Knowledge Representation and Reasoning

Stan Eduard George - Gr. 407

Key ideas

The clauses in a knowledge base can be divided in two categories:

- Facts – ground terms.
- Rules – conditionals that express new relations (production rules - IF conditions THEN actions).

  Negation, conjunction and disjunction of vague predicates:

$$\mu_{\neg P} = 1 - \mu_P$$
$$\mu_{P \wedge Q} = min(\mu_P, \mu_Q)$$
$$\mu_{P \vee Q} = max(\mu_P, \mu_Q)$$

1 Backward and Forward chaining

1.1 Rules

1. If a person is altruistic and has finished medical school, then they are a medic.
2. If a person is a medic and connects well with children, then they are a pediatrician.
3. If a person helps other people, then they are altruistic.
4. If a person is kind and playful, then they connect well with children.
5. If a person is empathetic, then they are kind.

1.2 Questions

1. Are you playful? (yes/no)
2. Are you empathetic? (yes/no)
3. Did you finish medical school? (yes/no)
4. Do you usually help other people? (yes/no)

1.3 Input

1. [n(is_altruistic), n(finished_medicine), is_medic]
2. [n(is_medic), n(connects_well_with_children), is_pediatrician]
3. [n(helps_people), is_altruistic]
4. [n(is_kind), n(is_playful), connects_well_with_children]
5. [n(is_empathetic), is_kind]

   The following atoms might be further added by asking the questions:
   [is_playful], [is_empathetic], [finished_medicine], [helps_people].

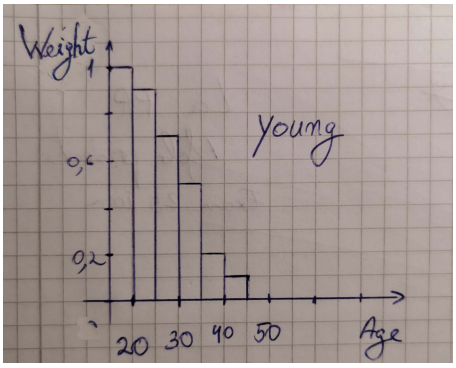
1.4 Definite Clause Grammar notation (DCG)

```
sentence --> if_statement, person_condition, [then], outcome.
if_statement --> [if, a, person].

person_condition --> is_altruistic, finished_medical_school.
person_condition --> is_medic, connects_well_with_children.
person_condition --> helps_others.
person_condition --> is_kind, is_playful.
person_condition --> is_empathetic.

outcome --> are_medic.
outcome --> pediatrician.
outcome --> are_altruistic.
outcome --> connect_well_with_children.
outcome --> are_kind.

is_altruistic --> [is, altruistic].
finished_medical_school --> [has, finished, medical, school].
is_medic --> [is, a, medic].
connects_well_with_children --> [connects, well, with, children].
helps_others --> [helps, other, people].
is_kind --> [is, kind].
is_playful --> [is, playful].
is_empathetic --> [is, empathetic].

are_medic --> [they, are, a, medic].
pediatrician --> [they, are, a, pediatrician].
are_altruistic --> [they, are, altruistic].
connect_well_with_children --> [connect, well, with, children].
are_kind --> [they, are, kind].
```

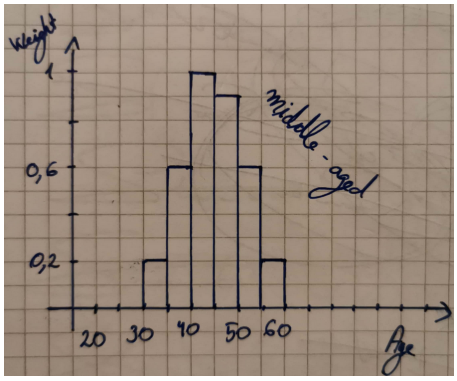
2 Defuzzification

2.1 Rules

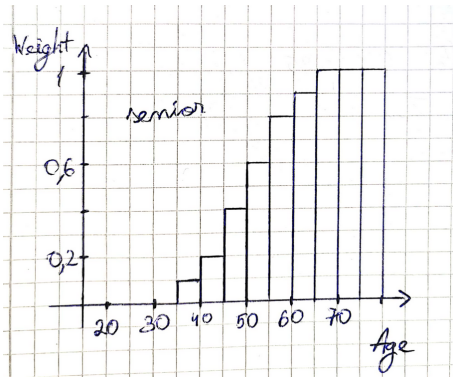
1. If the driver is senior or the car is new, then the insurance is cheap.
2. If the driver is middle-aged, then the insurance is mid-range.
3. If the driver is young or the car is old, then the insurance is expensive.



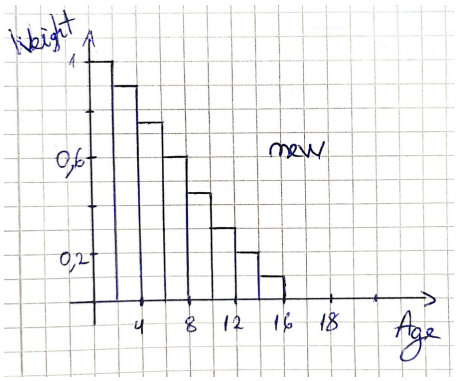
(a)



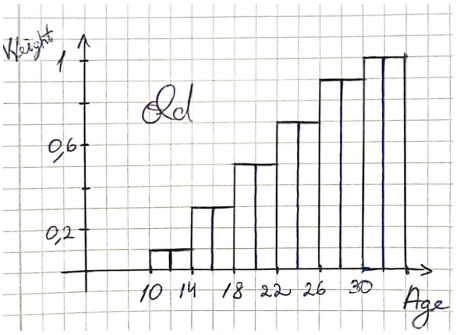
(b)



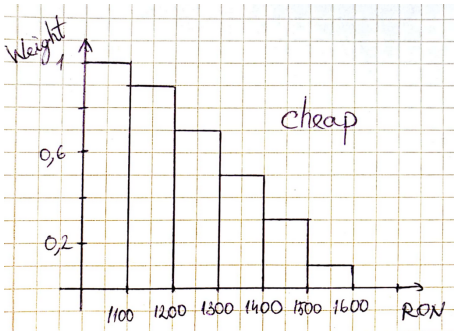
(c)



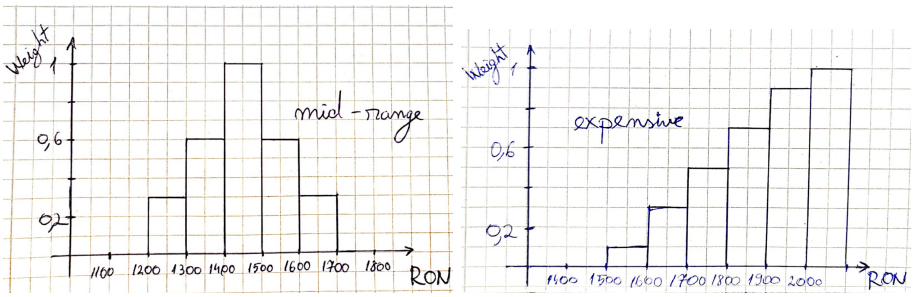
(d)



(e)



(f)



(g) (h)

Degree of curves

2.2 Input

1. [or, [driver/senior, car/new], cheap].
2. [and, [driver/middle_aged], mid_range].
3. [or, [driver/young, car/old], expensive]].

2.3 Definite Clause Grammar notation (DCG)

```
sentence --> [if], condition, [then], insurance_category.
condition --> driver_is_senior ; car_is_new.
condition --> driver_is_middle_aged.
condition --> driver_is_young ; car_is_old.

driver_is_young --> [the, driver, is, young].
driver_is_middle_aged --> [the, driver, is, middle_aged].
driver_is_senior --> [the, driver, is, senior].
car_is_new --> [the, car, is, new].
car_is_old --> [the, car, is, old].

insurance_category --> [the, insurance, is, cheap].
insurance_category --> [the, insurance, is, mid_range].
insurance_category --> [the, insurance, is, expensive].
```

2.4 Examples

How old are you? (number between 16-100) —: 70.
How old is your car? (number between 0-50) —: 25.
The insurance cost is 1432.29 RON.

How old are you? (number between 16-100) —: 24.
How old is your car? (number between 0-50) —: 14.
The insurance cost is 1766.67 RON.

How old are you? (number between 16-100) —: 45.
How old is your car? (number between 0-50) —: 2.
The insurance cost is 1158.57 RON.

3 Implementations

3.1 Backward Chaining

```
answer(yes).

afis_list([]):- nl.
afis_list([A|B]):-write(A), tab(2), afis_list(B).

concatenate([], L, L):- !.
concatenate([E|L1], L2, [E|L3]):- concatenate(L1, L2, L3).

create_clause([],[]).
create_clause([E1|L1], [E1/false|L2]):- create_clause(L1, L2).

questions(["Are you just a medic? (yes/no)" / is_medic,
           "Are you a pediatrician? (yes/no)" / is_pediatrician,
           "Do you connect well with children? (yes/no)" / connects_well_with_children]).

create_KB([],[]).
create_KB([L1|RawKB], [L2|NewKB]):- create_clause(L1, L2), create_KB(RawKB, NewKB).

update_clause(_, [], []).
update_clause(Atom, [Atom/false|RestL1], [Atom/true|RestL2]):- update_clause(Atom, RestL1, RestL2).
update_clause(Atom, [n(Atom)/false|RestL1], [n(Atom)/true|RestL2]):- update_clause(Atom, RestL1, RestL2).
update_clause(Atom, [C/A|RestL1], [C/A|RestL2]):- update_clause(Atom, RestL1, RestL2).

update_KB(_, [],[]).
update_KB(Atom, [L1|KB], [L2|NewKB]):- update_clause(Atom, L1, L2), update_KB(Atom, KB, NewKB).

update_more_atoms([], NewKB, NewKB).
update_more_atoms([Atom|RestAtoms], KB, NewKB):- update_KB(Atom, KB, TempKB1),
    update_KB(n(Atom), TempKB1, TempKB2), update_more_atoms(RestAtoms, TempKB2, NewKB).

make_all_true([], [], []).
make_all_true([n(E)/false|L1], [n(E)/true|L2], [E|RestChanged]):- make_all_true(L1, L2, RestChanged).
make_all_true([n(E)/true|L1], [n(E)/true|L2], RestChanged):- make_all_true(L1, L2, RestChanged).

check_for_relations_in_clause([E/false|L1], [E/false|L1], []):- !.
```

```
check_for_relations_in_clause([E/true|L1], [E/true|L2], Changed):- make_all_true(L1, L2, Changed).

check_for_relations([], [], []).
check_for_relations([L1|KB], [L2|NewKB], FutureChanged):- reverse(L1, RevL1),
        check_for_relations_in_clause(RevL1,RevL2,PresentChanged),
        concatenate(PresentChanged, PastChanged, FutureChanged),
        reverse(RevL2, L2), check_for_relations(KB, NewKB, PastChanged).

all_relations(KB, NewKB):- check_for_relations(KB, TempKB, Changed), (Changed == [],
       NewKB = TempKB; update_more_atoms(Changed, TempKB, AlmostNewKB),
       all_relations(AlmostNewKB, NewKB)).

iter_negatives([]).
iter_negatives([n(E)/true|SortedKB]):- write("- "), write_term(E, []), nl, iter_negatives(SortedKB).
iter_negatives([_|SortedKB]):- iter_negatives(SortedKB).

afis_unique(KB):- flatten(KB, FlatKB), sort(FlatKB, SortedKB), iter_negatives(SortedKB).

bkw(KB, Question/Atom):- write(Question), nl, read(Answer1), nl, nl,
        (
            answer(Answer1), update_KB(Atom, KB, TempKB), all_relations(TempKB, NewKB),
            write("Final KB:"), nl, afis_list(NewKB), nl, nl,
            write("You have the following qualities:"), nl,
            afis_unique(NewKB), nl, nl;
            write("Unsolved! Do you want to answer again to this question? (yes/no)"),
            nl, read(Answer2), nl,
            (
                answer(Answer2),
                bkw(KB, Question/Atom);
                nl, write("Okay. Your choice :D"), nl, nl
            )
        ).


main:- write("Based on your responses, I will determine what qualities you have."), nl,
       write("Press any key to continue."), nl, read(_), nl, nl,
       open('/home/edstan/Desktop/master_AI/krr/projects/project2/input1.txt', read, Stream),
       read_line_to_codes(Stream, KBcodes), KBcodes \= at_end_of_stream(Stream),
       read_term_from_codes(KBcodes, RawKB, []), close(Stream),
       create_KB(RawKB, KB), write("Initial KB:"), nl, afis_list(KB), nl,
       questions([Q1, Q2, Q3]), bkw(KB, Q1), bkw(KB, Q2), bkw(KB, Q3).
```

3.2 Forward Chaining

```
answer(yes).

afis_list([]):- nl.
afis_list([A|B]):-write(A), tab(2), afis_list(B).

create_clause([],[]).
create_clause([E1|L1], [E1/false|L2]):- create_clause(L1, L2).

create_KB([],[]).
create_KB([L1|RawKB], [L2|NewKB]):- create_clause(L1, L2), create_KB(RawKB, NewKB).

questions(["Are you a playful person? (yes/no)" / n(is_playful),
           "Are you empathetic? (yes/no)" / n(is_empathetic),
           "Did you finish medical school? (yes/no)" / n(finished_medicine),
           "Do you usually help other people? (yes/no)" / n(helps_people)]).

update_clause(_, [], []).
update_clause(Atom, [Atom/false|RestL1], [Atom/true|RestL2]):- update_clause(Atom, RestL1, RestL2).
update_clause(Atom, [C/A|RestL1], [C/A|RestL2]):- update_clause(Atom, RestL1, RestL2).

update_KB(_, [],[]).
update_KB(Atom, [L1|KB], [L2|NewKB]):- update_clause(Atom, L1, L2), update_KB(Atom, KB, NewKB).


check_for_relations_in_clause([], [], _):- !.
check_for_relations_in_clause([E/true|L1], [E/true|L2], Changed):-
        check_for_relations_in_clause(L1, L2, Changed).
check_for_relations_in_clause([n(E)/false|L1], [n(E)/false|L1], _Changed):- !.
check_for_relations_in_clause([E/false], [E/true], E):- !.

check_for_relations([], [], _).
check_for_relations([L1|KB], [L2|NewKB], Changed):- check_for_relations_in_clause(L1,L2,Changed),
       check_for_relations(KB, NewKB, Changed).

all_relations(KB, NewKB):- check_for_relations(KB, TempKB, Changed), (var(Changed),
       NewKB = TempKB; update_KB(n(Changed), TempKB, AlmostNewKB), all_relations(AlmostNewKB, NewKB)).

ask(NewKB, [], NewKB, NewQuestions, NewQuestions).
ask(KB, [Question/Atom|RestQuestions], LastKB, AddQuestion, LastQuestions):-
       (
              write(Question), nl, read(Answer), nl, nl,
```

```
                answer(Answer),
                update_KB(Atom, KB, TempKB), all_relations(TempKB, NewKB),
                ask(NewKB, RestQuestions, LastKB, AddQuestion, LastQuestions);
                ask(KB, RestQuestions, LastKB, [Question/Atom|AddQuestion], LastQuestions)
        ).

all_true([]).
all_true([_/true|NewKB]):- all_true(NewKB).

fwd(KB, Questions):- ask(KB, Questions, NewKB, [], NewQuestions),
        (
                flatten(NewKB, NewKBflat), not(all_true(NewKBflat)),
                (
                        write("Current KB:"), nl, afis_list(NewKB), nl,
                        write("Unsolved! You are not a pediatrician yet.
                        Do you want to continue answering questions? (yes/no)"),
                        nl, read(Answer), nl,
                        answer(Answer),
                        fwd(NewKB, NewQuestions);
                        nl, write("Okay. Your choice :D")
                );
                write("Final KB:"), nl, afis_list(NewKB), nl,
                write("You are a pediatrician!!")
        ).

main:- write("Based on your responses, I will determine wether you are a pediatrician."), nl,
        write("Press any key to continue."), nl, read(_), nl, nl,
        open('/home/edstan/Desktop/master_AI/krr/projects/project2/input1.txt', read, Stream),
        read_line_to_codes(Stream, KBcodes), KBcodes \= at_end_of_stream(Stream),
        read_term_from_codes(KBcodes, RawKB, []), close(Stream),
        create_KB(RawKB, KB), write("Initial KB:"),
        nl, afis_list(KB), nl, questions(Questions), fwd(KB, Questions).
```

3.3 Defuzzification

```
round_to_second_digit(Nr, Result) :-
    Result is round(Nr * 100) / 100.

afis_list([]).
afis_list([A|B]):-write(A), tab(2), afis_list(B).

vectorized_sum([[],[],[]],[]):- !.
vectorized_sum([[E1|L1], [E2|L2], [E3|L3]], [Efin|Lfin]):-
```

```
    Efin is E1+E2+E3, vectorized_sum([L1, L2, L3], Lfin).

young(Age, 1.0):- Age < 20, !.
young(Age, 0.9):- Age < 25, !.
young(Age, 0.7):- Age < 30, !.
young(Age, 0.5):- Age < 35, !.
young(Age, 0.2):- Age < 40, !.
young(Age, 0.1):- Age < 45, !.
young(_, 0):- !.

middle_aged(Age, 0.0) :- Age < 30, !.
middle_aged(Age, 0.2) :- Age < 35, !.
middle_aged(Age, 0.6) :- Age < 40, !.
middle_aged(Age, 1.0) :- Age < 45, !.
middle_aged(Age, 0.9) :- Age < 50, !.
middle_aged(Age, 0.6) :- Age < 55, !.
middle_aged(Age, 0.2) :- Age < 60, !.
middle_aged(_, 0) :- !.

senior(Age, 1.0):- Age > 65, !.
senior(Age, 0.9):- Age > 60, !.
senior(Age, 0.8):- Age > 55, !.
senior(Age, 0.6):- Age > 50, !.
senior(Age, 0.4):- Age > 45, !.
senior(Age, 0.2):- Age > 40, !.
senior(Age, 0.1):- Age > 35, !.
senior(_, 0):- !.

new(Age,  1.0):- Age < 2 , !.
new(Age,  0.9):- Age < 4 , !.
new(Age, 0.75):- Age < 6 , !.
new(Age,  0.6):- Age < 8 , !.
new(Age, 0.45):- Age < 10, !.
new(Age,  0.3):- Age < 12, !.
new(Age,  0.2):- Age < 14, !.
new(Age,  0.1):- Age < 16, !.
new(_, 0):- !.

old(Age, 1.0):- Age > 30, !.
old(Age, 0.9):- Age > 26, !.
old(Age, 0.7):- Age > 22, !.
old(Age, 0.5):- Age > 18, !.
```

```
old(Age, 0.3):- Age > 14, !.
old(Age, 0.1):- Age > 10, !.
old(_, 0):- !.

cheap(Price, 1.0) :- Price < 1000, !.
cheap(Price, 0.9) :- Price < 1100, !.
cheap(Price, 0.7) :- Price < 1200, !.
cheap(Price, 0.5) :- Price < 1300, !.
cheap(Price, 0.3) :- Price < 1400, !.
cheap(Price, 0.1) :- Price < 1500, !.
cheap(_, 0) :- !.

mid_range(Price, 0.0) :- Price < 1200, !.
mid_range(Price, 0.3) :- Price < 1300, !.
mid_range(Price, 0.6) :- Price < 1400, !.
mid_range(Price, 1.0) :- Price < 1500, !.
mid_range(Price, 0.6) :- Price < 1600, !.
mid_range(Price, 0.3) :- Price < 1700, !.
mid_range(_, 0) :- !.

expensive(Price, 0.1) :- Price < 1600, !.
expensive(Price, 0.3) :- Price < 1700, !.
expensive(Price, 0.5) :- Price < 1800, !.
expensive(Price, 0.7) :- Price < 1900, !.
expensive(Price, 0.9) :- Price < 2000, !.
expensive(Price, 1) :-   Price < 2300, !.

price_interval([800, 900, 1000, 1100, 1200, 1300, 1
        400, 1500, 1600, 1700, 1800, 1900, 2000, 2100, 2200]).

weights_for_interval(_, _, [], []):- !.
weights_for_interval(PriceDegree, ReferenceWeight, [Price|RestPriceInterval], [Weight|RestWeight]):-
    call(PriceDegree, Price, PriceWeight), PriceWeight =< ReferenceWeight,
    Weight is PriceWeight,
    weights_for_interval(PriceDegree, ReferenceWeight, RestPriceInterval, RestWeight);
    Weight is ReferenceWeight,
    weights_for_interval(PriceDegree, ReferenceWeight, RestPriceInterval, RestWeight).

and(WeightsList, Result):- min_list(WeightsList, Result).
or(WeightsList, Result):- max_list(WeightsList, Result).

compute_weights(_, [], []):- !.
```

```
compute_weights([Age|RestAges], [_/Degree|RestDegrees], [Weight|RestWeights]):-
    call(Degree, Age, Weight), compute_weights(RestAges, RestDegrees, RestWeights).

compute_area_under_curve(DriverAge, CarAge, [LogicalOperator, MembershipsList, PriceDegree], Area):-
    compute_weights([DriverAge, CarAge], MembershipsList, WeightsList),
    call(LogicalOperator, WeightsList, PriceWeight),
    price_interval(Interval),
    weights_for_interval(PriceDegree, PriceWeight, Interval, Area).


iter_rules(_,_,[],[]):- !.
iter_rules(DriverAge, CarAge, [Rule|RestRules], [Area|RestAreas]):-
    compute_area_under_curve(DriverAge, CarAge, Rule, Area),
    iter_rules(DriverAge, CarAge, RestRules, RestAreas).

weighted_mean([], [], []):- !.
weighted_mean([Weight|BigArea], [Price|Interval], [Result|RestResult]):-
    Result is Weight * Price, weighted_mean(BigArea, Interval, RestResult).


main_func:- write("How old are you? (number between 16-100)"), nl, read(DriverAge), nl,
       write("How old is your car? (number between 0-50)"), nl, read(CarAge), nl,

       open('/home/edstan/Desktop/master_AI/krr/projects/project2/input2.txt', read, Stream),
       read_line_to_codes(Stream, RulesCodes), RulesCodes \= at_end_of_stream(Stream),
       read_term_from_codes(RulesCodes, Rules, []), close(Stream),
       %afis_list(Rules),
       iter_rules(DriverAge, CarAge, Rules, Areas),
       %afis_list(Areas),
       vectorized_sum(Areas, BigArea), price_interval(Interval),
       weighted_mean(BigArea, Interval, WeightedPrices), sum_list(BigArea, BigWeight),
       sum_list(WeightedPrices, BigPrice), FinalPrice is BigPrice / BigWeight,
       round_to_second_digit(FinalPrice, RoundedFinalPrice),
       write("The insurance cost is "), write(RoundedFinalPrice), write(" RON.").
```