Logic Programming and Deductive Databases

Chapter 1: Introduction

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Summer 2024

http://www.informatik.uni-halle.de/~brass/lp24/

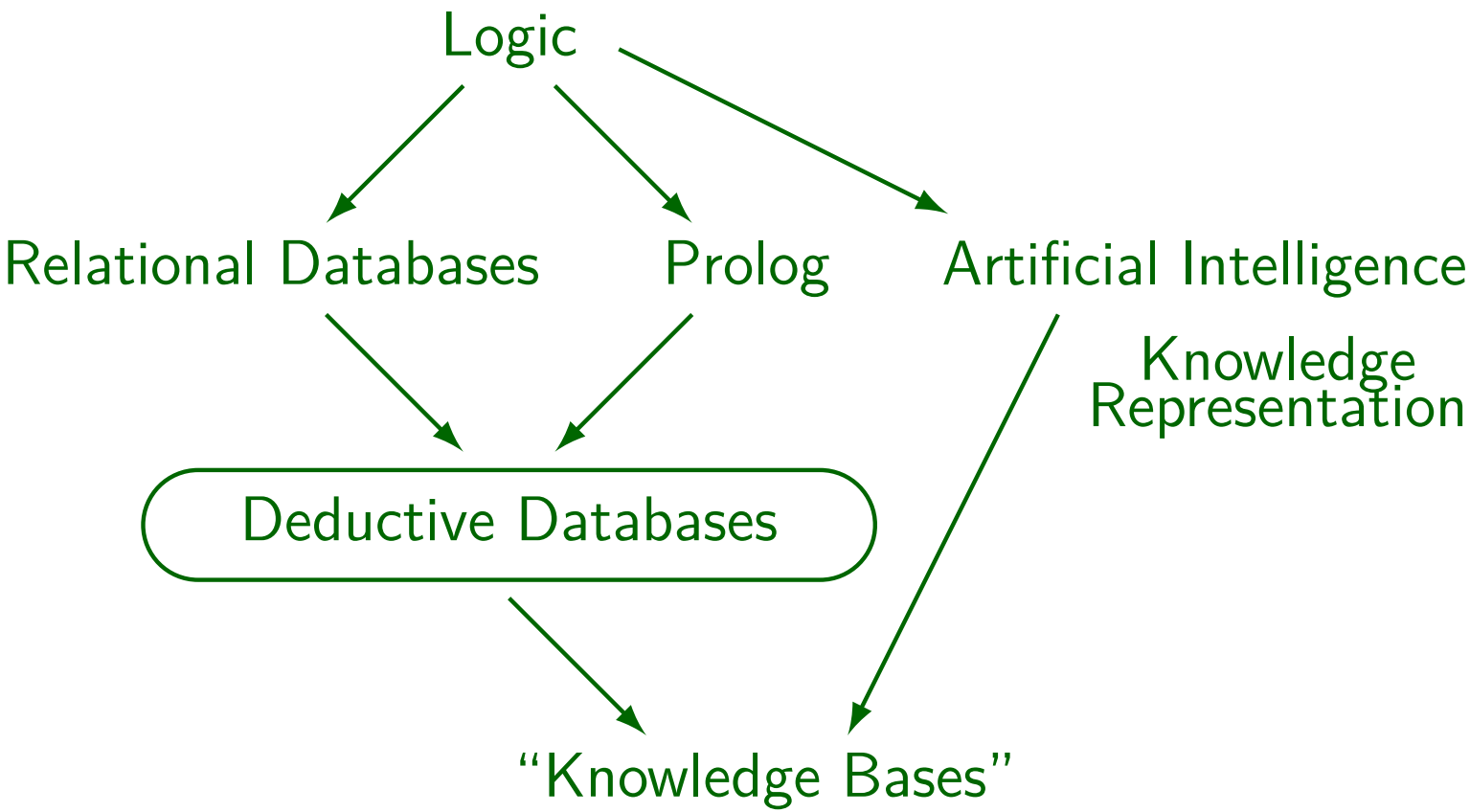
Objectives

After completing this chapter, you should be able to:

- explain the difference between declarative (logic) programming and imperative programming.
- explain what a deductive database is.
- explain the main strengths/advantages of deductive databases compared with classical relational DBs.
- develop simple Prolog/Datalog programs.

Contents

Area / Context



Logic Programming (1)

- Ideal of logic/declarative programming:
  - The program is a specification of the problem,
  - and the system automatically computes a possible solution that satisfies the given conditions.
- I.e. the programmer specifies
  - what is the problem, but not
  - how to compute a solution.
- I.e. the program is a set of axioms, and computation is a proof of a goal statement from the program.

Logic Programming (2)

- SQL is a declarative query language. The user specifies only conditions for the requested data:

```
SELECT X.HOMEWORK, X.POINTS
FROM   SOLVED X
WHERE  X.STUDENT = 'Ann Smith'
```

- Advantages of declarative languages:
  - Often simpler/shorter formulations: The user does not have to think about efficient execution. Enhanced productivity.
  - Easier adaptation to changing environments.

    For instance, parallel hardware (multi-core CPUs). Importance of Cache.
  - Better formalization, simpler verification, easier optimization.

Logic Programming (3)

- Can this work also for programming languages, not only query languages?
- Prolog ("Programming in Logic") is
  - a programming language based on these ideas.
  - used in industry for real problems.
  - not an ideal logic programming language.

    Sometimes one must know how it is executed and give some execution information. A Prolog program is not pure logic.
- Deductive databases are purer, but are still in the research state.

Logic Programming (4)

- Programming Inefficiency $*$ Runtime Inefficiency $\geq$ Constant (Robinson)
- In declarative languages,
  - the productivity of the programmer is often greater than in imperative languages, but

    E.g. Prolog program 10-fold shorter than similar C++ Program.
  - the runtime of the program is often longer.
- Programmers are expensive ("software crisis"), computers become faster and cheaper.

  I.e. the constant in the above inequality shrinks because of advances in hardware technology.

Logic Programming (5)

- Algorithm = Logic + Control (Kowalski)

  Imperative/Procedural Language: Explicit Control, Implicit Logic.

  Declarative/Descriptive Language: Explicit Logic, Implicit Control.

- Imperative languages (e.g. C) are coupled to the Von Neumann architecture of today's computers.
- Declarative languages have a larger independence of current hardware/software technology:
  - Simpler Parallelization
  - More powerful optimization

    This includes using new algorithms: If the next version of Oracle contains a new join algorithm, existing queries will profit from it. Already using a new index is a new algorithm.

Logic Programming (6)

- Naturally, logic programming languages are especially well-suited for knowledge-intensive tasks.

  However, e.g. compilers for Prolog are normally written in Prolog.

- E.g. expert systems and natural language processing are typical applications of Prolog.

- Also problems, where only constraints for a solution are given (development of time-tables, schedules) are well treated by logic programming.

  In Prolog or special "Constraint Logic Programming" languages.
  For NP-complete problems, there is anyway no good algorithm, so why bother to write one down?

Contents

Deductive Databases (1)

A Deductive Database is ...

- An integrated system consisting of a DB and a declarative programming language (Prolog-like).

  In my view, this is the most important motivation for working on deductive DBs. The combination DB+PL is needed, and is often done, but so far only with imperative languages (with problems at interface or non-declarative querying).

- "Pure Prolog" with special support for managing large sets of facts.

- A relational DB with a new query language (Datalog) and the possibility to define recursive views.

- A restricted theorem prover: It can handle only quite simple formulas, but very many of them.

  "deduce" = "compute a logical consequence".

Deductive Databases (2)

A Deductive Database Consists of ...

- a relational database (EDB),
  which defines relations/predicates "extensionally",
  i.e. by enumerating all tuples, and
- a logic program (IDB),
  which defines relations/predicates "intensionally",
  i.e. by giving rules (formulas of a particular kind).

Example for extensional vs. intensional:
time-table/schedule for busses (very regular data).

Datalog vs. Prolog (1)

Deductive DBs Permit Less Control, More Logic:

- Order of rules is not relevant.
- Order of conditions in the rule body often not relevant (depending on system/chosen optimizations).
- No cut (used in Prolog to prune the search tree).

  The cut will be explained further below. Sometimes, the cut only gives hints to the Prolog system for faster execution. But in practical Prolog programming it is used also in a way that modifies the logical meaning of the program. This violates the logic programming idea.

- Termination is guaranteed if no lists, term constructors or datatype functions are used.

Datalog vs. Prolog (2)

Reason (Database vs. Programming Language):

- The Prolog programmer knows how predicates are used (called): Which arguments are given (input), and which are variables (output).

  Closed system: There is only one main predicate, which calls all others.

- Databases allow very different queries.

  When a predicate (view) is defined, one normally does not know how it will be used in queries. Needs query optimizer.

- Query evaluation should be guaranteed to terminate, program execution cannot.

  But modern DDB systems allow the complete Pure Prolog.

Datalog vs. Prolog (3)

All Advantages of Databases:

- Persistence (Possibility to store data that live longer than a single program execution)
- Multi-User, Security, Access Control, Integrity.
- Transactions (Atomicity, Backup&Recovery, Concurrency Control).
- Support for "big data" (larger than main memory).
- But current DDB prototypes have not necessarily all database functions.

Contents

History of the Field (1)

∼322 BC	Syllogisms [Aristoteles]
∼300 BC	Axioms of Geometry [Euklid]
∼1700	Plan of Mathematical Logic [Leibniz]
1847	"Algebra of Logic" [Boole]
1879	"Begriffsschrift" (Early Logical Formulas) [Frege] (Member of Leopoldina in Halle)
∼1900	More natural formula syntax [Peano]
1910/13	Principia Mathematica (Collection of formal proofs) [Whitehead/Russel]
1930	Completeness Theorem [Gödel/Herbrand]
1936	Undecidability [Church/Turing]

History of the Field (2)

1960	First Theorem Prover [Gilmore/Davis/Putnam]
1963	Resolution-Method for Theorem proving [Robinson]
∼1969	Question Answering Systems [Green et.al.]
1970	Linear Resolution [Loveland/Luckham]
1970	Relational Data Model [Codd]
∼1973	Prolog [Colmerauer, Roussel, et.al.]

(Started as Theorem Prover for Natural Language Understanding)
(Compare with: Fortran 1954, Lisp 1962, Pascal 1970, Ada 1979)

∼1973	Algorithm = Logic + Control [Kowalski]
1976	Minimal Model Semantics [van Emden, Kowalski]

History of the Field (3)

1977	Conference "Logic and Databases" [Gallaire, Minker]
1977	First Compiler for Prolog [Warren]
1982	Start of the "Fifth Generation Project" in Japan (ended 1994) (caused research grants worldwide)
1986	"Magic Sets"
1986	Perfect Model Semantics [Przymusinski]
1986	First Deductive DB Systems
1987	CLP(R): Arithmetic Constraints [Jaffar]
1988	CHIP: Finite Domain Constraints [Van Hentenryck]
1988	Well-Founded and Stable Model Semantics [Van Gelder/Ross/Schlipf, Gelfond/Lifschitz]
$\sim$1989	First Textbooks on Deductive DBs

History of the Field (4)

∼1992	Second DDB Prototype System Generation
1996	ISO Standard for Prolog
1996	smodels: Answer Set Programming System (ASP) [Nimelä/Simons]
2002	LogicBlox founded (commercial deductive Database)
2007	clasp ($\rightarrow$ clingo, Potassco): Efficient ASP Systems [Torsten Schaub u.a., Potsdam]
2009	Datalog for Query-Answering over Ontologies [Calì, Gottlob, Lukasiewicz]
2010	Workshop "Datalog reloaded"
2010	Dedalus: Datalog in Time and Space (for Distributed Sys.) [Alvaro, Marczak, Conway, Hellerstein, Maier, Sears]
2016	Soufflé Project (Datalog for static analysis of Java)

Contents

Applications in Industry

- Prolog is being successfully used in industry.

  Boeing, British Airways, Kodak, Swiss Life, IBM (Machine Translation), Philips Research, SRI (natural language).
  Microsoft Windows NT used a small Prolog interpreter in order to generate optimal configurations for networks.
  Knowledgeware's Application Development Workbench (a CASE tool, 1995: 60.000 Licenses sold) contains around 250.000 lines of Prolog code. The company estimates that it would have been around 10 times larger if written in C. [Frühwirth/Abdennadher 1997]

- At Ericsson a special deductive DBMS for their own use has been developed.

- Constraint Logic Programming is very successful in industry.

  It is estimated that 1996 constraint technology for 100 million $ was sold. (1996, data mining tools for 120 million $ were sold. Microsoft's turnover was 10.000 million $.) [Frühwirth/Abdennadher 1997]

Applications of Recursion

- Important for processing hierarchical data, and data that has the form of a graph.
- E.g. the WWW is a directed graph of documents.
- Documents are hierarchically structured,
  e.g. XML defines tree-structure (plus `IDREF`).

  "Object Exchange Model", Models for semistructured data: graphs.
- CAD: Parts are often hierarchically composed out of smaller parts.
- CASE: Call structure of procedures is a graph.
- Getting an unknown number of pieces from a string.

  Exercise in database course: Cut off initials from president name.

  Structured strings are a bad database design, but that was given.