Département des Maths et Informatique

Master I, AD, outils formels

Corrigé type

I.
1. Faux
2. Faux
3. Vrai
4. Vrai
5. Vrai
6. Vrai
7. Vrai
8. Faux
9. Vrai
10. Faux

Exercice 1

```
fmod mots is
protecting INT .
protecting QID .
sort List .
subsort Qid < List .
op nil : -> List [ctor] .
op __ : List List -> List [ctor assoc id: nil] .
op words : List Int -> Int .
op wordoc : List Int -> Int .
op number : List -> Int .
op delete : List Int -> List .
var L : List .
var c : Qid .
var n : Int .
var i : Int .
ceq delete(c L, i) = delete(L, i - 1) if i > 0 .
ceq delete(c L, i) = c L if i == 0 .
eq delete(nil, i) = nil .
ceq words(c L, n) = words(L, n) if c == '  and L =/= nil .
ceq words(c L, n) = n if c == '  and L == nil .
ceq words(c L, n) = words(delete(L,wordoc(c L, 0)), n + 1) if c =/=
'  and L =/= nil .
eq words(nil, n) = n .
ceq words(c L, n) = n + 1 if c =/= ' and L == nil .
ceq wordoc(c L, i) = wordoc(L, i + 1) if c =/= ' and L =/= nil .
ceq wordoc(c L, i) = i if c == ' or L == nil .
eq number(L) = if L =/= nil then words(L, 0) else 0 fi .

endfm
```

Exercice 2

```
(omod Roundrobin is
protecting QID .
protecting INT .
sort list .
sort state .
subsort Qid < list .
subsort Qid < Oid .

ops wait ready run quit : -> state [ctor] .
op head : list -> Qid .
op delete : list -> list .
op add : list Qid -> list .
op nil : -> list [ctor] .
op __ : list list -> list [ctor assoc id: nil] .

vars P L T : Oid .
var l : list .
var n : Qid .
vars c q t : Int .
eq head(n l) = n .
eq head(nil) = nil .
eq delete(n l) = l .
eq delete(nil) = nil .
eq add(l, n) = l n .

class queue | List : list .
class time | quantum : Int, clock : Int .
class task | CPUtime : Int, State : state .

crl [turn] : < P : task | CPUtime : c, State : wait > < L : queue |
List : l > => < P : task | CPUtime : c, State : ready > < L : queue
| List : delete(l) > if P == head(l) .
rl [begin] : < P : task | CPUtime : c, State : ready > => < P : task
| CPUtime : c, State : run > .
crl [run] : < P : task | CPUtime : c, State : run > < T : time |
quantum : q, clock : t > => < P : task | CPUtime : c - 1, State :
run > < T : time | quantum : q - 1, clock : t + 1 > if q > 0 .
crl [quit] : < P : task | CPUtime : c, State : run > < T : time |
quantum : q, clock : t > => < P : task | CPUtime : c, State : quit >
< T : time  | quantum : 4, clock : t > if (q == 0 and c <= 0) .
crl [rewait] : < P : task | CPUtime : c, State : run > < T : time |
quantum : q, clock : t > < L : queue | List : l > => < P : task |
CPUtime : c, State : wait > < T : time | quantum : 4, clock : t > <
L : queue | List : add(l, P) > if (q == 0 and c > 0) .
endom)
```