1

Tools

- Side Effects
  `assert(C), asserta(C), assertz(C), retract(C), abolish(P, N)`
- Programs as Data
  `clause(RHS, LHS), =..(X, L), call(C)`
- Utilities
  `setof(X, P, S), bagof(X, P, S)`

©1993 John Mylopoulos

2

Side Effects

- `assert(C)` - adds `C` to the collection of clauses constituting the Prolog program
- `asserta(C)` - adds `C` to the start of the collection of clauses
- `assertz(C)` - adds `C` to the end of the collection
- `retract(C)` - deletes first clause that unifies with `C`
- `abolish(P, N)` - abolishes all clauses which use predicate `P` of arity `N`

These predicates all lead to self-modifications of a Prolog program

©1993 John Mylopoulos

3

Examples of Assertz

```
|?- [user].
| man(john).
| man(fred).
| ^D user con...
yes

|?- man(X).
X = john;
X = fred;
no

|?- assertz(man(tom)).
yes

|?- man(X).
X = john;
X = fred;
X = tom;
no
```


©1993 John Mylopoulos

4

Examples of Assertz (continued)

```
|?- male(X).
no

|?- man(X), assertz(male(X)).
X = john;          %performs assertz(male(john))
X = fred;          % ditto with male(fred)
X = tom;           % ditto with male(tom)
no

|?- male(X).
X = john;
X = fred;
X = tom;
no

|?- assertz(:-((human(X)),(man(X)))).
X = _0                  % the clause has been added

|?- human(X).
X = john:
X = fred;
X = tom;
no
```


©1993 John Mylopoulos

5

Examples of Assertx (continued)

```
   man(john).
   man(fred).

|?- asserta(man(tom)).
                % adds man(tom) at beginning
yes             % of list of clauses

|?- assert(man(george)).
                % adds man (george))
yes             % somewhere

|?- [user].
| a(X) :- asserta(a(3)), fail.
| ^D
yes

|?- a(X).
no

|?- a(X).
X = 3;                  % finds the "a" value asserted
no
```


©1993 John Mylopoulos

6

Examples of Assertx (continued)

```
|?- [user].
| e(X) :- asserta( :-((e(X)), (f(X)))),
fail.
| f(10).
| ^D user con...
yes

|?- e(Z).        % though goal failed, the clause
no              % e(X) :- f(X). has been added

|?- e(Z).
Z = 10
yes
```


©1993 John Mylopoulos

7

Examples of Assertx (continued)

```
|?- [user].
| b(X):-asserta(:-((c(X)),(d(X)))),fail.
| b(X) :- c(X).
| d(4).
| ^D user con...
yes

|?- b(Y).
Y = 4;
no              % first clause adds new clause then
                % fails; second clause generates
                % ?- c(Y) which is unified with new
                % clause to give the answer.
|?- b(Y).
Y = 4;
Y = 4;          % we got now two copies of the
                % c(X) :- d(X) clause, so we
                % let the same answer twice
no
```


©1993 John Mylopoulos

8

Examples of Retract

```
   married(john).                    (1)
   married(fred).                    (2)
   married(fred, sue).               (3)
   married(mark, jane).              (4)

|?- retract(married(fred, sue)).
yes

   married(john).                    (1)
   married(fred).                    (2)
   married(mark, jane).              (4)

|?- retract(married(X)).
X = john
yes

|?- retract(married(_)).
yes

married(mark, jane).                 (4)

|?- retract(foo(_)).
no
```


©1993 John Mylopoulos

9

Examples of abolish

```
married(john).                    (1)
married(fred).                    (2)
married(fred, sue).               (3)
married(mark, jane).              (4)

|?- abolish(married, 1).    % removes (1), (2)
yes

|?- abolish(married, 2).    % removes (3), (4)
yes

|?- abolish(married, 3).    % no effect, but...
yes
```

`abolish` allows global operations on a Prolog program

©1993 John Mylopoulos

10

Using Side Effects

```
liked(fred, judy).    liked(fred, sue).
liked(fred, flo).     liked(mark, fran).
liked(mark, judy).    liked(derek, june).
pop(X):-
      liked(X, Y), liked(X,Z),
      not(=(Y, Z)).


?- pop(X)
```

leads to

```
   X/fred      6 times
   X/mark      2 times
```

We need side effects to avoid double counting:

```
popular(X) :-pop(X),
                not(tagged(X)),
                assert(tagged(X)).
```


©1993 John Mylopoulos

11

Using Side Effects (continued)

```
|?- popular(X).
X = fred;
X = mark;
no              % each answer comes out once, but
                % we have added three tagged
                % clauses

|?- tagged(Z).
Z = fred;
Z = mark;
no              % ...but now we don't have any
                % popular persons, just tagged
                                    ones
|?- popular(X).
no

|?- abolish(tagged,1).
yes

|?- popular(X).
X = fred;
X = mark;
no
```


©1993 John Mylopoulos

12

Using Side Effects (continued)

- To remove all `liked` clauses of a `popular` person use:

```
changeOfMind(X) :- pop(X),
           retractall(liked(X,_)),
           retractall(liked(_,X)).
```

- Definition of `retractall`:

```
retractall(X) :- retract(X), fail.
retractall(_).
```


©1993 John Mylopoulos

13

Data as Programs

- `=..` translates a list into a function term
- `call` attempts to achieve a constructed goal

```
likes(john, mary).            (1)
likes(ellen, ted).            (2)
likes(ted,Y):-likes(Y,mary). (3)

|?- =..(X, [likes, john, mary]).
X = likes(john, mary);
no

|?- =..(X, [likes, john, Y]).
X = likes(john, _1)
Y = _1

|?- =..(X, [likes, john, Y]), call(X).
X = likes(john, _1)
Y = mary
```


©1993 John Mylopoulos

14

Programs as Data: `clause`

- `clause(LHS, RHS)` finds a clause of form "`LHS :- RHS`"

```
likes(john,mary).
likes(ellen,ted).
likes(ted,Y) :- likes(Y,mary).

|?- clause(likes(A,B),RHS).
A = john
B = mary
RHS = true;

A = ellen
B = ted
RHS = true;

A = ted
B = _1
RHS = likes(_1,mary);

no
```

The predicate symbol of LHS must be a constant

©1993 John Mylopoulos

15

Utilities: `setof`

```
   likes(tom, beer).
   likes(dick beer).
   likes(harry, beer).
   likes(bill, cider).
   likes(jan, cider).
   likes(tom, cider).

|?- setof(X,likes(X,  beer),S).
X = _0
S = [dick, harry, tom];
no

|?- setof(thirsty(X),likes(X,beer),S).
X = _0
S = [thirsty(dick),thirsty(harry),
                   thirsty(tom)];
no
```


©1993 John Mylopoulos

16

Utilities: `setof`

```
   likes(tom,beer).
   likes(dick,beer).
   likes(harry,beer).
   likes(bill,cider).
   likes(jan,cider).
   likes(tom,cider).

|?- setof(X,likes(X,Y),S).
X = _0
Y = beer
S = [dick,harry,tom];

X = _0
Y = cider
S = [bill,jan,tom];
no

|?- setof((X,Y),likes(X,Y),S).
X = _0
Y = _1
S = [(bill,cider),(dick,beer),
     (harry,beer),(jan,cider),
     (tom,beer),(tom,cider)];
no
```


©1993 John Mylopoulos

17

`setof` Variants

```
|?- setof((Y,S),setof(X,likes(X,Y),
                              S), SS)
Y = _0
S = _1
X = _6
SS = [(beer, [dick, harry, tom]),
      (cider, [bill, jan, tom])];
no
```

% ^ indicates existential quantification

```
|?- setof(X, Y^likes(X, Y), S)
X = _0
Y = _1
S = [bill, dick, harry, jan, tom]
```

% `bagof` is like setof but does not
% eliminate duplicates

```
|?- bagof(X, Y^likes(X, Y), S)
X = _0
Y = _1
S = [tom, dick, harry, bill, jan, tom];
no
```


©1993 John Mylopoulos

18

Utilities: `functor, arg`

- The predicate **`functor`** can be used to construct a functor of a given arity.

- The predicate **`arg`** pulls a specified argument out of a functor.

```
|?- functor(f(a, b, g(Z)), F, N).
Z = _23
F = f
N = 3

|?- arg(2,related(john,mother(jane)),X)
X = mother(jane)

|?- arg(2, [a, b, c], X).
X = ??
```


©1993 John Mylopoulos

`functor, arg` (continued)

This program takes a term of the form
f(arg_1, arg_2,...arg_n)
and outputs
"the f of arg_1 and arg_2 and ...and $arg_{(n-1)}$ is arg_n"

```
output(S) :- functor(S, Fn, _),
             write('the'),
             write(Fn),
             write('of'),
             outputAllButOne(S),
             write('is'), outputLast(S).

outputAllButOne(S) :- functor(S,_,0),!.
outputAllButOne(S) :- functor(S,_,Ar),
          N is Ar - 1, zeroArgs(1, N, S).

zeroArgs(N1, N2, _) :- N1 > N2, !.
zeroArgs(N1, N1, S) :- !, arg(N1, S, A),
             write(A).
zeroArgs(N1, N2, S) :- arg(N1, S, A),
             write(A), write('and'),
             N3 is N1 + 1,
             zeroArgs(N3, N2, S).
outputLast(S) :- functor(S, F, N),
   (N = 0 ; (arg(N, S, A), write(A))), !.
```


©1993 John Mylopoulos