DATA MODELLING IN ENGINEERING

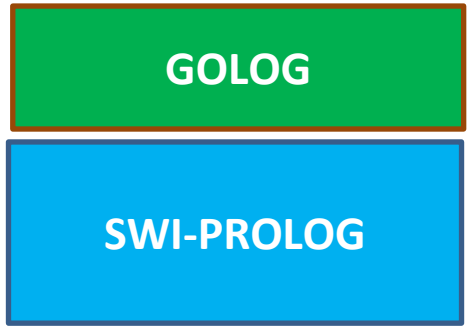
MODELING IN FRAMES – GOLOG – PART II -

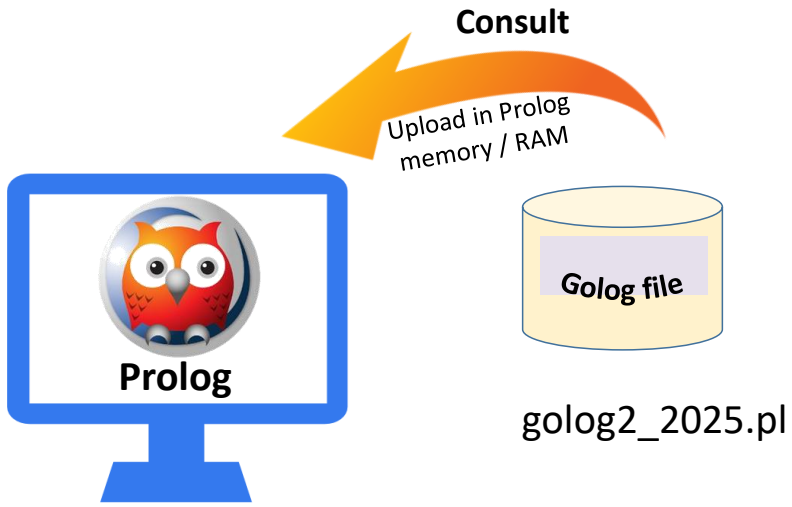
Ana Inês Oliveira aio@fct.unl.pt

2024 - 2025

Contents

Golog





golog2_2025.pl

A mini Frame Engine written in Prolog

- Creation and manipulation of frames and their slots and values.
- Definition of relations and inheritance mechanisms.
- Definition of methods and reactive programming.

GOLOG was inspired on a commercial product (KNOWLEDE CRAFT), one of the first frame engines

Golog Operations (review)

A. On frames

- **new_frame(F)** → *Creates a new frame with name F. At this stage the frame has no slots.*
- **frame_exists(F)** → *Checks if a frame with name F exists in memory*
- **show_frame(F)** → *Displays the frame with name F*
- **delete_frame(F)** → *Deletes the frame with name F from memory*

B. On slots

- **new_slot(F,S)** → *Adds a new slot with name S to the frame F. This slot has no value.*
- **new_slot(F,S,V)** → *Adds a new slot with name S to the frame F and puts the value V in the slot.*
- **delete_slot(F,S)** → *Deletes slot S from frame F.*
- **frame_local_slots(F,LS)** → *Obtains a list LS with the names of all slots locally defined in F.*
- **get_all_slots(F,LS)** → *Obtains a list LS with the names of all slots of F, including both the locally defined and the inherited ones.*

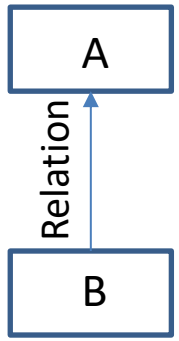
Golog Operations (review)

C. On values

- **new_value(F,S,V)** — *Writes a new value V in the slot S of frame F.*
- **new_values(F,S,LV)** — *Writes a list of values LV in the slot S of frame F.*
- **add_value(F,S,V)** — *Adds a new value V to the values already present in slot S of frame F.*
- **add_values(F,S,LV)** — *Adds a list of values LV to the values already present in slot S of frame F.*
- **get_value(F,S,V)** — *Obtains a value V from slot S of frame F. If the slot as a list of values, it gets the 1st value of the list*
- **get_values(F,S,LV)** — *Gets a list LV with all values of slot S of frame F.*
- **delete_value(F,S,V)** — *Deletes a value V from slot S of frame F.*
- **delete_values(F,S)** — *Deletes all values of slot S of frame F.*

Golog Operations (review)

Defines a new relation with name Relation.
The definition requires 4 parameters



B is related to A

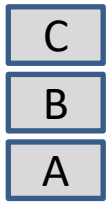
D. On relations

- **new_relation(Relation,Transitivity, Restriction, Inverse)**

Transitivity: transitive, intransitive

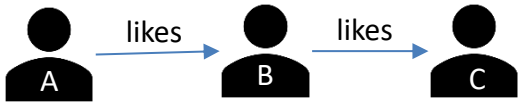
Restriction: all
none
inclusion(LS)
exclusion(LS)

Inverse: name of the inverse relation or *nil*



on-top-of

The relation "on-top-of" is transitive:
If B is on-top-of A and C is on-top-of B, then we can infer that C is on-top-of A



The relation "likes" is intransitive:
If A likes B and B likes C, we cannot infer that A likes C

- **delete_relation(Relation)**
- **frame_actual_relations(F,LR)**

To delete a relation of name Relation

Obtains a list LR with the names of all relations associated to a given frame F

Allows to automatically create an inverse relation. If we don't want it, put "nil" in this parameter

Specifies the level of inheritance:
all – B inherits all slots of A
none – B does not inherit any slot from A
inclusion(LS) – B only inherits from A the slots mentioned in list LS
exclusion(LS) – B inherits all slots from A except those indicated in list LS

Golog Operations: Methods

E. On Methods

A **method** in frames is a procedure associated with a **class**.
A **method** defines the behavior of the **class** and its sub-classes and instances.
A **method** is an action that an object (class) is able to perform.

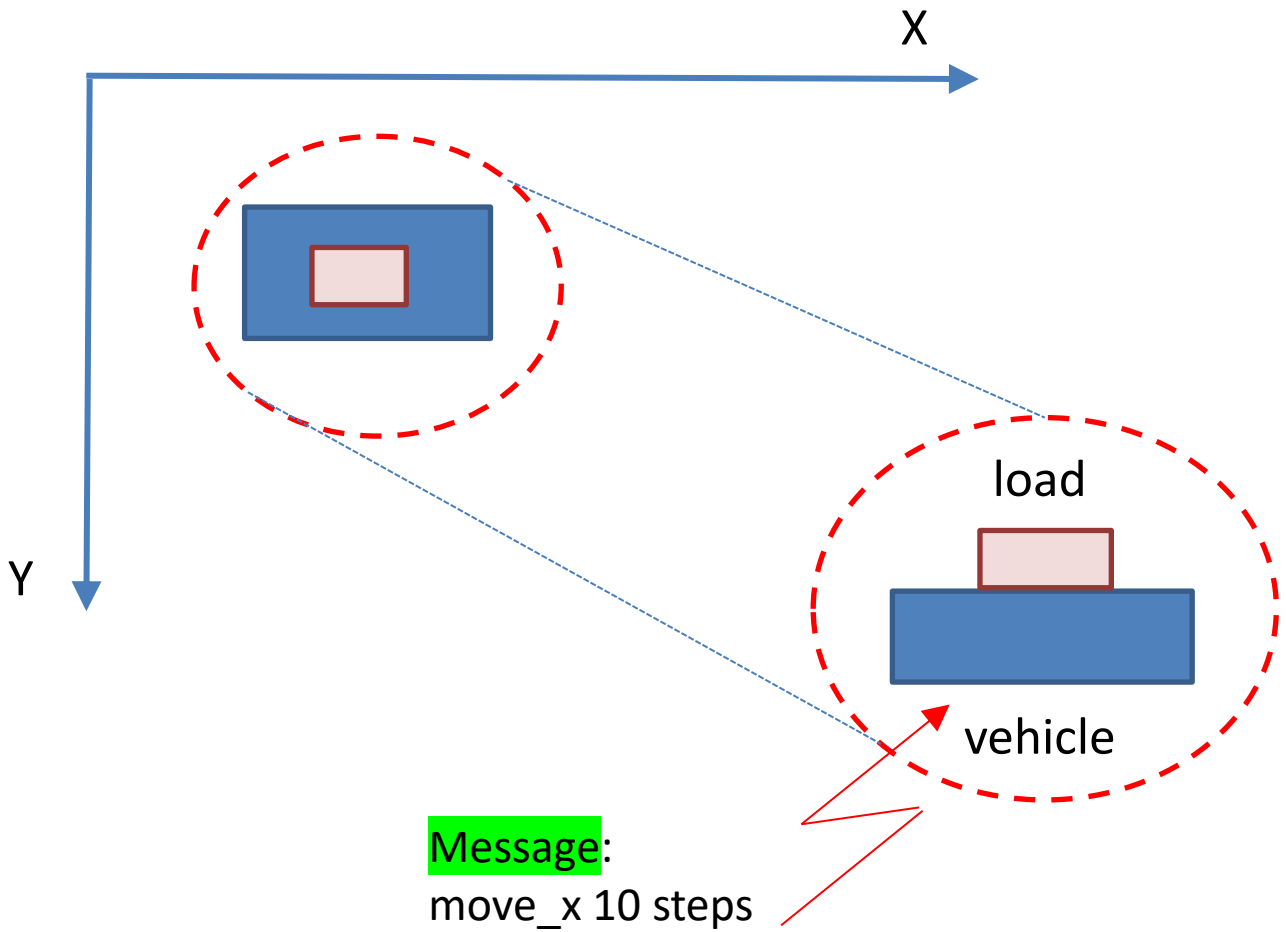
- **new_slot(F, S, Method)** → *Creates a new Method, which is "stored" in a new slot S.*
- **call_method(F, S, LPar)** → *Calls the method identified by slot S, passing the list of parameters Lpar to the corresponding procedure*
- **call_method_0(F, S)**
- **call_method_1(F, S, P)**
- **call_method_2(F, S, P1, P2)**
- **call_method_3(F, S, P1, P2, P3)**

Particular cases of call_method for the cases that the corresponding procedure has 0, 1, 2, or 3 parameters, respectively

call_method_0(F, S) equivalent to call_method(F,S,[])
call_method_1(F,S,P) " " call_method(F,S,[P])
call_method_2(F,S,P1,P2) " " call_method(F,S,[P1, P2])
call_method_2(F,S,P1,P2,P3) " " call_method(F,S,[P1, P2, P3])

Golog Operations: Methods

Example: Let's consider a vehicle moving in a 2D space and obeying to the commands *move_x* and *move_y*.



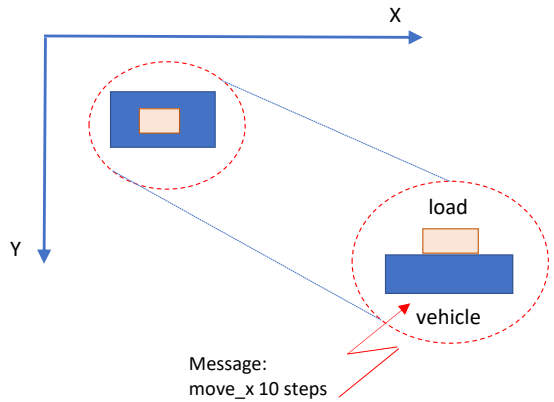

```
?-new_frame(vehicle).
?-new_slot(vehicle, position).
?-new_values(vehicle,position,[15,8]).

?-new_slot(vehicle, move_x,  move_along_x).

?-new_slot(vehicle, move_y, move_along_y).
```

The methods move_along_x and move_along_y can be defined in Prolog

Golog Operations: Methods

```
?-new_frame(vehicle).
?-new_slot(vehicle, position).
?-new_values(vehicle,position,[15,8]).
?-new_slot(vehicle, move_x,  move_along_x).
?-new_slot(vehicle, move_y, move_along_y).
```

Procedures that implement the methods move_x and move_y



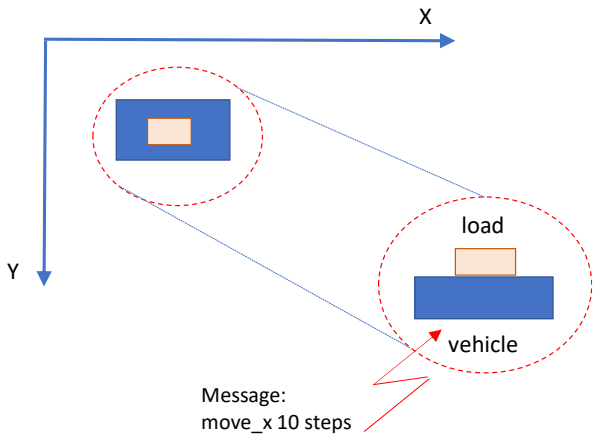

```
move_along_x(F,Dx) :- get_values(F,position,[X,Y]), Nx is X + Dx, new_values(F,position,[Nx,Y]), write('I moved '),
                      write(Dx),write(' steps along X'), nl.
move_along_y(F,Dy) :- get_values(F,position,[X,Y]), Ny is Y + Dy, new_values(F,position,[X,Ny]), write('I moved '),
                      write(Dy),write(' steps along Y'), nl.
```

Let's also define:

```
?-new_relation(on_top_of,transitive,inclusion([position]), under).
?-new_frame(load).
?-new_slot(load,on_top_of, vehicle).
```

The relation on_top_of allows the object "load" to inherit the "position" from the vehicle.

Golog Operations: Methods



?- get_values(load,position,P).
P = [15, 8].

?- call_method_1(vehicle,move_x, 10).
I moved 10 steps along X
true.

?- get_values(vehicle,position,P).
P = [25, 8].

?- get_values(load,position,P).
P = [25, 8].

We could have used
call_method_1(vehicle, move_y,5).

?- call_method(vehicle,move_y, [5]).
I moved 5 steps along Y
true.

?- get_values(vehicle, position, P).
P = [25, 13].

?- call_method(load,move_x,[2]).
false.

Because "load" does not inherit the method "move_x".
It only inherits "position"

Golog Operations: Methods

Let's consider a vehicle with the operations: move, stop.

vehicle	
speed	0
move	moveaction
stop	stopaction

```
stopaction(F):- new_value(F, speed, 0), write('I stopped ! '), nl.
moveaction(F, V):- new_value(F, speed, V),
                   write('I am moving at '), write(V),
                   write(' Km/h'), nl.
```

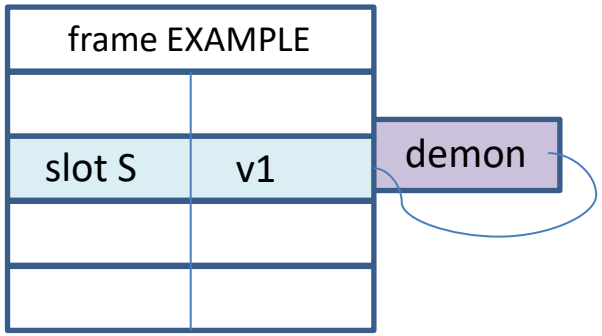
```
?-new_frame(car).
?-new_slot(car, speed, 0).
?-new_slot(car, move, moveaction).
?-new_slot(car, stop, stopaction).
```

```
?- call_method_1(car, move, 120).
I am moving at 120 Km/h
true.

?- show_frame(car).

    Frame: car {
      move: moveaction
      speed: 120
      stop: stopaction
    }

true.
```

```
?- call_method_0(car, stop).
I stopped !
true.

?- get_value(car,speed,V).
V = 0.
```

Golog Operations: Reactive Programming

F. On attached Predicates or Demons



A predicate associated to a slot (hidden) that reacts (executes) when certain actions are performed on the slot

- **new_demon(F,S,Demfunc, Demtype, When, Effect)**

Demtype:
- if_read — *Demon reacts when some reading action is performed on the slot. E.g. it reacts to get_value, get_values*
- if_write — *Demon reacts when some writing action is performed on the slot. E.g. it reacts to new_value, new_values, add_value, add_values*
- if_executed — *Demon reacts when a call_method is applied*
- if_deleted — *Demon reacts when a delete action is applied*

When:
- before
- after

Indicates if the demon is executed before or after the function that triggered it

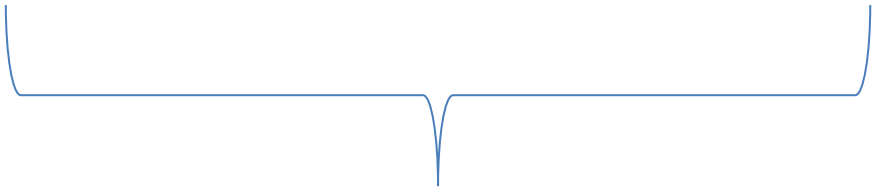
Effect:
- alter_value
- side_effect

Indicates if the demon interferes i.e. changes the parameter of the triggering function

Golog Operations: Reactive Programming

Format of the rule associated to the slot:
E.g.:

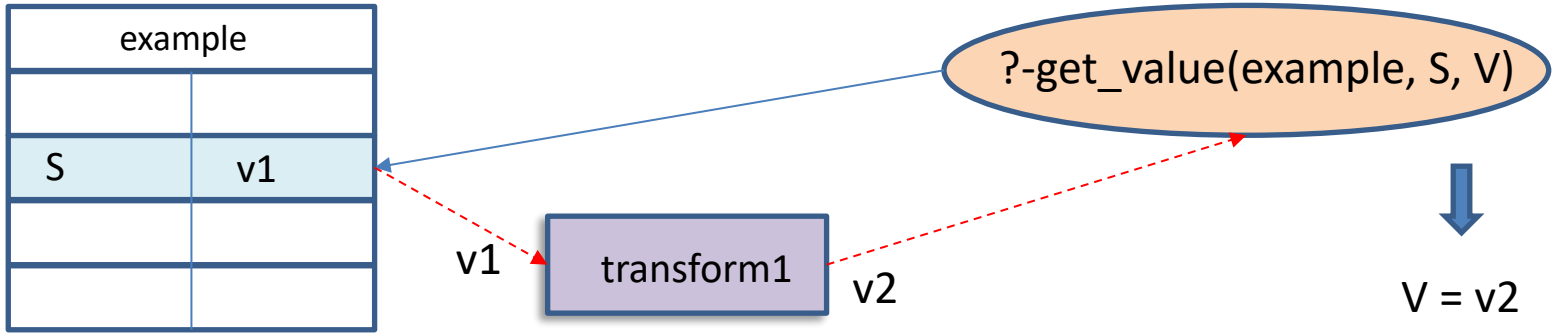
transform(Frame, Slot, Received_value, Returned_value) :- ....



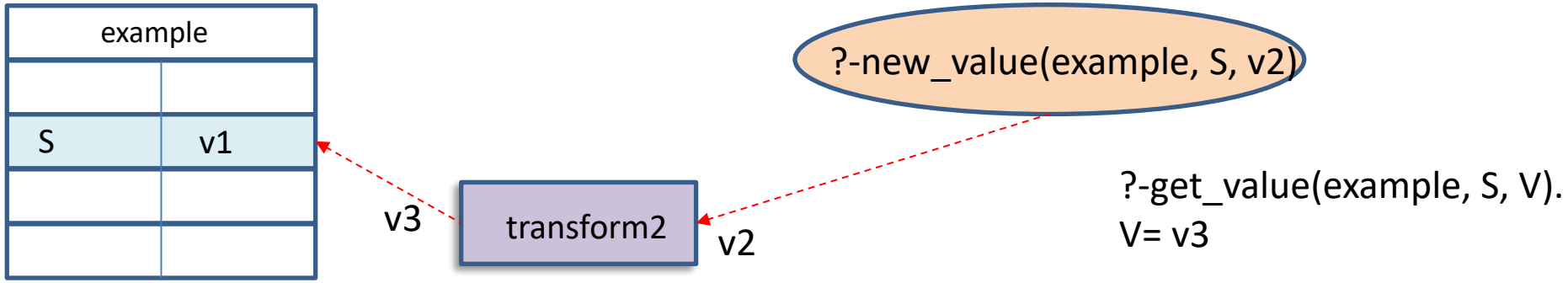
The function (rule) that implements the demon has always 4 parameters

Golog Operations: Reactive Programming

?-**new_demon(example, S, transform1, if_read, after, alter_value).**



?-**new_demon(example, S, transform2, if_write, before, alter_value).**



Golog Operations: Reactive Programming


```
?-new_frame(car).
?-new_slot(car, speed, 0).
?-new_slot(car, move, moveaction).
?-new_slot(car, stop, stopaction).

stopaction(F):- new_value(F, speed, 0), write('I stopped ! '), nl.
moveaction(F, V):- new_value(F, speed, V),
                   write('I am moving at '), write(V),
                   write(' Km/h'), nl.
```

```
?-new_slot(car, limit, 175).
?-new_demon(car, speed, police, if_write, before, side_effect).

police(F, _, V, V) :- get_value(F,limit,Vl), V =< Vl,!.
police(_, _, _, _)  :- write('Speed excess '), !, fail.
```

```
?- call_method_1(car,move,100).
I am moving at 100 Km/h
true.

?- get_value(car,speed,S).
S = 100.
```

In this case the police did not interfere (by rule 1)

```
?- call_method_1(car,move,200).
Speed excess
false.

?- get_value(car,speed,S).
S = 100
```

In this case the police interfered and did not allow the new speed (by rule 2)

Golog Operations: Reactive Programming

Write a demon that limits the speed to 120 Km/h (without aborting the program).

```
?-new_frame(car2).
?-new_slot(car2, speed, 0).
?-new_slot(car2, move, moveaction2).
?-new_slot(car2, stop, stopaction2).

stopaction2(F):- new_value(F, speed, 0), write('I stopped ! '), nl.
moveaction2(F, V):- new_value(F, speed, V), get_value(F,speed,RV),
                    write('I am moving at '), write(RV), write(' Km/h'), nl.
```

Notice that we read again the slot because we don't know if the demon interfered wit new_value.

```
?- call_method_1(car2,move,180).
I am moving at 120 Km/h
true.
```

!!!

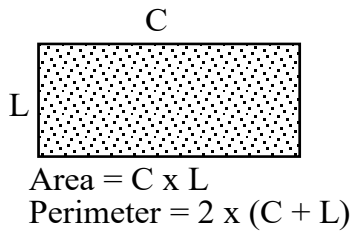
```
?- show_frame(car2).

    Frame: car2 {
      move: moveaction2
      speed: 120
        Demons:  limiter
      stop: stopaction2
    }

true.
```

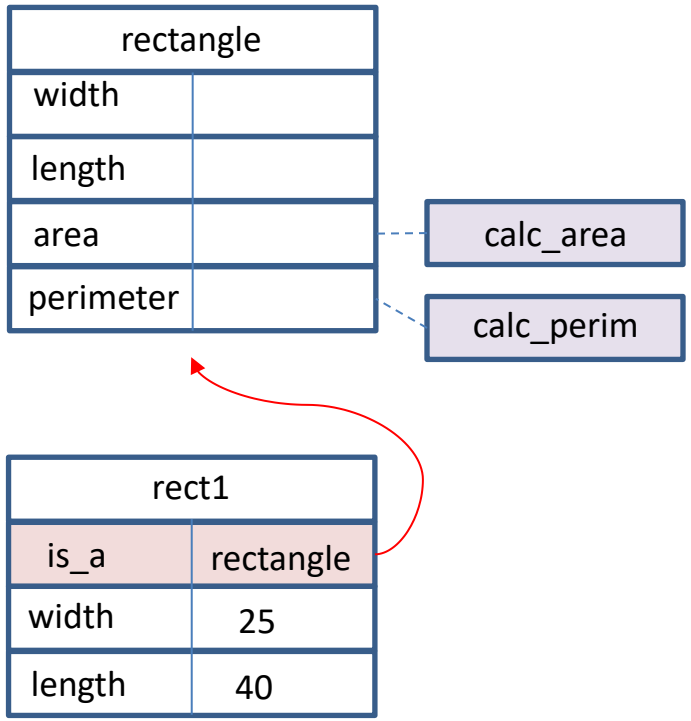
?-new_demon(car2, speed, limiter, if_write, before, alter_value).

```
limiter(_, _, V, V) :- V=< 120, !.
limiter(_, _, _, 120).
```

Golog Operations: Reactive Programming

Derived attributes:
dynamically calculated from other attributes.






```
?- new_relation(is_a,  transitive, all, nil).

?-new_frame(rectangle).
?-new_slot(rectangle, width).
?-new_slot(rectangle, length).
?-new_slot(rectangle,area).
?-new_slot(rectangle, perimeter).
?-new_frame(rect1).
?-new_slot(rect1, is_a, rectangle).
?-new_value(rect1, width, 25).
?-new_value(rect1, length, 40).
```

?-new_demon(rectangle, area, calc_area, if_read, after, alter_value).
?-new_demon(rectangle, perimeter, calc_perim, if_read, after, alter_value).

```
calc_area(F,_, _, A) :- get_value(F, width, L), get_value(F, length, C), A is C * L.
calc_perim(F,_, _, P):- get_value(F, width, L), get_value(F, length, C), P is 2 * (C + L).
```

```
?- get_value(rect1, area, A).
A = 1000

?- get_value(rect1, perimeter, B).
B = 130.
```

Golog Operations: Reactive Programming

Write a demon that forbids reducing the salary of an employee

```
?-new_frame(employee).
?-new_slot(employee,salary,0).
```

?-new_demon(employee, salary, protectsalary, if_write, before, side_effect).

```
protectsalary(F,S,V,V):-get_value(F,S,OldV), V>=OldV, !.
protectsalary(_,_,_,_):- write('It is illegal to reduce salary !!!'),fail.
```

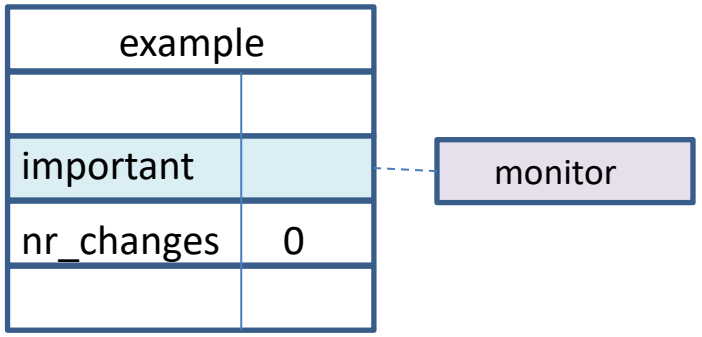
```
?- show_frame(employee).

    Frame: employee {
      salary: 0
        Demons:  protectsalary
    }

true.
```

```
?- new_value(employee, salary, 1000).
true

?- show_frame(employee).

    Frame: employee {
      salary: 1000
        Demons:  protectsalary
    }

true.
```

```
?- new_value(employee, salary, 500).
It is illegal to reduce salary !!!
false.

?- show_frame(employee).

    Frame: employee {
      salary: 1000
        Demons:  protectsalary
    }

true.
```


Golog Operations: Reactive Programming

Write a demon “monitor” that counts the number of times one given slot is changed (and stores this count in another slot).



?-new_frame(example).
?-new_slot(example,important).
?-new_slot(example,nr_changes,0).

?-**new_demon(example,important,monitor,if_write,after, side_effect).**

monitor(F,_,_,_):- get_value(F,nr_changes,N), N1 is N+1,
new_value(F, nr_changes, N1).

```
?- show_frame(example).

    Frame: example {
      important:
        Demons:  monitor
      nr_changes: 0
    }

true
```

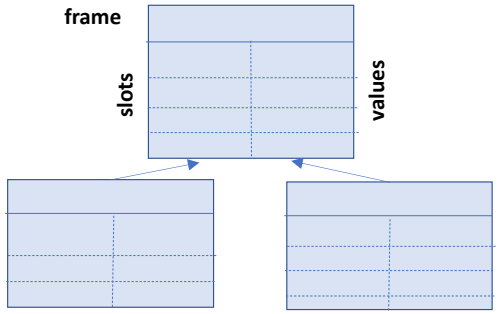
```
?- new_value(example, important, 5).
true.
```

```
?- show_frame(example).

    Frame: example {
      important: 5
        Demons:  monitor
      nr_changes: 1
    }

true.
```

```
?- add_value(example,important, 3).
true.
```

```
?- show_frame(example).

    Frame: example {
      important: 5 3
        Demons:  monitor
      nr_changes: 2
    }

true.
```

Golog Frame Engine Summary



A. On frames

- new_frame(F)
- frame_exists(F)
- show_frame(F)
- delete_frame(F)

B. On slots

- new_slot(F,S)
- new_slot(F,S,V)
- delete_slot(F,S)
- frame_local_slots(F,LS)
- get_all_slots(F,LS)

C. On values

- new_value(F,S,V)
- new_values(F,S,LV)
- add_value(F,S,V)
- add_values(F,S,LV)
- get_value(F,S,V)
- get_values(F,S,LV)
- delete_value(F,S,V)
- delete_values(F,S)

D. On relations

- new_relation(Relation,Transitivity, Restriction, Inverse)
- delete_relation(Relation)
- frame_actual_relations(F,LR)

E. On methods

- new_slot(F, S, Method)
- call_method(F, S, LPar)
- call_method_0(F, S)
- call_method_1(F, S, P)
- call_method_2(F, S, P1, P2)
- call_method_3(F, S, P1, P2, P3)

F. On attached predicates or demons

- new_demon(F,S,Demfunc, Demtype, When, Effect)
- add_demon(F,S,Demfunc, Demtype, When, Effect)
- remove_all_demons(F,S)

Golog Frame Engine

Some additional Functionalities

check

Golog2_2025.pl

- Mechanism to save the knowledge-base to a file

  ?- save_kb(FileName).

- Mechanism to show the actual knowledge-base

  ?-show_kb.

- Mechanism to delete the knowledge-base

  ?- delete_kb.

- Others…

```
% %%
% %%   Predicados Utilitarios
% %%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:-style_check(-singleton).
:-style_check(-discontiguous).

:-dynamic frame_/1.
:-dynamic relation_/1.

% http://stackoverflow.com/questions/130097/real-world-prolog-usage
%
%

save_kb(Filename):-
        tell(Filename),
        forall(show_kb,true),
        told.


show_kb:-
        (   (
          clause(frame_(_),true),
          listing(frame_),
```

Further reading

https://en.wikipedia.org/wiki/Frame_(artificial_intelligence)#Frame_language

https://www.cs.unm.edu/~luger/ai-final2/CH8_Natural%20Language%20Processing%20in%20Prolog.pdf

https://core.ac.uk/download/pdf/10675407.pdf

https://www.youtube.com/watch?v=y2HQmvqXON4

https://www.youtube.com/watch?v=V-O-RFSRe-E

https://drsmithbiology.weebly.com/further-reading.html

Good Work!

Ana Inês Oliveira

NOVA School of Sciences and Technology | FCT NOVA

aio@fct.unl.pt