# ROBUST LOGIC PROGRAMMING

Lorenzo Strigini
IEI del CNR, Pisa, Italy
Luca Simoncini
Universita` di Reggio Calabria, Italy

# ROBUST LOGIC PROGRAMMING

INTEREST OF THIS WORK:

- **IMMEDIATE**: PROGRAMMING TECHNIQUES TO IMPROVE PRODUCTS BASED ON PROLOG

- **GENERAL**: PROLOG AS AN EXAMPLE OF {NON-PROCEDURAL, NON-VON NEUMANN, ARTIFICIALLY INTELLIGENT} {LANGUAGE, PROGRAMMING, COMPUTING}

  HOW DOES IT FAIL? HOW CAN FAULTS BE TOLERATED?

## A.I. AND DEPENDABILITY

"A.I." MAY MEAN (PARNAS):

- **APPLICATIONS** THAT ARE NOT YET COMMON AND WELL UNDERSTOOD; HERE DEPENDABILITY PROBLEMS ARE RELATED TO FUZZINESS IN REQUIREMENT OR DIFFICULTIES IN MEASURING PERFORMANCE

- **METHODS** LIKE RULE-BASED PROGRAMMING.
  WE ARE INTERESTED IN DEPENDABILITY PROBLEMS SPECIFIC TO THESE METHODS: WE USE PROLOG AS AN EXAMPLE, AND THEN SEE WHICH CONCLUSIONS CAN BE GENERALIZED.

IS PROLOG TYPICAL OF A.I. METHODS? AT LEAST, IT IS PRAISED AND SOLD AS A PROGRAMMING LANGUAGE FOR A.I..

**PROLOG**
A NON-IMPERATIVE LANGUAGE.

A PROLOG PROGRAM (OR *DATABASE*) IS JUST A DESCRIPTION OF THE PROBLEM.

AN INTERPRETER (A PROBLEM SOLVING PROGRAM) CAN THEN USE THE PROGRAM TO ANSWER QUERIES.

INTERPRETER MAY MEAN: A CONVENTIONAL INTERPRETER, DEDICATED HARDWARE, RULES FOR COMPILING A PROGRAM INTO PROCEDURAL INSTRUCTIONS FOR SEARCHING THE DATABASE

```
child(jane,jill).
female(jill).
female(jane).
male(james).
male(arthur).

daughter(X,Y) :- child(X,Y), female(X).
son(X,Y) :- child(X,Y), male(X).
mother(X,Y) :- child(Y,X), female(Y).
father(X,Y) :- child(Y,X), male(Y).
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
```

EXAMPLES OF QUERIES:

```
?- male(jill).
   no
?- daughter(X,Y).
    X = jane, Y = jill ;
   no
?-
```

## PROS AND CONS OF PROLOG.

## THEORETICAL ADVANTAGES:

- **SIMPLICITY** - THE PROGRAMMER CAN DISREGARD HOW SOLUTIONS ARE OBTAINED AND CONCENTRATE ON WHAT THEY SHOULD BE;

- **EFFICIENCY** - THE INTERPRETER CAN BE MADE FASTER (E.G. BY PARALLELISM) WITHOUT CHANGING THE PROGRAMS;

- **FLEXIBILITY** - A DATA BASE CAN BE QUERIED TO OBTAIN ANY RESPONSE THAT IS DEDUCTIBLE FROM IT.

## DISADVANTAGES:

- **INEFFICIENCY** A GENERAL-PURPOSE INTERPRETER IS GENERALLY INEFFICIENT; A PROGRAM THAT ANSWERS SOME QUERIES EFFICIENTLY MAY BE VERY INEFFICIENT WITH OTHER QUERIES.

THIS REQUIRES: A) THE PROGRAMMER TO PAY MUCH ATTENTION TO HOW SOLUTIONS ARE OBTAINED; B) THE IMPLEMENTORS TO ADD A LOT OF PROCEDURAL AND META-LOGICAL CONSTRUCTS.

- **PROGRAMS USED IN UNFORESEEN WAYS MAY FAIL IN UNFORESEEN WAYS**

## EXAMPLES OF NON-LOGICAL BUILT-IN PREDICATES:

**CUT**: FREEZES THE CHOICES MADE FOR SOME VARIABLES SO THAT BACKTRACKING CANNOT CHANGE THEM

**ASSERT**: ADDS A CLAUSE TO THE DATABASE

**RETRACT**: REMOVES A CLAUSE FROM THE DATABASE

**VAR, NONVAR**: TRUE IF A TERM IS (IS NOT) AN UNINSTANTIATED VARIABLE WHEN THE PREDICATE IS ENCOUNTERED DURING EXECUTION

ETC.

## RELIABILITY PROBLEMS WITH PROLOG:

**ACCIDENTAL**: ARISING FROM YOUTH:

- MANY PROLOG ENVIRONMENTS ARE TOYS

- IDEAL PURITY IS CONSIDERED MORE THAN PRACTICAL NECESSITIES

**LACK OF**: TYPES, MODULES, NAME SCOPE LIMITATIONS, EXCEPTION HANDLING.
MOST TYPING ERRORS DON'T CAUSE ANY SYNTACTIC ERROR: THEY JUST CREATE A LEGAL WRONG PROGRAM.
(RECENT PROLOG IMPLEMENTATIONS INCLUDE MANY OF THESE FEATURES.
FURTHER IMPROVEMENTS ARE LIKELY WITH NEWER DEVELOPMENT ENVIRONMENTS AND/OR LANGUAGES)

**LACK OF STANDARDS**: DIFFERENT IMPLEMENTATIONS HAVE DIFFERENT SEMANTICS AND INCONSISTENT LIBRARIES

**PROBLEMS INHERENT** IN:

- THE SEQUENTIAL SEARCH STRATEGY OF ORDINARY PROLOG ?

- THE PROGRAMMING STYLE BASED ON FIRST-ORDER LOGIC?

- NON-PROCEDURAL PROGRAMMING?

- ....

?

## A POSSIBLE OBJECTION:

LOGICAL PROGRAMMING DOES NOT NEED FAULT-TOLERANCE, BECAUSE:

A. PROLOG IS LOGIC: YOU CAN PROVE CORRECTNESS IF YOU CARE TO;

B. A.I. PROGRAMS ARE NATURALLY ROBUST: IF THEY CANNOT FIND A SOLUTION AT FIRST, THEY WILL KEEP SEARCHING AND FIND ONE IF AT ALL POSSIBLE.

## ANSWERS:

A. PROLOG PROGRAMS ARE NOT PURE LOGIC.

- THE EXECUTION IS CONTROLLED BY SUCH THINGS AS THE ORDERING OF CLAUSES.

- PROGRAMS THAT ARE CORRECT DESCRIPTIONS OF PROBLEMS AND FAIL TO PRODUCE SOLUTIONS ARE COMMONPLACE.

- SUCCESSFUL PROLOGS HAVE LOTS OF PROCEDURAL OR META-LOGIC PREDICATES, WHOSE MEANING DEPENDS ON DETAILS OF THE WORKINGS OF THE INTERPRETER.

- IT HAS BEEN SHOWN THAT MANY PROLOG PROGRAMS ARE MENAT TO EXECUTE LIKE IMPERATIVE PROGRAMS.

- WHAT ABOUT FAULTS IN THE SUPPORT (HW PLUS OS PLUS INTERPRETER)?

B. WE WANT CORRECT SOLUTIONS, NOT JUST ANY SOLUTION.

REDUNDANCY IN LOGIC PROGRAMS CAN HAPPEN BY CHANCE AND EVEN BE HARMLESS, BUT USEFUL REDUNDANCY NEEDS PLANNING.

## SOFTWARE FAULT-TOLERANCE:

USING EXTRA INFORMATION ABOUT THE PROBLEM BESIDE THE INFORMATION NECESSARY TO FIND A SOLUTION.

A METHOD IS DEFINED BY:

- WHICH INFORMATION IS PROVIDED (NECESSARY CONDITIONS ABOUT THE SOLUTION, ALTERNATIVE WAYS OF COMPUTING THE SOLUTION)

- HOW THE REDUNDANT INFORMATION IS USED TO OBTAIN A MORE RELIABLE SOLUTION, OR TO SIGNAL AN ERROR

REDUNDANT INFORMATION IS EASILY EXPRESSED IN PROLOG.

EXAMPLES OF FAULT TOLERANCE IN PROLOG PROGRAMS

ASSERTIONS (PRE/POST CONDITIONS)

ASSERTIONS WITH BACKWARD RECOVERY

DIVERSITY WITH ADJUDICATION

```
/* this combination first uses <constructive
predicates (X,Y,..)> to find a solution, then
checks it by   assertion(X,Y,..) */

statement(X,Y,..) :- <constructive predicates
          (X,Y,..)>,  assertion(X,Y,..).
assertion(X,Y,...) :- nonvar(X),
          nonvar(Y),..,<appropriate
          predicates>, !.

/* notice the nonvar() and the cut  */


/* by adding this, we can treat a failed
assertion explicitly:*/

assertion(X,Y,..) :- <series of predicates to
          handle exception>.

/* assertions on preconditions */
statement(X,Y,..) :-  assertion(X,Y,..),
          <constructive predicates (X,Y,..)>.
assertion(X,Y,..) :- nonvar(...),..,
          <other predicates>.
```

```
/* if we can code two diverse ways to satisfy a
goal, and want to use backtracking as
backward recovery */
recoverable_goal(X,Y,..)  :-  procedure1(X,Y,..),
          assertion(X,Y,..).
recoverable_goal(X,Y,..)  :-
          procedure2(X,Y,..),  assertion(X,Y,..).
```

```
/* if we can code two diverse ways to satisfy a
goal, and want  static redundancy with
comparison */
duplex_goal(In1,  In2,..,Out1,Out2,..)  :-
          procedure1(In1,In2,.,Out1,Out2,..),
          procedure2(In1,In2,.Out1',Out2',..),
          consistency(Out1,Out2,..,Out1',
          Out2',..).

consistency(Out1,Out2,..,Out1',Out2',..)  :-
          <predicates(Out1,Out2,..,Out1',Out2
          ',..)>.

/* we can deal with mismatches by */
consistency(Out1,Out2,..,Out1',Out2',..)  :-
          <predicates(Out1,Out2,..,Out1',Out2'
          ,..)>,  !.
consistency(Out1,Out2,..,Out1',Out2',..)  :-
          <exception handling predicates)>.


/* serious problems arise with backtracking
and/or multiple solutions  */
```

```
/* in general for N diverse implementations */
Nuplex_goal(In1, In2,..,Out1,Out2,..) :-
        procedure1(In1, In2,..,Out11, Out21,
        ..), procedure2(In1,
        In2,..,Out12,Out22,..), ........,
        procedureN(In1, In2,..,Out1N,
        Out2N, ..),
        adjudicator(Out11,Out21,..,Out1N,
        Out2N,..).

/* This required all N procedures to succeed.
Instead, we may want threshold voting, which
is cumbersome without explicit non-
determinism */
```

## ASPECTS THAT NEED ATTENTION:

DISTRIBUTION

PARALLELISM

WAYS OF DIRECTING THE SEARCH STRATEGY

WAYS OF DIRECTING THE MAPPING OF EXECUTION ON HARDWARE MODULES (TO TOLERATE HARDWARE FAULTS)

## FAILURE MODES OF PROLOG PROGRAMS.

(FOR ANY CAUSE: PROGRAMMING ERROR OR FAILURE IN THE SUPPORT)

THE NORMAL OUTPUT OF A PROLOG PROCEDURE (PROGRAM) MAY BE:

- SUCCESS WITH INSTANTIATION OF 0 OR MORE VARIABLES
- FAILURE (NO ASSIGNMENT OF VALUES TO VARIABLES SATISFIES THE CALLING GOAL)

**FAILURE MODES:**

- FAILURE TO PRODUCE A SOLUTION THAT EXISTS, WITH INFINITE SEARCH
- FAILURE TO PRODUCE A SOLUTION THAT EXISTS (I.E. IS LOGICALLY DEDUCTIBLE FROM THE DATABASE)
- PRODUCTION OF A WRONG SOLUTION (LOGICAL ERROR)
- FAILURE TO PRODUCE A SOLUTION, BECAUSE IT IS NOT DEDUCTIBLE FROM THE DATABASE (LOGICAL ERROR)
- NON-DETERMINISM OF SOLUTION FROM REDUNDANT DATABASE

INVOLUNTARY/UNPREDICTABLE REDUNDANCY MAY ARISE FROM SELF MODIFYING PROGRAMS AS WELL AS FROM COMPLEXITY, E.G. THROUGH MAINTAINANCE

PROBLEMS WITH THE LOGICAL PARADIGM:

- CLOSED WORLD ASSUMPTION;

- LACK OF NEGATION

PROBLEMS WITH THE SEARCH STRATEGY:
VULNERABILITY TO

- UNBOUNDED RECURSION

- CIRCULAR REFERENCES

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- ancestor(X,Z), ancestor(Z,Y).
parent(abraham, isaac).
parent(isaac, jacob).


/* The query
?- ancestor(A,B).

obtains the answers:
    A = abraham, B = isaac ;
    A = isaac, B = jacob ;
    A = abraham, B = jacob ;
Not enough heap space.
*/
```

SOLUTIONS FOR THIS TYPE OF FAILURE: LIMITING DEPTH OF SEARCH; PARALLEL SEARCH

OF COURSE, THIS TOY EXAMPLE WOULD WORK CORRECTLY IF WRITTEN AS:

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :-  parent(X,Z), ancestor(Z,Y).
parent(abraham, isaac).
parent(isaac, jacob).
parent(jacob,a).

/* The query:
?-  ancestor(A,B).
gets the answers:
    A = abraham, B = isaac ;
    A = isaac, B = jacob ;
    A = abraham, B = jacob ;
   no
    */
```

## LANGUAGE FEATURES FOR IMPROVING RELIABILITY OF PROGRAMS:

NORMAL FEATURES OF HIGH-LEVEL LANGUAGES:

MODULES WITH NAME SCOPE RULES

ANNOTATION OF INPUT AND OUTPUT ARGUMENTS

LIMITATIONS TO SELF-MODIFIABILITY THROUGH ASSERT AND RETRACT

EXCEPTION HANDLING FOR:
- –INTERPRETER-GENERATED EXCEPTIONS;
- PROGRAMMER-DEFINED EXCEPTIONS

OTHER FEATURES FOR ROBUSTNESS AT RUN-TIME :

LIMITS ON DEPTH OF SEARCH (LOOP DETECTION, TIME-OUTS OR NUMBER OF STEPS?)

PARALLEL SEARCH: SIMPLIFIES REDUNDANT PROGRAMMING, RISKS NON-DETERMINISM

LIMITS ON EFFECTS OF SELF-MODIFICATION (ATOMICITY RULES)?

## TENTATIVE CONCLUSIONS:

- PROLOG HAS PROBLEMS THAT CAN BE CORRECTED BY NORMAL SOFTWARE ENGINEERING REMEDIES: STRUCTURE AND DISCIPLINE

- THE NON-PROCEDURAL STYLE HAS PROS AND CONS:

  SHIFT OF COMPLEXITY FROM APPLICATION TO INTERPRETER: +

  LESS CONTROL ON PROGRAM BEHAVIOUR: -

  NON-PORTABILITY OF PROGRAMS BETWEEN INTERPRETERS: -

  SIMPLICITY OF ADDING REDUNDANCY: +

  DIFFICULTY IN DIRECTING USE OF REDUNDANCY: -

- THERE ARE PECULIAR FAILURE MODES:

  DIFFERENCE BETWEEN "FAILED" PROCEDURE ACTIVATION (NO SOLUTION) AND WRONG SOLUTION

  UNCONTROLLED REDUNDANCY