EXPLOITING LOGIC PROGRAMMING IN ROBOT APPLICATIONS

ANTONIO NATALI

ANDREA OMICINI

FRANCESCO ZANICHELLI

Eighth Conference
on Logic Programming

GULP '93

Gizzeria Lido (CZ), June 18, 1993

Goal of this work

- there are particularly demanding application areas, such as **robotics**, where

- no standard solutions exist
- open problems are far more than solved ones

- may logic programming have a valuable impact on this fields?

  or more precisely

- may logic languages be extended so as to give positive contributions in terms of abstractions, architecture, programming environment?

Why robotics?

1) no standard robot architecture

- from planning-based to behaviour-based architectures
- spanning from highly symbolic to event-driven computations
- coexistence of different technologies

=> control in a logic language?

2) need for rapid prototyping programming environments

- highly experimental work
- no standard models, no fixed algorithms

=> advanced logic programming environment?

3) intrinsic parallelism

=> which model in a logic language?

The problem of control

1) controlling the program structure

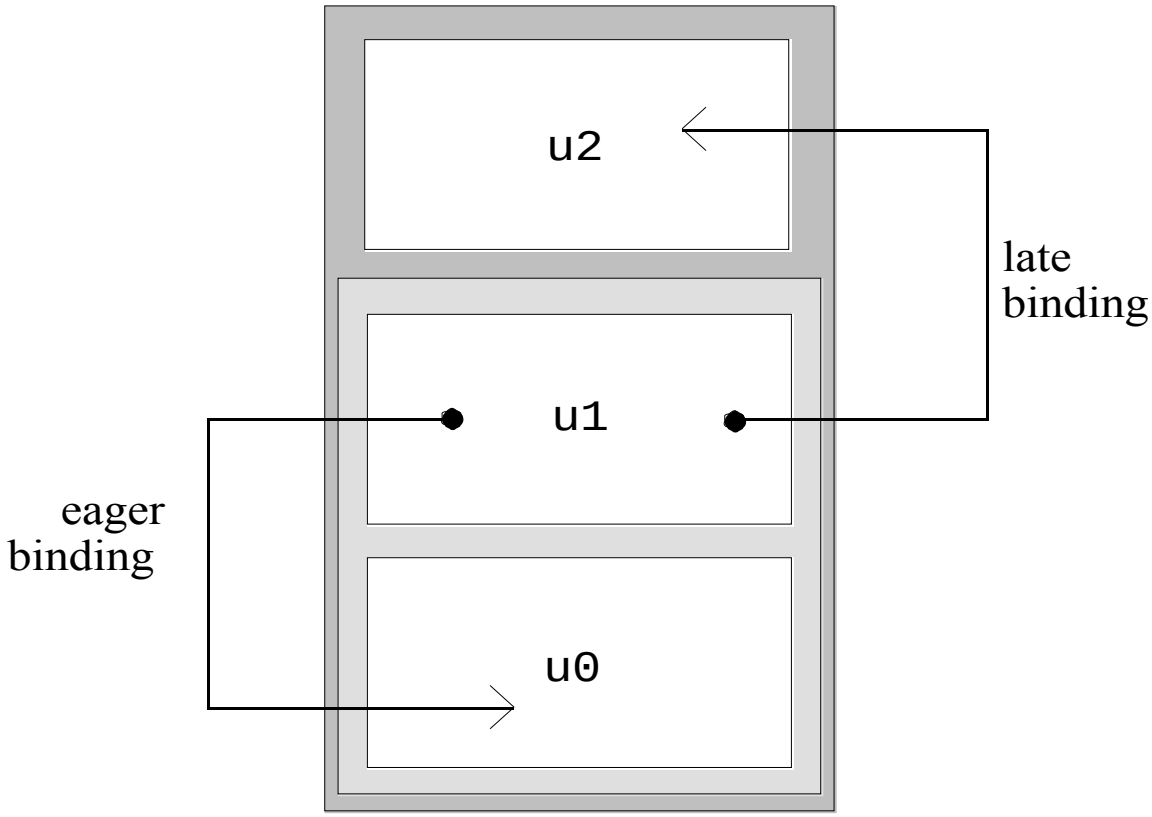
- blocks
- modules
- objects
- contexts

2) controlling the selection rule

- robotics calls for parallelism
    - streams
    - coroutine-based mechanisms
    - communicating units

Contextual logic programming

- static theories
  (*units*)
- units are open theories
  (=> delegation)
- units can be dynamically composed
  (=> hierarchies)
- structured theories
  contexts as lists of units
- different binding policies


u2
late
binding
u1
eager
binding
u0

Classes, instances and state

- class as an open world
- object as a closed world
- instance configuration as theory creation
- state modifications:
  no assert/retract
  - groundness
  - scope
  - backtrackable
  - delayed semantics
- reclassification as
  dynamic composition

=>evolving context

Context abstraction

- multiple interpretations

(*from a lower level to a higher level*)

- binding environment

- module
- updatable knowledge base

- unmodifiable object (instance)
- object class

- logic theory
- current line of reasoning

- uniformity of static and dynamic composition

=> the contextual extension provides a uniform support for a wide range of abstractions

Parallelism in logic programming?

- streams

  - not Prolog-compatible
  - no explicit control on process scheduling
  - low-level, point-to-point communication model

- coroutining

  - more conservative

- Communicating Prolog Units
  (Linda style)

=> communication through logic theories

- processes share knowledge in clausal form

Natural integration in the contextual model

=> Contextual Communicating Prolog Units

Communication mechanisms

- a communication primitive consists of a process interaction with a logic theory (context) via meta-operators

  e.g. Linda/CPU *read* is a *demo* in a logic theory

- CPU world abstraction as an evolving context

=> new communication policies may be easily provided through high level specialized demo operators

– An example: new world method

Specification: when new data arrive from sensors

- a waiting agent must be reactivated
- data consuption by the agent must be logical only (agent may backtrack on failure and try other messages)
- when no other data from sensors are available, failure must be induced

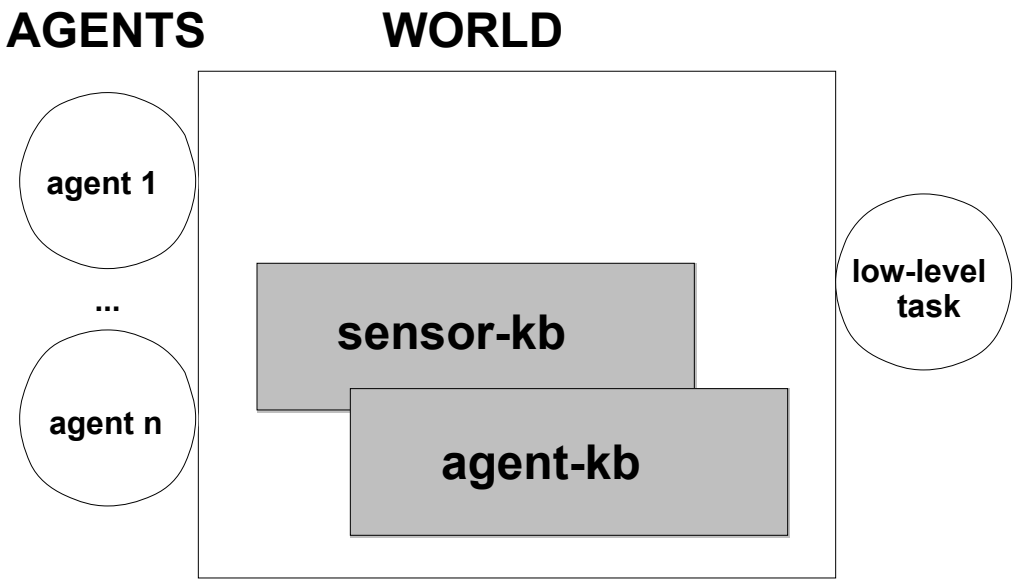
An example: a possible code

```
readAtLeastOne( M ) :-
        curr_process( P ),
        makeFunctorCopy( M, M1 ),  %free vars in M1
        #lastaccess( M1, P, TOld ),!,
        getMsg( M, P, TOld ).

getMsg( M, P, TOld ) :-
% if there are messages
        copy_term( M, MM ),        %MM is a new term
        #message( MM, TNew ),
        TNew > TOld, !,
        currentTime( TLimit ),
% then take them
        takeMsg( M, P, TOld, TLimit ).
getMsg( M, P, TOld ) :-
% otherwise suspend the agent
        suspend( M,P ),            % eager binding
% and retry when reactivated
        getMsg( M, P, TOld ).

takeMsg( M, P, TOld, TLimit ) :-
% logically consume all messages (in backtracking)
% between TOld and TLimit
        message( M, TNew ),
        TNew > TOld, TNew =< TLimit,
        makeFunctorCopy( M, M1 ),
% and update the agent last access time
        +lastaccess( M1, P, TNew ).
takeMsg( M, P, TOld, TLimit ) :-
% look for further messages beyond TLimit
        copy_term( M, MM ),
        currentTime( TCur ), !,
        TCur > TLimit,
        #message( MM, TNew ),
        TNew > TLimit, !,
% if there are messages, then consume them
        takeMsg( M, P, TLimit, TCur ).
% otherwise fail.
```

Exploiting the CCPU model: CARA

Contextual Agent Robot Architecture

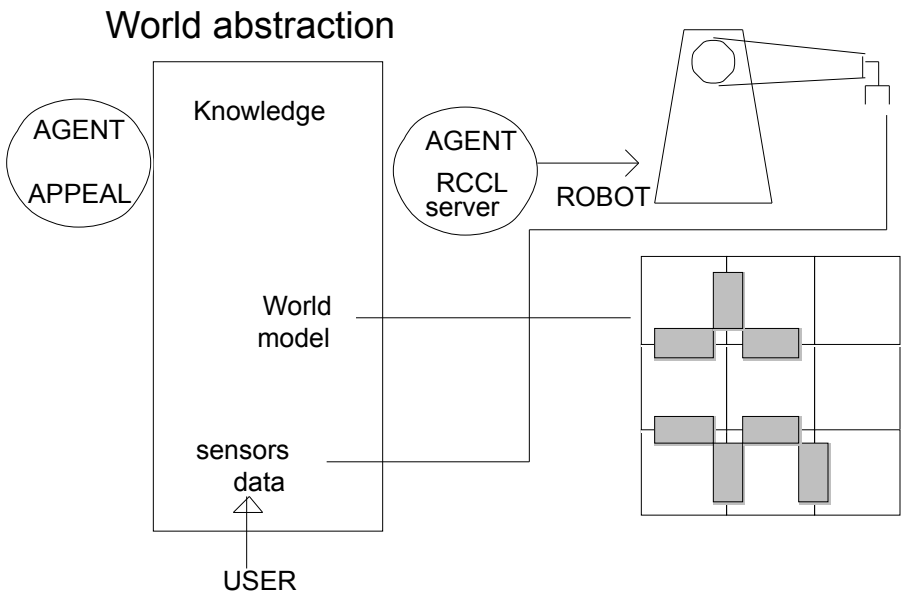
- abstract, general-purpose architecture



- heterogeneous agents
  (agreement only on interface)

- full range of hybrid architectures can be supported

A case study



Viable for

- experimenting rapid prototyping techniques
- exploring different communication protocols
- exploiting different architectural solutions

Four steps (i)

1) basic agent

- purely reactive
- event-driven, asynchronous computation
  but
- it is open!

2) meta control

- subsuming basic behaviour
- revoking moves on failure
- new world method => new protocol

3) situated reasoning

- all knowledge acquired is available
- reasoning on a logic theory

4) robot driving

- redefining move methods
  simply by pushing a “robot” unit

```
:- unit smartagent.
.......

smart_agent( Path,pos(N1,N2),Res ):-
   checkpos( N1,N2 ),!, user:reverse( Path, Res ).
smart_agent( Path, Dest, Res ):-
   Dest = pos(D1,D2),
   world <- pos( N1,N2,_ ),!,         %get current position
   (D1 > N1 -> toeast >:> smart_agent(Path,Dest,Res);
    D1 < N1 -> towest >:> smart_agent(Path,Dest,Res);
    D1 = N1 -> toelsewhere >:> smart_agent(Path,Dest,Res)).

action( N1,N2,PrevDir,Dir,P1,P2,Path,Dest,Res ) :-
   opposite( Dir,OpDir ),
   ( ( opposite( PrevDir, Dir );
       world <- message( cell(P1,P2,OpDir), _ );
       world <- message( cell(N1,N2,Dir), _ )
     ) ->
         \+ loop( [Dir|Path] ),robotmove( Dir ),
         smart_agent( [Dir|Path], Dest, Res ) ;
         neverbefore( N1,N2 ),
         sendcommand( torccl( sonar( pos(N1,N2),_ ))),
         world <- readcwa( cell(N1,N2,NextDir) ),
         Dir = NextDir,!,
         \+ loop( [Dir|Path] ),
         robotmove( Dir ),
         smart_agent( [Dir|Path], Dest, Res )
   ).

%------------------------------

:- unit toeast.

smart_agent( Path, Dest, Res ):-
   Dest = pos(D1,D2),                % get destination
   world <- pos( N1,N2,PrevDir ),!,  % get current position
   g0 <- ( rows(NRows), columns(NCols) ),
   ( To1 is N1+1, To1 < NCols, To2 = N2, Dir = e;
                                     % try to go east
     ( D2 < N2
       -> (To2 is N2-1, To2 >= 0, To1 = N1, Dir = s;
                                     % try to go south
           To2 is N2+1, To2 < NRows, To1 = N1, Dir = n)
                                     % try to go north
        ; (To2 is N2+1, To2 < NRows, To1 = N1, Dir = n;
                                     % try to go north
           To2 is N2-1, To2 >= 0, To1 = N1, Dir = s)
                                     % try to go south
     );
     To1 is N1-1, To1 >= 0, To2 = N2, Dir = o
                                     % try to go west
   ),
   action(N1,N2,PrevDir,Dir,To1,To2,Path,Dest,Res).
                                     % eager call!!
```

Conclusions

- *logic programming* seems to represent a promising approach to some of *robot programming* open problems

- logic programming must be suitably *extended* for advanced robot programming, in particular, *control* issues have to be addressed

- (*evolving*) *contexts* represent a unifying metaphor subsuming most of abstractions and mechanisms needed

- a fully-working *contextual logic programming environment*, increased with the notion of *concurrency*, is an effective framework where to build advanced robot applications, according to *different architectures* and behaviours