A second life for Prolog

Declarative programming

Jan Wielemaker
J.Wielemaker@cwi.nl

8th Language & Technology Conference
November 17-19, 2017, Poznań, Poland

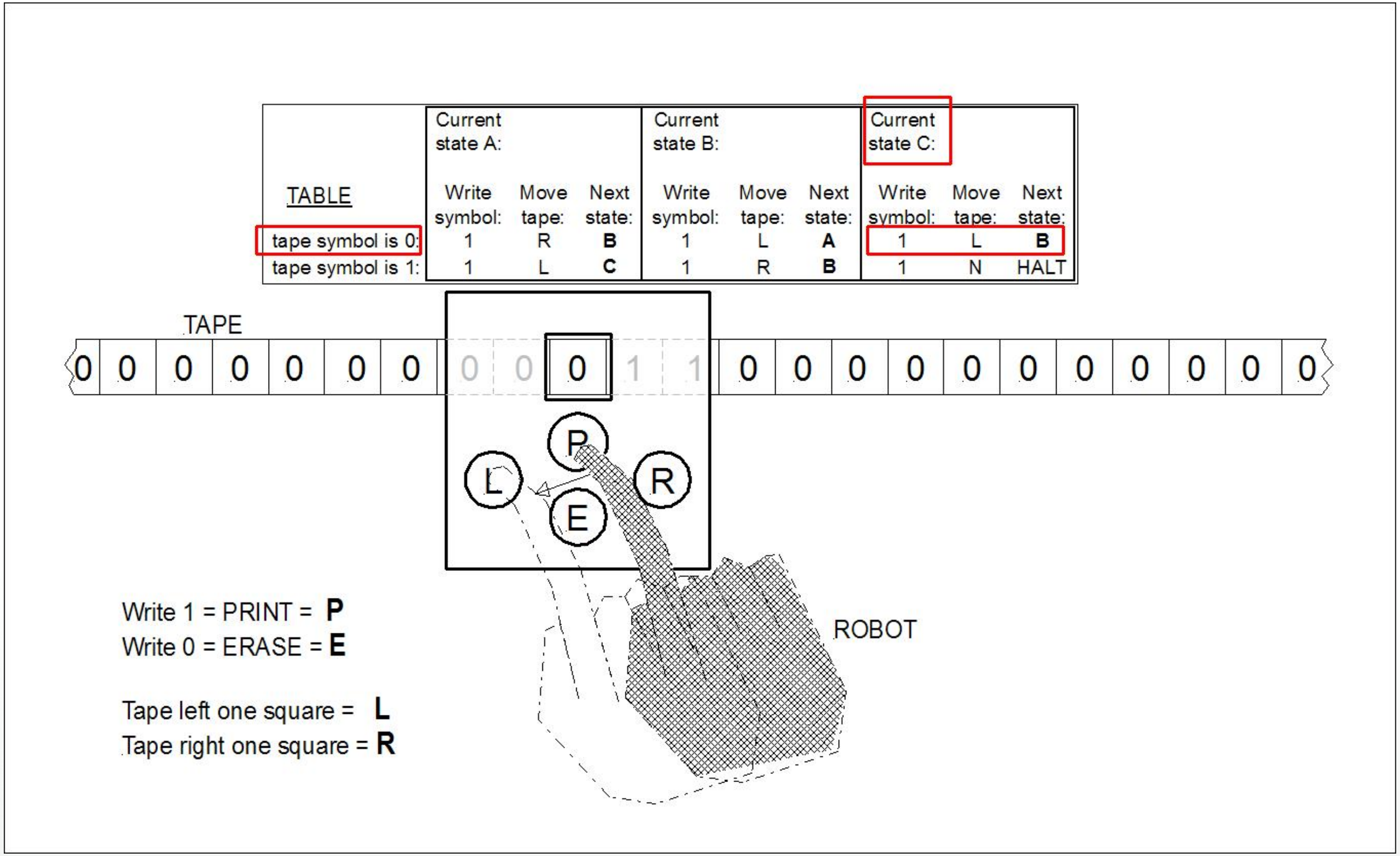
This research was partially supported by the VRE4EIC project, a project that has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 676247.

Sessions

1) Introducing declarative programming and Prolog
   - Programming paradigms
   - Basics of Prolog
2) Algorithm = Logic + Control
   - Advanced control: tabling, constraints, continuations
   - Data and aggregation
3) Prolog as unifying framework
   - Accessing the outside world

Programming paradigms

The Turing Machine

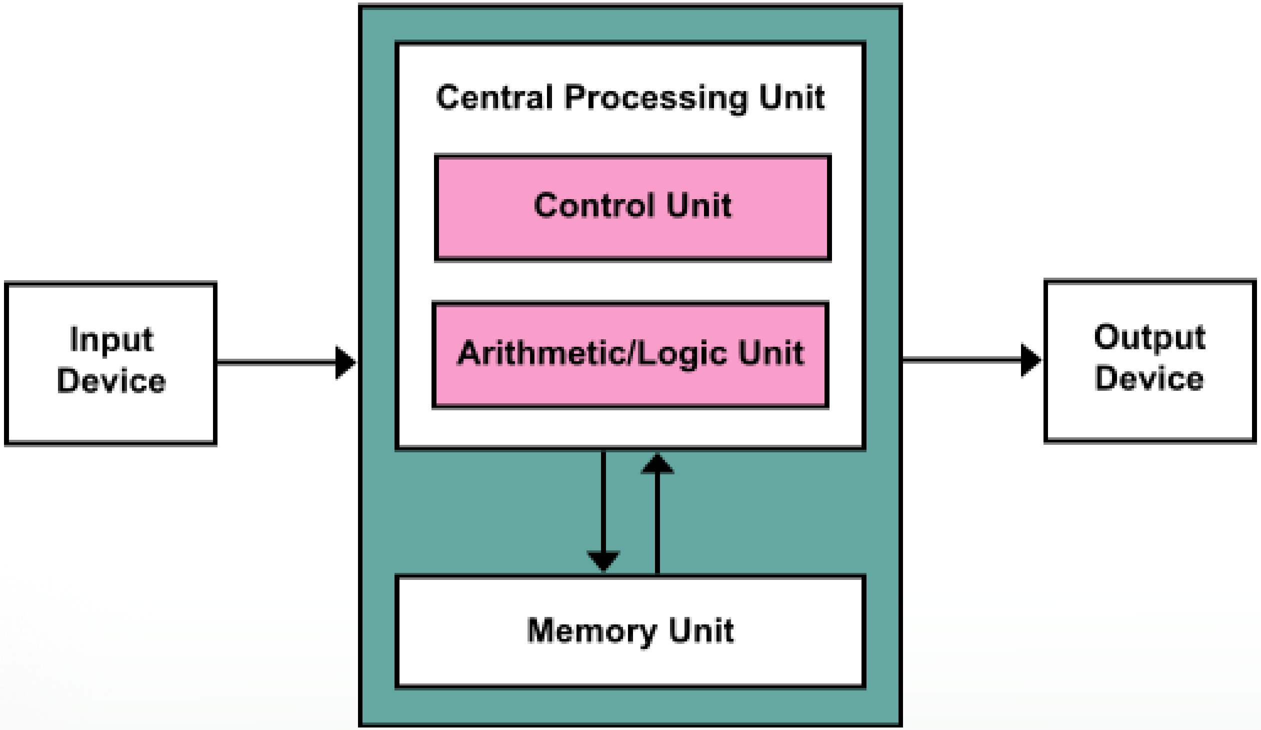


The universal computing device!

- "The evidence seems to indicate that every algorithm for any computing device has an equivalent Turing machine algorithm ... if [Church's thesis] is true, it is certainly remarkable that Turing machines, with their extremely primitive operations, are capable of performing any computation that any other device can perform, regardless of how complex a device we choose."

(Stone (1972), p. 13)

Von Neumann architecture



Imperative Programming

- Imperative: *giving an authoritative command*
- *X = Y + Z*
  - *Take the values stored in the memory locations of Y and Z, add them and store the result in the memory location of X.*
- *X = X + 1*
  - *To a mathematician this simply false*
  - *A programmer doesn't even see the problem!*
- *With state everywhere, it gets hard to*
  - *Understand the computation*
  - *Reorder it computation (exploit concurrency)*

Just a test ...

```
do
{ double r = floor(e0/e1);
  double e00 = e0, p00 = p0, q00 = q0;
  volatile double p1_q1;

  e0 = e1;
  p0 = p1;
  q0 = q1;
  e1 = e00 - r*e1;
  p1 = p00 - r*p1;
  q1 = q00 - r*q1;

  p1_q1 = p1/q1;
  d = p1_q1 - n1->value.f;
} while(fabs(d) > DBL_EPSILON);
```

Functional programming

- A function specifies its (return) value as an expression over its inputs
  - There are no more variables storing state
  - Easier to reason about and execute concurrently
- Functions as primary objects (Lambda expressions)
- Lisp (1958), Scheme (1970), Clojure (2007)
- Most are hybrid language: they do provide for traditional variables but discourage their use
- JavaScript, Python. Even Java and to some extend C.

Logic programming

- Use (predicate) logic to describe the relation between values:

```
parent(bob, jane).
parent(jan, jane).

mother(X,Y) :- parent(X,Y), female(Y).
father(X,Y)  :- parent(X,Y), male(Y).

brother(X,Y) :- parent(X,P), parent(Y,P), male(Y).
```

Twists wrt. logic

- P → Q (P implies Q) is written as Q :- P (Q is true if P is true).
- Q (the head) is only a single *atom*, i.e., we cannot write
  - A, B :- P.

Some logic based languages

- Prolog (Alain Colmerauer, 1972)
  - Goal-directed backward chaining (top-down)
- Datalog (~1977)
  - Forward chaining (bottom-up)
- Mercury (1995)
  - Avoid need for extra-logical constructs using more declarations
- Picat (2015)
  - Aim at combinatorial problems
- Answer set programming (1993)
  - Pure

Prolog data

- Constants
  - bob, 'Bob', 'Nice wheather'
  - =, ==, <, ... (sequences of *glueing* punctuation characters)
  - !, |, … (non-glueing punctuation characters)
- Numbers
  - 0, 42, 67.3
  - 7‘012345, 0‘a (97), 0b010101, 0xa0, 0o77
- Strings (SWI, LPA, ECLiPSe)
  - “abc“ (SWI, ECLiPSe), `abc` (LPA)
- Compounds
  - age('Bob', 42)
- Variables
  - X, Myvar, _var

Syntactic sugar

- Operators (user extensible):
  - X = Y ↔ =(X,Y)
  - Type: **fx, fy, xfx, xfy, yfx, xf, yf**
  - Priority (0..1200)
- Lists (In SWI-Prolog **.** is **[|]**):
  - [a,b,c] ↔ .(a, .(b, .(c, [])))
  - [a,b|C] ↔ .(a, .(b, C))
  - “ab“ or \`ab\` (SWI) ↔ .(97,.(98, []))
- Others
  - { a,b } ↔ {}((a,b)).

Logical variables: unification

- T1 = T2, is true if there is a substitution of variables with values that makes the two terms identical (T1 == T2)

X = a	a = a
a = X	a = a
X = Y	XY = XY
X = Y, Y = a	a = a
f(1) = f(X)	f(1) = f(1)
1 = 2	false
f(X,1) = f(Y,2)	false

8th Language & Technology Conference
November 17-19, 2017, Poznań, Poland

Logical variables

- After unifying with a non-variable, the variable is *gone*!
  - ?- T = f(X), X = 1.
  - T = f(1).
- Unify-once:
  - X = 1, X = 2 → false!

Observations

- Objects (terms) that do not contain variables are ground.
- There is no *new*. Writing an object (term) creates it. As terms are *immutable one instance suffices*.
- The program is represented as a term
  - We can reason about programs
  - We can do dynamic programming

databases and a queries

- The data
  - mother(bob, jane).
    mother(jan, jane).
- Queries
  - ?- mother(bob, jane).
  - **true**.
  - ?- mother(bob, jennifer).
  - false.

databases and a queries (cont)

- The data
  - mother(bob, jane).
  - mother(jan, jane).
- Queries
  - ?- mother(X, jane).
  - X = bob ;
  - X = jan ;
  - false.

databases and a queries (cont)

- The data
  - parent(jane, bob). female(jane). male(bob).
  - parent(jane, jan). male(jan).
  - parent(jane, mary). female(mary).
- Queries
  - ?- parent(jane, X), female(X).
  - X = mary.

databases, rules and a queries

- The data
  - parent(jane, bob). female(jane). male(bob).
    parent(jane, jan). male(jan).
  - parent(jane, mary). female(mary).
- Rule
  - daughter(Mother, Daughter) :-
    parent(Mother, Daughter),
    female(Daughter).
- Queries
  - ?- daughter(X, jane).
  - X = mary.

Exercises

- Please play around with the exercise below:
  - https://swish.swi-prolog.org/p/ltc_family.swinb
  -
- All material is linked from
  - https://swish.swi-prolog.org/p/LTC2017.swinb

Einstein‘s Riddle

- Also known as „*The zebra puzzle*“
  - https://swish.swi-prolog.org/example/houses_puzzle.pl
    1. Five colored houses in a row, each with an owner, a pet, cigarettes, and a drink.
    2. The English lives in the red house.
    3. The Spanish has a dog.
    4. They drink coffee in the green house.
    5. The Ukrainian drinks tea.
    6. The green house is next to the white house.
    7. The Winston smoker has a serpent.
    8. In the yellow house they smoke Kool.
    9. In the middle house they drink milk.
    10. The Norwegian lives in the first house from the left.
    11. The Chesterfield smoker lives near the man with the fox.
    12. In the house near the house with the horse they smoke Kool.
    13. The Lucky Strike smoker drinks juice.
    14. The Japanese smokes Kent.
    15. The Norwegian lives near the blue house.

Five colored houses in a **row**, each with an **owner**, a **pet**, **cigarettes**, and a **drink**.

```
Houses = [ house(Owner,Pet,Cigarettes,Drink,Color),
           house(_,_,_,_,_),
           house(_,_,_,_,_),
           house(_,_,_,_,_),
           house(_,_,_,_,_)
         ].
```

The English lives in the red house

```
member(house(english,_,_,_,red), Houses)
```

```
member(X, [X|_]).
member(X, [_|T]) :- member(X, T).
```

In the middle house they drink milk.

Houses = [_,_,house(_,_,_,milk,_),_,_].

The Chesterfield smoker lives near the man with the fox

```
next(house(_,fox,_,_,_),
        house(_,_,chesterfield,_,_),
        Houses)
```

```
next(A, B, Ls) :- append(_, [A,B|_], Ls).
next(A, B, Ls) :- append(_, [B,A|_], Ls).
```

Exercises

- Map coloring
  - https://swish.swi-prolog.org/p/ltc_mapcolor.swinb
- Learn Prolog Now! (2.2 Proof Search)
  - http://lpn.swi-prolog.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse6