Semantic and Declarative Technologies

View PDF
Instructor(s):

Péter Szeredi

Short Description of the Course:
The course presents two novel computing technologies: **declarative programming** and the **semantic web**. Both these approaches are based on mathematical logic and formal reasoning.

*Declarative programs use mathematical statements to describe the problem (**what** is to be solved), as opposed to traditional, imperative programs, containing detailed instructions on **how** to solve the problem.* Within declarative programming, the course focuses on logic and constraint programming. We discuss the principal features of **Prolog**, the main logic programming language; as well as its extensions covering **constraint programming**. We use puzzles, such as Sudoku, as motivating examples; we show how to develop programs for checking puzzle solutions and solving puzzles.

The vision of the *semantic web* is to make computers able to **understand** the information available on **the web**, as opposed to current search engines, which merely *read* this information. We discuss the standard languages developed by the World Wide Web Consortium (W3C) for formalizing knowledge on the web, such as the **RDF** Resource Description Framework, the RDF Schema **RDFS**, and the Web Ontology Language **OWL**. We also provide a detailed overview of **description logics** (DLs), the mathematical formalism behind these languages. We briefly discuss the most important reasoning algorithms for DLs, as well as interactive development environments, such as the **Protégé** ontology editor.

Aim of the Course:
The course aims at introducing the students to some new approaches for solving **practical** problems in computer science, which use techniques rooted in **mathematical logic**. In the first part of the semester we focus on declarative, logic-based approaches in programming, while in the second part we discuss the semantic web, where logic is used for representing and searching various types of information on the web.

A common feature of these technologies is that the underlying execution/search mechanism is in fact based on **logical deduction**. This makes it possible to provide a high level, **human-oriented** approach both in programming and in knowledge-based search on the web. The high level view also means that these technologies can be implemented efficiently in multi-processor and **multi-core** environments.

Prerequisites:
The course requires some familiarity with first order mathematical logic, its connectives, quantifiers, etc., as well as with basic programming concepts, such as if-then-else construct and recursion. The ability to formulate recursive function definitions is crucial.

Detailed Program and Class Schedule:
The first part of the course introduces *declarative programming*, the *Prolog* language and its extensions for handling *constraints*.

The slogan of declarative programming is “*what* rather than *how*”. This is because a declarative program focuses on the task being solved (*what* to solve), rather than on providing specific instructions for *how* to solve the problem. A declarative program consists of statements (in the mathematical sense), describing the universe of the problem to be solved. A declarative program uses *single assignment* variables, which denote a possibly unknown entity, similarly to variables used in mathematical arguments. Declarative programs rely on recursion, instead of loops.

Within the declarative paradigm there are two main directions: *functional programming*, which is built around functions; and *logic programming*, which uses relations. The LISP language, developed in late 1950s, was the first functional language, followed by several more sophisticated languages, such as SML, Haskell and Erlang, just to mention a few. The Prolog language, originating in the early 1970s, is the first and still the most widespread logic programming language.

A Prolog program can be read in two ways: as a collection of formal statements (*declarative semantics*) and as instructions for how to reduce a problem to solving some other (sub)problems (*procedural semantics*). These two faces of Prolog reflect the *what* and the *how* side of problem solving, although the *how* part is on a much higher level than in imperative languages. This is because the Prolog problem reduction mechanism is based on *pattern matching*, and is *non-deterministic*. The latter means that Prolog uses backtracking to try multiple reductions, until a successful one is found.

The *constraint logic programming* (CLP) paradigm was introduced in the late 1980s, and is implemented mostly as an extension to Prolog. The generic schema CLP(X) extends Prolog with specific relations (constraints) within the domain X, and also adds strong reasoning capabilities in this domain. The most developed and most widely used variant is CLP(FD), where FD stands for *finite domain*. CLP(FD) uses the techniques of *Constraint Satisfaction Problems* (CSP), a branch of Artificial Intelligence.

The first part of the course is a practical introduction to Prolog and CLP(FD) using SICStus Prolog, one of the most widely used and most efficient Prolog implementations. As motivating examples we develop several variants of checkers and solvers for puzzles such as Sudoku.

The second part of the course gives an introduction to *semantic technologies* focusing on the *Semantic Web*. The course also discusses *Description Logics*, as the mathematical background for semantic technologies.

Semantic technologies are about bringing semantics, i.e. *meaning* into various computer applications. In the forefront of this research is the Semantic Web initiative, which aims at making web search much more intelligent than today. It's vision is to make it possible for the computers to *understand* the information available on the web, as opposed to current search engines, which merely *read* this information.

Several steps are needed for this vision to come true. First, the textual information on the web has to be transformed – either manually, or, preferably, automatically – into formal, structured statements. Second, the common background knowledge, which we take for granted when reading a text, has to be formalized, i.e. transformed into a computer-processable format. Note that a collection of formal statements of both these kinds is often called an *ontology*. Third, a *reasoning engine* has to be developed, which is capable of deriving new pieces of information from the old ones.

The World Wide Web Consortium has published several standards for formalizing knowledge on the web. The Resource Description Framework (RDF) makes it possible to include formal statements in the web pages, while the RDF Schema (RDFS) allows for formulating simple forms of background knowledge. The Web Ontology Language (OWL) supports more complex forms of background knowledge, while still offering efficient reasoning capabilities.

Description Logics (DLs) is a family of formal languages each of which is a subset of first order logic. DLs provide the mathematical background for the RDF, RDFS and OWL languages, while theorem-proving algorithms for DLs are used in the reasoning engines developed for the Semantic Web languages.

Schedule of the course

There are two sessions of 100 minutes per week. Most sessions include both a lecture by the instructor and a hands-on lab or problem-solving practice.

W1:

- Background: propositional and first order logic.
- Semantic and declarative technologies, an overview.
- Imperative and declarative programming, a comparison.
- Basic constructs of the Prolog language: clauses, variables, constants.
- Simple Prolog programs: declarative reading and procedural execution.

W2:

- Advanced Prolog control constructs: disjunction, negation, if-then-else.
- Compound data structures.
- The unification algorithm.
- Execution models for Prolog: goal reduction and procedure-box.
- Tracing Prolog programs.

W3:

- Lists as syntactic extensions.
- Basic list handling predicates: append, naïve and efficient reverse, member, select.
- Operators.
- A summary of Prolog syntax.

W4:

- Efficient programming in Prolog: the cut predicate.
- Indexing.
- Tail recursion, accumulators.
- Transforming an algorithm with mutable variables to Prolog.
- Definite clause grammars.

W5:

- An overview of built-in predicate groups.
- All solution predicates.
- Meta-predicates.
- Dynamic predicates.
- The loop notation.

W6:

- An overview of SICStus Prolog libraries.
- Constraint Logic Programming (CLP) – basic principles.
- The background of finite domain constraint solving: Constraint Satisfaction Problems.

W7:

- The CLP(FD) library: arithmetic and membership constraints.
- Domain- and bound-consistency.
- Implementing constraints using daemons.
- Labeling options: variable/value selection, direction, user-defined labeling.

W8:

- Reified constraints, domain- and bound-entailment.
- Propositional constraints.
- Combinatorial constraints.
- Practical applications of the constraint technology – highlights.

W9:

- Semantic Web – the vision.
- Basic technologies of the today's web: HTML, XML.
- A roadmap from the today's internet to the Semantic Web.

The basics of Resource Description Framework (RDF): Uniform Resource Identifiers (URIs), triples, notational variants.

W10:

- Advanced RDF constructs: reification, containers, collections.
- The RDF Schema: classes, properties, hierarchies, restrictions.
- Querying RDF resources.

W11:

- Description Logics.
- Terminological and assertional knowledge: the TBox and the ABox.
- The main building stones of DLs: concepts and roles.
- A hierarchy of DL languages: AL, ALC, SHIQ, SROIQ and beyond.
- Reasoning tasks for DLs.

W12:

- The Web Ontology Language OWL.
- Constructs for building classes: property restriction, intersection, union, complement.
- Class axioms: subclasses, equivalence and disjointness.
- Property axioms: subproperties, equivalent and inverse properties.

W13:

- Building and using ontologies
- The Protégé ontology editor, its user interface and services.
- Analyzing and querying ontologies.
- The underlying reasoning algorithms – a brief overview.

W14:

- Declarative and Semantic technologies – an outlook.
- Major implementations of declarative languages.
- Application areas of declarative programming.
- Declarative technologies and parallel/multi-core computing.
- Semantic aspects of Enterprise Application Integration.
- Semantic Web Services.
- Related technologies: decision support systems, rule-based languages.

Method of instruction:
Classes consist of lectures, problem solving sessions and hands-on labs. The two latter activity types help students to understand better and to use in practice the material presented in the lectures.

In the Declarative Programming part the hands-on lab sessions help students to get acquainted with the programming environment by writing smaller programs on their laptops, guided by the instructor. The problem solving sessions in the Semantic Technologies part are concerned with e.g. modelling problems in Description Logics, and performing reasoning tasks on the obtained model (knowledge base).

In the Declarative Programming part students are given 2-3 homeworks, each containing several small-to-mid sized programming tasks, to be solved individually and submitted to an automated test system. There are two large programming assignments as well, usually related to solving Sudoku-type puzzles.

Further information can be obtained from the current course homepage: http://cs.bme.hu/~szeredi/ait/

Grading:
The final grade is made up of the following components.

Class Participation and Homeworks:	20%	
Two Programming Assignments (20% each)	40%	
Mid-term test (Declarative Programming)	20%	sample
Final exam (Semantic Technologies)	20%	sample

The maximum cutoffs for letter grades A, B, etc. will be at the traditional 90%, 80%, etc. with plus and minus grades given at the top or bottom 3% of the intervals.

Textbooks:
Clocksin, W.F., Mellish, C.S.: *Programming in Prolog using the ISO standard*. Springer Verlag, 2003.
Szeredi, P., Lukácsy, G., Benk?, T.: *The Semantic Web explained.* Cambridge University Press, 2014.

Instructors' bio:

Péter Szeredi (born 1949) is an Associate Professor at the Budapest University of Technology and Economics. Between 1972 and 2003, he worked for several software R&D companies in Hungary. In the mid 1970s he authored the first Hungarian Prolog interpreter, and led the development of the MProlog system, a pioneering Hungarian software product sold worldwide in the 1980s. He worked as a research fellow at UK universities (Manchester and Bristol, 1987-1990) and at the Swedish Institute of Computer Science (1998-1999). His main research fields are semantic technologies, as well as logic and constraint programming. He edited and co-authored a textbook on the Semantic Web to be published by Cambridge University Press. He is the author or co-author of about 90 peer-reviewed publications, including 14 books and book chapters. He has received several academic awards and is among the 15 researchers recognized by the Association of Logic Programming as “Founders of the field of Logic Programming”.