exercice01

a) les sources du problème d'apprentissage

1. La premi‘ere correspond au nombre de cycles durant lesquels la performance de l'agent reste sous-optimale pour la tˆache de d ecision donnee.

2. La seconde correspond aux ressources de calcul n ecessaires durant chaque cycle ‘a l'agent pour r eviser sa strat egie et choisir une action.

b) une d efinition du modéle d'apprentissage

mod‘ele d'apprentissage est un cadre formel donnant une mesure de ces deux sources de complexit e. Les observations, les actions et le feedback peuvent influer sur la difficult e de l'apprentissage

c) Expliquer comment les observations, les actions et le feedback peuvent influer sur la difficult e de l'apprentissage

1. L'observation: l'espace des observations la dimension peut etre immense, voire infini les valeurs de certains attributs peuvent ˆetre impr´ecises, erron´ees, ou encore absentes. environnements partiellement observables alors 'une situation inceain
2. L'action: les actions sont des decision soit simple ou complex, l'espace des d´ecisions poss‘ede une structure combinatoire; les d´ecisions peuvent prendre la forme d'arbres, de graphes, ou encore d'hypergraphes Les actions simples peuvent avoir un impact sur la difficult´e de l'apprentissage selon la mani‘ere dont elles influent l'environnement (´episodique/s´equentiel)
3. Feedback:Le type de feedback d´efinit le mode d'apprentissage

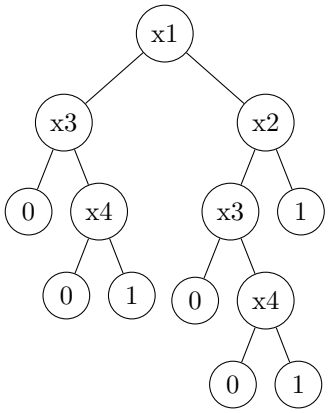
exercice2

Donner avec explication les composants du probl‘eme l'apprentissage supervis e de porte logique XOR

Une fonction XOr doit renvoyer une valeur vraie si les deux entr ees ne sont pas egales et une valeur fausse si elles sont egales -l'espace des entr ees est X =(0, 1)n et l'espace des sorties est Y = 0,1. 'hypothèse associée à = (w,) est définie par h(x) = tanh (x) le produit scalaire entre w et x et tanh(z) est la fonction qui retourne le signe du scalaire z

exercice03

x1 x2 x3 x4



exercice04

a) la diff erence entre une requˆete d'appartenance et une requˆete d' equivalence est

1. Une requˆete d'appartenance (MQ) associe 'a une instance x pos ee par l'apprenant la r eponse oui si h(x) = 1, et non sinon.
2. Une requˆete d' equivalence (EQ) associe 'a une hypoth'ese h pos e par l'apprenant la r eponse oui si h = h, et non sinon

b) Si pour toute instance (x1, x2, x3) quelle est l'hypoth'ese (le concept) le plus sp ecifiqu

est h1((x1,x2,x3)) = x1

c) Si h(x1,x2,x3) = x1 x2 et la requˆete est h((0,1,1)) = 0

1. le type de cette requˆete est requˆete d'appartenance (MQ)
2. et si h*((0,1,1)) = 1, requˆete d' equivalence (EQ)
3. Si le contre exemple est (0,1,0) mettre 'a jour h. h((0,1,0)) = 1

exercice05

exercice06

la diff erence entre le mod‘ele agnostique et le mod‘ele PAC

Dans le mod‘ele “agnostique”, la distribution D est arbitraire, ce quiimplique qu’il n’existe a priori aucune d ependance fonctionnelle entre une instance x et une d ecision y dans un exemple tir e dans D. En revanche, dans le mod‘ele (PAC), nous supposons qu’il existe uned ependance fonctionnelle gouvern ee par une fonction cible

---

PAL

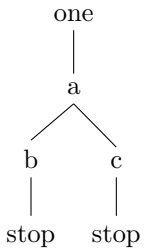
exercice01

1. A[]B = B[]A → Commutativité
2. A[]stop = stop[]A = A → Zéro absorption
3. A[](B[]C) = (A[]B)[]C → Associativité

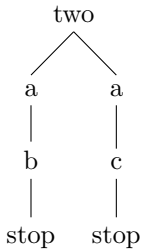
exercice02

exercice03

process one [a,b,c] a; (b; stop [] c; stop) endproc

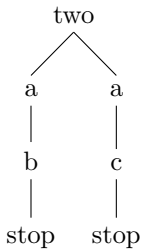


process two [a,b,c] a; b; stop [] a; c; stop Endproc

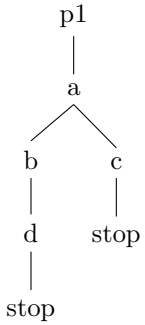


P1 := a; (b; d; stop [] c; stop)

process two [a,b,c] a; b; stop [] a; c; stop Endproc



P1 := a; (b; d; stop [] c; stop)



exercice04

Ecrire des spécifications lotos pour les circuits logiques: and, or et xor

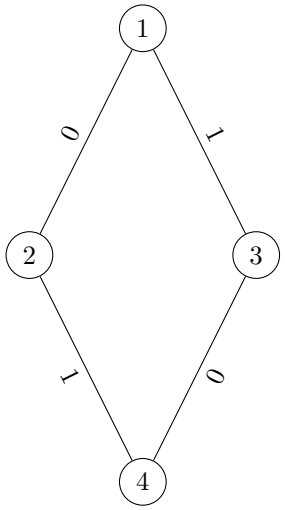
AND

specification circuitlogiqueAND [a,b,c] :noexit
type BIT is
sorts BIT
opns 0 (*! constructor *),
1 (*! constructor *) : -¿ BIT
and : BIT ,BIT-¿ BIT
eqns
ofsort BIT
and (0,0) = 0;
and (0,1) = 0;
and (0,1) = 0;
and (1,1) = 1;
endtype
behaviour
gate$_a nd[a, b, c]$
where
$processgate_A ND[a, b]$: *noexit* :=
$a?aa : Bit; b?bb : Bit; c!and(aa, bb);$ *stop*
endproc
endspec

OR

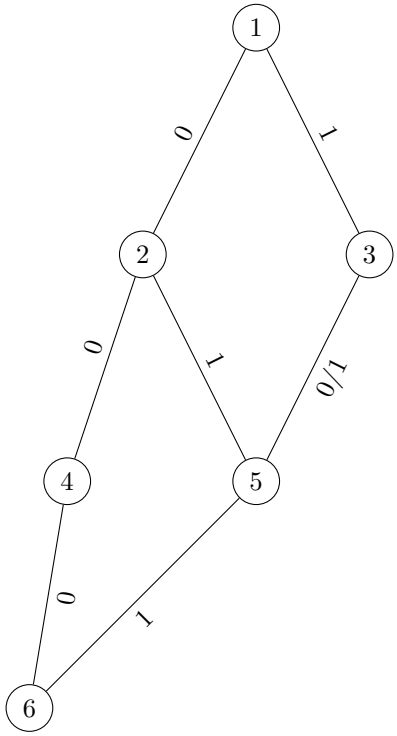
specification circuitlogiqueOR [a,b,c] :noexit
type BIT is
sorts BIT
opns 0 (*! constructor *),
1 (*! constructor *) : -¿ BIT
or : BIT ,BIT-¿ BIT
eqns
ofsort BIT
or (0,0) = 0;
or (0,1) = 1;
or (0,1) = 1;
or (1,1) = 1;
endtype
behaviour
gate$_O R[a, b, c]$

where
$processgate_O R[a, b]$: *noexit* :=
$a?aa : Bit; b?bb : Bit; c!or(aa, bb); stop$
endproc
endspec



XOR

specification circuitlogiqueXOR [a,b,c] :noexit
type BIT is
sorts BIT
opns 0 (*! constructor *),
1 (*! constructor *) : -¿ BIT
xor : BIT ,BIT-¿ BIT
eqns
ofsort BIT
xor (0,0) = 0;
xor (0,1) = 1;
xor (0,1) = 1;
xor (1,1) = 0;
endtype
behaviour
gate$_X OR[a, b, c]$
where
$processgate_X Or[a, b]$: *noexit* :=
$a?aa : Bit; b?bb : Bit; c!xor(aa, bb); stop$
endproc
endspec


1
0
1
2
3
0
1
0/1
4
5
0
1
6