Exam Guidelines – Declarative Languages

NAME: PROGRAMME OF STUDY:
ROOM: PC: LOGIN:

- You are only allowed to use the following materials:
  - Copies of the course notes and slides, where applicable. Handwritten notes on these handouts are allowed.
  - "Prolog Programming for Artificial Intelligence" by I. Bratko.
  - "Introduction to Haskell" by T. Schrijvers.
  - Digitally: All materials (manuals, slides, etc.) made available through esystant.
- The duration of the examen is at most 3 hours.
- Solving only one of the two programming tasks is **not sufficient** to succeed for the exam.
- Read the **questions** carefully.
- Go to the folder `/localhost/examen/loginnaam/`. It contains important files. Backups of these files are put in the map `/localhost/examen/loginnaam/OPGAVEN`. The backups are readonly.

  For Prolog, you get the files `myprolog.pl` and `esystant.pl`. Put your Prolog code in the file `myprolog.pl`. The tests in `esystant.pl` can be run by the command `swipl -f esystant.pl`. If needed there can be more Prolog files with example facts related to the question.

  For Haskell, you add your code to `MyHaskell.hs`. To test your code use `runhaskell MyHaskellTest.hs`.

  Do not forget to save the source code before running the tests.
- At the end of the exam you submit the two files (`myprolog.pl` and `MyHaskell.hs`):
  - put your name, student number and programme of study at the beginning of the two files;
  - make sure that the two files are in the correct directory, namely in `/localhost/examen/loginnaam/`;
  - open a terminal and execute the command `submitfiles_dt`. It will check whether the necessary files exist and these files are then submitted;
  - log out and hand in your exam sheets.

- **Getting started**
  For Prolog, you are allowed to use swipl, swish and/or e-systant.
  - swipl: open a terminal and type in `swipl` to start swipl.
  - swish: open a terminal and type in `startswish`. Do not close this terminal. This starts swish, after which you can open the graphical user interface on http://localhost:3050/ in a browser.
  - e-systant: Use your exam account to login on the e-systant exam server `http://esystant-exam.cs.kuleuven.be`. Do not forget to submit your exam with the procedure explained on the first page; **submitting through e-systant does not count**.

  For Haskell, you can use GHCi and/or e-systant.
  - GHCi: open a terminal and type in `ghci` to start the GHCi session.
  - e-systant: Use your exam account to login on the e-systant exam server `http://esystant-exam.cs.kuleuven.be`. Do not forget to submit your exam with the procedure explained on the first page; **submitting through e-systant does not count**.

- **Manuals**
  For the SWI-Prolog manual browse to the file
  `/localhost/examen/DOC/Manual-swi/www.swi-prolog.org/pldoc/refman/index.html`.
  You can also access the manual of SWI-Prolog by the following queries in swipl: `?- help(Topic).` or `?- apropos(Word).`

  For the Ghci manual browse to the file `/localhost/examen/DOC/Manual-haskell/base/index.html`. In this subdirectory you also find `Prelude.html` and `Data-List.html`.

  Furthermore, in e-systant, you can find additional digital material such as the haskell course text, slides, or swish notebooks under the header 'Documents' in the sidebar.

- **To avoid ... in your Prolog output**
  Use the following query in **swipl**.

```
?-current_prolog_flag(answer_write_options,K),
select(max_depth(_),K,L), L1 = [ max_depth(100)|L],
set_prolog_flag(answer_write_options,L1).
```

Rummikub Lite

NAAM: RICHTING:

Gerda is een grote fan van het bordspelletje Rummikub, en zou graag de uitkomst kunnen voorspellen. Ze vraagt jou dan ook om een tool te schrijven, die kan voorspellen welke speler gaat winnen. De bedoeling van het spel is om al je blokjes (elk bestaande uit een nummer en een kleur) op de tafel (het spelbord) te kunnen afleggen. Het bord bestaat uit rijen van blokjes (telkens minstens 3 blokjes lang), waarvan ofwel

- het nummer gelijk is, en de kleuren allemaal verschillen.
- de kleuren gelijk zijn en de nummers opeenvolgend zijn.

Concreet mag een speler op zijn beurt dus nieuwe rijtjes maken van zijn blokjes, zijn blokjes bij andere bestaande rijtjes toevoegen of een nieuw blokje uit de doos rapen (indien hij niets kan afleggen). Voor deze opgave vereenvoudigen we het spel lichtjes :

- Een speler mag een rijtje enkel langs de uiteindes uitbreiden, en kan dus geen rijen opbreken.
- Een speler mag maar exact 1 actie uitvoeren tijdens zijn beurt : ofwel voegt hij 1 blokje toe aan een bestaande rij, ofwel legt hij 1 nieuw rijtje (van lengte 3) op het bord, ofwel raapt hij 1 nieuw blokje.

Opdracht 1 : Valsspelers!

Voor dit deel van de opgave schrijf je een predicaat dat kan verifiëren of het spelbord legaal is. We willen namelijk zeker zijn dat er geen spelers vals spelen.

In deze opgave stellen we blokjes voor met `block(Number,Color)` termen. Voor `Number` gebruiken we gewoon de build-in integers van Prolog. Voor `Color` hebben we de volgende 4 termen als mogelijkheden : `red`, `blue`, `yellow` en `black`. Je mag er altijd van uitgaan dat **elk blokje uniek** is. Bijvoorbeeld, er bestaan meerdere rode blokjes, en meerdere blokjes met het nummer 5, maar er kan maar maximaal één rode 5 in het spel zitten.

Een rij stellen we vervolgens voor met `crow(Blocks)` voor een kleuren rij, of `nrow(Blocks)` voor een nummer rij, waarbij `Blocks` telkens de lijst met blokjes in de rij voorstelt. De tafel (het spelbord) ten slotte stellen we gewoon voor als een lijst van rijen.

Schrijf nu een predicaat `valid_table(Table)` wat voor elk van de rijen op het spelbord checkt of de rij geldig is. Meer concreet, een rij moet uit minimaal 3 blokjes bestaan. En bovendien

- In een `crow` moeten alle nummers identiek zijn, en de kleuren moeten allemaal verschillen.
- In een `nrow` moeten alle kleuren identiek zijn, en de nummers moeten opeenvolgend (**gesorteerd**) zijn.

```
?- Table1 = [ crow( [block(5,red) , block(5,yellow) , block(5,black)] )
            , nrow( [block(5,blue) , block(6,blue) , block(7,blue) , block(8,blue)] ) ],
   valid_table(Table1).
True
```

```
?- Table2 = [ crow( [block(5,red) , block(5,yellow)] ) ],
   valid_table(Table2).
False

?- Table3 = [ crow( [block(5,red) , block(5,yellow) , block(4,blue)] ) ],
   valid_table(Table3).
False

?- Table4 = [ nrow( [block(2,blue) , block(3,blue) , block(4,blue) , block(5,red)] ) ],
   valid_table(Table4).
False

?- Table5 = [ nrow( [block(5,blue) , block(6,blue) , block(8,blue)] ) ],
   valid_table(Table5).
False

?- Table6 = [ nrow( [block(5,blue) , block(7,blue) , block(6,blue)] ) ],
   valid_table(Table6).
False
```

Opdracht 2 : Play the Game

Voor deze opgave is het de bedoeling dat je een predicaat `play_game(P1,P2,Table,Bag)` schrijft om het spel te simuleren. Meer concreet zijn `P1` en `P2` de twee spelers, waarbij de linkse speler `P1` aan beurt is, voorgesteld door een `player(Blocks,Actions)` term. Hierbij is `Blocks` een lijst met de huidige blokjes van deze speler, en `Actions` is de lijst met **alle** opeenvolgende acties die de speler dit spel uitvoert. Het argument `Table` is het huidige spelbord en `Bag` is de lijst met de blokjes die geraapt kunnen worden. We beschouwen in de tests de 2 `Blocks` lijsten, de `Table` en `Bag` als gegeven, en de 2 `Actions` lijsten als de 'output' van het predicaat.

Zoals eerder vermeld, vereenvoudigen we het spel lichtjes, door spelers te beperken tot exact 1 actie per beurt. De actieve speler probeert dus telkens 1 van deze acties uit te voeren, waarbij dit toegevoegd wordt in zijn `Actions` lijst :

- `playrow(crow(Blocks))` : Maak een nieuwe rij `crow(Blocks)` van exact 3 blokjes lang (uit de voorraad van deze speler), en leg deze op tafel. Elke blokje in `Blocks` moet hetzelfde getal en een verschillende kleur bevatten. Sorteer deze rij met `sort`, om de testen te doen slagen!
- `playrow(nrow(Blocks))` : Maak een nieuwe rij `nrow(Blocks)` van exact 3 blokjes lang (uit de voorraad van deze speler), en leg deze op tafel. Elke blokje in `Blocks` dient dezelfde kleur te hebben en opeenvolgende (gesorteerde) getallen te bevatten.

- `playblock(Block,Row)` : Breid een bestaande rij op het spelbord uit tot de nieuwe rij `Row`, door 1 blokje `Block` uit de voorraad van de speler aan
  - een uiteinde van een bestaande `nrow` toe te voegen.
  - aan een bestaande `crow` toe te voegen en opnieuw te sorteren.
- `draw(Block)` : Enkel als de bovenstaande acties niet lukken, neem dan 1 nieuw blokje `Block`, het **eerste** blokje uit de `Bag`.

Je vindt een voorbeeld van de mogelijke acties in Tabel 1.

Het spel eindigt wanneer een van de spelers geen blokjes meer over heeft, waarbij deze speler dan het spel wint. Voeg hiervoor de finale actie `win` en `lose` toe voor de winnende en verliezende speler respectievelijk. Indien beide spelers geen blokjes meer hebben, of indien een speler een blokje moet trekken en de `Bag` leeg is, is het een gelijkspel en voeg je een actie `draw` toe voor beide spelers.

Merk op dat er geen prioriteit aan deze acties toegekend is. Het is dus de bedoeling dat je **alle mogelijke oplossingen** teruggeeft.

Let op dat het echter **niet** de bedoeling is om dezelfde actie meerdere keren terug te geven. Gebruik hiervoor best `sort/2` of `list_to_set/2`. Je mag ervan uit gaan dat elke blokje in het spel uniek is, en dat er bijgevolg dus ook op tafel niet meerdere dezelfde rijen kunnen liggen.

Voor de duidelijkheid : De volgorde van je acties, is **wel** belangrijk. Eerst blok A op tafel leggen en in je volgende beurt blok B leggen, is wel degelijk een andere oplossing dan eerst blok B en dan blok A op tafel leggen.

Onderstaande voorbeeldjes zijn geïndenteerd voor leesbaarheid.

```
?- P1Blocks = [block(1,red),block(2,red),block(3,red),block(2,blue)],
   P2Blocks = [block(5,red),block(5,yellow),block(5,black),block(4,red)],
   play_game(player(P1Blocks,P1Act),player(P2Blocks,P2Act),[],[block(5,blue)]).

P1Act = [ playrow(nrow([block(1, red), block(2, red), block(3, red)]))
        , draw(block(5, blue))
        , draw],
P2Act = [ playblock(block(4, red),
              nrow([block(1, red), block(2, red), block(3, red), block(4, red)]))
        , playblock(block(5, red),
              nrow([block(1, red), block(2, red), block(3, red), block(4, red), block(5, red)]))
        , draw]

P1Act = [ playrow(nrow([block(1, red), block(2, red), block(3, red)]))
        , draw(block(5, blue))
        , lose],
P2Act = [ playblock(block(4, red),
              nrow([block(1, red), block(2, red), block(3, red), block(4, red)]))
        , playrow(crow([block(5, black), block(5, red), block(5, yellow)]))
```

```
        , win]

P1Act = [ playrow(nrow([block(1, red), block(2, red), block(3, red)]))
        , draw(block(5, blue))
        , lose],
P2Act = [ playrow(crow([block(5, black), block(5, red), block(5, yellow)]))
        , playblock(block(4, red),
             nrow([block(1, red), block(2, red), block(3, red), block(4, red)]))
        , win]

?- P1Blocks = [block(2,red),block(3,red),block(4,red),block(2,blue),block(2,black)],
   P2Blocks = [block(5,red),block(6,red),block(8,blue),block(8,black)],
   play_game(player(P1Blocks,P1Act),player(P2Blocks,P2Act),[],
             [block(7,red),block(8,red),block(9,red)]).

P1Act = [ playrow(crow([block(2, black), block(2, blue), block(2, red)]))
        , draw(block(8, red))
        , playblock(block(8, red),
             nrow([block(5, red), block(6, red), block(7, red), block(8, red)]))
        , playblock(block(4, red),
             nrow([block(4, red), block(5, red), block(6, red), block(7, red), block(8, red)]))
        , playblock(block(3, red),
             nrow([block(3, red), block(4, red), block(5, red), block(6, red), block(7, red),
                   block(8, red), block(9, red)]))
        , win],
P2Act = [ draw(block(7, red))
        , playrow(nrow([block(5, red), block(6, red), block(7, red)]))
        , draw(block(9, red))
        , playblock(block(9, red),
             nrow([block(4, red), block(5, red), block(6, red), block(7, red), block(8, red),
                   block(9, red)]))
        , lose]

P1Act = [ playrow(crow([block(2, black), block(2, blue), block(2, red)]))
        , draw(block(8, red))
        , playblock(block(4, red),
             nrow([block(4, red), block(5, red), block(6, red), block(7, red)]))
        , playblock(block(8, red),
             nrow([block(4, red), block(5, red), block(6, red), block(7, red), block(8, red)]))
        , playblock(block(3, red),
             nrow([block(3, red), block(4, red), block(5, red), block(6, red), block(7, red),
```

```
                    block(8, red), block(9, red)]))
        , win],
P2Act = [ draw(block(7, red))
        , playrow(nrow([block(5, red), block(6, red), block(7, red)]))
        , draw(block(9, red))
        , playblock(block(9, red),
             nrow([block(4, red), block(5, red), block(6, red), block(7, red), block(8, red),
                   block(9, red)]))
        , lose]

P1Act = [ playrow(crow([block(2, black), block(2, blue), block(2, red)]))
        , draw(block(8, red))
        , playblock(block(4, red),
             nrow([block(4, red), block(5, red), block(6, red), block(7, red)]))
        , playblock(block(3, red),
             nrow([block(3, red), block(4, red), block(5, red), block(6, red), block(7, red)]))
        , draw],
P2Act = [ draw(block(7, red))
        , playrow(nrow([block(5, red), block(6, red), block(7, red)]))
        , draw(block(9, red))
        , draw]

P1Act = [ playrow(nrow([block(2, red), block(3, red), block(4, red)]))
        , draw(block(7, red))
        , playblock(block(7, red),
             nrow([block(2, red), block(3, red), block(4, red), block(5, red), block(6, red),
                   block(7, red)]))
        , draw(block(9, red))
        , playblock(block(9, red),
             nrow([block(2, red), block(3, red), block(4, red), block(5, red), block(6, red),
                   block(7, red), block(8, red), block(9, red)]))
        , draw],
P2Act = [ playblock(block(5, red),
             nrow([block(2, red), block(3, red), block(4, red), block(5, red)]))
        , playblock(block(6, red),
             nrow([block(2, red), block(3, red), block(4, red), block(5, red), block(6, red)]))
        , draw(block(8, red))
        , playblock(block(8, red),
             nrow([block(2, red), block(3, red), block(4, red), block(5, red), block(6, red),
                   block(7, red), block(8, red)]))
        , draw]
```

Tabel 1: Mogelijke Acties

Actie	Speler blokken	Tafel	Nieuwe Tafel
`playrow(crow(Row))`	4 red, 4 blue, 4 black		4 black, 4 blue, 4 red
`playrow(nrow(Row))`	4 red, 5 red, 6 red		4 red, 5 red, 6 red
`playblock(Block,NRow)`	7 red	4 red, 5 red, 6 red	4 red, 5 red, 6 red, 7 red
`playblock(Block,NRow)`	3 red	4 red, 5 red, 6 red	3 red, 4 red, 5 red, 6 red
`playblock(Block,NRow)`	4 red	4 black, 4 blue, 4 yellow	4 black, 4 blue, 4 red, 4 yellow

```
P1Act = [ playrow(nrow([block(2, red), block(3, red), block(4, red)]))
        , draw(block(7, red))
        , playblock(block(7, red),
             nrow([block(2, red), block(3, red), block(4, red), block(5, red), block(6, red),
                   block(7, red)]))
        , draw(block(9, red))
        , lose],
P2Act = [ playblock(block(5, red),
             nrow([block(2, red), block(3, red), block(4, red), block(5, red)]))
        , playblock(block(6, red),
             nrow([block(2, red), block(3, red), block(4, red), block(5, red), block(6, red)]))
        , draw(block(8, red))
        , playrow(crow([block(8, black), block(8, blue), block(8, red)]))
        , win]
```

Veel succes!