Logic Overview

1. Propositional logic and truth tables; tautologies, consistent formulas, and contradictions; propositional equivalences; normal forms (Chapter 1).
2. Natural deduction for propositional logic (Chapter 3).
3. Goal-directed deduction for propositional logic, including ready for computation form (Chapter 4).
4. Syntax and semantics of first-order logic; quantifier equivalences; models; theorems (valid formulas), consistent/satisfiable, and inconsistent/unsatisfiable formulas (Chapter 7, in particular Section 7.4 and exercises).
5. Goal-directed deduction for first-order logic, including unification and Skolemization (Chapter 8).

Prolog Overview

1. Prolog syntax and terminology, matching, goal execution.
2. Lists: notation, head/tail-pattern, built-in predicates for list manipulation.
3. Arithmetic expressions: arithmetic evaluation using the `is`-operator, built-in arithmetic functions/operators and relations.
4. Operators: precedence and associativity, defining operators using `op/3`.

Prolog Overview (continued)

5. Backtracking: how it works, benefits and problems; cuts: how they work, benefits and problems; negation as failure: closed world assumption, the `\+`-operator; disjunction using `;` (semicolon).
6. Logic foundations: translation of Prolog clauses into first-order sentences, Horn formulas, goal execution as a deductive process in resolution or the goal-directed system (the latter point we have not discussed in detail).

Lists

Recall how lists can be represented in Prolog:

- Normal list notation:

  ```
  [1, 2, 3, X, dog]
  ```

  In particular, `[]` is the empty list, `[E]` is a list with exactly one element, etc.

- Bar notation:

  ```
  [1, 2, 3, X | Rest]
  ```

  The term after the bar always represents a list itself.

- Head/tail-pattern (a special case of the bar notation):

  ```
  [Head | Tail]
  ```

Matching and Arithmetic Evaluation

- To evaluate an arithmetic expression and assign the result to a variable use the `is`-operator. Example:

  `N is M + 3` (assuming `M` is instantiated to a number)

- To compare two arithmetic expression use `=:=` (or other arithmetic relations). Example:

  `3 + 4 =:= max( 2, 7)`

- In all other cases in which you might be tempted to use `is` or `=:=` you want to use `=` (term matching).

Matching Exercises

Do the following queries succeed? If yes, what would be the variable instantiations made by Prolog?

```
?- [elephant] = [_ | X].

?- [1, 2, 3] = [H | T], X = [T | [H]].

?- 5 = (5 = 5).

?- X = 27 + 3, Y is X.
```

Matching Solutions

```
?- [elephant] = [_ | X].
X = []
Yes

?- [1, 2, 3] = [H | T], X = [T | [H]].
H = 1
T = [2, 3]
X = [[2, 3], 1]
Yes
```


Matching Solutions (continued)

```
?- 5 = (5 = 5).
No

?- X = 27 + 3, Y is X.
X = 27+3
Y = 30
Yes
```