OBJECTS WITH STATE IN CSM

ANTONIO NATALI

ANDREA OMICINI

SECOND COMPULOG-NETWORK
SUBGROUP MEETING
ON PROGRAMMING LANGUAGES

joint with

WORKSHOP ON LOGIC LANGUAGES
PROGETTO FINALIZZATO INFORMATICA

Pisa, May 6, 1993

Logic programming style (1)

- relational approach

- objects (and attributes) as terms

```
resistor(10,5,2)
```

- relationship between objects as clauses

```
resistor(V,R,I) :- V = R * I.
```

? complex objects as terms are painful

=> clausal representation

=> object as a logic theory

Logic programming style (2)

- objects of a class

```
resistor(V,R,I) :- V = R * I.
```

– specification of a class of resistor exemplars

```
?- resistor(V,5,2).
```
V = 10

– substitution as instance state configuration

```
resistor(10,5,2)
```

- everything up to the caller

? no locality, no encapsulation

? unsatisfactorily representation of instance/class relationship

Contextual logic programming

- static theories
  (*units*)

- units are open theories
  (=> delegation)

- units can be dynamically composed
  (=> hierarchies)

- structured theories
  (*contexts*)

=> contexts can be used to represent objects in a hierarchy of classes

The Dumbo example

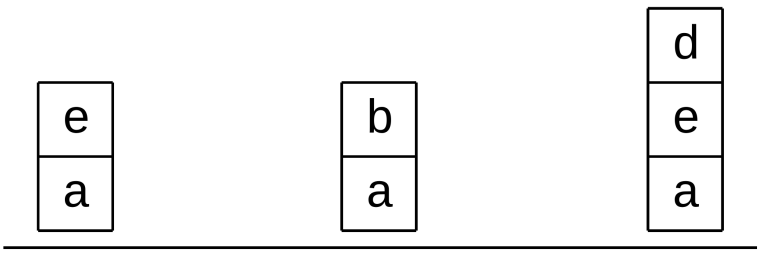
```
:- unit animal.                :- unit elephant.
move(walk)  :- #legs(_).       legs(4).
move(run)   :- #legs(2).       ears(2).
move(gallop):- #legs(4).       trunk(1).
move(fly)   :- #wings(2).

:- unit dumbo.                 :- unit bird.
flyingEars(X) :- ears(X).      legs(2).
wings(X) :- flyingEars(X).     wings(2).
```

?- `animal>>elephant>>move(X).`
X = walk;
X = gallop;
no

?- `animal>>bird>>move(X)`
X = walk;
X = run;
X = fly;
no

?- `animal>>elephant>>dumbo>>move(X)`
X = walk;
X = run;
X = fly;
no



CSM extensions

- context switch

```
?- [dumbo,elephant,animal] <- move(X).
```

X = walk; X = run; X = fly;
no

- context identifier

```
?- createContext(sweetDumbo,
        [dumbo, elephant, animal]),
   sweetDumbo <- move(X).
```

X = walk; X = gallop; X = fly;
no

- static definition

```
:- unit animal.
....
:- unit elephant inherits animal.
....
:- unit bird inherits animal.
....
:- unit dumbo inherits elephant.
....
```

```
?- dumbo <- move(X).
```

X = walk; X = gallop; X = fly;
no

What is an instance in CSM?

- new instance: new variables

```
:- unit resistor.
   resistor(V,R,I) :- V = R * I.
   madeOf(silicon).
```

=> encapsulation and locality principles still unattended; uniformity is lost

- instance memoing of information computed for a goal `O<-G`

```
:- resistor <- resistor(V,5,2).
```
V = 10

- memoing information in a context `r` so that it is possible

```
:- r <- resistor(V,R,I).
```
V = 10
R = 5
I = 2

instance state configuration
as
theory creation

i) `<<=/2` operator

```
TInstance <<= TClass(....)
```

– atomic operation

– undone on backtracking

– unit `TInstance` dynamic creation

– context `TClass` extension

=> fixed theory `[TInstance|TClass]` representing a non-modifiable instance of class `TClass` whose state can be accessed through self (lazy) binding

? how to specify and force instance state configuration

ii) constructor

- specific “method” has to be selected

– rule: functor like theory name

– normal Prolog definition clause(s)

- specifying whole state configuration

? how

iii)`+/1` operator

- in a constructor clause

- specifying a part of the *self* unit configuration in terms of a ground fact

The Resistor example

```
:- unit resistor.
resistor(V,R,I) :- V = R * I,
                   +voltage(V),
                   +value(R),
                   +current(I).
```

```
?- r1 <<= resistor(V,5,2),
  r2 <<= resistor(V,R,5).
    V= 10, R = 2

?- r1 <- (voltage(V1),current(I1)),
  r2 <- (voltage(V1),current(I2)).
    V1 = 10, I1 = 2, I2 = 5
```

```
:- unit resistor.
resistor(V,R,I) :-
   voltage(V), value(R),
    current(I), !.
resistor(V,R,I) :-
    V = R * I, +voltage(V),
    +value(R), +current(I).
```

```
?- r1 <<= resistor(V,5,2),
  r1 <- resistor(V1,R1,I1).
    V= 10, V1 = 10, R1 = 5, I1 = 2
```

State representation is not state modification

- state modification in Prolog is passing `OldState / NewState` parameters through goals

- contexts can be passed like every other structure

? creating a new fixed theory for every update is not satisfactorily

=> must consider *theory updating*

evolving context
for
object evolution

Evolving contexts

- `+/1` in any clause

- informal semantics

– puts a ground fact on top of current instance database
=> similar to an `asserta` in the top (state) unit of the current (self) context

– `+M` is different from `asserta(M)` in that

  a) M must be ground
  b) effect local to current self unit
  c) undone on backtracking
  d) delayed semantics

– (d) to distinguish between *deduction* and *updating actions*

- evolving context =
  fixed context + extendible unit

– class methods and instance state

- monotonic extension of a logic theory

Prototypes

- sharing state

– reducing costs of new object creation

– propagating evolution of a prototype to its extension objects

- `islike` operator

– if `r2 islike r1`, `r2` shares
  - class of `r1` as a context
  - state of `r1` as clauses

  i.e. `r2` has the same first subcontext of `r1` and a different self unit sharing all clauses of `r1` self unit

– an update operation on `r1` extends both `r1` and `r2`, while an update operation on `r2` extends `r2` only

A Library example

```
:- unit library.
   ....
 addBook(author,title) :-
       +book(author,title).
```

```
?- lib1 <<= library(...),
   lib1 <- addBook(ritchie,c),
   lib2 islike lib1,
   lib2 <- addBook(brooks,hook),
   lib1 <- addBook(pippo,pensieri).
```

```
?- lib1 <- book(A,T).
        A = pippo, T = pensieri  ;
        A = ritchie, T = c  ;
   no
```

```
?- lib2 <- book(A,T).
        A = brooks, T = hook  ;
        A = pippo, T = pensieri  ;
        A = ritchie, T = c  ;
   no
```

Final considerations

- Extension alone

– cannot directly capture *one shot* notions (like “age”)

=> disciplined access through

  - access predicates
  - pruning
  - user-handled meta knowledge
  - introducing negative knowledge
  - logic overriding
  - logic retraction

– may be too eager to handle

=> knowledge retraction mechanisms

  - implicit (overriding)
  - explicit (`-/1` ?)

- Permanent modifications should be allowed?

? back to assert? How much?