploaded in ILIAS: 6_FOL.pdf (slides 15-64)

The complete playlist is available at: `https://mediathek.hhu.de/playlist/691`.

Note: you have to log in with your HHU account (Uni-Kennung) to see the lecture videos!

Exercise 2
Define the following statements in propositional logic:

1. The system heats if the cover is closed and the power is on.
2. If the lamp shines, the power is on.
3. The lamp shines.
4. The cover is closed.

Rewrite above statements to conjunctive normal form and prove by contradiction and resolution that the system heats.

Exercise 3 (Simplifying Propositional Formulae)
In this exercise we want to analyse and manipulate propositional formulae. We represent propositional formulae as terms in Prolog while propositional statements are represented as Prolog atoms. For instance, the propositional formula

$$a \wedge b \vee c \tag{1}$$

is represented as the term

```
1 or(and(a, b), c)
```

[1] `https://mediathek.hhu.de/watch/221dc19e-567a-4673-a89d-1e754f3b66e2`
[2] `https://mediathek.hhu.de/watch/65662f18-f9b6-4814-acee-47aea268003d`

in Prolog. Implement the following predicates:

a) `is_atomic_expr(+Term)` is true iff `Term` is a propositional statement.

b) `is_literal(+Term)` is true iff `Term` is a literal (positive or negative propositional statement).

c) `simplify_expr(+Term, -Simplified)` simplifies a given propositional formula by applying the following rules:

- Remove double negations: $\neg\neg A \equiv A$
- DeMorgan's Laws
  (a) $\neg(A \wedge B) \equiv \neg A \vee \neg B$
  (b) $\neg(A \vee B) \equiv \neg A \wedge \neg B$

d) `is_clause(+Term)` is true iff `Term` is a clause after simplification.

e) `is_horn_clause(+Term)` is true iff `Term` is a horn-clause after simplification.

f) `is_denial(Term)` is true iff `Term` is a denial after simplification.

Exercise 4 (Greatest Common Divisor and Applications)
In the fourth exercise sheet, we implemented a predicate `gcd(+X, +Y, -Gcd)`, which calculates the greatest common divisor of two natural numbers `X` and `Y`, and `coprime(+X, +Y)`, which is true if the two numbers `X` and `Y` are coprime (their GCD is equal to 1). We provide implementations for these two predicates in the Prolog file containing the unit tests for this exercise.

a) Implement a predicate `range_1(+M, -L)` which computes a list of numbers within a given range. A call to `range_1(M, L)` unifies `L` with a list of numbers n with $1 \leq n < M$.

Example:

```
?- range_1(10, L).
L = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

We provide a partial implementation for `range_1/2` to start with in the Prolog file containing the unit tests for this exercise.

b) Implement a predicate `phi(+M, -N)` which counts all numbers n with $1 \leq n < M$ which are coprime to `M`.

Use `range_1/2` and `coprime(+X, +Y)`.

Example:

```
?- phi(10, N).
N = 4
```