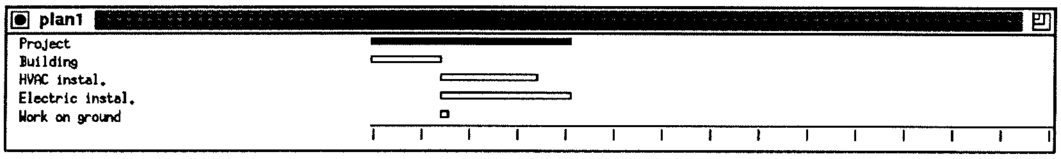
precondition `when/2`, where the starting time of the activities, based on the global state, are set equal to the activities' *first possible* starting time), there is just one plan, which is the plan presented above.

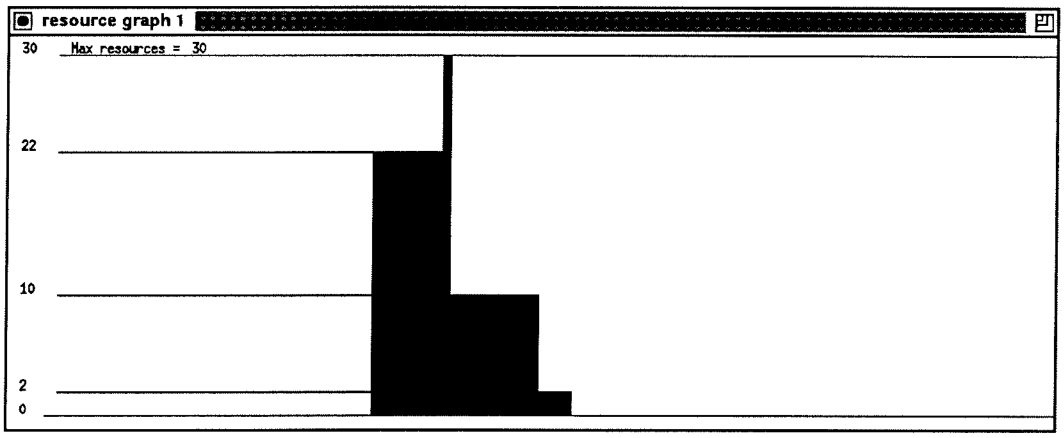
9.6.5 Priority Among Activities

It is easy to specify different priorities among the activities to implement different scheduling strategies. A simple example is to extend the top level strategy for the scheduling phase with a call to a priority relation, which specifies some ordering among the activities.

```
priority(akt(1,_,_)).
priority(akt(X,_,_)) :- X \== 1.

schedule <=
        (ready_sch <- true),          % All activities scheduled?
        (((priority(Act) ->
           search(akt(_,_,_),I,
             d_left(Act,I,
              a_left(_,I,
               weak_all(akt(_,_,_),
                weak_all(performed(_,_,_,_),schedule_pre)),

               a_left(_,I,
                weak_all(state(_,_,_),
                 weak_all(akt(_,_,_),schedule_calc)),

                  v_left_all(schedule_post((A \- C)))))))))
        <- (schedule -> (A \- C))),
        false).
```

Note that the only extension to the original schedule strategy is the call to `priority(Act)`, and that `Act` is passed to the first `d_left/3` inference rule call. Compare the following plan and resource graph with the first ones in section 9.6.2:





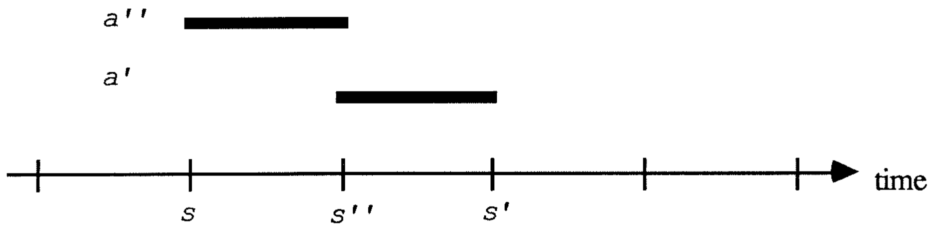
We can note that the activity `Work on ground` has got the maximum allocation of resources (10 teams with 2 persons in each), and that the other activities in parallel with `Work on ground` have to manage with the resources that are left. Notably, the activity `Electric installation` has got only one team allocated.

10. Discussion

We have described a system which spans the set of all possible plans given a design. The system has no expert knowledge about what plans are good and what plans are poor, and which method to apply in a given situation. But the system has expert knowledge about what different systematical ways there are to solve a complex task. The next step would be to add expert knowledge about what plans should be preferred over other plans, for example, if there are several methods to choose among, the system should be able to pick one which seems better based on the overall information. As for the described system, this additional information should be editable by the user, to reflect his personal knowledge and preferences. There are different ways to do that, but one promising way is to use decision theory, which is used to calculate a priority when there exist choices, for example resource allocation to competing activities. By this an ordering of the possible plans can be imposed, which in turn guides both the method choosing algorithm and the scheduling algorithm. Whether it is possible to find priority functions for preference of methods and resource allocation, and if the expressiveness of them is enough for experts in the field, is not clear, but to us it seems the most promising approach.

The basis of the planning theory described here is very basic. It should be possible to generalize it to other domains as well. The basic entities of our planning theory are *activities* which are applied to and change a *global state* when *resources* are allocated to it. A *scheduler* reasons about when and with what resources different activities are allocated, as soon as the activities are known. To generate the activities, *methods* are used to systematically group activities, that solve a more complex task. The overall goal is expressed as the design of a building, representing a state change. We hope to be able to apply these ideas to some other domain as well, to see what parts are general and what are parts are specific for this domain.

There is one known logical deficiency that ought to be mentioned. The application described here lacks the possibility to add constraints on the global state. For example, assume that there is one activity a' changing the state into a state s', and then an activity a'' that changes the state *before* state s' into a state s'', from which s' cannot be reached. This means that s' must contain the information that all states before s' must satisfy some constraints that should not violate it.



It is possible to order the activities for the scheduling phase by imposing conditions in the preconditions, that cause such critical activities as a' to be performed after a'', when the state s'' is known. This is quite natural, since one often delays an activity a' that is known to be dependent on another activity a'' until its duration and place in time are calculated.

To be able to manage the search space, the user must have ways to guide the generation of plans to obtain plans he is interested in. Some such guiding possibilities have been sketched in the paper, e.g. partially specifying what plans the choice-of-method phase should produce, interaction with the user to fill in values in the preparation phase between the choice-of-method phase and the scheduling phase, and one could also add user interaction in the scheduling phase when there are several ways to solve the produced constraints. The user interaction could take place in several ways, not only as events generated by the program as sketched in section 9.6.4, but also give the user possibilities to change the control level, as suggested in section 9.6.5. This gives him the possibility to write priority functions for different kinds of situations, the way he himself prioritizes some solution over another.

Since the programmer has got hold of the object level structure in each deduction step, if he wants to, it is easy to extend and manipulate the search behaviour in GCLA. The programmer is not forced to specify for each step what is to be done, but he has the opportunity. As he gathers more knowledge about what to do and when to do it, only the control level needs to be extended, as it is completely separated from the declarative definition. Since it is easy to build modular search strategies, the programmer can change one particular search strategy without affecting the rest, thus preserving the overall structure. To the programmers help there are also tools for diagnosing the behaviour of an application (see the performance package of GCLA in [Aro91a]).

There is a difference of a factor of 5 to 10 in execution time between this application and a Prolog application of the same system, in favour of the Prolog application. In GCLA's favour it must be said that we feel that we have a clearer understanding of what we are doing in GCLA. It is easier to change parts of the system, or to add some further functionality to the system. Also, we feel that the clear distinction between declarative knowledge, in this case methods and activities, and procedural knowledge, here how to plan using the methods and activities, has increased the ability for us to reason about what is declarative during the development of the application, and what the interesting possibilities there are to use that declarative knowledge. For example, the ability to reason about all objects satisfying a partially instantiated term through `a(Var)`-terms got its solution when we turned from Prolog to GCLA. In Prolog we did not have the underlying framework to sort things out, while in GCLA the way of thinking about those partially instantiated terms as a partial inductive definition helped us to sort the different problems out. This is not to say that the application cannot be implemented in *any* language (Prolog in particular since GCLA is implemented on top of Prolog), but it is to say that the higher level of the GCLA language helped us to reason on a higher level, which was what was needed in order to understand the nature and solutions of the high level problems.

Note that apart from arithmetic rules and some simple new rules, the whole control part consists of strategies, and thus is a 'clean' GCLA program, without any obscure things done hidden in new, very specialized rules. We think that one should try to stick to the general GCLA rules as much as possible until one knows what one is doing, i.e. has understood the problem at hand, and then, gain efficiency by introducing more specialized rules. For the application described in this paper, we are convinced that efficiency could be increased by some new rules, which would implement some of the things that are now handled by strategies. For example, all weakening could be replaced by a rule, which performs all weakening in a proviso.

We are planning a continuation of this project, and we will then perform the next step of the development methodology described in [Aro92], which would be to write new rules of the kind described above, followed by more specialized provisos. With these improvements, we would gain efficiency, and still know what we are doing. The system described in this paper stands as a model for the next version.

Acknowledgements

The author wishes to thank the members of the former MDA project at SICS, especially the leader of the project Adina Jägbeck and Anna-Lena Ereback, with whom the author has had a lot of fruitful discussions, and who are partly responsible for the underlying ideas (the non-GCLA stuff) of the material presented herein.

The author also wishes to thank both the FOPKBS group at SICS, especially Per Kreuger with whom he has had fruitful discussions around GCLA-related topics, and my supervisor Lars Hallnäs at Chalmers University of Technology, who among other things pointed out to me that index functions could be useful in this context, and also for his never-ending enthusiasm, which has helped me several times. Many thanks also to Torkel Franzen at SICS for improving the language of the paper.

References

[Aro91a] M. Aronsson, *GCLA User's manual*, Technical Report SICS T91:21, 1991. An extended manual is submitted with the GCLA system.

[Aro91b] M. Aronsson, *A Definitional Approach to the Combination of Functional and Relational Programming*, SICS Research Report R91:10.

[Aro91c] M. Aronsson, L.H. Eriksson, L. Hallnäs, P. Kreuger, *A Survey of GCLA: A Definitional Approach to Logic Programming*, Extensions of Logic Programming: Proceedings of a workshop held at the SNS, Universität Tübingen, November 1989, in Springer Lecture Notes in Artificial Intelligence no. 475, 1991.

[Aro92] M. Aronsson, *Methodology and Programming Techniques in GCLA II*, Extensions of Logic Programming: Proceedings of a workshop held at the SICS, January 1991, in Springer Lecture Notes in Artificial Intelligence. Also available as SICS Research Report R92:05.

[AC87] P.E. Agre and D. Chapman, *Pengi: An implementation of Theory of Activity*, AAAI-87, 1987

[BD90] J. Bresina, M. Drummond, *Integrating Planning and Reaction, A preliminary report*, NASA Ames Research Center, 1990

[BPN] Svensk Byggtjänst, *Byggbranschens Prisguide, Nyproduktion*, two loose-leafs, Svensk byggtjänst, 1991

[ET89] F. Engvall, S. Thelander, *Utvärdering av datorstödd projektplanering; Mjukvarans anpassning till planering och styrning på byggarbetsplatsen*, Examensarbete 233, Inst. för byggnadsekonomi och byggnads-organisation, KTH 1989 (in Swedish).

[Eri92] Lars-Henrik Eriksson, *A Finitary Version of the Calculus of Partial Inductive Definitions*, Extensions of Logic Programming: Proceedings of a workshop held at the SICS, January 1991, in Springer Lecture Notes in Artificial Intelligence. Also available as SICS Research Report R92:08.

[Geo88] M. P. Georgeff, *Reasoning about Plans and Actions*, in *Exploring Artificial Intelligence - Survey talks from the National Conferences on AI*, eds. Howard E. Shrobe, 1987

[GN87] M. R. Genesereth, N. J. Nilsson; *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann Publishers 1987

[GL87] M. P. Georgeff, A. L. Lansky, *Procedural Knowledge*, Technical note 411, SRI International, 1987

[GI89] M.P. Georgeff, F.F. Ingrand, *Decision-Making in an Embedded Reasoning System*, IJCAI-89

[Ham89] K. J. Hammond, *Case-Based Planning*, Academic Press, 1989

[HS-H90] L. Hallnäs, P. Schroeder-Heister, *A Proof-Theoretic Approach to Logic Programming. I, Clauses as Rules*, Journal of Logic and Computation vol. 1 no. 2, pp 261 - 283, 1990.

[HS-H91] L. Hallnäs, P. Schroeder-Heister, *A Proof-Theoretic Approach to Logic Programming. II, Programs as Definitions*, Journal of Logic and Computation vol. 1 no. 5, pp 635 - 660, 1991.

[Hal91] L. Hallnäs, *Partial Inductive Definitions*, Theoretical Computer Science 87, pp 115 - 142, 1991.

[Kae86] L. P. Kaelbling, *An Architecture for Intelligent Reactive Systems*, Technical note 400, SRI International, 1986

[Kae88] L. P. Kaelbling, *Goals as Parallel Program Specification*, SRI International, 1988

[Kre92] P. Kreuger, *GCLA II, A Definitional Approach to Control*, Extensions of Logic Programming: Proceedings of a workshop held at the SICS, January 1991, in Springer Lecture Notes in Artificial Intelligence. Also available as SICS Research Report R92:09.

[Kre93] P. Kreuger, *Issues in Symmetry and Iterated Sequents, Notes on Declarative Control*, unpublished, january 1993.

[Lau91] A. Laurell, *The Construction Methods Presentator, A Multi Media Presentation Tool for Construction Methods*, SICS technical report T91:07

[Metod&Data] Byggförbundet, *Metod och Data*, Byggförlaget, Stockholm

[MDA] *Modell till datorstöd för planering och styrning på byggarbetsplatsen*, two videos, the first produced by Eckerud Reklam AB together with Multivision, the second produced by SICS, KTH and EFI. The video is distributed by SICS.

[Wil89] D. E. Wilkins, *Can AI Planners Solve Practical Problems*, Technical note 468R, SRI International, 1989

Appendix A:
A Complete Listing of the Example Building

The example building constitutes the following design database

```
%%% Patches to simulate a design database consisting of these elements as well as
%%% the supporting parts below.
design([1,a(_),a(_)],[1,0,a(_)],a(_),[area(3000)]).
design([3,2,a(_)],[1,0,a(_)],a(_),[]).
design([3,4,a(_)],[1,2,a(_)],a(_),[area(n(288)),length(n(24)),width(n(12))]).
design([3,6,a(_)],[1,1,a(_)],a(_),[area(n(0))]).
design([3,6,a(_)],[1,2,a(_)],a(_),[area(n(0))]).
design([3,7,a(_)],[1,1,a(_)],a(_),[area(n(1150))]).
design([3,7,a(_)],[1,2,a(_)],a(_),[area(n(1150))]).
design([5,a(_),a(_)],[1,a(_),a(_)],a(_),[]).
design([6,a(_),a(_)],[1,a(_),a(_)],a(_),[]).

%%% Supporting parts
design([3,3,1],[1,1,0001],[e,3,3,2,_],
    [grid([a11]),construction_weight_kg_per_sqm(n(15)),single_layer,
     length(n(24)),height(n(3)),thickness(n(0.20)),area(n(72)),
     volume(n(14.4)),relative_height(n(1))]).
design([3,3,1],[1,1,0002],[e,3,3,2,_],
    [grid([a12]),construction_weight_kg_per_sqm(n(15)),single_layer,
     length(n(24)),height(n(3)),thickness(n(0.20)),area(n(72)),
     volume(n(14.4)),relative_height(n(1))]).
design([3,3,1],[1,1,0003],[e,3,3,2,_],
    [grid([a21]),construction_weight_kg_per_sqm(n(15)),single_layer,
     length(n(12)),height(n(3)),thickness(n(0.20)),area(n(36)),
     volume(n(7.2)),relative_height(n(1))]).
design([3,3,1],[1,1,0004],[e,3,3,2,_],
    [grid([a22]),construction_weight_kg_per_sqm(n(15)),single_layer,
     length(n(12)),height(n(3)),thickness(n(0.20)),area(n(36)),
     volume(n(7.2)),relative_height(n(1))]).
design([3,3,1],[1,1,0005],[e,3,3,2,_],
    [grid([a23]),construction_weight_kg_per_sqm(n(15)),single_layer,
     length(n(12)),height(n(3)),thickness(n(0.20)),area(n(36)),
     volume(n(7.2)),relative_height(n(1))]).
design([3,3,1],[1,1,0006],[e,3,3,2,_],
    [grid([a24]),construction_weight_kg_per_sqm(n(15)),single_layer,
     length(n(12)),height(n(3)),thickness(n(0.20)),
     volume(n(7.2)),relative_height(n(1))]).

design([3,3,1],[1,2,0007],[e,3,3,2,_],
    [grid([a11]),construction_weight_kg_per_sqm(n(15)),single_layer,
     length(n(24)),height(n(3)),thickness(n(0.20)),area(n(72)),
     volume(n(14.4)),relative_height(n(4))]).
design([3,3,1],[1,2,0008],[e,3,3,2,_],
    [grid([a12]),construction_weight_kg_per_sqm(n(15)),single_layer,
     length(n(24)),height(n(3)),thickness(n(0.20)),area(n(72)),
     volume(n(14.4)),relative_height(n(4))]).
design([3,3,1],[1,2,0009],[e,3,3,2,_],
    [grid([a21]),construction_weight_kg_per_sqm(n(15)),single_layer,
     length(n(12)),height(n(3)),thickness(n(0.20)),area(n(36)),
     volume(n(7.2)),relative_height(n(4))]).
design([3,3,1],[1,2,0010],[e,3,3,2,_],
    [grid([a22]),construction_weight_kg_per_sqm(n(15)),single_layer,
     length(n(12)),height(n(3)),thickness(n(0.20)),area(n(36)),
     volume(n(7.2)),relative_height(n(4))]).
design([3,3,1],[1,2,0011],[e,3,3,2,_],
    [grid([a23]),construction_weight_kg_per_sqm(n(15)),single_layer,
     length(n(12)),height(n(3)),thickness(n(0.20)),area(n(36)),
     volume(n(7.2)),relative_height(n(4))]).
design([3,3,1],[1,2,0012],[e,3,3,2,_],
    [grid([a24]),construction_weight_kg_per_sqm(n(15)),single_layer,
        length(n(12)),height(n(3)),thickness(n(0.20)),area(n(36)),
        volume(n(7.2)),relative_height(n(4))]).

design([3,3,4],[1,0,0013],[e,3,2,1,_],
     [grid([a11,a12,a21,a22,a23,a24]),construction_weight_kg_per_sqm(n(30)),
      area(n(288)),volume(n(43.2)),
      width(n(12)),length(n(24)),thickness(n(0.15)),relative_height(n(0))]).
design([3,3,4],[1,1,0014],[e,3,3,3,_],
     [grid([a11,a12,a21,a22]),construction_weight_kg_per_sqm(n(30)),
      area(n(96)),volume(n(14.4)),
      width(n(8)),length(n(12)),thickness(n(0.15)),relative_height(n(3))]).
design([3,3,4],[1,1,0015],[e,3,3,3,_],
     [grid([a11,a12,a22,a23]),construction_weight_kg_per_sqm(n(30)),
      area(n(96)),volume(n(14.4)),
      width(n(8)),length(n(12)),thickness(n(0.15)),relative_height(n(3))]).
design([3,3,4],[1,1,0016],[e,3,3,3,_],
     [grid([a11,a12,a23,a24]),construction_weight_kg_per_sqm(n(30)),
      area(n(96)),volume(n(14.4)),
      width(n(8)),length(n(12)),thickness(n(0.15)),relative_height(n(3))]).

design([3,3,4],[1,2,0017],[e,3,3,3,_],
     [grid([a11,a12,a21,a22]),construction_weight_kg_per_sqm(n(30)),
      area(n(96)),volume(n(14.4)),
      width(n(8)),length(n(12)),thickness(n(0.15)),relative_height(n(6))]).
design([3,3,4],[1,2,0018],[e,3,3,3,_],
     [grid([a11,a12,a22,a23]),construction_weight_kg_per_sqm(n(30)),
      area(n(96)),volume(n(14.4)),
      width(n(8)),length(n(12)),thickness(n(0.15)),relative_height(n(6))]).
design([3,3,4],[1,2,0019],[e,3,3,3,_],
     [grid([a11,a12,a23,a24]),construction_weight_kg_per_sqm(n(30)),
      area(n(96)),volume(n(14.4)),
      width(n(8)),length(n(12)),thickness(n(0.15)),relative_height(n(6))]).

%%% Fasade
design([3,5,3],[1,1,0020],[e,3,3,2,_],
     [grid([a11]),length(n(24)),height(n(3)),brick,thickness(n(0.20))]).
design([3,5,3],[1,1,0021],[e,3,3,2,_],
     [grid([a12]),length(n(24)),height(n(3)),brick,thickness(n(0.20))]).
design([3,5,3],[1,1,0022],[e,3,3,2,_],
     [grid([a21]),length(n(12)),height(n(3)),brick,thickness(n(0.20))]).
design([3,5,3],[1,1,0023],[e,3,3,2,_],
     [grid([a22]),length(n(12)),height(n(3)),brick,thickness(n(0.20))]).

design([3,5,3],[1,2,0024],[e,3,3,2,_],
     [grid([a11]),area(n(72)),length(n(24)),height(n(3)),brick,thickness(n(0.20))]).
design([3,5,3],[1,2,0025],[e,3,3,2,_],
     [grid([a12]),area(n(72)),length(n(24)),height(n(3)),brick,thickness(n(0.20))]).
design([3,5,3],[1,2,0026],[e,3,3,2,_],
     [grid([a21]),area(n(36)),length(n(12)),height(n(3)),brick,thickness(n(0.20))]).
design([3,5,3],[1,2,0027],[e,3,3,2,_],
     [grid([a22]),area(n(36)),length(n(12)),height(n(3)),brick,thickness(n(0.20))]).

max_resources(man,_,n(30)).
```

Appendix B:

A Complete Listing of the Example Methods and Section Divisions

The Object Level Code

```
plan(N,A,X) <= expand(X,Y) -> pl(N,A,Y).

expand([],[]).
expand([X|Y],[X1|Y1]) <= expand(X,X1),expand(Y,Y1).
expand(plan(N,A,X),Z) <= plan(N,A,X) -> Z.
expand(X,X1)#{X \= [],X \= [_|_],X \= plan(_,_,_)} <= expand_one(_,X) -> X1.
```

```
expand_one(1,X) <= X.
expand_one(2,X) <= m(_,_,X).
expand_one(N,X)#{N \= 1,N \= 2} <= e(_,_,X).

member(X,[X|_]).
member(X,[X1|R])#{f(X,X1) \= f(W,W)} <= member(X,R).

%%%--------------------------------
%%% Method DB

m(0,[],activity(0,[a(V1),a(V2),a(V3)],[V4,V5,V6],_,Tst,Tsl,Res)) <=
  plan(main_groups,
       activity(0,[a(V1),a(V2),a(V3)],[V4,V5,V6],_,Tst,Tsl,Res),
  [activity(3,[3,a(V2),a(V3)],[V4,V5,V6],_,_,_,_),      % building
   activity(81,[5,a(V2),a(V3)],[V4,V5,V6],_,_,_,_),     % Installations, HVAC
   activity(83,[6,a(V2),a(V3)],[V4,V5,V6],_,_,_,_),     % el. installations
   activity(1,[1,a(V2),a(V3)],[V4,V5,V6],_,_,_,_)]).    % ground
m(1,[],activity(3,[3,a(V2),a(V3)],[V4,V5,V6],_,Tst,Tsl,Res)) <=
  plan(main_groups_of_building,
       activity(3,[3,a(V2),a(V3)],[V4,V5,V6],_,Tst,Tsl,Res),
  [activity(2,[3,2,a(V3)],[V4,0,V6],_,_,_,_),            % basis
   activity(33,[3,3,a(V3)],[V4,V5,V6],[e|_],_,_,_),      % supporting parts
   activity('4_8',[3,4,a(V3)],[V4,V5,V6],_,_,_,_),       % outer roof
   activity(6,[3,5,a(V3)],[V4,V5,V6],_,_,_,_),           % outer walls
   activity(7,[3,6,a(V3)],[V4,V5,V6],_,_,_,_),           % rooming in
   activity(66,[3,7,a(V3)],[V4,V5,V6],_,_,_,_)]).        % inner surfaces
m(2,[],activity(33,[3,3,a(V3)],[V4,0,V6],_,Tst,Tsl,Res)) <=
  plan(supporting_parts,
       activity(33,[3,3,a(V3)],[V4,0,V6],_,Tst,Tsl,Res),
  [activity('3_2',[3,3,4],[V4,0,V6],_,_,_,_)]).  % slab on ground = slab 0
m(3,[],activity(33,[3,3,a(V3)],[V4,V5,a(V6)],_,Tst,Tsl,Res))
       #{V5\=0,V5\=a(_)} <=
  plan(supporting_parts,
       activity(33,[3,3,a(V3)],[V4,V5,a],_,Tst,Tsl,Res),
  [activity('3_3',[3,3,1],[V4,V5,a(V6)],_,_,_,_),   % supporting walls
   activity('3_4',[3,3,2],[V4,V5,a(V6)],_,_,_,_),   % pillars
   activity('3_6',[3,3,4],[V4,V5,a(V6)],_,_,_,_),   % supporting slabs
   activity('3_7',[3,3,6],[V4,V5,a(V6)],_,_,_,_),   % supporting parts of stairs
   activity('4_8',[3,3,7],[V4,V5,a(V6)],_,_,_,_)]). %supporting parts of outer roof

%%%--------------------------------
%%% Partition DB
e(1,building_partitioning,
  activity(X,[3,V2,V3],[a(V4),V5,a(V6)],Bsab1,Tst,Tsl,Res)) <=
   bagof([V2,V3,Bsab2],Hus,
         (design([3,V2,V3],[Hus,V5,a(V6)],Bsab1,a(_)),number(n(Hus))),Tmp),
   sort(Tmp,List1),
   bagof([TST,TSL,RES],activity(X,[3,V2,V3],[V4,V5,a(V6)],Bsab1,TST,TSL,RES),
        member(V4,List1),List)
   -> plan(building_partitioning,
       activity(X,[3,V2,V3],[a(V4),V5,a(V6)],Bsab1,Tst,Tsl,Res),List).
e(2,floor_partitioning,activity(X,[3,V2,V3],[V4,a(V5),a(V6)],
       Bsab,Tst,Tsl,Res))
       #{V2 \= 2,V2 \= 4,V2 \= 6} <=
   bagof([V2,V3,Bsab],Van,
         (design([3,V2,V3],[V4,Van,a(V6)],Bsab,a(_)),number(n(Van))),Tmp),
   sort(Tmp,List1),
   bagof([TST,TSL,RES],activity(X,[3,V2,V3],[V4,V5,a(V6)],Bsab,TST,TSL,RES),
        member(V5,List1),List)
   -> plan(floor_partitioning,
       activity(X,[3,V2,V3],[V4,a(V5),a(V6)],Bsab,Tst,Tsl,Res), List).
```

The Control Code

```
%%% From rules.rul, added that B should not be false, and if true
%%% then do not execute the sequent (P \- true), succeed at once.
d_right(C,PT) <=
  atom(C),
  clause(C,B),
  B \== false,
  (B == true ; B \== true,(PT -> (P \- B)))
  -> (P \- C).

%%%==========================================
%%% Choice-of-method phase
a_left2((A -> C1),I,PT,PT1,left) <=
        data(C1) -> (I@[(A -> C1)|_] \- _).
a_left2((A -> C1),I,PT,PT1,left) <=
        a_left1(X,I,PT,PT1).
a_left2((A -> C1),I,PT,PT1,right) <=
        not(data(C1)) -> (I@[(A -> C1)|_] \- _).
a_left2((A -> C1),I,PT,PT1,right) <=
        a_left(X,I,PT,PT1).

a_left1((A -> C1),I,PT,PT1) <=
        (PT1 -> (I@[C1|Y] \- C)),
        (PT -> (I@Y \- A))
        -> (I@[(A -> C1)|Y] \- C).

d_right1(C,PT) <= (not(functor(C,plan,_)),not(functor(C,pl,_)) -> (_ \- C)).
d_right1(C,PT) <= d_right(C,PT).

d_left1(C,I,PT) <=
       (not(functor(C,pl,_)),
        not(functor(C,activity,_)),
        not(functor(C,state,_))
       -> (I@[C|_] \- _)).
d_left1(T,X,PT) <=
       atom(T),
       definiens(T,Dp,N),
       N > 0,
       (PT -> (X@[Dp|Y] \- C))
        -> (X@[T|Y] \- C).

sort_right <=
       sort(X,Y) ->
       (_ \- sort(X,Y)).

findall_right(PT) <=
       lift_from_a(B,B1,[],_),
       (i([A],PT^Ass^B1^(PT -> (Ass \- B1))) -> C) ->
       (Ass \- findall(A,B,C)).

bagof_right(PT) <=
       lift_from_a(B,B1,[],Vars),
       append(EVars,Vars,Vars1),
       (i([A],Vars1^PT^(PT -> (Ass \- B1))) -> C) ->
       (Ass \- bagof(EVars,A,B,C)).

axiom1(T,C,I) <=
       data(T)
       -> (I@[T|_] \- C).
axiom1(T,C,I) <=
       axiom(T,C,I).

lift_from_a(V,V,L,L) :- var(V).
lift_from_a(A,V,L,[V|L]) :- functor(A,a,1),A = a(V).
lift_from_a(Atom,Atom,L,L) :- atomic(Atom).
lift_from_a(X,[F1|R1],L,L2) :- nonvar(X),X = [F|R],
       lift_from_a(F,F1,L,L1),
       lift_from_a(R,R1,L1,L2).
lift_from_a(Str,Str1,L,L1) :- nonvar(Str),
       Str =..[S|A],S \== '.',S \== a,A \== [],
       lift_from_a(A,A1,L,L1),
       Str1 =..[S|A1].

data(X) :- functor(X,pl,_).
data(X) :- functor(X,activity,_).
```

```
number_right <=
       number(C) ->
       (_ \- number(n(C))).

right1(PT) <=
       sort_right,
       findall_right(PT),
       bagof_right(PT),
       number_right,
       v_right(_,PT,PT),
       a_right(_,PT),
       o_right(_,_,PT),
       true_right,
       d_right1(_,PT).
left1(PT) <=
       v_left(_,_,PT),
       a_left2(_,_,PT,PT,_),
       o_left(_,_,PT,PT),
       d_left1(_,_,PT),
       pi_left(_,_,PT),
       false_left(_).

plan <= axiom1(_,_,_),right1(plan),left1(plan).
```

Appendix C:
A Complete Listing of the Intermediate Step

The Object Level Code

```
flatten_act([pl(N,activity(A,B,C,D,E,F,Res),L)],[(E,F)]) <=
       cons(akt(-1,activity(A,B,C,D,E,F,Res),times(E,F,List)),
            flatten_act(L,List)).
flatten_act([pl(N,activity(A,B,C,D,E,G,Res),L),F|R],[(E,G)|R1]) <=
       cons(akt(-1,activity(A,B,C,D,E,G,Res),times(E,G,List)),
            append(flatten_act(L,List),flatten_act([F|R],R1))).
flatten_act([activity(A,B,C,D,E,G,Res),F|R],[(E,G)|R1]) <=
       cons(akt(A,activity(A,B,C,D,E,G,Res),N),flatten_act([F|R],R1)).
flatten_act([activity(A,B,C,D,E,G,Res)],[(E,G)]) <=
       cons(akt(A,activity(A,B,C,D,E,G,Res),N),[]).

append([],L) <= L.
append([F|R],L) <= cons(F,append(R,L)).
append(X,Y)#{X \= [],X \= [_|_]} <=
       ((X -> Z),(Z = [] ; Z = [_|_]) -> append(Z,Y)).
```

The Control Level

```
%%% Intermediate step, convert from tree structure to flat structure.
%%% Also, keep track of and collect time slots of each activity,
%%% so that superactivities have a list of their subactivities
%%% time slots.
flatten <=
       axiom_flatten(_,_,_),
       left_flatten(flatten),
       right_flatten(flatten).

axiom_flatten(T,C,I) <=
       (functor(T,akt,3) ;
        functor(T,[],0)  ;
        functor(T,'.',2))
       -> (I@[T|_] \- C).
axiom_flatten(T,C,I) <= axiom(T,C,I).

d_right_flatten(C,PT) <= functor(C,=,2) -> (_ \- C).
d_right_flatten(C,PT) <= d_right(C,PT).
```

```
right_flatten(PT) <=
        true_right,
        d_right_flatten(_,PT),
        o_right(_,_,PT),
        a_right(_,PT),
        v_right(_,PT,PT).

left_flatten(PT) <=
        d_left_flatten(C,I,PT),
        a_left(_,_,PT,PT).

d_left_flatten(A,I,PT) <=
        (functor(A,flatten_act,2) ;
         functor(A,append,2) ;
         functor(A,cons,2))
        -> (I@[A|_] \- _).
d_left_flatten(A,B,PT) <= d_left(A,B,PT).
```

Appendix D:
A Complete Listing of the Example Activities

The Object Level Code

```
akt(0,activity(0,[a(V1),a(V2),a(V3)],[V4,V5,V6],B,Time,End,NT),project) <=
     when(started(official_start,_,_),T) ->
     ((area(design([3,3,4],[a(_),a(_),a(_)])) -> n(TotArea)),
      (area(design([3,3,4],[a(_),0,a(_)])) -> n(BasArea)),
      (defun((time_formula -> n(9) * n(TotArea))),
       defun((building_area -> n(Area))),
       defun((team_size(general_workers) -> n(2))),
       defun((place_coeff -> n(1.0))),
       defun((density(general_workers) -> (n(BasArea) / n(25)))) ->
       (get_manpower(team_size(general_workers),density(general_workers),NT)
        -> n(Res)),
       (consume(T,n(Res),time_formula,End,
                activity(0,[a(V1),a(V2),a(V3)],[V4,V5,V6],B,Time,End,NT))
        -> quote(Act)))
     -> (change(started([a(V1),a(V2),a(V3)],[V4,V5,V6],B),End),Act)).

akt(1,activity(1,[1,a(V2),a(V3)],[V4,V5,V6],B,Time,End,NT),work_on_ground) <=
     when(started([3,a(V2),a(V3)],[V4,V5,V6],a(_)),Time) ->
     ((area(design([1,a(V2),a(V3)],[a(_),0,a(_)])) -> n(Area)),
      (defun((time_formula -> ground_area * n(0.2))),
       defun((ground_area -> n(Area))),
       defun((teamsize(gardenworkers) -> n(2))),
       defun((density(gardenworkers) -> (ground_area / n(300)))),
       defun((place_coeff -> n(1.0))) ->
       (get_manpower(teamsize(gardenworkers),density(gardenworkers),NT) -> n(Res)),
       (consume(Time,n(Res),time_formula,End,
                activity(1,[1,a(V2),a(V3)],[V4,V5,V6],B,Time,End,NT)) -> quote(Act)))
     -> (change(started([1,a(V2),a(V3)],[V4,V5,V6],B),End),Act)).

akt(3,activity(3,[3,a(V2),a(V3)],[V4,a(V5),a(V6)],B,Time,End,NT),actual_building) <=
     when(started(official_start,_,_),Time) ->
     ((area(design([3,3,4],[V4,0,a(V6)])) -> n(BasArea)),
      (area(design([3,3,4],[V4,a(V5),a(V6)])) -> n(TotArea)),
      (defun((timeformula -> n(6) * n(TotArea))),
       defun((building_area -> n(Area))),
       defun((teamsize(general_workers) -> n(2))),
       defun((density(general_workers) -> (n(BasArea) / n(25)))) ->
       (get_manpower(teamsize(general_workers),density(general_workers),NT)
        -> n(Res)),
       (consume(Time,n(Res),timeformula,End,
                activity(3,[3,a(V2),a(V3)],[V4,a(V5),a(V6)],B,Time,End,NT))
        -> quote(Act)))
     -> (change(started([3,a(V2),a(V3)],[V4,a(V5),a(V6)],B),End),Act)).
 akt(3,activity(3,[3,a(V2),a(V3)],[V4,0,a(V6)],B,Time,End,NT),
        actual_building_basement) <=
     when(started(official_start,_,_),Time) ->
     ((area(design([3,3,4],[V4,0,a(V6)])) -> n(Area)),
      (defun((timeformula -> n(6) * building_area)),
       defun((building_area -> n(Area))),
       defun((teamsize(general_workers) -> n(2))),
       defun((density(general_workers) -> (building_area / n(25)))) ->
       (get_manpower(teamsize(general_workers),density(general_workers),NT)
        -> n(Res)),
       (consume(Time,n(Res),timeformula,End,
                activity(3,[3,a(V2),a(V3)],[V4,0,a(V6)],B,Time,End,NT))
        -> quote(Act)))
     -> (change(started([3,a(V2),a(V3)],[V4,0,a(V6)],B),End),Act)).
 akt(3,activity(3,[3,a(V2),a(V3)],[V4,V5,a(V6)],B,Time,End,NT),building_floors)
    #(V5 \= a(_)) <=
     ((n(V5) - n(1) -> n(V5p)),
      when(started([3,a(V2),a(V3)],[V4,V5p,a(V6)],a(_)),Time)) ->
     ((area(design([3,3,4],[V4,V5,a(V6)])) -> n(Area)),
      (defun((timeformula -> n(6) * buildingarea)),
       defun((buildingarea -> n(Area))),
       defun((teamsize(general_workers) -> n(2))),
       defun((density(general_workers) -> (buildingarea / n(25)))) ->
       (get_manpower(teamsize(general_workers),density(general_workers),NT)
        -> n(Res)),
       (consume(Time,n(Res),timeformula,End,
                activity(3,[3,a(V2),a(V3)],[V4,V5,a(V6)],B,Time,End,NT))
        -> quote(Act)))
     -> (change(started([3,a(V2),a(V3)],[V4,V5,a(V6)],B),End),Act)).

 akt(81,activity(81,[5,a(V2),a(V3)],[V4,V5,V6],_,Time,End,NT),
        'HVAC installations') <=
     when(started([3,3,a(V3)],[V4,V5,V6],a(_)),Time) ->
     ((area(design([3,3,4],[V4,V5,V6])) -> n(Area)),
      (defun((timeformula -> area * n(3.0))),
       defun((area -> n(Area))),
       defun((teamsize(plumber) -> n(2))),
       defun((density(plumber) -> (area / n(100)))),
       defun((place_coeff -> n(1))) ->
       (get_manpower(teamsize(plumber),density(plumber),NT) -> n(Res)),
       (consume(Time,n(Res),timeformula,End,
                activity(81,[5,a(V2),a(V3)],[V4,V5,V6],_,Time,End,NT))
        -> quote(Act)))
     -> (change(started([5,a(V2),a(V3)],[V4,V5,V6],_),End),Act)).

 akt(83,activity(83,[6,a(V2),a(V3)],[V4,V5,V6],_,Time,End,NT),install_electric) <=
     when(started([3,3,a(V3)],[V4,V5,V6],a(_)),Time) ->
     ((area(design([3,3,4],[a(_),a(_),a(_)])) -> n(Area)),
      (defun((timeformula -> area * n(1.0))),
       defun((area -> n(Area))),
       defun((teamsize(electrician) -> n(1))),
       defun((density(electrician) -> (area / n(100)))),
       defun((place_coeff -> n(1.0))) ->
       (get_manpower(teamsize(electrician),density(electrician),NT) -> n(Res)),
       (consume(Time,n(Res),timeformula,End,
                activity(83,[6,a(V2),a(V3)],[V4,V5,V6],_,Time,End,NT))
        -> quote(Act)))
     -> (change(started([6,a(V2),a(V3)],[V4,V5,V6],_),End),Act)).

%%% For method 2
akt(2,activity(2,[3,2,a(V3)],[V4,0,V6],_,Time,End,NT),work_on_ground) <=
    when(started(official_start,_,_),Time) ->
    ((area(design([3,3,4],[V4,0,a(_)])) -> n(Area)),
     (defun((timeformula -> area_slab0 * n(0.5))),
      defun((area_slab0 -> n(Area))),
      defun((answer_unit -> hours)),
      defun((teamsize(moulders) -> n(3))),
      defun((density(moulders) -> (area_slab0 / n(1000)))),
      defun((place_coeff -> n(1.0))) ->
```

```
          (get_manpower(teamsize(moulders),density(moulders),NT) -> n(Res)),
          (consume(Time,n(Res),timeformula,End,
                   activity(2,[3,2,a(V3)],[V4,0,V6],_,Time,End,NT)) -> quote(Act)))
        -> (change(started([3,2,a(V3)],[V4,0,V6],_),End),Act)).

 akt(33,activity(33,[3,3,a(V3)],[V4,V5,V6],_,Time,End,NT),
        'Constructing supporting parts of building') <=
     when(started([3,2,a(V3)],[V4,V5,V6],a(_)),Time) ->
     ((area(design([3,3,4],[V4,V5,V6])) -> n(Area)),
      (defun((timeformula -> floor_area * n(2.5))),
       defun((floor_area -> n(Area))),
       defun((answer_unit -> hours)),
       defun((teamsize(moulders) -> n(4))),
       defun((density(moulders) -> (floor_area / n(250)))),
       defun((driftkoefficient -> n(1.0))) ->
       (get_manpower(teamsize(moulders),density(moulders),NT) -> n(Res)),
       (consume(Time,n(Res),timeformula,End,
                activity(33,[3,3,a(V3)],[V4,V5,V6],_,Time,End,NT)) -> quote(Act)))
     -> (change(started([3,3,a(V3)],[V4,V5,V6],_),End),Act)).

 akt('4_8',activity('4_8',[3,4,a(V3)],[V4,V5,V6],_,Time,End,NT),work_on_outer_roof)
 <=
     when(started([3,3,a(V3)],[V4,V5,V6],a(_)),Time) ->
     ((area(design([3,4,a(V3)],[V4,V5,V6])) -> n(Area)),
      (defun((timeformula -> 3.5 * roof_area)),
       defun((roof_area -> n(Area))),
       defun((answer_unit -> hours)),
       defun((teamsize(carpenters) -> n(10))),
       defun((density(carpenters) -> (roof_area / n(500)))),
       defun((driftkoefficient -> n(1))) ->
       (get_manpower(teamsize(carpenters),density(carpenters),NT) -> n(Res)),
       (consume(Time,n(Res),timeformula,End,
                activity('4_8',[3,4,a(V3)],[V4,V5,V6],_,Time,End,NT)) -> quote(Act)))
     -> (change(started([3,4,a(V3)],[V4,V5,V6],_),End),Act)).

 akt(6,activity(6,[3,5,a(_)],[V4,V5,V6],_,Time,End,NT),work_on_outer_walls) <=
     when(started([3,3,a(V3)],[V4,V5,V6],a(_)),Time) ->
     ((area(design([3,5,a(V3)],[V4,V5,V6])) -> n(Area)),
      (defun((timeformula -> n(2.0) * area_outer_walls)),
       defun((area_outer_walls -> n(Area))),
       defun((answer_unit -> hours)),
       defun((teamsize(brick_layer) -> n(6))),
       defun((density(brick_layer) -> (area_outer_walls / n(100)))),
       defun((place_coeff -> n(1))) ->
       (get_manpower(teamsize(brick_layer),density(brick_layer),NT) -> n(Res)),
       (consume(Time,n(Res),timeformula,End,
                activity(6,[3,5,a(_)],[V4,V5,V6],_,Time,End,NT)) -> quote(Act)))
     -> (change(started([3,5,a(V3)],[V4,V5,V6],_),End),Act)).

 akt(7,activity(7,[3,6,a(V3)],[V4,V5,V6],_,Time,End,NT),stomkomplettering) <=
     when(started([3,3,a(V3)],[V4,V5,V6],a(_)),Time) ->
     ((area(design([3,3,4],[V4,V5,V6])) -> n(Area)),
      (defun((timeformula -> floor_area * n(2.0))),
       defun((floor_area -> n(Area))),
       defun((answer_unit -> hours)),
       defun((team_size(carpenters) -> n(5))),
       defun((density(carpenters) -> (floor_area / n(200)))),
       defun((place_coeff -> n(1))) ->
       (get_manpower(antalman(carpenters),antal_lag(carpenters),NT) -> n(Res)),
       (consume(Time,n(Res),timeformula,End,
                activity(7,[3,6,a(V3)],[V4,V5,V6],_,Time,End,NT)) -> quote(Act)))
     -> (change(started([3,6,a(V3)],[V4,V5,V6],_),End),Act)).

 akt(66,activity(66,[3,7,a(V3)],[V4,V5,V6],_,Time,End,NT),inner_surfaces) <=
     when(started([3,6,a(V3)],[V4,V5,V6],a(_)),Time) ->
     ((area(design([3,3,a(V3)],[a(_),a(_),a(_)])) -> n(Area)),
      (defun((timeformula -> area * n(0.1))),
       defun((area -> n(Area))),
       defun((answer_unit -> hours)),
       defun((teamsize(painters) -> n(1))),
        defun((density(painters) -> (area / n(250)))),
        defun((driftkoefficient -> n(1))) ->
        (get_manpower(teamsize(painters),density(painters),NT) -> n(Res)),
        (consume(Time,n(Res),timeformula,End,
                activity(66,[3,7,a(V3)],[V4,V5,V6],_,Time,End,NT)) -> quote(Act)))
      -> (change(started([3,7,a(V3)],[V4,V5,V6],_),End),Act)).


akt(-1,Act,times(Min,Max,List)) <=
        true
        -> (find_max_min(List,Min,Max)
            -> performed(Act,Min,Max,List)).

%%%======================
find_max_min(L,Tst,Tsl) <=
        find_max_min1(L,[],[],Tst,Tsl).

find_max_min1([],Min,Max,n(Tst),n(Tsl)) <=
        constr((Tst = min(Min))),
        constr((Tsl = max(Max))).
find_max_min1([(n(Tst),n(Tsl))|R],Min,Max,ST,SL) <=
        find_max_min1(R,[Tst|Min],[Tsl|Max],ST,SL).

get_manpower(TeamSize, TeamFormula, n(NoOfTeams)) <=
        max_resources(man,_,MaxSize),
        (int(MaxSize / TeamSize) -> n(MaxSize1)),
        (int(TeamFormula) -> n(TeamNum)),
        (min([TeamNum,MaxSize1]) -> n(Top)),
        (gen(n(1),n(Top)) -> n(NoOfTeams)) ->
        TeamSize * n(NoOfTeams).

cons(X,Y) <= ((X -> X1), (Y -> Y1) -> [X1|Y1]).

max(A) <= (constr((X = max(A))) -> n(X)).
min(A) <= (constr((X = min(A))) -> n(X)).

gen(A,B) <= ((int(B) -> n(B1)) -> gen1(A,n(B1),_)).

gen1(n(_),n(X),1) <= n(X).
gen1(n(Y),n(X),Z)#{Z \= 1} <= n(Y) < n(X) -> gen1(n(Y),n(X) - n(1),_).
gen1(X,Y,_)#{Y \= n(_)} <= (Y -> Z) -> gen1(X,Z,_).

when((A,B),Time) <=
        when(A,Time1),
        when(B,Time2),
        constr((Time = max([Time1,Time2]))).
when(started(B1,B2,B3),n(Time)) <=        %started means when some state
        forall([B1,B2,B3,Prop,T1,T2],(design(B1,B2,B3,Prop) -> %started to exist
                    state(bsab(B1,B2,B3),T1,T2))),
        findall(Start,state(bsab(B1,B2,B3),n(Start),_),List),
        constr((Time = max(List))),
        findall(End,state(bsab(B1,B2,B3),_,n(End)),List1),
        constr((Time = min(List1))).
when(finished(B1,B2,B3),n(Time)) <=            %finished means when some state
        forall([B1,B2,B3,Prop,T1,End],(design(B1,B2,B3,Prop) -> %finished to exist
                    state(bsab(B1,B2,B3),T1,n(End)))),
        findall(E,state(bsab(B1,B2,B3),_,n(E)),List),
        (max(List) -> n(Time)).

change(started(B1,B2,B3),n(Time)) <=
        forall([B1,B2,B3,Prop,End],
                (design(B1,B2,B3,Prop) ->
                 state(bsab(B1,B2,B3),n(Time),n(End)))).
change(finished(B1,B2,B3),n(Time)) <=
        forall([B1,B2,B3,Prop,St],
                (design(B1,B2,B3,Prop),
                 state(bsab(B1,B2,B3),St,n(Time)) -> true)).
change((A,B),Time) <= change(A,Time),change(B,Time).

consume(n(Time),n(0),TimeFormula,n(End),Act) <=
        call(constr(End = Time)) ->
         quote(performed(Act,n(Time),n(Time),n(0))).
consume(n(Time),Resources,TimeFormula,n(End),Act)#{Resources \= n(0)} <=
        ((int((TimeFormula / Resources) / n(8)) -> n(Duration)),
         call(constr(End = Time + Duration)) ->
         overload_check(Act,n(Time),n(Duration),Resources)).

overload_check(Act,Xst,Dur,Res) <=
        max_resources(man,Xst,MR),
        (Xst + Dur -> Xsl),
        findall(n(R),(performed(_,Xst1,Xsl1,n(R)),Xst1=<Xst,Xsl1>Xst),Lstart),
        (sum(Lstart) -> n(NStart)),
        (n(NStart) + Res -> StartTotal),
        StartTotal =< MR,           % checking start
        findall(Xst1-n(R),(performed(_,Xst1,_,n(R)),Xst<Xst1,Xsl>Xst1),
               Lst_in),
        findall(Xsl1-R1,(performed(_,_,Xsl1,n(R)),
                         Xst<Xsl1,Xsl>Xsl1,(n(0)-n(R)->R1)),Lsl_in),
        append(Lst_in,Lsl_in,L),
        keysort(L,L1),
        check_all_res_changes(L1,StartTotal,MR)
        -> quote(performed(Act,Xst,Xsl,Res)).

append([],L,L).
append([F|R],L,[F|R1]) <= append(R,L,R1).

check_all_res_changes([],_,_).
check_all_res_changes([N-X],Sofar,MaxRes) <=
        (Sofar + X -> Sofar1),
        Sofar1 =< MaxRes.
check_all_res_changes([N-X,N-Y|R],Sofar,MaxResource) <=
        (X + Y -> Z),
        check_all_res_changes([N-Z|R],Sofar,MaxResource).
check_all_res_changes([N-X,N1-Y|R],Sofar,MaxResource)#{f(N,N1)\=f(W,W)} <=
        (Sofar + X -> Sofar1),
        Sofar1 =< MaxResource,
        check_all_res_changes([N1-Y|R],Sofar1,MaxResource).

X = X.

%%%----------------------
%%% Some general routines for arithmetics
area(design(A,B)) <=
        call(bagof([C,D],X,(design(A,B,D,C),member(area(X),C)),L)) -> sum(L).
volume(design(A,B)) <=
        call(bagof([C,D],X,(design(A,B,D,C),member(volume(X),C)),L)) -> sum(L).
length(design(A,B)) <=
        call(bagof([C,D],X,(design(A,B,D,C),member(length(X),C)),L)) -> sum(L).
reinforcement_weight(design(A,B)) <=
        call(bagof([C,D,X,Y],Z,
                   (design(A,B,D,C),
                    member(construction_weight_kg_per_sqm(X),C),
                    member(area(Y),C),(X * Y -> Z)),L))
        -> sum(L).
call(X) <= X.

%%%----------------------
%%% Used when all activities are scheduled, and the plans should be plotted
plot <= findall(Y-(X,Y,Z,W),performed(X,Y,Z,W),L),
        present_plans(L).
```

The Control Level

```
%%%==========================================
%%% Scheduling phase, new rules

%%% Forall is a kind of pi-declaration.
%%% Special generalized index function used:
%%% i(Constructor, Base-constant, Template, Call), where Constructor is the functor
%%% used for building the answer structure, and Base-constant is the empty structure
%%% if there are no solutions to Call.
forall_r(forall(X,(A -> C)),PT) <=
       lift_from_a((A -> C),(A1 -> C1),[],Vars),
       (i(',',true,C1,X^Vars^Y^definiens(A1,true,Y)) -> Struct),
       (PT -> (Ass \- Struct))
       -> (Ass \- forall(X,(A -> C))).

schedule_forall <=
       right_sch_forall(schedule_forall),
       axiom_sch(_,_,_).

right_sch_forall(PT) <=
       v_right(_,PT,PT),
       true_right,
       d_right_sch(_,PT).

%%% In principle, this can be done by first doing a lot of contractions,
%%% then eliminate the forall quantification with the elements produced by
%%% the bagof.
forall_l_generate_all(forall(X,(A -> G)),I,PT,PTbag,2) <=
       lift_from_a((A -> G),(A1 -> G1),[],Vars),
       (i([G1],X^Vars^(PTbag -> ([] \- A1))) -> Goals),
       (PT -> (I@Goals@R \- C))
       -> (I@[forall(X,(A -> G))|R] \- C).

%%% Strategy for eliminate all forall-conditions
all_forall_l_generate_all(PT) <=
       (forall_l_generate_all(_,_,(A \- C),ra_sch,2) <-
        (all_forall_l_generate_all(PT) -> (A \- C))),
       PT.

ra_sch <= (right(ra_sch) <- true),axiom(_,_,_).
%%% --------------------------------------------
%%% General strategies for eleminating terms through weakening
weak_all(T,PT) <=
       copy_term(T,T1) ->
       ((search1(_,I,weak_l(T1,I,(A \- C))) <-
        (weak_all(T,PT) -> (A \- C))),PT).

v_left_all(PT) <=
       (v_left(_,I,(A \- C)) <- (v_left_all(PT) -> (A \- C))),PT.

%%% --------------------------------------------
%%% General strategy for searching the first applicable assumption
search1(N,I,PT) <= length(Ass,N) -> (Ass \- _).
search1(N,I,PT) <= search([],I,N,PT).

search(I,I1,N2,PT) <=
       (length(I,N1), N2 > N1) -> (Ass \- C).
search(I,I1,N,PT) <=
        (((I = I1 -> PT) <- true), search([_|I],I1,N,PT)).

%%% --------------------------------------------
%%% Top level strategy, for the scheduling phase
incorporate([F|R],I,PT) <=
       (PT -> (I@[F|R]@R1 \- C)) ->
       (I@[[F|R]|R1] \- C).

schedule <=
       (ready_sch <- true),
       ((search1(_,I,
          d_left(akt(_,_,_),I,
             a_left(_,I,
              weak_all(akt(_,_,_),weak_all(performed(_,_,_,_),schedule_pre)),
             a_left(_,I,
                 weak_all(state(_,_,_),weak_all(akt(_,_,_),schedule_calc)),
                 v_left_all(schedule_post((A \- C)))))))
        <- (schedule -> (A \- C))),
        false).                          % No act possible
```

```
ready_sch <= check_assumptions(Ass) -> (Ass \- _).
ready_sch <= (plot <- true),axiom(_,_,_).

check_assumptions([]).
check_assumptions([A|R]) :-
        functor(A,F,_),
        (F == state ;
         F == performed),
        check_assumptions(R).

schedule_pre <=
        (a_right(_,math) <- true),
        true_right,
        d_right(_,schedule_pre),
        v_right(_,schedule_pre,schedule_pre),
        forall_r(_,schedule_forall),
        findall_right(schedule_findall),
        constr_right(_).

schedule_calc <=
        (mathing <- true),
        (right_sch(schedule_calc),
         axiom_sch(_,_,_),
         left_sch(schedule_calc)).

schedule_post(PT) <=
    (d_left(change(_,_),_,(A \- C)) <- (schedule_post(PT) -> (A \- C))),
    (forall_l_generate_all(_,_,(A \- C),r,2)
     <- (schedule_post(PT) -> (A \- C))),
    PT.

keysort_right(keysort(L,L1)) <=
        keysort(L,L1) -> (_ \- keysort(L,L1)).

%%%--------------------------
%%% Strategies used by schedule_calc
right_sch(PT) <= (not(functor(C,state,_)) -> (_ \- C)).
right_sch(PT) <=
        relations,
        findall_right(schedule_findall),
        bagof_right(PT),
        forall_r(C,schedule_forall),
        constr_right(C),
        v_right(_,PT,PT),
        a_right(_,v_left_all(PT)),
        o_right(_,_,PT),
        true_right,
        d_right_sch(_,PT),
        keysort_right(_).

left_sch(PT) <=
        forall_l_generate_all(C,I,PT,r,2),
        v_left(_,_,PT),
        a_left(_,_,PT,v_left_all(PT)),
        o_left(_,_,PT,PT),
        d_left_sch(_,_,PT),
        pi_left(_,_,PT).

axiom_sch(T,C,I) <= axiom(T,C,I).

d_left_sch(C,I,PT) <= (not(functor(C,performed,4)) -> (I@[C|_] \- _)).
d_left_sch(C,I,PT) <= (not(functor(C,state,3)) -> (I@[C|_] \- _)).
d_left_sch(C,I,PT) <= d_left(C,I,PT).

d_right_sch(C,PT) <= not(functor(C,plot,_)) -> (_ \- C).
d_right_sch(C,PT) <= d_right(C,PT).

schedule_findall <=
        (a_right(_,(A \- C)) <- (math -> (A \- C))),
        right_sch_forall(schedule_findall),
        axiom_sch(_,_,_),
        relations.

%%% --------------------------------------------
%%% Some general strategies
mathing <= (var(C) ; functor(C,n,1) ; functor(C,quote,1)) -> (_ \- C).
mathing <= math.

constr_right(constr(_)) <=
        constraint(Constr) -> (_ \- constr(Constr)).

%%% Overwrites the definition in math.rul.
if_statement <= schedule_calc.

%%%---------------------------
%%% Strategy for plotting a plan.
plot <= (_ \- plot).
plot <= d_right(_,v_right(_,findall_right(axiom(_,_,_)),presenting)).

presenting <=
        plot_plan(L)
        -> (_ \- present_plans(L)).

%%% PROVISOS
constructor(findall,3).
constructor(bagof,4).
constructor(forall,2).
constructor(constr,1).
constructor(keysort,2).

append([],L,L).
append([F|R],L,[F|R1]) :- append(R,L,R1).

%%%===================
%%% CONSTRAINTS
constraint((X =< Y)) :- prolog:when((ground(X),ground(Y)),X =< Y).
constraint((X >= Y)) :- prolog:when((ground(X),ground(Y)),X >= Y).
constraint((X > Y)) :- prolog:when((ground(X),ground(Y)),X > Y).
constraint((X < Y)) :- prolog:when((ground(X),ground(Y)),X < Y).
constraint((X = Y)) :- not(functor(Y,max,1)),not(functor(Y,min,1)),
        prolog:when(ground(Y),X is Y).
constraint((X = max(L))) :- user:max(L,X).  % max(X,L): X is max of L's elements
constraint((X = min(L))) :- user:min(L,X).  % min(X,L): X is min of L's elements
```

Appendix E:

The Mathematical Rules

```
%%% Rewritten rules
%%% For evaluating user-defined functions
d_left_fun(T,I,PT) <=
        atom(T),
        not(num(T)),
        not(functor(T,'.',2)),
        definiens(T,Dp,N),
        N > 0,                              % a function is never absurd
        (PT -> (I@[Dp|Y] \- C))
        -> (I@[T|Y] \- C).

%%% Overwrites the pi_left rule in rules.rul
pi_left((pi X \ C),I,PT) <=
  inst(X,C,C2),
  (PT -> (I@[C2|R] \- C1))
  -> (I@[(pi X \ C)|R] \- C1).

%%%================================
%%% New rules
```

```
numax(T,C) <=
        (num(T) ; functor(T,'.',2) ; T == [] ; functor(T,quote,1)),
         unify(C,T)
         -> (I@[T|_] \- C).

integer_left(int(X),I,PT) <=
        (PT -> (I@[X|R] \- n(X1))),
        Y is integer(X1),
        (PT -> (I@[n(Y)|R] \- C))
        -> (I@[int(X)|R] \- C).

mod_left(mod(A,B),I,PT) <=
        (PT -> (I@[A|R] \- n(A1))),
        (PT -> (I@[B|R] \- n(B1))),
        X is A1 mod B1,
        (PT -> (I@[n(X)|R] \- C))
        -> (I@[(A mod B)|R] \- C).

add_left(+(A,B),I,PT) <=
        (PT -> (I@[A|R] \- n(A1))),
        (PT -> (I@[B|R] \- n(B1))),
        X is A1 + B1,
        (PT -> (I@[n(X)|R] \- C))
        -> (I@[(A + B)|R] \- C).
mul_left(*(A,B),I,PT) <=
        (PT -> (I@[A|R] \- n(A1))),
        (PT -> (I@[B|R] \- n(B1))),
        X is A1 * B1,
        (PT -> (I@[n(X)|R] \- C))
        -> (I@[(A * B)|R] \- C).
div_left(/(A,B),I,PT) <=
        (PT -> (I@[A|R] \- n(A1))),
        (PT -> (I@[B|R] \- n(B1))),
        X is A1 / B1,
        (PT -> (I@[n(X)|R] \- C))
        -> (I@[(A / B)|R] \- C).
minus_left(-(A,B),I,PT) <=
        (PT -> (I@[A|R] \- n(A1))),
        (PT -> (I@[B|R] \- n(B1))),
        X is A1 - B1,
        (PT -> (I@[n(X)|R] \- C))
        -> (I@[(A - B)|R] \- C).

sum_left(sum(_),I,PT,PT1) <=
        (functor(L,'.',2),strip_n(L,L1) ;
         not(functor(L,'.',2)),
          (i([X],(PT1 -> (I@[L|R] \- n(X)))) -> L1)),
        add_list(L1,S),
        (PT -> (I@[n(S)|R] \- C))
        -> (I@[sum(L)|R] \- C).

strip_n([],[]).
strip_n([n(X)|R],[X|R1]) :-
        strip_n(R,R1).
strip_n([F|R],[F|R1]) :-
        not(functor(F,n,1)),
        strip_n(R,R1).

add_list(L,N) :- add_list(L,0,N).

add_list([],N,N).
add_list([F|R],N,Answ) :-
        N1 is N + F,
        add_list(R,N1,Answ).

%%% For closures
defun_left(defun(X),I,PT) <=
        (PT -> (I@[X|Y] \- C))
        -> (I@[defun(X)|Y] \- C).
```

```
less_than <=
        X < Y -> (_ \- n(X) < n(Y)).
greater_than <=
        X > Y -> (_ \- n(X) > n(Y)).

greater_or_equal_than <=
        X >= Y -> (_ \- n(X) >= n(Y)).
equal_or_less_than <=
        X =< Y -> (_ \- n(X) =< n(Y)).

%%%===================================
%%% New strategies
math <= math(math).

relations <=
        less_than,
        greater_than,
        greater_or_equal_than,
        equal_or_less_than.

math(PT) <=
        (system_defined(math(PT)) <- true),
         eager(math(PT)).

system_defined(PT) <=
        numax(_,_),
        integer_left(_,_,PT),
        mod_left(_,_,PT),
        add_left(_,_,PT),
        mul_left(_,_,PT),
        div_left(_,_,PT),
        minus_left(_,_,PT),
        sum_left(_,_,PT,sumstrat).

%%% Default summing strategy. Can be replaced.
sumstrat <= math.

%%%-----------------------------
%%% To handle user-defined functions.
eager(PT) <=
        ((closure(_,PT) <- true),
         user_defined(C,I,PT)).

user_defined(C,I,PT) <=
        d_left_fun(C,I,handle(PT)).

handle(PT) <=
        pi_left(_,_,handle(PT)),
        a_left(_,_,if_or_eval(PT),PT),         % if-statement, or evaluation
        and_l(_,_,eager1(PT)),                 % case-statement
        PT.

if_or_eval(PT) <=
        v_right(_,if_or_eval(PT),if_or_eval(PT)),
        a_right(_,PT),                         % evaluation
        relations,                             % if-statement
        d_right(_,if_statement).               % if-statement

if_statement <= gcla.                          % default strategy

%%%-----------------------------
%%% To handle a closure
closure(I,PT) <= (I@[defun(_)|R] \- _).
closure(I,PT) <=
        contr_l(_,I,defun_left(defun(_),I,closure1(I,PT))).

closure1(I,PT) <=
        pi_left(_,_,closure1(I,PT)),
        a_left(_,I,axiom(_,C,I1),weak_l(C,_,handle(PT))),
        and_l(_,I,closure1(I,PT)).
```

```
%%%==============================
%%% Provisos
constructor(int,1).
constructor(<,2).
constructor(>=,2).
constructor(>,2).
constructor(=<,2).
constructor('*',2).
constructor('/',2).
constructor('+',2).
constructor('-',2).
constructor(n,1).
constructor(quote,1).
%constructor([],0).
%constructor('.',2).
constructor(defun,1).

num(X) :- functor(X,n,1).
```

Appendix F:

The General Library Rules

```
:- multifile(constructor/2).

true_right <= (_ \- true).

false_left(I) <= functor(C,false,0) -> (I@[C|_] \- _).

axiom(T,C,I) <=
  term(T),
  term(C),
  unify(T,C)
  -> (I@[T|_] \- C).
axiom(T,I) <=
  term(T),
  term(C),
  unify(T,C)
  -> (I@[T|_] \- C).

d_right(C,PT) <=
  atom(C),
  clause(C,B),
  (PT -> (P \- B))
  -> (P \- C).

d_left(T,I,PT) <=
  atom(T),
  definiens(T,Dp,N),
  (PT -> (I@[Dp|Y] \- C))
  -> (I@[T|Y] \- C).

a_left((A -> C1),I,PT,PT1) <=
  (PT -> (I@Y \- A)),
  (PT1 -> (I@[C1|Y] \- C))
  -> (I@[(A -> C1)|Y] \- C).

a_right((A -> C),PT) <=
  (PT -> ([A|P] \- C))
  -> (P \- (A -> C)).

o_right(1,(C1 ; C2),PT) <=
  (PT -> (Ass \- C1))
  -> (Ass \- (C1 ; C2)).
```

```
o_right(2,(C1 ; C2),PT) <=
  (PT -> (Ass \- C2))
  -> (Ass \- (C1 ; C2)).

o_left((A1 ; A2),I,PT1,PT2) <=
  (PT1 -> (I@[A1|R] \- C)),
  (PT2 -> (I@[A2|R] \- C))
  -> (I@[(A1 ; A2)|R] \- C).

v_right((C1,C2),PT1,PT2) <=
  (PT1 -> (A \- C1)),
  (PT2 -> (A \- C2))
  -> (A \- (C1,C2)).

v_left((C1,C2),I,PT) <=
  (PT -> (I@[C1,C2|Y] \- C))
  -> (I@[(C1,C2)|Y] \- C).

pi_left((pi X \ C),I,PT) <=
  inst(X,C,C1),
  (PT -> (I@[C1|R] \- Concl))
  -> (I@[(pi X \ C)|R] \- Concl).

%%% Additional rules that can be incorporated if the
%%% programmer wishes to do so.

sigma_right((X^C),PT) <=
  inst(X,C,C1),
  (PT -> (A \- C1))
  -> (A \- (X^C)).

%%% Principal way to implement comma-left without contraction as
%%% a strategy on top of the ordinary GCLA v_left-rule.
%and_l((A,B),I,I1,PT) <= append(I,[A],I1) -> (I@[(A,B)|_] \- _).
%and_l((A,B),I,I1,PT) <=
%       v_left((A,B),I,weak_l(B,I1,PT)),
%       v_left((A,B),I,weak_l(A,I,PT)).

%%% More efficient version as a new rule, better to use.
and_l((A,B),I,PT) <=
       ((PT -> (I@[A|R] \- C)) ;
        (PT -> (I@[B|R] \- C)))
       -> (I@[(A,B)|R] \- C).

weak_l(T,I,PT) <=
       (PT -> (I@R \- C))
       -> (I@[T|R] \- C).

contr_l(T,I,PT) <=
       (PT -> (I@[T,T|R] \- C))
       -> (I@[T|R] \- C).

add_l(add_def(X,Y),I,PT) <=
       add(X),
       (PT -> (I@[Y|R] \- C)) ->
       (I@[add_def(X,Y)|R] \- C).
rem_l(rem_def(X,Y),I,PT) <=
       rem(X),
       (PT -> (I@[Y|R] \- C)) ->
       (I@[rem_def(X,Y)|R] \- C).
add_r(PT) <=
       add(X),
       (PT -> (A \- Y)) ->
       (A \- add_def(X,Y)).
rem_r(PT) <=
       rem(X),
       (PT -> (A \- Y)) ->
       (A \- rem_def(X,Y)).

%%%------------------------
```

```
%%% Provisos

constructor(';',2).
constructor((->),2).
constructor(true,0).
constructor(false,0).
constructor(',',2).
constructor(pi,1).
constructor(contr,1).
constructor(add_def,2).
constructor(rem_def,2).

%%========================
% Strategies
gcla <= arl.

arl <= axiom(_,_,_),right(arl),left(arl).
alr <= axiom(_,_,_),left(alr),right(alr).
lra <= left(lra),right(lra),axiom(_,_,_).

ar <= axiom(_,_,_),right(ar).
al <= axiom(_,_,_),left(al).

ra <= right(ra), axiom(_,_,_).
la <= left(la), axiom(_,_,_).

rl <= right(rl), left(rl).
lr <= left(lr), right(lr).

r <= right(r).
l <= left(l).

right(C,PT) <=
        user_add_right(C,PT),
        v_right(C,PT,PT),
        a_right(C,PT),
        o_right(_,C,PT),
        true_right,
        d_right(C,PT).
right(PT) <=
        user_add_right(_,PT),
        v_right(_,PT,PT),
        a_right(_,PT),
        o_right(_,_,PT),
        true_right,
        d_right(_,PT).

c_right(PT) <=
        v_right(_,PT,PT),
        a_right(_,PT),
        o_right(_,_,PT),
        true_right.
c_right(C,PT) <=
        v_right(C,PT,PT),
        a_right(C,PT),
        o_right(_,C,PT),
        true_right.

left(PT) <=
        user_add_left(_,_,PT),
        false_left(_),
        v_left(_,_,PT),
        a_left(_,_,PT,PT),
        o_left(_,_,PT,PT),
        d_left(_,_,PT),
        pi_left(_,_,PT).
left(C,I,PT) <=
        user_add_left(C,I,PT),
        false_left(I),
        v_left(C,I,PT),
        a_left(C,I,PT,PT),
        o_left(C,I,PT,PT),
        d_left(C,I,PT),
        pi_left(C,I,PT).

c_left(PT) <=
        false_left(_),
        v_left(_,_,PT),
        a_left(_,_,PT,PT),
        o_left(_,_,PT,PT),
        d_left(_,_,PT),
        pi_left(_,_,PT).
c_left(C,I,PT) <=
        false_left(I),
        v_left(C,I,PT),
        a_left(C,I,PT,PT),
        o_left(C,I,PT,PT),
        d_left(C,I,PT),
        pi_left(C,I,PT).
```