

NOM: \_\_\_\_\_

PG211

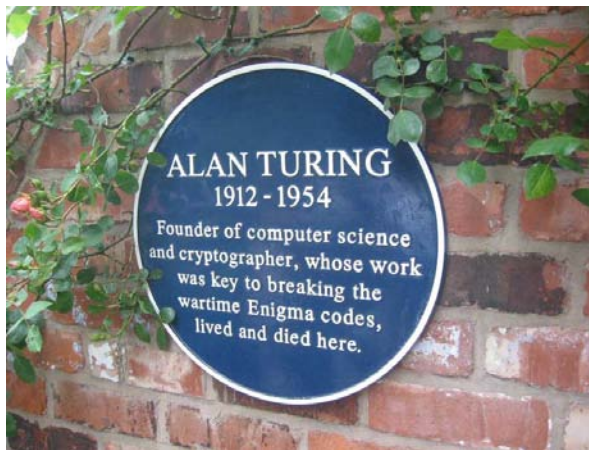
Prénoms: \_\_\_\_\_

Notes: I\_\_\_ II\_\_\_ III\_\_\_

Durée: deux heures et dix-sept secondes.

Accès Internet limité à la page personnelle de Paul Y Gloess (<http://www.enseirb.fr/~gloess> et toute sa descendance, y compris la documentation en ligne de CHIP ; accès au système CHIP autorisé ; communication (“e-mail”, “chat”, ou autre procédé) interdite.

On rappelle que **Prolog pur** (“pure Prolog”) n'utilise dans ses programmes que des clauses de Horn définies, écrites en syntaxe Edinburgh (adoptée ici) sous la forme “c.” ou la forme “c :- h<sub>1</sub>, ..., h<sub>n</sub>.” où la conclusion “c” et les hypothèses “h<sub>1</sub>”, ..., “h<sub>n</sub>” sont des littéraux positifs. Prolog pur n'autorise ni coupure, ni négation, ni symbole prédéfini tel que “=”, “+”, “\*”, ...; on utilisera la notation Edinburgh pour les listes: [ ] pour la liste vide; [E | L] pour le “cons” de E et L.



### I. [7 pts] Pure Prolog Xerox Turing machine

La machine de Turing est une machine à états finis, constituée d’une bande, infinie vers la gauche et la droite, et d’une tête de lecture/écriture, qui sert à lire ou écrire un symbole, pris dans un ensemble fini, dont le « blanc », représenté ici par le ‘#’. Elle exécute un programme à base de 3 instructions :

- stateIn : i\o, left, goto stateOut  
La machine étant dans l’état stateIn, la tête lit le symbole ‘i’, le remplace par le symbole ‘o’, puis se déplace d’une case (un symbole) vers la gauche ; la machine se met dans l’état stateOut ;
- stateIn : i\o, right, goto stateOut  
Comportement identique, sauf que la tête se déplace vers la droite ;
- stateIn : i|i, halt  
La machine étant dans l’état stateIn, lisant le symbole ‘i’, s’arrête.

Voici une machine réalisant l’image miroir d’une suite de symboles a, b, c, en détruisant l’original. Partant par exemple de la bande :

#####abcb#



Elle laisse la bande dans l’état :

bcba#\*\*\*\*#



La position de la tête de lecture/écriture est indiquée par le ▲.

Le programme utilisé est donné en page suivante.

```

init:   ##, right, goto lookForLetter
lookForLetter: ##, left, goto allDone
          *\*, right, goto lookForLetter
          a\*, left, goto copyA
          b\*, left, goto copyB
          c\*, left, goto copyC

copyA:  ##, left, goto outputA
          *\*, left, goto copyA

outputA: a\a, left, goto outputA
          b\b, left, goto outputA
          c\c, left, goto outputA
          #\a, right, goto data

copyB:  ##, left, goto outputB
          *\*, left, goto copyB

outputB: a\a, left, goto outputB
          b\b, left, goto outputB
          c\c, left, goto outputB
          #\b, right, goto data

copyC:  ##, left, goto outputC
          *\*, left, goto copyC

outputC: a\a, left, goto outputC
          b\b, left, goto outputC
          c\c, left, goto outputC
          #\c, right, goto data

data:   a\a, right, goto data
          b\b, right, goto data
          c\c, right, goto data
          ##, right, goto lookForLetter

allDone: *\*, left, goto allDone
          ##, halt
    
```

Semblable à copyA, outputA

Semblable à copyA, outputA

On se propose de traduire le programme ci-dessous en Prolog pur, en associant à chaque état un prédicat de même nom, d'arité 4, et à chaque instruction, une clause. On se limitera aux deux fragments encadrés. On représente la bande, plus exactement un voisinage de la tête de lecture/écriture, par un triplet [L, H, R] où :

- L est une liste des symboles situés à gauche de la tête, en commençant par le plus proche ;
- H est le symbole lu par la tête ;
- R est une liste des symboles situés à droite de la tête, dans l'ordre.

Exemple :

- 1abc0defg1  
▲
- L = [c, b, a, 1], H = 0, R = [d, e, f, g, 1]

Cette représentation se prête bien aux mouvements de la tête : un déplacement vers la droite (right) supprime le premier élément de R, et ajoute un élément au début de R ; un déplacement vers la gauche (left) supprime le premier élément de L, et ajoute un élément au début de L.

La sémantique des prédicats associés aux états est uniforme. Pour le prédicat `init`, par exemple, on a :

`init(Lin, Hin, Rin, TapeOut)`  $\stackrel{def}{\equiv}$  `[Lin, Hin, Rin]` représente l'état de la bande et de la tête à l'entrée (état init du programme) et `TapeOut` est un triplet qui représente l'état de la bande et de la tête à l'arrêt (halt).

```

init(L, '#', [H | R], Tape) :-
  lookForLetter(['#' | L], H, R, Tape).

lookForLetter(_____, Tape) :-
  allDone(_____, Tape).
lookForLetter(_____, Tape) :-
  lookForLetter(_____, Tape).
lookForLetter(_____, Tape) :-
  _____.
lookForLetter(_____, Tape) :-
  _____.
lookForLetter(_____, Tape) :-
  _____.

copyA(_____, Tape) :-
  _____.
copyA(_____, Tape) :-
  _____.

outputA(_____, Tape) :-
  _____.
outputA(_____, Tape) :-
  _____.
outputA(_____, Tape) :-
  _____.
outputA(_____, Tape) :-
  _____.

data(_____, Tape) :-
  _____.
data(_____, Tape) :-
  data_____.
data(_____, Tape) :-
  _____.
data(_____, Tape) :-
  _____.

allDone([H | L], *, R, Tape) :-
  allDone(L, H, [* | R], Tape).
allDone(L, '#', R, [L, '#', R]).

```

See [http://www.enseirb.fr/~gloess/enseignement/chip/2008\\_2009/test/turingTemplate.pl](http://www.enseirb.fr/~gloess/enseignement/chip/2008_2009/test/turingTemplate.pl) for a full program skeleton.

## II. [7 Pts] Pure Prolog Turing machine again

On veut maintenant exécuter un programme de Turing quelconque, passé en argument, comme une liste de quintuples de la forme [InputState, InputSymbol, OutputSymbol, InstructionType, OutputState] :

Instruction de Turing	Quintuple correspondant
$e1 : \#\backslash\#, \text{halt}$	[e1, #, unused, halt, unused]
$e1 : 1\backslash\#, \text{right, goto } e2$	[e1, 1, #, right, e2]
$e3 : \#\backslash 1, \text{left, goto } e4$	[e3, #, 1, left, e4]

$\text{turing}(\text{IS}, \text{IT}, \text{OS}, \text{OT}, \text{P}) \stackrel{\text{def}}{\equiv}$  OS est l'état final de la machine, OT, l'état final de la bande, après exécution du programme P à partir des états IS et IT.

$\text{instruction}(\text{S}, \text{T}, \text{P}, \text{I}) \stackrel{\text{def}}{\equiv}$  I est l'instruction du programme P à exécuter dans l'état S de la machine et l'état T de la bande et tête de lecture.

$\text{turing}(\text{IS}, \text{IT}, \text{OS}, \text{OT}, \text{P}, \text{I}) \stackrel{\text{def}}{\equiv}$   $\text{turing}(\text{IS}, \text{IT}, \text{OS}, \text{OT}, \text{P})$ , sachant que I est l'instruction à exécuter.

```
turing(IState, ITape, OState, OTape, Program) :-
    instruction(IState, ITape, Program, Instruction),
    turing(IState, ITape, OState, OTape, Program, Instruction).
instruction(State, [_ , H, _], Program,
    [State, H, OH, InstructionCode, OState]) :-
    member([State, H, OH, InstructionCode, OState], Program).

turing(IState, [L, H, R], IState, [L, H, R], _,
    [IState, H, _, halt | _]).

turing(IState, [_____, _____, _____], OState, OTape, Program,
    [IState, IH, OH, left, State]) :-
    turing(State, [_____], OState, OTape, Program).

%%% Fin de bande à gauche: on simule un blanc '#':
turing(IState, [[], IH, IR], OState, OTape, Program,
    [IState, IH, OH, left, State]) :-
    turing(State, [_____], OState, OTape, Program).

turing(IState, [_____, _____, _____], OState, OTape, Program,
    [IState, IH, OH, right, State]) :-
    turing(State, [_____], OState, OTape, Program).

%%% Fin de bande à droite: on simule un blanc '#':
turing(IState, [IL, IH, []], OState, OTape, Program,
    [IState, IH, OH, right, State]) :-
    turing(State, [_____], OState, OTape, Program).
```

### III. [6 pts] CLP search tree

On se place dans le cadre CLP( $Q_{lin}$ ), de la programmation logique avec contraintes linéaires en nombres rationnels. On reprend l'exemple introductif du cours, sous forme normalisée, dans la syntaxe CHIP:

$sumto(N, S) \text{ :- } 0 \wedge N, 0 \wedge S. \quad (sumtoZ)$

$sumto(N, S) \text{ :- } 1 \wedge \leq N, N \wedge \leq S, N-1 \wedge NP, S-N \wedge SN, \quad (sumtoP)$   
 $sumto(NP, SN).$

Dessiner l'arbre de recherche du but " $2 * N \wedge S, sumto(N, S)$ " en faisant apparaître la ou les solution(s), le ou les ensemble(s) de contraintes insatisfaisables. [On pourra utiliser les notations " $\wedge$ " pour " $\wedge$ ", " $\leq$ " pour " $\wedge \leq$ ", " $1 \leq N \leq S$ " pour " $1 \wedge \leq N, N \wedge \leq S$ "; ne pas oublier de numéroter les variables des clauses utilisées en fonction de la profondeur.]

Dans un souci d'économie, on ne copiera pas les contraintes héritées du nœud parent (et de ses ancêtres).

See [http://www.enseirb.fr/~gloess/enseignement/chip/2008\\_2009/test/sumto.pl](http://www.enseirb.fr/~gloess/enseignement/chip/2008_2009/test/sumto.pl) for sumto program and queries.