

Programação em Lógica

Aplicação para horários

Nelson Jorge Silva Rodrigues
Ricardo Jorge Marques Veloso



Universidade do Porto

Faculdade de Engenharia

FEUP

Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias, s/n, 4200-465 Porto, Portugal

Dezembro de 2002

Programação em Lógica

Damas3D

Nelson Jorge Silva Rodrigues
Ricardo Jorge Marques Veloso



Universidade do Porto
Faculdade de Engenharia
FEUP

Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias, s/n, 4200-465 Porto, Portugal

Novembro de 2002

Resumo

Relatório da execução da segunda proposta de trabalho da cadeira de Programação em Lógica do primeiro semestre do terceiro ano da Licenciatura em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto.

A aplicação tem como objectivo criar horários atendendo a restrições impostas.

È apresentado neste relatório as opções de implementação descrevendo os pormenores mais importantes e por fim, as conclusões retiradas pelos autores do trabalho.

Descrição da aplicação

Este trabalho consistiu em desenvolver uma aplicação para obtenção de horário impondo restrições, usando para isso a biblioteca *clpfd* para restrições em domínios finitos fornecida pelo *SICStus Prolog*.

As restrições obrigatoriamente impostas (“*Hard Constraints*”), geram horários sem eventos sobrepostos, eventos em salas com as características necessárias a cada evento e com a capacidade para os alunos inscritos num evento, evitar que eventos que tenham o mesmo aluno inscrito decorram ao mesmo tempo.

As restrições opcionais (“*Soft Constraints*”), melhoram os horários, tentando evitar que um aluno tenha aulas ao último tempo, um aluno ter mais de dois eventos consecutivos e um aluno ter uma única aula num dia.

Por fim pretende-se a impressão da informação para o ecrã e para um ficheiro de texto.

Na aplicação começa-se por criar duas listas, uma de *slots* e outra de salas, com o tamanho do número de eventos.

Para a lista de *slots* é imposto um domínio de 1 até ao número de *slots*.

A estratégia usada para restringir eventos a salas com as características requeridas e a salas que tenham a capacidade para os alunos inscritos no evento foi, criar para cada evento uma lista das salas que satisfazem estas restrições. A restrição é imposta por *in_set*, que garante que o evento apenas pode decorrer numa das salas do *Set*.

Para estas restrições é chamado o predicado *room_constraints* que recebe a lista de salas onde cada posição desta lista, que corresponde ao número de cada evento, vai sofrer estas operações de restrições.

Na restrição de eventos sobrepostos percorre-se a lista de *slots* e a lista de salas, para cada posição destas listas em simultâneo, obrigando que posições simultâneas superiores nas mesmas listas não infrinjam as restrições impostas. Obriga eventos com o mesmo aluno inscrito a ocorrerem em *slots* diferentes. No caso em que esta restrição não se verifique os eventos são postos em salas diferentes ou *slots* diferentes.

Para a restrição de eventos sobrepostos é chamado o predicado *overlapping_constraints* que recebe a lista de *slots* e lista de salas a serem percorridas. No decorrer deste predicado há uma chamada a *student_constraints* que verifica a intersecção de alunos nos eventos a serem restringidos no durante *overlapping_constraints*. Neste predicado não há a implementação de nenhuma restrição.

No final das chamadas de *room_constraints* e de *overlapping_constraints*, com as restrições impostas, os predicados *labeling* são chamados para cada uma das listas criadas, dando valores às variáveis de domínio e originando um horário possível, atendendo às restrições impostas. *ffc*, foi usado como opção do *labeling*, conseguindo assim melhor eficiência e obtenção de resultados sobre problemas até então não conseguidos.

Com o horário criado, segue-se a sua impressão das salas, com eventos que nelas irão decorrer, no ecrã e num ficheiro de texto. É imprimido também informação sobre o tempo de execução da aplicação, e ainda para o ficheiro de texto a data e hora actual. Para estas informações adicionais foi utilizada a biblioteca *system*.

Perspectivas ao desenvolvimento

Neste trabalho tem certamente possível muitos melhoramentos, como a implementação de *Soft Constraints*, que faria esta aplicação bastante optimizada.

Outro melhoramento é a impressão de horário por alunos.

Comentários ao desenvolvimento

Foram encontradas dificuldades no ponto de partida para a implementação da aplicação, assim como na investigação das funcionalidades de predicados utilizados em restrições. O trabalho tornou-se ainda mais complicado com o aparecimento de propostas não dadas a conhecer inicialmente.

Conclusões

Apesar de não muita experiência com *prolog*, e nenhuma na programação de restrições, esta mostrou ser uma técnica que, além de poder simplificar a resolução de problemas complexos, é bastante poderosa.

Bibliografia

- Leon Sterling, Ehud Shapiro. The Art of Prolog, 2a edição. MIT Press, 1994.
- Adventure In Prolog by AMZI –
<http://oopweb.com/Prolog/Documents/AdventureInProlog/VolumeFrames.html>

Anexo

```
/*
*****
*/
/* Nelson Jorge Silva Rodrigues      ei00070 */
/* Ricardo Jorge Marques Veloso     ei00125 */
*****
*/

:- use_module(library(clpfd)).
:- use_module(library(lists)).
:- use_module(library(system)).

campo( 6 ).

i :- inicio.

inicio :-

    write('Ficheiro a ler: '),
    read(File),nl,
    reconsult(File),

    write('Ficheiro a gravar: '),
    read(Sav_file),

    open( Sav_file, append, Out ), assert( stream(Out) ),

    datetime(datetime(Year, Month, Day, Hour, Min, Sec)),
    format(Out, '***Date: ~p~p~p   Time: ~ph~pm~ps***/n\n', [Year, Month,
Day, Hour, Min, Sec] ),

    statistics( runtime, [Start, _] ),

    problem( N_events, N_rooms, N_features, N_students ),
    slots( _, _, N_slots ),

    length( Slots_list, N_events ), % cria lista para os blocos
    length( Rooms_list, N_events ), % cria lista para as salas

    domain( Slots_list, 1, N_slots ),

    room_constraints( Rooms_list, N_events, N_rooms ),
    /*event_constraints( Slots_list, N_rooms, N_slots ),
    event_constraints( Rooms_list, N_slots, N_rooms ),*/
    overlapping_constraints( N_events, Slots_list, Rooms_list ),
```

```

labeling( [ffc], Slots_list ),
labeling( [ffc], Rooms_list ),

statistics( runtime, [End, _] ),

horario( Slots_list, Rooms_list, N_rooms, N_events ),

Time is End - Start,
escreve('Elapsed time: '),escreve(Time),escreve('ms\n\n\n'),

close(Out),

retractall(stream(X)).

horario( Slots, Rooms, 0, _).
horario( Slots, Rooms, N_room, N_events ) :-
    N2 is N_room - 1,
    horario( Slots, Rooms, N2, N_events ),
    %write('TOU NO HORARIO'),nl,
    gera_lista( Slots, Rooms, N_room, N_events, Lista_solucoes ),
    keysort( Lista_solucoes, Lista_ordenada ),
    escreve('Room: '),escreve(N_room),
    slots(_, N_times, N_slots ),escreve('\n'),
    print_time( N_times ),
    print_horario( Lista_ordenada, 1, N_slots, 0 ),
    escreve('\n\n\n').

gera_lista( _, _, _, 0, [] ).
gera_lista( [S | Resto], [N_room | Resto2], N_room, N_event, [ S-N_event | Z ] ) :-
    N is N_event - 1,
    gera_lista( Resto, Resto2, N_room, N, Z ).
gera_lista( [_ | Resto], [_ | Resto2], N_room, N_event, Z ) :-
    N is N_event - 1,
    gera_lista( Resto, Resto2, N_room, N, Z ).

print_time( 0 ) :-
    escreve('      ').
print_time( N_times ) :-
    N is N_times - 1,
    print_time( N ),
    escreve('Time '),escreve( N_times ), escreve(' ').

print_horario( _, Slots, N_slots, _ ) :-
    S is N_slots + 1,

```

```

S == Slots.
print_horario( [Slot-N_event | Resto], Slot, N_slots, Dia ) :-
%       write( Slot ), write( ' evento: ' ), write(N_event),nl,

muda_dia(Dia, Slot, D, N_event),
%write(N_event),write(' '),

S is Slot + 1,
print_horario( Resto, S, N_slots, D ).
print_horario( Resto, Slot, N_slots, Dia ) :-

muda_dia(Dia, Slot, D, '-'),escreve(' '),
%escreve('-'),escreve(' '),

S is Slot + 1,
print_horario( Resto, S, N_slots, D ).

muda_dia( Dia, Slot, D, N_event ) :-
(
    slots( _, N_times, _ ),
    N is Slot + N_times - 1,
    M is N mod N_times,
    M == 0,
    D is Dia + 1,
    escreve('\nDay '),escreve(D),escreve(': ')
    ;
    D = Dia
),
escreve_evento(N_event).

escreve(X) :-
    stream(Out),write(X),write(Out,X).

escreve_evento(X) :-
    stream(Out),write(X),write(Out,X),
    (
        number(X),
        get_size(X, Size),
        campo( Campo ),
        S is Campo - Size,
        escreve_espacos(S, Out)
        ;
        true
    ).

```

```

get_size( X, Size ) :-
    X < 10, Size = 0.
get_size( X, Size ) :-
    X2 is X // 10, get_size(X2, S), Size is S + 1.

escreve_espacos( 0, _ ).
escreve_espacos( Size, Stream ) :-
    %write('TOU A ESCREVER ESPACOS\n'),
    write(' '),write(Stream, ' '),
    N is Size - 1,
    %      write(N),
    escreve_espacos( N, Stream ).

/*****
/*      Event constraints      */
*****/

/*event_constraints( Lista, Count, N_max ) :-
    event_constraints( 1, Lista, Count, N_max ).

event_constraints( N_max, _, _, N_max ) :- !.
event_constraints( Val, Lista, Count, N_max ) :-
    count( Val, Lista, #=<=, Count ),
    V is Val + 1,
    event_constraints( V, Lista, Count, N_max ).*/

/*****
/*      Room constraints      */
*****/

% ATENÇÃO: Lista_rooms fica invertida devido a comecar as restricoes pelo
ultimo evento

room_constraints( _, 0, _ ) :- !.
room_constraints( [ Room | Z ], Event_number, N_rooms ) :-
    %write('TOU NO ROOM_CONSTRAINTS'),nl,
    event( Event_number, Students_list, Event_feats_list ),

    length( Students_list, X ),

    %write('Event number: '),write(Event_number),nl,

    room_event_constraints( Possible_rooms_list, N_rooms, X, Event_feats_list, [] ),
    %write(Possible_rooms_list),nl,

```

```

list_to_fdset( Possible_rooms_list, Set ),
Room in_set Set,

Next_event is Event_number - 1,
room_constraints( Z, Next_event, N_rooms ).

room_event_constraints( Possible_rooms_list, 0, _, _, Possible_rooms_list ) :- !.
room_event_constraints( Possible_rooms_list, Room_number, Students,
Event_feats_list, Aux_list ) :-
    room( Room_number, Room_size, Room_feats_list ),
    Next_room is Room_number - 1,
    (
        Room_size >= Students,
        intersection( Room_feats_list, Event_feats_list, Intersection ),
        length( Event_feats_list, Y ),
        length( Intersection, Z ),
        Y == Z,
        %write('Room number: '),write(Room_number),nl,
        append( [Room_number], Aux_list, Aux ),
        room_event_constraints( Possible_rooms_list, Next_room, Students,
Event_feats_list, Aux )
    );
    room_event_constraints( Possible_rooms_list, Next_room, Students,
Event_feats_list, Aux_list )
).

/*****
/*      Event constraints      */
*****/

/*event_constraints( Lista, Count, N_max ) :-
    event_constraints( 1, Lista, Count, N_max ).

event_constraints( N_max, _, _, N_max ) :- !.
event_constraints( Val, Lista, Count, N_max ) :-
    count( Val, Lista, #=<, Count ),
    V is Val + 1,
    event_constraints( V, Lista, Count, N_max ).*/

/*****
/*      Overlapping constraints      */
*****/

% n deixar sobrepor eventos
% para cada evento percorrer as listas todas à procura a impor restricoes

% Para cada evento

```

```

overlapping_constraints( 0, _, _ ) :- !.
overlapping_constraints( N_eventos, Lista_slots, Lista_rooms ) :-
    Evento is N_eventos - 1,
    overlapping_constraints( Evento, Lista_slots, Lista_rooms ),
    overlapping( N_eventos, Evento, Lista_slots, Lista_rooms ).

% comparar para todos os outros e impor restricoes
overlapping( _, 0, _, _ ) :- !.
overlapping( Evento_A, Evento_B, Lista_slots, Lista_rooms ) :-
    Evento is Evento_B - 1,

    nth( Evento_A, Lista_slots, Slot_A ),
    nth( Evento_B, Lista_slots, Slot_B ),
    nth( Evento_A, Lista_rooms, Room_A ),
    nth( Evento_B, Lista_rooms, Room_B ),
    (
        student_constraints( Evento_A, Evento_B ),
        Slot_A #\= Slot_B % caso em que ha alunos repetidos nos 2 eventos
        ;
        Slot_A #\= Slot_B #\ Room_A #\= Room_B
    ),
    overlapping( Evento_A, Evento, Lista_slots, Lista_rooms ).

/*****
/*      Students constraints      */
*****/

% n pode haver alunos repetidos nos eventos em slots iguais

student_constraints( Evento_A, Evento_B ) :-
    event( Evento_A, Alunos_A, _ ),
    event( Evento_B, Alunos_B, _ ),

    intersection( Alunos_A, Alunos_B, Alunos ),
    %write(' students intersection: '),write(Alunos),nl,
    length( Alunos, N ),
    N \= 0. % n ha interseccao => n ha alunos repetidos

/*****
/*  cria lista inteseccões      */
*****/

intersection([X|Y],M,[X|Z]) :- member(X,M), intersection(Y,M,Z).
intersection([X|Y],M,Z) :- \+ member(X,M), intersection(Y,M,Z).
intersection([],M,[]).

```