CSE579 Individual Project Report

Written by Thien Tran*

Arizona State University
Tempe
AZ 85281, United States
tptran2@asu.edu

Problem statement

"Students are tasked to incorporate the knowledge learned in CSE 579 to apply to a real world problem. They have to build a knowledge base system for either the automated warehouse scenario or the insurance referee assignment project. To help students to achieve the core competencies of the project, the course provides a guidance system via a series of intermittent milestones and mini-assignments to build up core competencies of the students. Upon completing the project, students will be equipped fully with the necessary tools and devices at disposal to tackle real world challenges in the knowledge representation domain."[1]

Project background (e.g., basic knowledge of ASP, external online materials that helped in your project)

The fulfillment of the project requires 2 main components: core compentencies and complements. Core competencies are those that are necessary and sufficient to achieve the Minimum Viable Product (MVP) concept, also known as the acceptance criteria. The complements are nice-to-have features.

The core competencies can be broken down to the followings:

- Logic, reasoning and representation- Natural deduction,first order logic.
- Combinatorial search - Answer set programming (ASP).
- Knowledge representation: Frame problem, Soft constraints, Bayesian network, LPMLN.

Complements:

- Knowledge graph[2]
- PDDL[3]
- ASP action states[4]
- Project management[5]

Your approach to solving the problem

As previously discussed, the agile approach is utilized to provide the solution. Unlike any other problem. an optimization problem can be seen as going through the search space to find a good-enough solution which does not need to be the most optimal one, though ideally it should be. A greedy search is performed at the local level to find the optimal solution. This means that a latter solution is guaranteed to be better than the former one. This strategy is also aligned with the Dr Lee's generate-define-test approach[4] in the sense that adding more constraints at each level help the project closer to the optimal solution.

The project is divided into 4 iterative phases:

- Generate/define/test search space.
- Define/Test payment constraints.
- Define/Test region/preference constraints.
- Define/Test workload constraints.

Main results and analysis

Generate search space

Problem formulation: In order to define a search space, the project follows the four step approach: Problem formulation, problem representation, output, solution[6]. The insurance problem can be roughly summarized that whenever an accident happens, the clients report the claims. The insurance agency sends referee to investigate the claim. The project goal is to minimize the operating expenses which are impacted by mainly five different subjects: referee, case, preference, region and damage. Each of these subject has different properties with variable weights. **Problem representation:** These subjects are modeled using the RDF-like graph[7](Appendix A). These variables in the RDF graph represent the initial state. The next step is to define the goal state of the solution. A solution S is deemed satisfiable if it satisfies the hard-constraints. The ideal solution S* is a solution whose cost function is less than that of any other solution. It is defined as in:

Hard-constraints implies S.
S* = Argmin(S*) Cost(S*)
Output:

Assign(Cid,Rid): We generate all the possible combinations of the referees and cases in the universe of the insurance referee projects. The assign function is the mapping function between the referees and the cases. At first, it has been assumed that each referee can only be assigned to one case at a time, but according to the test cases, their cardinality should be that each referee can be assigned zero, one or more cases but each case can only be assigned to one referee. Thus, this is defined as a function where the cases is in the domain, and the referees are in the codomain[8]. In other words, for every case in the domain, there is exactly one referee in the codomain.[8]

Solution: To find the solution, hard-constraints and soft-constraints are translated from English into the first order logic (Appendix B,C). These represent the variables of the goal states. Once the initial state and the final goal state are realized, note that this is the frame problem[3][6]. The backward search and forward search are performed simultaneously to find the functions needed to go from the start to the goal (Appendix B,C). According to the graph, while some of the variables can reach the goal without any intermediary step, the others require auxiliary functions.

Define/Test payment constraints

The payment constraints consist of 1 hard constraints and 2 soft constraints. While the maximum effort hard constraint and the total payment to the external referees soft constraint (cA) are straight forward, only requires at most 2 auxiliary function. The individual external payment penalty cB is by far the most tedious constraints as they require many auxiliary functions.

Payment(Cid,Rid,Type,Payment)(Appendix C): is a function that assigns payment from each case to the assigned external referee.

newEffort(Cid,Rid,Effort): when a case is assigned to a referee, the effort of the case is recorded to calculate the workload constraint.

totalEff(Rid,Effort1): since possibly there are more than one cases that can be assigned to a referee each day, this function calculates the total effort of each referee for a given day.

orid(Rid,e,SumOrid): According to the requirement, it is the sum of payments made for the total case and the previous payment of each referee.

sumOrid(SumO): the sum of all the orid tuples. **numExternals(N**: the number of all the externals.

average_payment_external(Avg): is the average of the total payment made to all of the external referees.

divergenceCost(Rid,e,(SumD-Avg)): For each external referee, the divergenceCost represents the deviation from the average payment made.

Define/Test preference/region constraints

These are straight forward from the graph. They are the combination of preference/region, case and referee functions. **prefCost(Rid,Cost)**: For each case assign to a referee, the cost is the difference between three and the amount of preference assigned.

Define/Test workload constraints

workLoad(Rid,Type,Workload): The total workload changes whenever a new case is assigned to a referee. This function ensures that the new amount accounts for the new cases.

sumOfWorkLoad(Sum): Represents the total amount of workload assigned to the all referees.

numOfRef(N): Is the total number of referees.

averageWorkLoad(To): Is the average amount of workload when divided by the number of referees.

Conclusion

The project passes all the test cases provided in the description file, except for test case six. Although the assignments of referees are not the most optimal ones, they are good enough to save cost for the company. If the project had used the #sum function to calculate the aggregated cost, the result would have been more optimal; this means that the algorithm would exponentially consume more memories and computational power. Even if there were hundreds of referees each in a relatively medium firm, the computational amount would be massive. As with all the optimization problem, there is trade off between performance and result.

Opportunities for future work

I used various methods to my Quality Assurance line of work. Recently, I have learned about text-driven software testing using Artificial Intelligence. The method directly translates English text into test cases. It is a culmination of AI, machine learning and natural language processing. This is why I wanted to take CSE579. After learning the concepts in CSE579, I have an idea of how to use the description logics to translate business requirements and ASP to find flaws in the components. Furthermore, in my business courses, the scheduling problem has always been a big problem. While we used different mathematical models along with lengthy lines of reasoning to manage thousands of the resources each week, this can be done in a few line of code.

References

[1] T. Tran, "CSE579 Project Progress Report", Arizona State University, Tempe, 2022.

[2] J. Lee, "Knowledge Graph", Arizona State University, Tempe, 2022.

[3] S. Srivastava, "Automated Search Control", Arizona State University, Tempe, 2021.

[4] J. Lee, "Methodology of ASP", Arizona State University, Tempe, 2022.

[5] PMI, A guide to the project management body of knowledge. Project Management Institute, Inc., 2017.

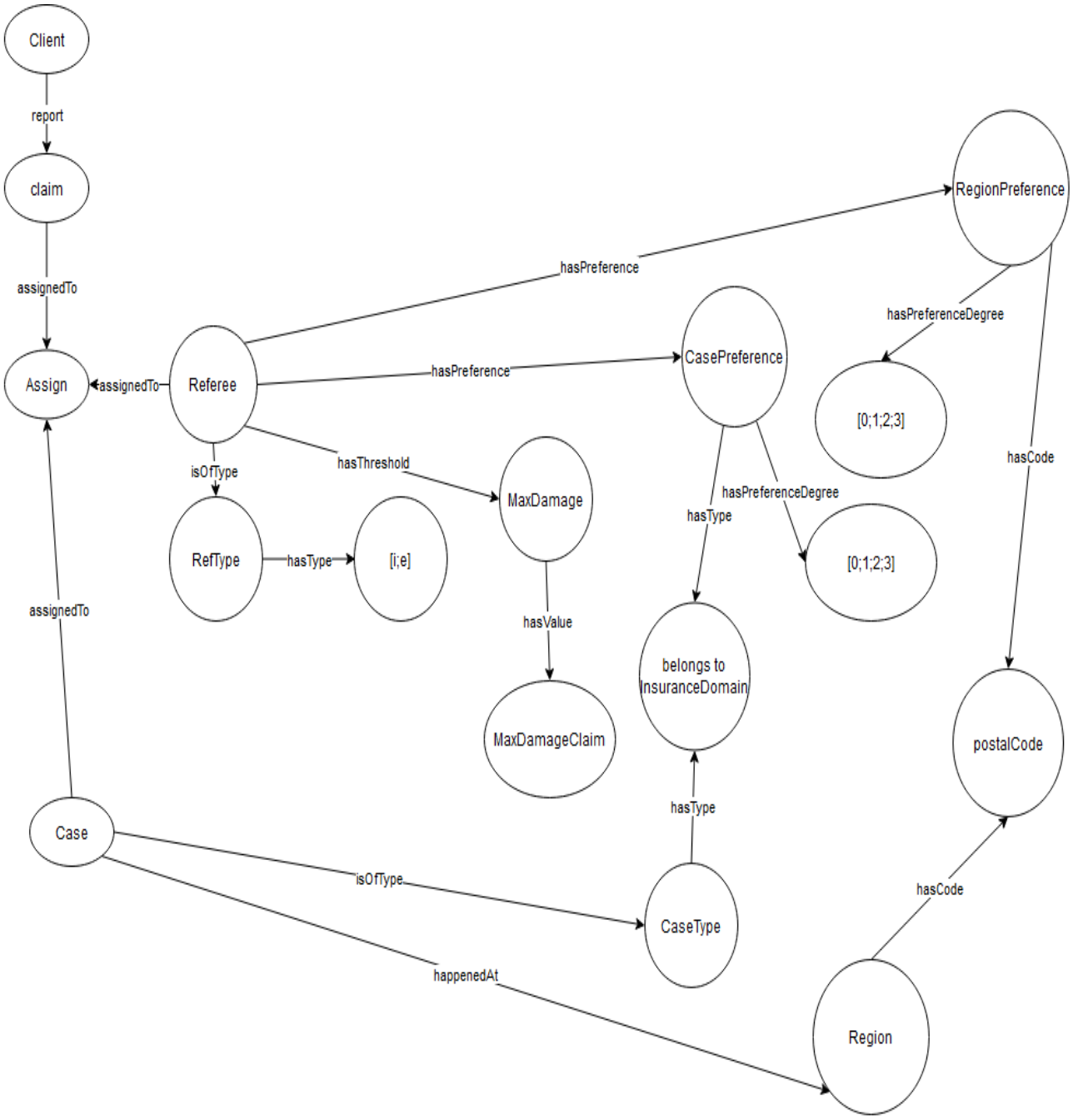
[6] J. Lee, "Introduction to Answer Set Programming", Arizona State University, Tempe, 2022.

[7] F. Iacobelli, "FOL(First Order Logic)," in youtube, 2021.

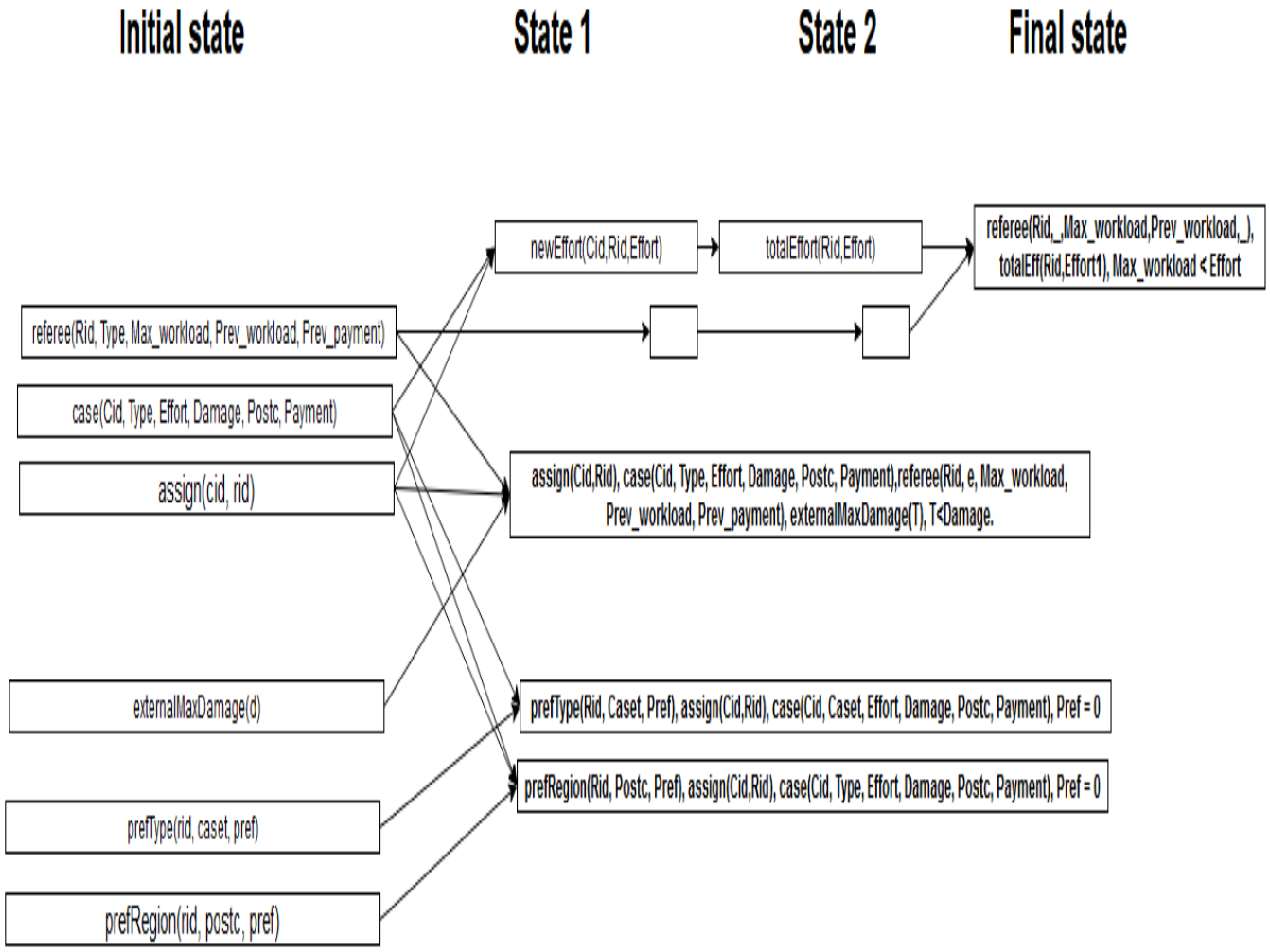
[8] J. Lee, "Representing Functions in ASP", Arizona State University, Tempe, 2022.

Appendices

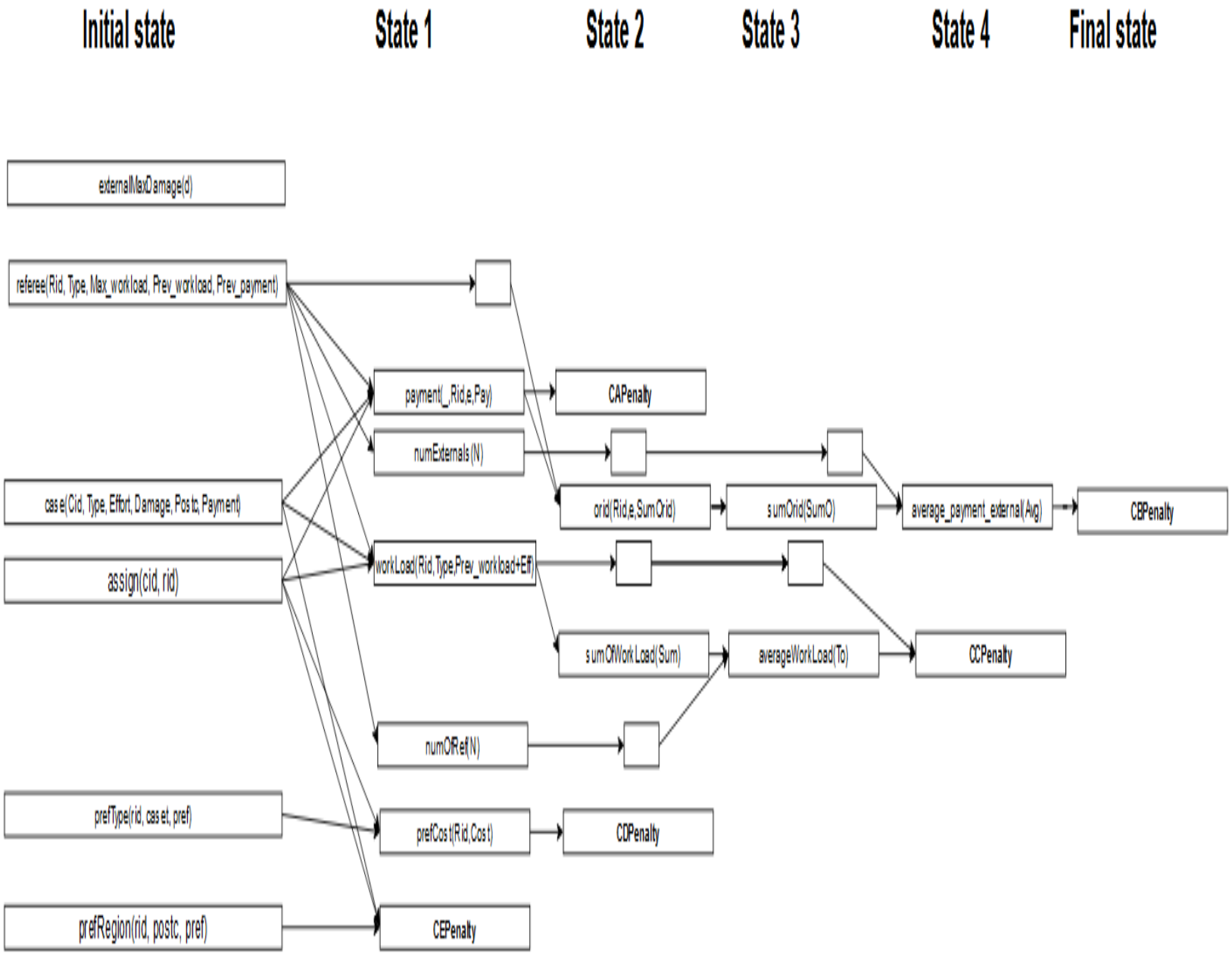
Appendix A: Insurance Universe



Appendix B : Hard-Constraints



Appendix C: Soft constraints



Appendix D: Code

```
%Generate

%referee(Rid, Type, Max_workload, Prev_workload, Prev_payment).

%case(Cid, Type, Effort, Damage, Postc, Payment).

%output

%Define domain and codomain of the assign

{assign(Cid,Rid):referee(Rid,_,_,_,_)}=1:-case(Cid,_,_,_,_,_).



%:- assign(Cid,Rid1), assign(Cid,Rid2), Rid1!=Rid2 .

newEffort(Cid,Rid,Effort):- assign(Cid,Rid), case(Cid,_,Effort,_,_,_).

%totalEff(Rid,Effort):- newEffort(Cid,Rid,Effort).

%totalEff(Rid,Effort):- newEffort(Cid1,Rid,Effort1), newEffort(Cid2,Rid,Effort2), Cid1 != Cid2, Effort =
Effort1+Effort2.

%#show totalEff/2.

totalEff(Rid,Effort1):- Effort1 = #sum{Effort,Cid,Rid: newEffort(Cid,Rid,Effort)}, referee(Rid, Type,
Max_workload, Prev_workload, Prev_payment).

%#show totalEff/2.

%payment

%{payment(Cid,Rid,Type,Payment): case(Cid,_,_,_,_,Payment)}=1 :- assign(Cid,Rid),
referee(Rid,Type,_,_,_).

{payment(Cid,Rid,e,Payment): case(Cid,_,_,_,_,Payment)}=1 :- assign(Cid,Rid), referee(Rid,e,_,_,_).

%:- payment(Cid,Rid,Type,Payment),payment(Cid1,Rid,Type,Payment),Cid!=Cid1.

%:- {payment(Cid,Rid,Type,Payment): assign(Cid,Rid)} =0, case(Cid,_,_,_,_,Payment).
```

%:- payment(_,_,X,_), X=e.

%%%% Hard Constraint %%%%%

%:~ referee(Rid,_,Max_workload,Prev_workload,_), totalEff(Rid,Effort1), Temp = Prev_workload + Effort1, Max_workload < Temp.[1@1]

%:- referee(Rid,_,Max_workload,Prev_workload,_), totalEff(Rid,Effort1), Temp = Prev_workload + Effort1, Max_workload < Temp.

:- referee(Rid,_,Max_workload,Prev_workload,_), totalEff(Rid,Effort1), Temp = Effort1, Max_workload < Temp.

%#show referee/5.

%#show totalEff/2.

%maxWL(Rid,Max_workload,Temp) :- referee(Rid,_,Max_workload,Prev_workload,_), totalEff(Rid,Effort1), Temp = Prev_workload + Effort1.

%#show maxWL/3.

%prefRegion(Rid, Postc, Pref).

%:~ prefRegion(Rid, Postc, Pref), assign(Cid,Rid), case(Cid, Type, Effort, Damage, Postc, Payment), Pref = 0. [1@1]

:- prefRegion(Rid, Postc, Pref), assign(Cid,Rid), case(Cid, Type, Effort, Damage, Postc, Payment), Pref = 0.

%prefType(Rid, Caset, Pref).

%:~ prefType(Rid, Caset, Pref), assign(Cid,Rid), case(Cid, Caset, Effort, Damage, Postc, Payment), Pref = 0. [1@1]

:- prefType(Rid, Caset, Pref), assign(Cid,Rid), case(Cid, Caset, Effort, Damage, Postc, Payment), Pref = 0.

%:~ assign(Cid,Rid), case(Cid, Type, Effort, Damage, Postc, Payment),referee(Rid, e, Max_workload, Prev_workload, Prev_payment), externalMaxDamage(T), T<Damage. [1@1]

:- assign(Cid,Rid), case(Cid, Type, Effort, Damage, Postc, Payment),referee(Rid, e, Max_workload, Prev_workload, Prev_payment), externalMaxDamage(T), T<Damage.

%totalEff(Rid,Effort1):- Effort1 = #sum{Effort,Cid,Rid: newEffort(Cid,Rid,Effort)}, referee(Rid, Type, Max_workload, Prev_workload, Prev_payment).

%newDamage(Cid,Rid,Dmg):- assign(Cid,Rid), case(Cid,_,_,Dmg,_,_).

%totalDamage(Rid,Dmg):- Dmg = #sum{Damage,Cid,Rid: newDamage(Cid,Rid,Damage)}, referee(Rid, Type, Max_workload, Prev_workload, Prev_payment).

%:- referee(Rid,e,Max_workload,Prev_workload,_), totalDamage(Rid,Dmg), externalMaxDamage(Max),Dmg>=Max.

%%%%%%%%%%%%%%%%%%%%%%%% Soft Constraints %%%%%%%%%%%%%%%%%%%%%%

%paymentToExternals(Total) :- Total = #sum{Pay,Rid,e: payment(_,Rid,e,Pay)}.

:~ Total = #sum{Pay,Rid,e: payment(_,Rid,e,Pay)}, CAPenalty =16*Total. [CAPenalty@0,1]

orid(Rid,e,SumOrid) :- referee(Rid, e, _, _, Prev),payment(Cid,Rid,e,Payment), SumOrid=Prev+Payment.

%sumOfPay(Rid,e,Sum):- referee(Rid, e, _, _, Prev), Sum = #sum{Payment,Rid:payment(Cid,Rid,e,Payment)}.

%orid(Rid,e,SumOrid) :- referee(Rid, e, _, _, Prev),sumOfPay(Rid,e,Sum), SumOrid=Prev+Sum.

%orid(Rid,e,SumOrid) :- referee(Rid, e, _, _, Prev),Pay= #sum{Payment,Rid:payment(Cid,Rid,e,Payment)}, SumOrid=Prev+Pay.

sumOrid(SumO) :- SumO= #sum{SumA,Rid: orid(Rid,e,SumA)}.

```
numExternals(N):- N= #count{Rid,e: referee(Rid,e,_,_,_)},N>0.

% For your average payment external, write the A and B in separate line.
average_payment_external(Avg):- sumOrid(A),numExternals(B), Avg=A/B.




divergenceCost(Rid,e,|SumD-Avg|) :- orid(Rid,e,SumD), average_payment_external(Avg).

:~ orid(Rid,e,SumD), average_payment_external(Avg), CBPenalty = 7*(|SumD -Avg|). [CBPenalty@0,2]



workLoad(Rid,Type,Prev_workload+Eff):-
referee(Rid,Type,_,Prev_workload,_),assign(Cid,Rid),case(Cid,_,Eff,_,_,_).
sumOfWorkLoad(Sum) :- Sum = #sum{Wl,Rid: workLoad(Rid,Type,Wl)}.
numOfRef(N) :- N = #count{Rid: referee(Rid,_,_,_,_)}.
averageWorkLoad(To) :- sumOfWorkLoad(Sum), numOfRef(N), To=Sum/N.

%costWL(Rid,|Wl-To|) :- workLoad(Rid,Type,Wl), averageWorkLoad(To).

:~ CCPenalty = 9 * |Wl-To|,workLoad(Rid,Type,Wl),averageWorkLoad(To). [CCPenalty@0,3]

%sumCostWL(S) :- S=#sum{W,Rid: costWL(Rid,W)}.

%#minimize{W,Rid: costWL(Rid,W)}.
```

```
%%%%% preference constraints %%%%%

%case(3, b, 120, 800, 1190, 31).

%prefType(1, a, 1).

%referee(1, i, 600, 220, 0).

% Hard constraint

:- prefType(Rid, Type, 0), case(Cid, Type,_, _, _, _), assign(Cid,Rid).

% computation

prefCost(Rid,Cost) :- prefType(Rid, Type, Pref), assign(Cid,Rid), Cost=3-Pref, case(Cid, Type,_, _, _, _).

%sumPrefCost(Cost):- Cost=#sum{C,Rid: prefCost(Rid,C) }.

:~ CDPenalty = 34 * Cost, Cost =#sum{C,Rid: prefCost(Rid,C) }. [CDPenalty,4]


%

%#show sumPrefCost/1.

%#show assign/2.

%#show prefCost/2.


%%%%%% Region Constraints %%%%%%

%case(Cid, Type, Effort, Damage, Postc, Payment).

%prefRegion(Rid, Postc, Pref)

:- prefRegion(Rid,Postc,0), case(Cid, Type, Effort, Damage, Postc, Payment), assign(Cid,Rid).

%prefRegionCost(Rid,Cost) :- prefRegion(Rid,Postc,Pref), assign(Cid,Rid),case(Cid, Type, Effort, Damage,
Postc, Payment) , Cost = 3-Pref.

:~ CEPenalty = 34 * Cost, prefRegion(Rid,Postc,Pref), assign(Cid,Rid),case(Cid, Type, Effort, Damage,
Postc, Payment) , Cost = 3-Pref. [CEPenalty,5]




%CATotal(CAPenalty):- Total = #sum{Pay,Rid,e: payment(_,Rid,e,Pay)}, CAPenalty =16*Total.
```

%cATotal(Total*16):- Total = #sum{Pay,Rid,e: payment(_,Rid,e,Pay)}.

%cBTotal(

%cDTotal(Cost*34):- Cost =#sum{C,Rid: prefCost(Rid,C) }.

%div(Rid,e, CBPenalty) :- orid(Rid,e,SumD), average_payment_external(Avg), CBPenalty = 7*(|SumD - Avg|).

%cBTotal(Total) :- Total = #sum{CBPenalty,Rid:div(Rid,e, CBPenalty)}.

%:~ orid(Rid,e,SumD), average_payment_external(Avg), CBPenalty = 7*(|SumD -Avg|). [CBPenalty@0]