Representación de Conocimiento
Tarea 1

Arturo Márquez Flores
Maestría en Inteligencia Artificial
Universidad Veracruzana
CIIA – Centro de Investigación en Inteligencia Artificial
Sebastián Camacho No 5, Xalapa, Ver., México 91000
arturomf94@gmail.com
https://github.com/arturomf94/rc-mia

26 de marzo del 2019

Pregunta 1

PEAS para un semáforo inteligente:

Agente	Desempeño	Ambiente	Actuadores	Sensores
Semáforo Inteligente	Flujo de tráfico vial y paso peatonal sujeto a mínimo de accidentes	Estructuras viales como carreteras y calles	Luces, sonido sistema de control	Cámaras, botones para que el peatón pida paso

Los actuadores y sensores pueden ser utilizados por el agente como artefactos. Por ejemplo, como actuadores, las luces, sonidos y el sistema de control pueden tener métodos para que el agente pueda emitir las luces y sondidos adecuadas para coordinar a los vehículos y los peatones. Por otro lado, los snesores también son artefactos con sus respectivos métodos para activarse y poder responder a las necesidades del agente. Por ejemplo, la cámara puede tener la funcionalidad de cambiar de dirección dependiendo de lo que indique el agente.

Una función de desempeño en este caso tendría que considerar tanto el flujo de vehículos como de peatones, penalizando el número de accidentes que puedan ocurrir. Podría formularse una función por partes que penalice al semáforo si hay accidentes con un número negativo. Por ejemplo:

$$R = \begin{cases} w_v V + w_p P & \text{si A <0} \\ -\infty & \text{si A >0} \end{cases}$$

Donde w_v y w_p son pesos para ponderar qué tanto valora el semáforo el flujo de vehículos sobre el de peatones, respectivamente, y V y P son el número de vehículos y peatones que cruzan por hora.

Pregunta 3

Primero demostramos que $p \to q \vdash \neg p \vee q$. Tomemos como premisa que $p \to q$ y supongamos dos casos (por LEM): primero supongamos que p. Entonces, como $p \to q$, obtenemos que q (MP) y, por lo tanto, $\neg p \vee q$ (por la introducción de la disyunción). En el otro caso, suponemos que $\neg p$, de lo cual se sigue automáticamente que $\neg p \vee q$, otra vez por la introducción de la disyunción.

Ahora mostremos que $\neg p \vee q \vdash p \to q$. Tomemos $\neg p \vee q$ como premisa y supongamos p y, para llegar a una contradicción, que $\neg q$. Entonces tenemos que $p \wedge \neg q$, lo cual equivale a $\neg(\neg p \vee q)$, lo cual es una contradicción de nuestra premisa. Por lo tanto, asumimos que q, y entonces tenemos que $p \to q$.

La lógica proposicional es robusta, por lo cual el haber mostrado que $p \to q$ y $\neg p \vee q$ son equivalentes sintácticamente es suficiente para saber que también lo son semánticamente.

Pregunta 4

Archivos:

- arturo-marquez-2.6.sen
- arturo-marquez-2.6.wld
- arturo-marquez-2.6(1).wld

Pregunta 5

Archivos:

- arturo-marquez-2.7.sen

Pregunta 6

Archivos:

- arturo-marquez-2.15.sen
- arturo-marquez-2.15.wld

Pregunta 7

Como es especificado en las notas de clase, el algoritmo para convertir una fórmula bien formada proposicional en su equivalente en forma normal conjuntiva está dada por la Figura 1, que a la vez utiliza el algoritmo de la Figura 2.

Algoritmo 4.1 CNF

```
1:  function CNF(φ)                                   ▷ φ es una fbf proposicional
2:      ψ ← NNF(IMPL_FREE(φ))
3:      switch ψ do
4:          case literal(ψ)
5:              return ψ
6:          case ψ1 ∧ ψ2
7:              return CNF(ψ1) ∧ CNF(ψ2)
8:          case ψ1 ∨ ψ2
9:              return DISTR(CNF(ψ1), CNF(ψ2))
10: end function
```

Figura 1: CNF

Algoritmo 4.2 NNF

```
1: function NNF(φ)                                   ▷ φ es libre de implicaciones
2:     switch φ do
3:         case literal(φ)
4:             return φ
5:         case ¬¬φ1
6:             return φ1
7:         case φ1 ∧ φ2
8:             return NNF(φ1) ∧ NNF(φ2)
9:         case φ1 ∨ φ2
10:            return NNF(φ1) ∨ NNF(φ2)
11:        case ¬(φ1 ∧ φ2)
12:            return NNF(¬φ1) ∨ NNF(¬φ2)
13:        case ¬(φ1 ∨ φ2)
14:            return NNF(¬φ1) ∧ NNF(¬φ2)
15: end function
```

Figura 2: NNF

En la Tabla 1 observamos la implementación de este algoritmo en Prolog.

```
1
2       :- op(4, xfy, imp).
3       :- op(3, xfy, or).
4       :- op(2, xfy, and).
5       :- op(1, fy,  neg).
6
7       free_impl(X,FBF) :- X = A or B,
8           free_impl(A, C), free_impl(B, D),
9           FBF = (C or D), !.
10
11      free_impl(X,FBF) :- X = A and B,
12          free_impl(A, C), free_impl(B, D),
13          FBF = (C and D), !.
14
15      free_impl(X,FBF) :- X = A imp B,
16          free_impl(A, C), free_impl(B, D),
17          FBF = (neg (C) or D), !.
18
19      free_impl(X,FBF):- FBF = X.
20
21      nnf(neg (A and B),NNF):- nnf(neg A, C),
22          nnf(neg B, D), NNF = (C or D), !.
23
24      nnf(neg (A or B),NNF):- nnf(neg A, C),
25          nnf(neg B, D), NNF = (C and D), !.
26
27      nnf(X,NNF):- X = A and B,
28          nnf(A, C), nnf(B, D),
29          NNF = (C and D), !.
30
31      nnf(X,NNF):- X = A or B,
32          nnf(A, C), nnf(B, D),
33          NNF = (C or D), !.
34
35      nnf(neg (neg X),NNF):- NNF = X,!.
36
37      nnf(X,NNF):- NNF = X.
38
39      distrib(X,Y,DISTRIB):- X=P and Q,
40          distrib(P,Y,DISTRIB2), distrib(Q,Y, DISTRIB3),
41          DISTRIB = DISTRIB2 and DISTRIB3, !.
42
43      distrib(X,Y,DISTRIB):- Y=P and Q,
44          distrib(X,P,DISTRIB2),
45          distrib(X,Q,DISTRIB3),
46          DISTRIB = DISTRIB2 and DISTRIB3, !.
47
48      distrib(X,Y,DISTRIB):- DISTRIB = X or Y.
49
50      cnf_aux(Z,CNF):- Z = X and Y,
51          cnf_aux(X,P), cnf_aux(Y,Q),
52          CNF = P and Q, !.
53
54      cnf_aux(Z,CNF):- Z= X or Y,
55          cnf_aux(X,P), cnf_aux(Y,Q),
56          distrib(P,Q,CNFAUX), CNF=CNFAUX, !.
57
58      cnf_aux(Z,CNF):- Z=CNF.
59
60      cnf(Z, CNF):- free_impl(Z,Q),
61      nnf(Q, R), cnf_aux(R, CNF), !.
```

Cuadro 1: CNF en Prolog

Aquí la cláusula $free\_impl$ se encarga de convertir una fbf a una fbf libre

de implicaciones. En su ejecución, podemos verificar esto con el ejemplo de las notas de clase:

```
%%% ?- free_impl(neg p and q imp p and (r imp q), R).
%%% R = neg (neg p and q)or p and (neg r or q).
```

Cuadro 2: free_impl en Prolog

La cláusula nnf se encarga de ejecutar lo especificado en la Figura 2, mientras la cláusula $distrib$ es auxiliar para el algoritmo en la Figura 1. Con todo, una ejecución de la clásula cnf para el mismo ejemplo que la tabla anterior es mostrada en la Tabla 3.

```
%%% ?- cnf(neg p and q imp p and (r imp q), R).
%%% R = ((p or neg q)or p)and ((p or neg q)or neg r or q).
```

Cuadro 3: CNF en Prolog