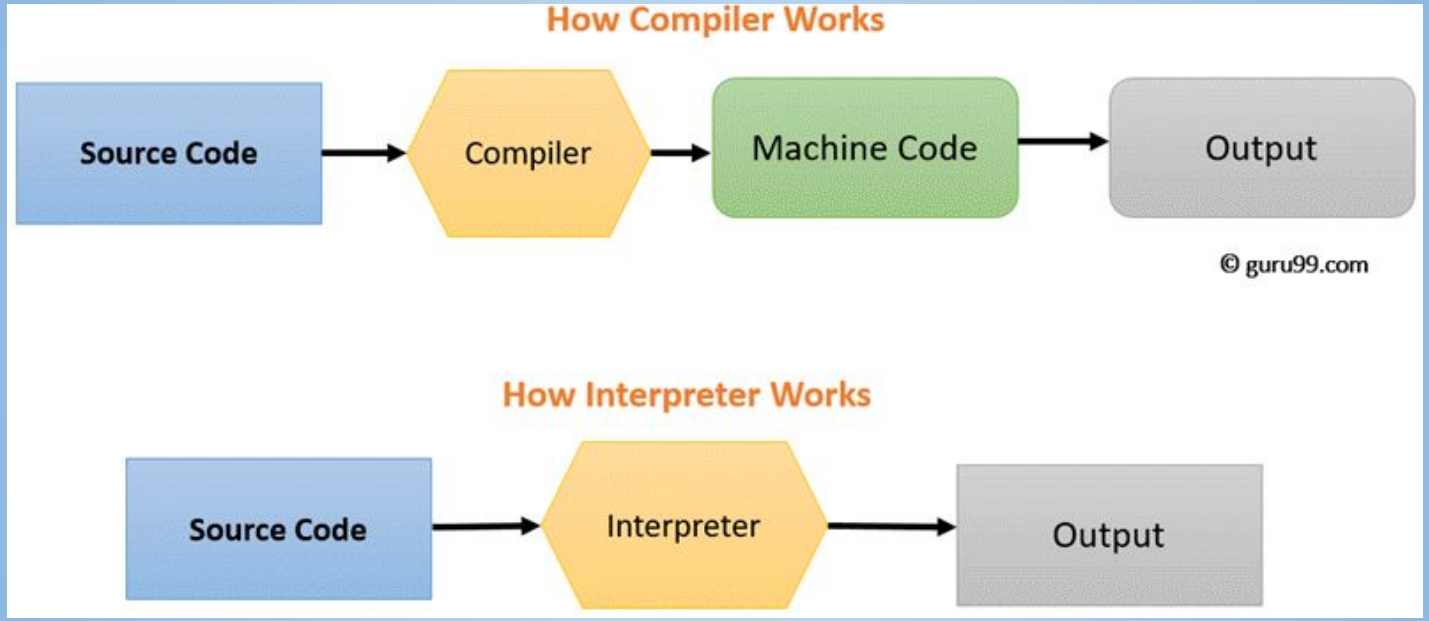
# Logical programming languages implementation

Speakers: Joel Aumedes and Marc Cervera

# Index of contents

# Introduction

- The implementation of a programming language is which provides a way to execute a program in a concrete combination of software and hardware.
- There exist two ways of implement a programming language:
  - Compilation: Process that translates a program written in a programming language to another equivalent programming language.
  - Interpretation: Is a meanings assignation to the well-formed formulas of a formal language.
- We're going to focus on PROLOG to perform this lecture.
- PROLOG is a logic programming language associated wit AI and computational linguistics.
- PROLOG has its roots in CP1, and unlike many other programming languages, PROLOG is intended as a declarative programming language.
- Logical programming is a programming paradigm which is based on formal logic.


How Compiler Works
Source Code
Compiler
Machine Code
Output
© guru99.com
How Interpreter Works
Source Code
Interpreter
Output

# PROLOG, compiled or interpreted?

- The first PROLOG implementation was completed in 1972 using the compiler of ALGOL W programming language, based on a proposal for ALGOL X by Nikla Wirth and Tony Hoare and the actual basic aspects of the language were concluded un 1973.
- Hence, PROLOG is a compiled language.

# PROLOG's interpreter program


SWI Prolog

# PROLOG's implementation: ISO PROLOG


- ISO = International organization of standarization.
- The ISO PROLOG has two parts:
  - ISO/IEC 13211-1:
    - Published in 1995.
    - As a target: standardize the existing practices of the many implementations of the core elements of PROLOG.
  - ISO/IEC 13211-2:
    - Published in 2000.
    - Adds support for the modules to the standard

# PROLOG’s implementation: Compilation

- For efficiency, PROLOG code is typically compiled to abstract machine code, often influenced by the register-based Warren abstract machine instruction set.
- Some implementations employ abstract interpretation to derive type and mode information of predicates at compile time, or compile to real machine code for high performance.
- Devising efficient implementation methods for PROLOG code is a field of active research in the logic programming community, and various other execution methods are employed in some implementations.

# PROLOG’s implementation: Tail recursion

- PROLOG systems typically implement a well-known optimization method called tail call optimization for deterministic predicates exhibiting tail recursion or, more generally, tail calls: A clause’s stack frame is discarded before performing a call in a tail position.
    - Tail calls can be implemented without adding a new stack frame to the call stack.
    - Most of the frame of the current procedure is no longer needed, and can be replaced by the frame of the tail call, modified as appropriate.
    - The program can then jump to the called subroutine.
    - Producing such code instead of a standard sequence is called tail call optimization.
- Therefore, deterministic tail-recursive predicates are executed with constant stack space, like loops in other languages.

# PROLOG’s implementation: Term indexing

- Finding clauses that are unifiable with a term in a query is linear in the number of clauses.
- Term indexing uses a data structure that enables sub-linear-time lookups.
- Indexing only affects program performance, it does not affect semantics.
- Most PROLOGs only use indexing on the first term, as indexing on all terms is expensive, but techniques based on field-encoded words or superimposed codewords provide fast indexing across the full query and head.

# PROLOG’s implementation: Hashing

- Some PROLOG systems, such as WIN-PROLOG and SWI-PROLOG, now implement hashing to help handle large datasets more efficiently.
- This tends to yield very large performance gain when working with large corpora such as WordNet.

# PROLOG's implementation: Tabling

- Some PROLOG systems, implement a memoization method called *tabling*, which frees the user from manually storing intermediate results.
- Tabling is a space-time tradeoff, execution time can be reduced by using more memory to store intermediate results:
  - Subgoals encountered in a query evaluation are maintained in a table, along with answers to these subgoals.
  - If a subgoal is re-encountered, the evaluation reuses information from the table rather than re-performing resolution against program clauses.
- Tabling can be extended in various directions.
  - It can support recursive predicates through SLG-resolution or linear tabling.
  - In a multi-threaded PROLOG system tabling results could be kept private to a thread among all threads.
  - And in incremental tabling, might react to changes.

# PROLOG's implementation: Implementation in hardware

- During the fifth generation computer systems project, there attempt to implement PROLOG in hardware with the aim of achieving faster execution with dedicated architectures.
- Furthermore, PROLOG has a number of properties that may allow speed-up through parallel execution.
- A more recent approach has been to compile restricted PROLOG programs to a field programmable gate array (FPGA).
    - FPGA is an integrated circuit designed to be configured by a customer or a designer after manufacturing.
- However, rapid progress in general-purpose hardware has consistently overtaken more specialised architectures.

# Limitations

- PROLOG and other logic programming languages have not had a significant impact on the computer industry in general.
- Most apps are small by industrial standards.
- “Programming  in the large” is considered to be complicated because not all PROLOG compilers support modules (compatibility problems).
- Portability of PROLOG code across implementations
- Software developed in PROLOG has been criticised for having a high performance penalty.
- PROLOG is not pure declarative.

# Extensions

- Various implementations have been developed from PROLOG to extend logic programming capabilities in numerous directions.
  - The extensions are:
    - Types.
    - Modes.
    - Constraints.
    - Object-orientation.
    - Graphics.
    - Concurrency.
    - Web programming.

# Related languages

- Gödel.
- Visual prolog.
- Datalog.
- Mercury.
- Planner (PROLOG is a subset of Planner).
- GraphTalk (is a proprietary implementation of WAM's, with additional object-oriented properties.)
- AgentSpeak.
- Erlang.
- Pilog.