

Programming Languages (CSCI 4430/6969)

History, Syntax, Semantics, Essentials, Paradigms

Carlos Varela
Rensselaer Polytechnic Institute

September 1, 2009

C. Varela

1

The first programmer ever

Ada Augusta, the Countess of Lovelace, the daughter of the poet Lord Byron

Circa 1843

Using Babbage's Analytical Engine

C. Varela

2

The first "high-level" (compiled) programming language

FORTRAN

1954

Backus at IBM

It was called "an automatic coding system", not a "programming language"

Used for numerical computing

C. Varela

3

The first functional programming language

Lisp

1958

McCarthy at Stanford

For LISTS Processing---lists represent both code and data

Used for symbolic manipulation

C. Varela

4

The first object oriented programming language

Simula

1962

Dahl and Nygaard at University of Oslo, Norway

Used for computer simulations

C. Varela

5

The first logic programming language

Prolog

1972

Roussel and Colmerauer at Marseilles University, France

For "PROgrammation en LOGique".

Used for natural language processing and automated theorem proving

C. Varela

6

The first concurrent programming language

Concurrent Pascal
1974
Hansen at Caltech
Used for operating systems development

C. Varela 7

The first scripting language

REXX
1982
Cowlishaw at IBM
Only one data type: character strings
Used for "macro" programming and prototyping

C. Varela 8

The first multi-paradigm programming language

Oz
1995
Smolka at Saarland University, Germany
A logic, functional, imperative, object-oriented, concurrent, and distributed programming language
Used for teaching programming and prototyping

C. Varela 9

Other programming languages

<p>Imperative</p> <ul style="list-style-type: none"> Algol (Naur 1958) Cobol (Hopper 1959) BASIC (Kennedy and Kurtz 1964) Pascal (Wirth 1970) C (Kernighan and Ritchie 1971) Ada (Whitaker 1979) 	<p>Functional</p> <ul style="list-style-type: none"> ML (Milner 1973) Scheme (Sussman and Steele 1975) Haskell (Hughes et al 1987)
<p>Object-Oriented</p> <ul style="list-style-type: none"> Smalltalk (Kay 1980) C++ (Stroustrup 1980) Eiffel (Meyer 1985) Java (Gosling 1994) C# (Hejlsberg 2000) 	<p>Actor-Oriented</p> <ul style="list-style-type: none"> PLASMA (Hewitt 1975) Act (Lieberman 1981) ABCL (Yonezawa 1988) Actalk (Briot 1989) Erlang (Armstrong 1990) E (Miller et al 1998) SALSA (Varela and Agha 1999)
<p>Scripting</p> <ul style="list-style-type: none"> Python (van Rossum 1985) Perl (Wall 1987) Tcl (Ousterhout 1988) Lua (Teuscalmschky et al 1994) JavaScript (Eich 1995) PHP (Lerdorf 1995) Ruby (Matsumoto 1995) 	

C. Varela 10

Logic Programming (PLP 11)

Horn Clauses
Introduction to Prolog: Resolution, Unification

C. Varela 11

An Early (1971) "Conversation"

<p>USER: Cats kill mice. Tom is a cat who does not like mice who eat cheese. Jerry is a mouse who eats cheese. Max is not a mouse. What does Tom do? COMPUTER: Tom does not like mice who eat cheese. Tom kills mice. USER: Who is a cat? COMPUTER: Tom.</p>	<p>USER: What does Jerry eat? COMPUTER: Cheese. USER: Who does not like mice who eat cheese? COMPUTER: Tom. USER: What does Tom eat? COMPUTER: What cats who do not like mice who eat cheese eat.</p>
--	---

C. Varela 12

Another Conversation

USER:

Every psychiatrist is a person.
 Every person he analyzes is sick.
 Jacques is a psychiatrist in Marseille.
 Is Jacques a person?
 Where is Jacques?
 Is Jacques sick?

COMPUTER:

Yes.
 In Marseille.
 I don't know.

C. Varela

13

Logic programming

- A program is a collection of *axioms*, from which theorems can be proven.
- A *goal* states the theorem to be proved.
- A logic programming language implementation attempts to satisfy the goal given the axioms and built-in inference mechanism.

C. Varela

14

Horn clauses

- A standard form for writing axioms, e.g.:

$$\text{father}(X, Y) \Leftarrow \text{parent}(X, Y), \text{male}(X).$$

- The Horn clause consists of:
 - A *head* or consequent term H_1 , and
 - A *body* consisting of terms B_i

$$H \Leftarrow B_0, B_1, \dots, B_n$$

- The semantics is:

« If B_0, B_1, \dots, B_n , then H »

C. Varela

15

Terms

- Constants

rpi
troy

- Variables

University
City

- Predicates

located_at(rpi, troy)
pair(a, pair(b, c))

Can be nested.

C. Varela

16

Resolution

- To derive new statements, Robinson's resolution principle says that if two Horn clauses:

$$H_1 \Leftarrow B_{11}, B_{12}, \dots, B_{1m}$$

$$H_2 \Leftarrow B_{21}, B_{22}, \dots, B_{2n}$$

are such that H_1 matches B_{2p} , then we can replace B_{2p} with $B_{11}, B_{12}, \dots, B_{1m}$:

$$H_2 \Leftarrow B_{21}, B_{22}, \dots, B_{2(p-1)}, \underline{B_{11}, B_{12}, \dots, B_{1m}}, B_{2(p+1)}, \dots, B_{2n}$$

- For example:

$$\frac{C \Leftarrow A, B}{D \Leftarrow C}$$

$$D \Leftarrow A, B$$

C. Varela

17

Resolution Example

$$\text{father}(X, Y) :- \text{parent}(X, Y), \text{male}(X).$$

$$\text{ancestor}(X, Y) :- \text{father}(X, Y).$$

$$\text{ancestor}(X, Y) :- \text{parent}(X, Y), \text{male}(X).$$

$:-$ is Prolog's notation (syntax) for \Leftarrow .

C. Varela

18

Unification

- During *resolution*, free variables acquire values through *unification* with expressions in matching terms.
- For example:

```
male(carlos).
parent(carlos, tatiana).
father(X,Y) :- parent(X,Y), male(X).
-----
father(carlos, tatiana).
```

C. Varela

19

Unification Process

- A **constant** unifies only with itself.
- Two **predicates** unify if and only if they have
 - the same *functor*,
 - the same number of *arguments*, and
 - the corresponding arguments *unify*.
- A **variable** unifies with anything.
 - If the other thing has a *value*, then the variable is *instantiated*.
 - If it is an *uninstantiated variable*, then the two variables are *associated*.

C. Varela

20

Backtracking

- *Forward chaining* goes from axioms forward into goals.
- *Backward chaining* starts from goals and works backwards to prove them with existing axioms.

C. Varela

21

Backtracking example

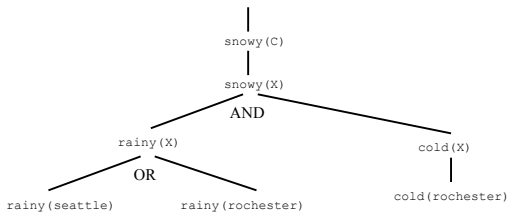
```
rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), cold(X).
```

C. Varela

22

Backtracking example

```
rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), cold(X).
```

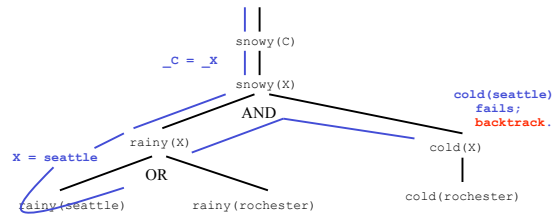


C. Varela

23

Backtracking example

```
rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), cold(X).
```

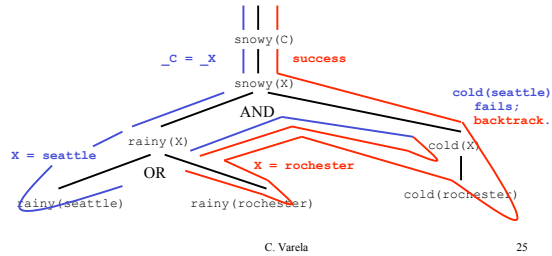


C. Varela

24

Backtracking example

```
rainy(seattle).  
rainy(rochester).  
cold(rochester).  
snowy(X) :- rainy(X), cold(X).
```



Exercises

1. Download SWI Prolog and install it in your laptop.
2. Execute the `snowy(City)` example. Use "tracing" to follow backtracking step by step.
3. Create a knowledge base with Prolog facts about your family members using predicates and constants. Create Prolog rules using variables to define the following: brother, sister, uncle, aunt, nephew, niece, grandfather, grandmother, etc. Query your program for family relationships.

C. Varela

26