POPL : Logic Programming

Srijith P K

Declarative Programming language

- **declarative programming** is a programming paradigm—a style of building the structure and elements of computer programs—that expresses the logic of a computation without describing its control flow. programming is done with expressions or declarations instead of statements.
- Defines *what* the program must accomplish in terms of the problem domain, rather than describe *how* to accomplish it as a sequence of the programming language primitives(the *how* being left up to the language's implementation).
- “imperative” programming means that the **computer get a list of commands and executes them in order**, when “procedural programming” (which is also imperative) **allows splitting those instructions into procedures (or functions)**.

Declarative vs imperative languages

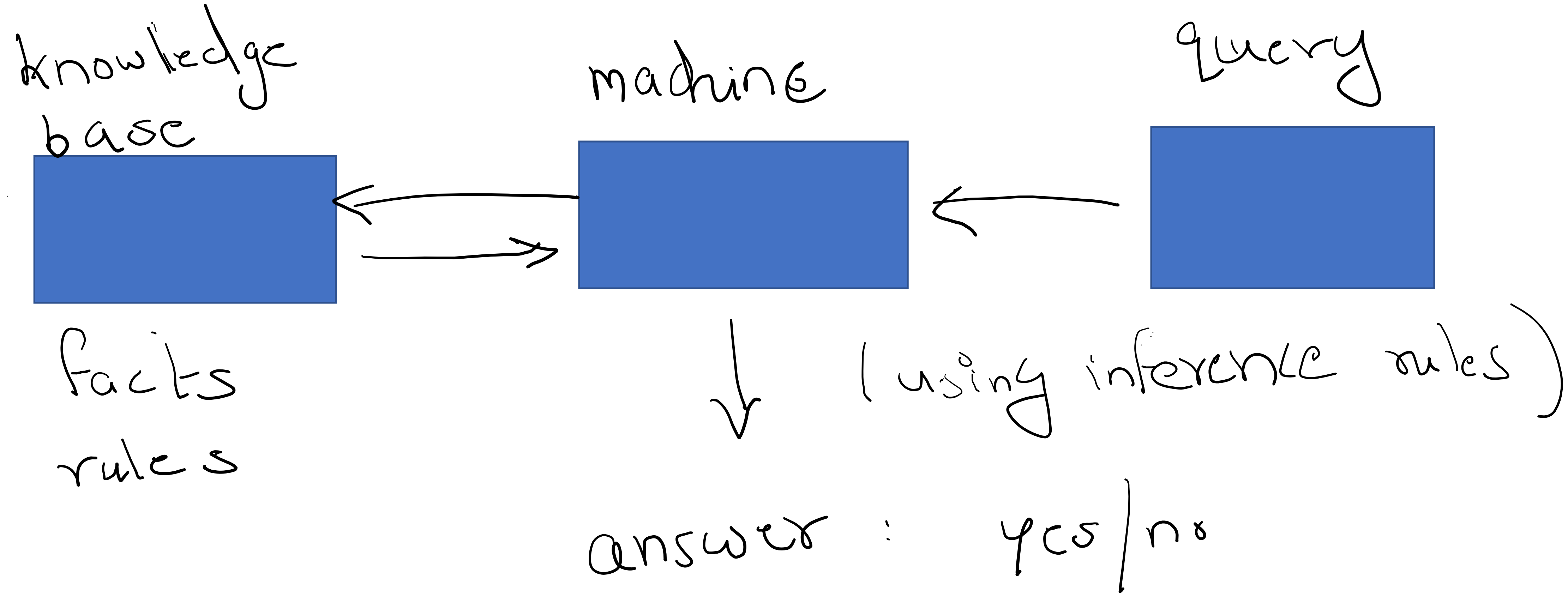
	Imperative	Declarative
Paradigm	Describe HOW TO solve the problem	Describe WHAT the problem is
Program	A sequence of commands	A set of statements
Examples	C, Fortran, Ada, Java	Prolog, Pure Lisp, Haskell, ML
Advantages	Fast, specialized programs	General, readable, correct(?) programs.

Declarative description A grandchild to x is a child of one of x's children.

Imperative description I To find a grandchild of x, first find a child of x. Then find a child of that child.

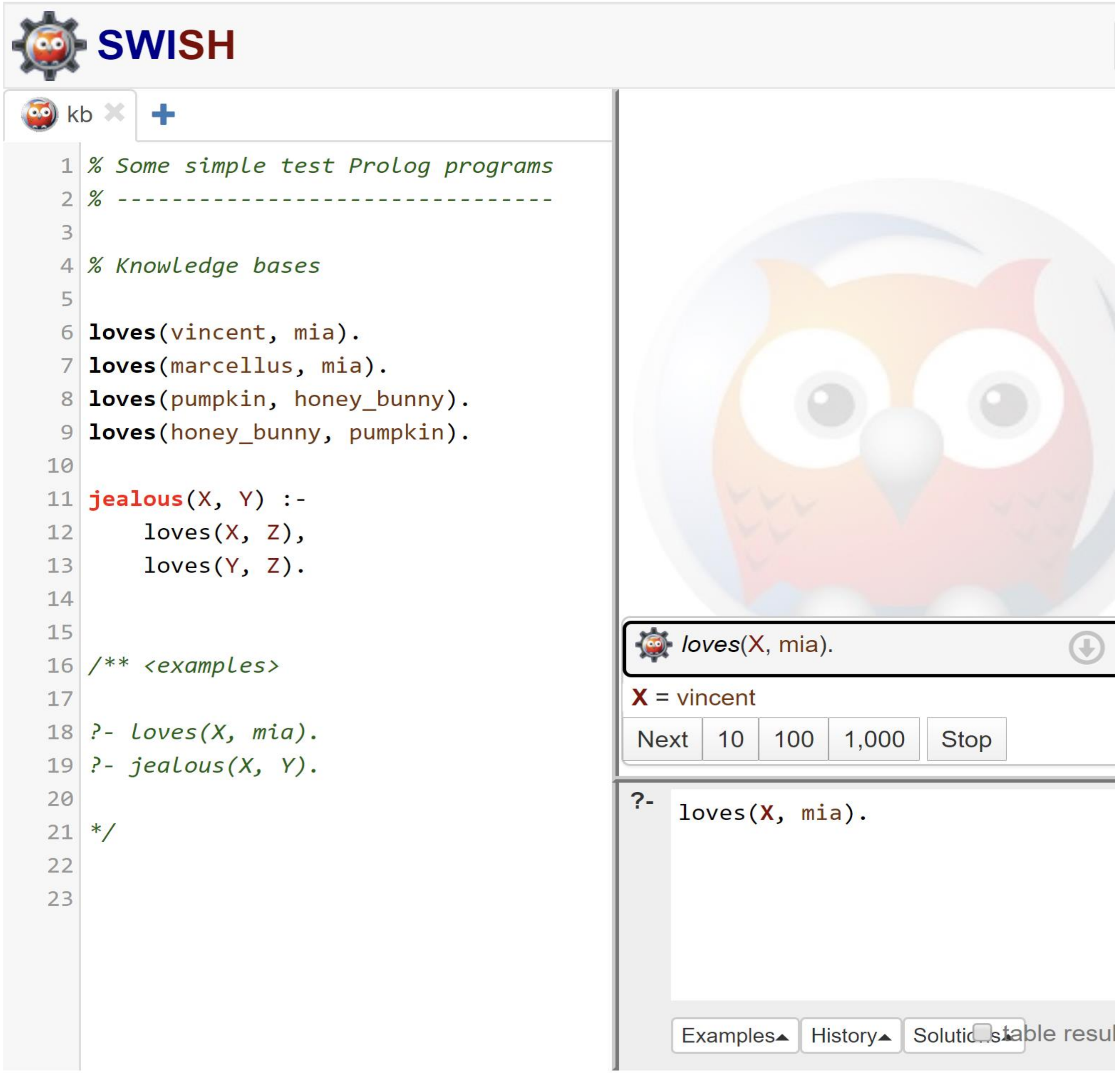
Imperative description II To find a grandchild of x, first find a parent-child pair and then check if the parent is a child of x.

Logic programming



Prolog

- Prolog has its roots in first-order logic
- Prolog is intended primarily as a declarative programming language: the program logic is expressed in terms of relations, represented as facts and rules.
- A computation is initiated by running a query over these relations
- Application in Artificial Intelligence, computational linguistics, automated planning etc.



```
% Some simple test Prolog programs
% -------------------------------

% Knowledge bases

loves(vincent, mia).
loves(marcellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).

jealous(X, Y) :-
    loves(X, Z),
    loves(Y, Z).


/** <examples>

?- loves(X, mia).
?- jealous(X, Y).

*/
```

```
loves(X, mia).
X = vincent
```

```
?- loves(X, mia).
```

Logic programming references

- Prolog, programming for Artificial Intelligence by Ivan Bratko 4th Edition
- Learn Prolog now by Patrick Blackburn, Johan Bos, and Kristina Striegnitz
- http://www.swi-prolog.org/

Applications of Logic Programming

- Theorem proving
- Natural language processing
- Pattern matching
- Combinatorial test case generation
- Extract sub structures from structured data such as an XML document
- Symbolic computation i.e. calculus
- Deductive databases
- Expert systems
- Artificial Intelligence
- Parsing
- Query languages

History of logic programming

- automated theorem proving from which it took the notion of a deduction.
- “computation as deduction” : 1930s Kurt Godel and Jacques Herbrand
- Automated deduction : 1965 Alan Robinson A machine-oriented logic based on the resolution principle. Journal of the ACM.
- resolution principle, the notion of unification and a unification algorithm, resolution method one can prove theorems of first-order logic
- 1974 by Robert Kowalski in his paper Predicate logic as a programming language introduced logic programs with a restricted form of resolution
- Alain Colmerauer worked on programming language for natural language processing led to **prolog** in 1973 (implemented by Philippe Roussel)

History of Logic programming

- Creation of **deductive databases** extend the relational databases by
- providing deduction capabilities
- Japanese Fifth Generation Project for intelligent computing systems
- inductive logic programming, a logic based approach to machine learning.

Computation vs Deduction

- To compute we start from a given expression and, according to a fixed set of rules (the program) generate a result. For example, 15 + 26 → (1 + 2 + 1)1 → (3 + 1)1 → 41.
- To deduce we start from a conjecture and, according to a fixed set of rules (the axioms and inference rules), try to construct a proof of the conjecture.
- **Deductive reasoning**, also **deductive logic**, is the process of reasoning from one or more statements (premises) to reach a logically certain conclusion. Deductive reasoning goes in the same direction as that of the conditionals, and links premises with conclusions.
- Boolean algebra : logical operations in the same way that elementary algebra describes numerical operations
- Logic program computation proceeds by proof search according to a fixed strategy

Logic

- Proof is an evidence, which is obtained as a deduction using inference rules of the form

$$\frac{J_1 \ldots J_n}{J}\ R$$

- R is the name of the rule, J is the **judgment** established by the inference (the conclusion), and J1, . . . , Jn are the premisses of the rule. If J1 and · · · and Jn then we can conclude J by virtue of rule R
- **Judgement** is an object of knowledge
  - **truth of a proposition, A true, A false (A is false),**
  - **A true at t (A is true at time t)**

Logic

- Represent natural numbers 0, 1, 2, . . . as terms of the form z, s(z), s(s(z)), . . ., using two function symbols (z of arity 0 and s of arity 1)
- Consider predicate ‘even’ of arity 1 representing even numbers, its represented by two inference rules.

$$\frac{}{\mathsf{even}(\mathsf{z})}\ \mathsf{evz} \qquad\qquad \frac{\mathsf{even}(N)}{\mathsf{even}(\mathsf{s}(\mathsf{s}(N)))}\ \mathsf{evs}$$

The first rule, evz, expresses that 0 is even. It has no premiss and therefore is like an axiom. The second rule, evs, expresses that if N is even, then s(s(N)) is also even.

Deduction

- Show that 4 is even

$$\dfrac{\dfrac{\dfrac{}{\text{even}(z)}\ \text{evz}}{\text{even}(s(s(z)))}\ \text{evs}}{\text{even}(s(s(s(s(z)))))}\ \text{evs}$$

Inference rules to logic programming

- **Goal directed or top down :** search backward from the conjecture, growing a proof tree upward
- **Forward reasoning or bottom-up :** work forwards from the axioms
- applying rules until we arrive at the conjecture.
- Logic programming was conceived with goal-directed search (prolog)
- goal-directed strategy : given a conjecture (called the *goal*) we determine which inference rules might have been applied to arrive at this conclusion. Select one of them and then recursively apply our strategy to all the premisses as subgoals. If there are no premisses we have completed the proof of the goal.

Goal directed strategy

- conjecture even(s(s(z))).

$$\dfrac{\begin{array}{c}\vdots\\ \mathsf{even}(\mathsf{z})\end{array}}{\mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{z})))}\;\mathsf{evs}$$

only rule with a matching conclusion is evs.

- Considering the subgoal even(z)

$$\dfrac{\dfrac{}{\mathsf{even}(\mathsf{z})}\;\mathsf{evz}}{\mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{z})))}\;\mathsf{evs.}$$

only the rule evz could have this conclusion.
this rule has no premisses so the computation terminates successfully

logic programming will answer yes if a proof has been found, which means the conjecture was true.

Goal directed strategy

- goal even(s(s(s(z)))).

$$\dfrac{\begin{array}{c}\vdots\\ \mathsf{even}(\mathsf{s}(\mathsf{z}))\end{array}}{\mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z}))))}\ \mathsf{evs}.$$

- no rule whose conclusion matches the goal even(s(z)). search *fails*, even(s(z)) is false , *negation as failure* – common technique in logic programming
- If conjecture was true, search constructed an explicit proof which provides evidence for its truth