Corrigé du partiel de compilation 2015

Exercice 1 Question 1

Il fallait ajouter l'instruction *switch* dans les instructions et les 2 non terminaux *cases* et *cas* :

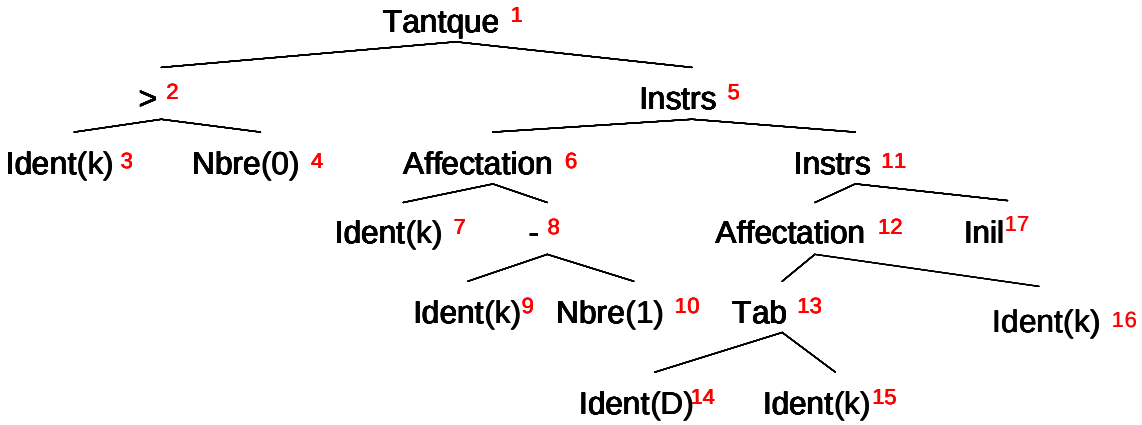
instr → switch "(" exp ")" "{" cases "}"	switch ($3, $6)
cases → case cases | vide	cases($1, $2) | cnil
case → case fact ":" instrs	case($2, $4)

La syntaxe abtraite engendrée est un arbre *switch* ayant 2 fils, un arbre *cases* ayant 2 fils et un arbre *case* ayant 2 fils.

Question 2 L'arbre *switch* est une instruction. On introduit des arbres de type CASES pour représenter une liste de cas et CAS pour représenter un cas.

switch : EXP × CASES	→ INSTR
cases : CAS × CASES	→ CASES
cnil :	→ CASES
case : EXP × INSTRS	→ CAS

Question 3



Question 4 : Règles de traduction

switch (e) {

case v_1 : is_1 ; case v_2 : is_2 ; … case v_{n-1} : is_{n-1} ; case v_n : is_n ;}

est traduite par l'imbrication de conditionnelles suivante :

if e==v_1 {is_1 ;}
else if e==v_2 {is_2 ;}
else …
if e==v_{n-1} {is_{n-1} ;}
else if e==v_n {is_n ;}

ou par la séquence de conditionnelles suivante (ce qui est une moins bonne solution) :
if e==v_1 {is_1 ;} ;
if e==v_2 {is_2 ;} ;
…
if e==v_{n-1} {is_{n-1} ;} ;
if e==v_n {is_n ;} ;

Solution par règle de réécriture :

Cette traduction peut-être définie par des règles de transformation des arbres de syntaxe abtraite *switch, case, cases* et *cnil* en arbres déjà défini dans la SA du langage Mini-Jaja décrit en cours. Les règles de transformation sont décrites par les fonctions appelée T et T', T' ayant deux arguments, l'expression du *switch* et l'arbre de SA à transformer. *e*, e_1 dénotent des expressions, *cs* une liste de cas et *is* une liste d'instructions.

T(switch(e, cs)) = T'(e, cs)
T'(e, cases(case(e_1, is), cs) = si(=(e, e_1), is, T'(e, cs))
T'(e, cnil) = inil

Solution avec des règles d'interprétation des nœuds :

[switch] $\dfrac{m|\text{-eval-}e \Rightarrow v,\ <nv, v, var, entier>.m\ |\text{-- } cs \rightarrow m_1}{m\ |\text{--switch}(e, cs) \rightarrow m_1}$

où *nv* est un nouveau nom de variable qui n'est pas dans *m*

[cases] $\dfrac{m\ |\text{--}c \rightarrow m_1,\ m_1|\text{--}cs \rightarrow m_2}{m\ |\text{--cases}(c, cs) \rightarrow m_2}$

[casevrai] $\dfrac{<nv, v, var, entier>.m\ |\text{-eval-=}(nv, v) \Rightarrow true,\ <nv, v, var, entier>.m\ |\text{--}is \rightarrow m_1}{<nv, v, var, entier>.m\ |\text{--case}(v, is) \rightarrow m_1}$

[casefaux] $\dfrac{<nv, v, var, entier>.m\ |\text{-eval-=}(e, v) \Rightarrow false}{<nv, v, var, entier>.m\ |\text{--case}(e, is) \rightarrow m}$

[cnil] $m|\text{--cnil} \rightarrow m$

Question 5

L'état mémoire m_0 est défini ainsi :
m_0 = <k, 2, var, entier >. m où
m = <j, 1, var, entier >.
<D, @tas_D, tab, entier >.
<N, 7, cst, entier >.
<P_2015, ω, var, * >. []
La valeur du tas à l'adresse @tas_D est [ω, ω, 2, 3, 4, 5, 6].

Question 6

Voir ci-dessous l'interprétation du while après 6 boucles complètes où k=1 :

L'état mémoire m_r est défini ainsi :
m_r = <k, 0, var, entier >. m
La valeur du tas à l'adresse @tas_D est [0, 1, 2, 3, 4, 5, 6].

Seule la valeur de *k* est modifiée ainsi que les deux valeurs D[0] et D[1] dans le tas.

Soit m_1 = <k, 1, var, entier >. m
l'état mémoire obtenu par exécution du while après 6 boucles complètes
où la valeur du tas à l'adresse @tas$_D$ est [ω, 1, 2, 3, 4, 5, 6].

Le résultat de l'interprétation développée ci-dessous est bien l'état mémoire m_r défini ci-dessus

Question 7

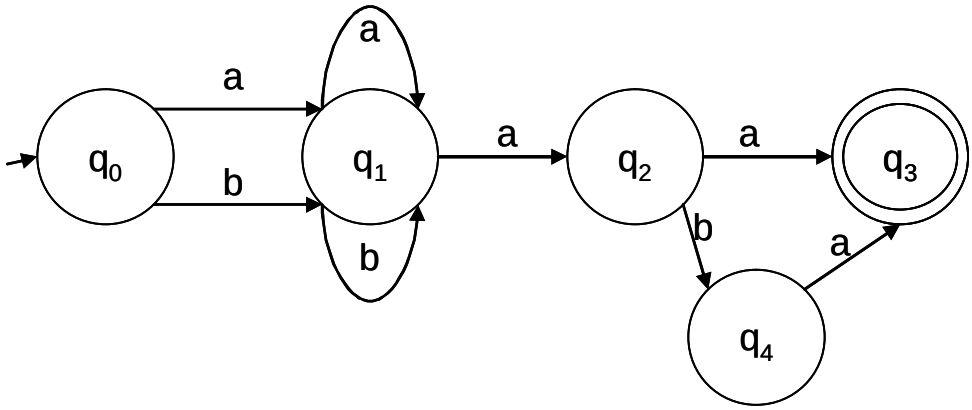
L'état mémoire m_f obtenu après exécution du *switch* est défini ainsi :
m_f = <k, 0, var, entier >. m
La valeur du tas à l'adresse @tas$_D$ est [0, 3, 2, 3, 4, 5, 6].
Seule la valeur D[1] est modifiée dans le tas, elle prend la valeur de D[3].

Question 6 interprétation

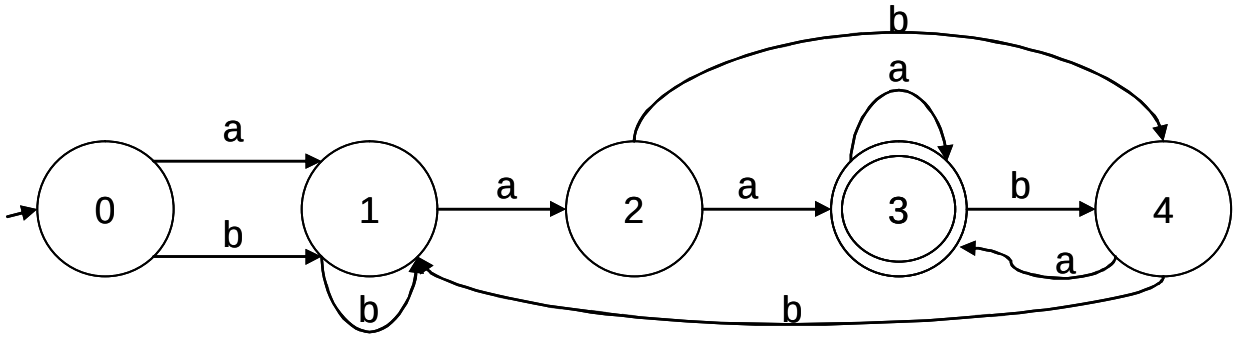
[tantquevrai]$_1$ m_1 |-- tantque(2, 5) → m_r
[op$_2$]$_2$ m_1 |-eval- >(3, 4) ⇒ $v_1 = v_2 > v_3 = 1>0$ = true
[ident]$_3$ m_1 |-eval- ident(k) ⇒ v_2 = 1
[nbre]$_4$ m_1 |- eval - nbre(0) ⇒ v_3 =0
[instrs]$_5$ m_1 |-- instrs(6, 11) → m_2
[affectation]$_6$ m_1 |-- affectation(ident(k), 8) → m_3
AffecterVal(k, v_4, m_1)
donc m_3 = AffecterVal(k, 0, m_1), donc
m_3 = <k, 0, var, entier >. m
[op$_2$]$_8$ m_1 |-eval- -(ident(k), Nbre(1)) ⇒ $v_4 = v_5 - v_6 = 1-1 = 0$
[ident]$_9$ m_1 |-eval- ident(k) ⇒ v_5 = 1
[nbre]$_{10}$ m_1 |- eval - nbre(1) ⇒ v_6 =1
[instrs]$_{11}$ m_3 |-- instrs(12, 14) → m_2
[affectationT]$_{12}$ m_3-- affectation(tab(ident(D), 15), 16) → $m_4 = m_3$
AffecterValT(D, v_8, v_7, m_3)
donc m_4 = AffecterValT(D, 0, 0, m_3),
$m_4 = m_3$ où @tas$_D$ est [0, 1, 2, 3, 4, 5, 6]
[ident]$_{16}$ m_3 |-eval- ident(k) ⇒ v_7 = 0
[ident]$_{15}$ m_3 |-eval- ident(k) ⇒ v_8 = 0
[inil]$_{14}$ m_4|-- inil → $m_2 = m_4$
[tantquefaux]$_1$ m_2 |-- tantque(2, 5) → $m_r = m_2$
[op$_2$]$_2$ m_2 |- eval - >(3, 4) ⇒ $v_9 = v_{10} > v_{11} = 0>0$ = false
[ident]$_3$ m_2 |-eval- ident(k) ⇒ v_{10} = 0
[nbre]$_4$ m_2 |-eval- nbre(0) ⇒ v_{11} = 0

Exercice 2

Le langage régulier `(a|b)+ a (a|(a b))` est reconnu par l'automate non déterministe suivant :



L'automate déterministe est le suivant :



Sa table de transition déterministe est la suivante :

	q_0 (0)	q_1 (1)	q_1, q_2 (2)	q_1, q_2, q_3 (3)	q_1, q_4 (4)
a	q_1 (1)	q_1, q_2 (2)	q_1, q_2, q_3 (3)	q_1, q_2, q_3 (3)	q_1, q_2, q_3 (3)
b	q_1 (1)	q_1 (1)	q_1, q_4 (4)	q_1, q_4 (4)	q_1 (1)

Il est minimal.

Exercice 3

Le Jaja Code traduisant l'itération donnée :
while (k > 0) {k = k-1 ; x = k - (x + 2) ; } ;
est par exemple le suivant :

1 load(k) 2 push(0) 3 sup 4 not 5 if(17) 6 load(k) 7 push(1) 8 sub 9 store(k) 10 load(k)
11 load(x) 12 push(2) 13 add 14 sub 15 store(x) 16 goto(1)

Exercice 4 (non donné)

La grammaire transformée est la suivante :

P → variables DS debut IS fin
DS → D DS'
DS' → ';' D DS' | vide
IS → I IS'
IS' → ';' I IS' | vide