Université Laval
Faculté des sciences et de génie
Département d'informatique et de génie logiciel
IFT-3101

Danny Dubé
Automne 2020
Version: 3 décembre

Travail pratique #2
Traduction orientée-syntaxe → Génération de code

Questions

1. **Définitions orientées-syntaxe**. Considérons le langage

$$L_1 = \left\{ w \in \{\texttt{c}, \texttt{d}, \texttt{i}, \texttt{o}, \texttt{v}\}^* \;\middle|\; \begin{array}{l} w \text{ contient la sous-chaîne } \texttt{covid} \\ \text{exactement 2019 fois} \end{array} \right\}.$$

La DOS suivante sert à reconnaître L_1 en ce sens que, pour une chaîne w tirée de l'alphabet $\{\texttt{c}, \texttt{d}, \texttt{i}, \texttt{o}, \texttt{v}\}$, l'attribut $S.ok$ va contenir le booléen vrai si et seulement si $w \in L_1$. La DOS comporte des parties manquantes. Vous devez comprendre le fonctionnement de la DOS et déterminer quelles sont les parties manquantes. J'indique seulement que les noms des attributs $A.l$ et $A.n$ ont été choisis parce qu'ils contiennent une *longueur* et un *nombre*, respectivement.

Productions	Règles sémantiques			
$S \to A$	$S.ok$	$:=$	$(A.n = 2019)$	
$A \to \texttt{c}\, A_1$	$A.l$	$:=$	**if** $A_1.l = 4$ **then** 5 **else** 0	// $\texttt{c} \cdot \texttt{ovid}$?
	$A.n$	$:=$	$\boxed{}$	
$A \to \texttt{d}\, A_1$	$A.l$	$:=$	1	// $\texttt{d}$
	$A.n$	$:=$	$A_1.n$	
$A \to \texttt{i}\, A_1$	$A.l$	$:=$	**if** $A_1.l = 1$ **then** 2 **else** 0	// $\texttt{i} \cdot \texttt{d}$?
	$A.n$	$:=$	$A_1.n$	
$A \to \texttt{o}\, A_1$	$A.l$	$:=$	$\boxed{}$	// $\texttt{o} \cdot \texttt{vid}$?
	$A.n$	$:=$	$A_1.n$	
$A \to \texttt{v}\, A_1$	$A.l$	$:=$	**if** $A_1.l = 2$ **then** 3 **else** 0	// $\texttt{v} \cdot \texttt{id}$?
	$A.n$	$:=$	$A_1.n$	
$A \to \epsilon$	$A.l$	$:=$	0	// ϵ
	$A.n$	$:=$	0	

(a) $\boxed{A.l}$ $\boxed{\textbf{if } A_1.l = 4 \textbf{ then } 3 \textbf{ else } 0}$

(b) $\boxed{A_1.n + 1}$ $\boxed{\textbf{if } A_1.l = 4 \textbf{ then } 3 \textbf{ else } 0}$

(c) $\boxed{\textbf{if } A.l = 5 \textbf{ then } A_1.n + 1 \textbf{ else } A_1.n}$ $\boxed{\textbf{if } A_1.l = 3 \textbf{ then } 4 \textbf{ else } 0}$

(d) $\boxed{A_1.n}$ $\boxed{\textbf{if } A_1.l = 3 \textbf{ then } 4 \textbf{ else } 1}$

2. **Définitions orientées-syntaxe**. Considérons le langage L_1 à nouveau. Cette fois-ci, on a une DOS qui utilise des attributs hérités. Cette DOS a malheureusement un défaut qui l'empêche de faire le travail de reconnaissance attendu. Essayez de comprendre la DOS et identifiez son défaut parmi les choix donnés.

Productions	Règles sémantiques		
$S \to A$	$A.l := 0$		
	$A.n := 0$		
$A \to \texttt{c}\, A_1$	$A_1.l := 1$		// `c`
	$A_1.n := A.n$		
$A \to \texttt{d}\, A_1$	$A_1.l :=$ **if** $A.l = 4$ **then** 5 **else** 0		// `covi` $\cdot$ `d` ?
	$A_1.n :=$ **if** $A_1.l = 5$ **then** $A.n + 1$ **else** $A.n$		
$A \to \texttt{i}\, A_1$	$A_1.l :=$ **if** $A.l = 3$ **then** 4 **else** 0		// `cov` $\cdot$ `i` ?
	$A_1.n := A.n$		
$A \to \texttt{o}\, A_1$	$A_1.l :=$ **if** $A.l = 1$ **then** 2 **else** 0		// `c` $\cdot$ `o` ?
	$A_1.n := A.n$		
$A \to \texttt{v}\, A_1$	$A_1.l :=$ **if** $A.l = 2$ **then** 3 **else** 0		// `co` $\cdot$ `v` ?
	$A_1.n := A.n$		
$A \to \epsilon$	$S.ok := (A.n = 2019)$		

(a) Chaque apparition de `covid` est comptée deux fois.

(b) L'attribut $A.l$ devrait plutôt contenir la longueur du *suffixe* de `covid` en présence.

(c) La DOS compte plutôt le nombre d'apparitions de `divoc` dans la chaîne d'entrée.

(d) Le comptage est bien effectué mais il faudrait recopier le compte obtenu du bas de l'arbre de dérivation jusqu'à sa racine.

3. **Définitions orientées-syntaxe**. Considérons le langage L_1 encore une fois. Une stratégie de comptage complètement différentes des stratégies précédentes est utilisée ici. Essayez de comprendre la DOS suivante. Identifiez l'énoncé qui est *faux* parmi les choix proposés.

PROD.	RÈGLES SÉMANTIQUES
$S \to A$	$A.a := \texttt{'x'}; \quad A.b := \texttt{'x'}$ $S.ok := (A.n = 2019)$
$A \to \texttt{c}\, A_1$	$A_1.a := A.b; \quad A_1.b := A.c; \quad A.c := \texttt{'c'}; \quad A.d := A_1.c; \quad A.e := A_1.d$ $A.n := A_1.n$
$A \to \texttt{d}\, A_1$	$A_1.a := A.b; \quad A_1.b := A.c; \quad A.c := \texttt{'d'}; \quad A.d := A_1.c; \quad A.e := A_1.d$ $A.n := A_1.n$
$A \to \texttt{i}\, A_1$	$A_1.a := A.b; \quad A_1.b := A.c; \quad A.c := \texttt{'i'}; \quad A.d := A_1.c; \quad A.e := A_1.d$ $A.n := A_1.n$
$A \to \texttt{o}\, A_1$	$A_1.a := A.b; \quad A_1.b := A.c; \quad A.c := \texttt{'o'}; \quad A.d := A_1.c; \quad A.e := A_1.d$ $A.n := A_1.n$
$A \to \texttt{v}\, A_1$	$A_1.a := A.b; \quad A_1.b := A.c; \quad A.c := \texttt{'v'}; \quad A.d := A_1.c; \quad A.e := A_1.d$ $A.n :=$ **if** $(A.a = \texttt{'c'})$ **and** $(A.b = \texttt{'o'})$ **and** $(A.c = \texttt{'v'})$ **and** $(A.d = \texttt{'i'})$ **and** $(A.e = \texttt{'d'})$ **then** $A_1.n + 1$ **else** $A_1.n$
$A \to \epsilon$	$A.c := \texttt{'x'}; \quad A.d := \texttt{'x'}; \quad A.e := \texttt{'x'}$ $A.n := 0$

(a) La DOS détermine correctement si la chaîne d'entrée fait partie de L_1.

(b) La DOS crée des dépendances circulaires entre des attributs, ce qui empêche de calculer tous les attributs, en général.

(c) Les symboles arbitraires 'x' qui sont injectés à la première production et à la dernière production servent à éviter de détecter à tort des apparitions de `covid`.

(d) Dans la deuxième règle sémantique associée à la production $A \to \texttt{v}\, A_1$, il est redondant (inutile, superflu) de tester si $A.c = \texttt{'v'}$.

4. **Langages formels**. Dans quelle classe de langages peut-on placer le langage L_1?

(a) Il est fini.

(b) Il n'est pas fini mais il est régulier.

(c) Il n'est pas régulier mais il est hors-contexte.

(d) Il n'est pas hors-contexte.

5. **Définitions orientées-syntaxe**. Lors de l'examen intra, on avait vu le langage L_5. On se souviendra qu'il n'était pas si simple de concevoir une définition régulière qui le génère. Ici, on va tester l'appartenance à L_5 à l'aide d'une DOS. Indiquez les formules qui manquent à l'intérieur des règles.

$$L_5 \;=\; \left\{ w \in \{\mathtt{a},\mathtt{b},\mathtt{c}\}^* \;\middle|\; \begin{array}{l} w \text{ contient la sous-chaîne } \mathtt{abc} \text{ et} \\ w \text{ contient la sous-séquence } \mathtt{cba} \end{array} \right\}$$

Prod.			Règles sémantiques			
S	$\to$	A	$S.ok$	$:=$	$A.ok$ **and** $(A.l_2 = 3)$	
A	$\to$	$\mathtt{a}\,A_1$	$A.l_1$	$:=$	**if** $A_1.l_1 = 2$ **then** 3 **else** 0	// $\mathtt{a}\cdot\mathtt{bc}$?
			$A.ok$	$:=$	$\boxed{}$	
			$A.l_2$	$:=$	$\boxed{}$	// ...$\mathtt{a}$... ?
A	$\to$	$\mathtt{b}\,A_1$	$A.l_1$	$:=$	**if** $A_1.l_1 = 1$ **then** 2 **else** 0	// $\mathtt{b}\cdot\mathtt{c}$?
			$A.ok$	$:=$	$A_1.ok$	
			$A.l_2$	$:=$	**if** $A_1.l_2 = 1$ **then** 2 **else** $A_1.l_2$	// ...$\mathtt{b}$...$\mathtt{a}$... ?
A	$\to$	$\mathtt{c}\,A_1$	$A.l_1$	$:=$	1	// $\mathtt{c}$
			$A.ok$	$:=$	$A_1.ok$	
			$A.l_2$	$:=$	**if** $A_1.l_2 = 2$ **then** 3 **else** $A_1.l_2$	// ...$\mathtt{c}$...$\mathtt{b}$...$\mathtt{a}$... ?
A	$\to$	ϵ	$A.l_1$	$:=$	0	
			$A.ok$	$:=$	*faux*	
			$A.l_2$	$:=$	0	

(a) $\boxed{A_1.ok \textbf{ or } (A.l_1 = 3)}$ $\qquad$ $\boxed{\textbf{if } A_1.l_2 = 0 \textbf{ then } 1 \textbf{ else } A_1.l_2}$

(b) $\boxed{A_1.ok \textbf{ or } (A.l_2 = 3)}$ $\qquad$ $\boxed{\textbf{if } A_1.l_2 = 0 \textbf{ then } 1 \textbf{ else } A_1.l_2}$

(c) $\boxed{(A.l_1 = 3)}$ $\qquad$ $\boxed{1}$

(d) $\boxed{A_1.ok \textbf{ and } (A.l_1 = 3)}$ $\qquad$ $\boxed{A_1.l_2 + 1}$

6. **Systèmes de traduction**. Ici, on transforme la DOS de la question 3 en un système de traduction. Cependant, une action quelque part n'est pas placée au bon endroit. Indiquez dans quelle production on retrouve le problème.

S	$\to$	$\{A.a := \texttt{'x'}\}\ \{A.b := \texttt{'x'}\}\ A\ \{S.ok := (A.n = 2019)\}$
A	$\to$	$\texttt{c}\ \{A.c := \texttt{'c'}\}\ \{A_1.a := A.b\}\ \{A_1.b := A.c\}\ A_1\ \{A.n := A_1.n\}$ $\{A.d := A_1.c\}\ \{A.e := A_1.d\}$
A	$\to$	$\texttt{d}\ \{A.c := \texttt{'d'}\}\ \{A_1.a := A.b\}\ \{A_1.b := A.c\}\ A_1\ \{A.n := A_1.n\}$ $\{A.d := A_1.c\}\ \{A.e := A_1.d\}$
A	$\to$	$\texttt{i}\ \{A.c := \texttt{'i'}\}\ \{A_1.a := A.b\}\ \{A_1.b := A.c\}\ A_1\ \{A.n := A_1.n\}$ $\{A.d := A_1.c\}\ \{A.e := A_1.d\}$
A	$\to$	$\texttt{o}\ \{A.c := \texttt{'o'}\}\ \{A_1.a := A.b\}\ \{A_1.b := A.c\}\ A_1\ \{A.n := A_1.n\}$ $\{A.d := A_1.c\}\ \{A.e := A_1.d\}$
A	$\to$	$\texttt{v}\ \{A.c := \texttt{'v'}\}\ \{A_1.a := A.b\}\ \{A_1.b := A.c\}\ A_1$ $\left\{ A.n := \textbf{if}\ (A.a = \texttt{'c'})\ \textbf{and}\ (A.b = \texttt{'o'})\ \textbf{and}\ (A.c = \texttt{'v'})\ \textbf{and}\ (A.d = \texttt{'i'})\ \textbf{and}\ (A.e = \texttt{'d'})\ \textbf{then}\ A_1.n + 1\ \textbf{else}\ A_1.n \right\}$ $\{A.d := A_1.c\}\ \{A.e := A_1.d\}$
A	$\to$	$\{A.c := \texttt{'x'}\}\ \{A.n := 0\}\ \{A.d := \texttt{'x'}\}\ \{A.e := \texttt{'x'}\}$

(a) La S-production.

(b) La première A-production.

(c) L'avant-dernière A-production.

(d) La dernière A-production.

7. **Systèmes de traduction**. Dans la chapitre 5, on voit comment éliminer la récursion à gauche dans un système de traduction. Ici, on veut s'attaquer à un cas juste un peu plus complexe, en ayant le non-terminal Z en plus du non-terminal Y dans la production récursive. Indiquez comment on devrait transformer le système de traduction.

$$\begin{array}{lcll} A & \rightarrow & A_1\ Y\ Z & \{A.a := g(A_1.a,\ Y.y,\ Z.z)\} \\ A & \rightarrow & X & \{A.a := f(X.x)\} \end{array}$$

(a)
$$\begin{array}{lcll} A & \rightarrow & X & \{R.i := f(X.x)\} \quad R\ \{A.a := R.s\} \\ R & \rightarrow & Y & \{Z.i := g_1(R.i,\ Y.y)\} \quad Z\ \{R_1.i := g_2(Z.z)\} \quad R_1\ \{R.s := R_1.s\} \\ R & \rightarrow & \epsilon & \{R.s := R.i\} \end{array}$$

(b)
$$\begin{array}{lcll} A & \rightarrow & X & \{R.i := f(X.x)\} \quad R\ \{A.a := R.s\} \\ R & \rightarrow & Y\ Z & \{R_1.i := g(R.i,\ Y.y,\ Z.z)\} \quad R_1\ \{R.s := R_1.s\} \\ R & \rightarrow & \epsilon & \{R.s := R.i\} \end{array}$$

(c)
$$\begin{array}{lcll} A & \rightarrow & X\ R & \{R.i := f(X.x)\} \quad \{A.a := R.s\} \\ R & \rightarrow & Y\ Z\ R_1 & \{R_1.i := g(R.i,\ Y.y,\ Z.z)\} \quad \{R.s := R_1.s\} \\ R & \rightarrow & \epsilon & \{R.s := R.i\} \end{array}$$

(d)
$$\begin{array}{lcll} A & \rightarrow & X & \{R.i_1 := f_1(X.x)\} \quad \{R.i_2 := f_2(X.x)\} \\ & & R & \{A.a := R.s\} \\ R & \rightarrow & Y\ Z & \{R_1.i_1 := g_1(R.i_1,\ Y.y)\} \quad \{R_1.i_2 := g_2(R.i_2,\ Z.z)\} \\ & & R_1 & \{R.s := R_1.s\} \\ R & \rightarrow & \epsilon & \{R.s := g_3(R.i_1,\ R.i_2)\} \end{array}$$

8. **Génération de code intermédiaire**. Dans la chapitre 6, on a vu comment générer du code intermédiaire pour les opérateurs logiques "ET" et "OU", entre autres. Ici, on vous demande de générer du code intermédiaire classique pour le "OU exclusif". Une DOS à cet effet est présentée ici mais il manque une règle sémantique ou un morceau de règle sémantique quelque part. Indiquez l'élément qui manque.

PRODUCTIONS	RÈGLES
$E \rightarrow E_1$ **xor** E_2	$E.L_{\text{sortie}}$:= newLabel() $E.code$:= $E_1.code$ ‖ $E_2.code$ ‖ $gen(E.place$ ':=' $E_1.place)$ ‖ $gen($'**if**' $E_2.place$ '=' '0' '**goto**' $E.L_{\text{sortie}})$ ‖ $gen(E.place$ ':=' $E_2.place)$ ‖ $gen($'**if**' $E_1.place$ '=' '0' '**goto**' $E.L_{\text{sortie}})$ ‖ $gen(E.place$ ':=' '0') ‖ $gen(E.L_{\text{sortie}}$ '**:**')

Notez qu'on choisit de générer du code classique tant pour E que pour ses deux sous-expressions E_1 et E_2. Supposons que le langage de programmation hypothétique impose que la valeur vraie qui est retournée par un OU exclusif est la valeur retournée par la (seule) sous-expression qui a une valeur vraie.

(a) Il manque une règle qui crée la variable qui doit contenir la valeur de E.

(b) Il manque du code pour traiter le cas où les deux sous-expressions sont vraies.

(c) Il manque des règles pour faire hériter aux deux sous-expressions les étiquettes où elles doivent brancher.

(d) Il manque la création d'au moins une autre étiquette car la logique d'un OU exclusif ne peut pas être respectée par un code n'utilisant qu'une seule étiquette.

9. **Génération de code intermédiaire**. Comme à la question précédente, on s'attaque à la tâche de générer du code intermédiaire pour un "OU exclusif". Cette fois-ci, on veut obtenir du code à court-circuit pour le OU exclusif. Toutefois, une chose qui n'a jamais été faite dans les notes de cours, c'est de générer du code avec une stratégie différente chez l'une ou l'autre des sous-expressions que chez l'expression parente. Ici, on se considère libre de changer de stratégie chez les sous-expressions. Indiquez quelle stratégie de génération de code est employée pour chacune des sous-expressions. Notez que la DOS suivante ne fait pas la distinction entre les noms d'attributs *code* et *ccode*, pour éviter de donner des indices trop flagrants.

PRODUCTIONS	RÈGLES
$E \rightarrow E_1$ **xor** E_2	$E.L_1$:= newLabel() $E.L_2$:= newLabel() $E_2.L_{\text{true}} := E.L_1$ $E_2.L_{\text{false}} := E.L_2$ $E.code$:= $E_1.code \parallel$ $E_2.code \parallel$ $gen(E.L_1$ '**:**') $gen($'**if**' $E_1.place$ '=' '0' '**goto**' $E.L_{\text{true}}) \parallel$ $gen($'**goto**' $E.L_{\text{false}}) \parallel$ $gen(E.L_2$ '**:**') $gen($'**if**' $E_1.place$ '=' '0' '**goto**' $E.L_{\text{false}}) \parallel$ $gen($'**goto**' $E.L_{\text{true}})$

(a) E_1 et E_2 sont traduites en code à court-circuit.

(b) E_1 est traduite en code à court-circuit et E_2 est traduite en code classique.

(c) E_1 est traduite en code classique et E_2 est traduite en code à court-circuit.

(d) E_1 et E_2 sont traduites en code classique.

10. **Génération de code intermédiaire**. Nous poursuivons avec la génération de code intermédiaire pour un autre opérateur logique: le "si et seulement si" (SSI). Soit la production $E \rightarrow E_1 \Leftrightarrow E_2$. Un SSI devient vrai lorsque ses deux sous-expressions ont la même valeur de vérité. Supposons que notre langage de programmation hypothétique fixe les valeurs produites par le SSI ainsi. Lorsque le SSI voit E_1 et E_2 être vraies, le SSI doit retourner la valeur exacte produite par E_2. Ensuite, lorsque le SSI voit E_1 et E_2 être fausses, le SSI doit retourner la valeur 1. Enfin, les cas restants font en sorte que le SSI est faux.

    Supposons qu'on s'intéresse ici à faire générer du code classique pour un SSI. Quelle affirmation est incorrecte parmi les choix de réponses quant à la stratégie de génération de code pour les sous-expressions?

    (a) E_1 pourrait être traduite en code classique.

    (b) E_1 pourrait être traduite en code à court-circuit.

    (c) E_2 pourrait être traduite en code classique.

    (d) E_2 pourrait être traduite en code à court-circuit.

11. **Typage**. Au chapitre 6, on a étudié des rudiments du typage. Supposons qu'on ajoute le type interne *float* au langage hypothétique étudié. Supposons aussi qu'on ajoute l'opérateur d'addition. La production $E \rightarrow E_1 + E_2$ introduit la syntaxe source pour l'expression d'addition.

    Comme on a maintenant deux types numériques et qu'on souhaite que l'opérateur d'addition s'applique aux deux types de nombres, on ajoute les spécifications suivantes. Les deux sous-expressions doivent chacune être d'un type numérique. Quand les deux sous-expressions ont le même type, alors le résultat de l'addition est du même type que les sous-expressions. Quand les deux sous-expressions ont des types numériques différents, alors le résultat de l'addition est du type *float*.

    Parmi les règles sémantiques suivantes, une ne parvient pas à faire le typage tel que souhaité. Laquelle?

    (a) $E.type :=$ **if** $E_1.type = integer$ **then** $E_2.type$
    else if $E_1.type = float$ **then** *float*
    else *type_error*

    (b) $E.type :=$ **if** $E_1.type = E_2.type = integer$ **then** *integer*
    else if $E_1.type \in \{integer, float\}$ **and** $E_2.type \in \{integer, float\}$
    then *float*
    else *type_error*

    (c) $E.type :=$ **if** $E_1.type \notin \{integer, float\}$ **then** *type_error*
    else if $E_2.type \notin \{integer, float\}$ **then** *type_error*
    else if $E_1.type = float$ **then** *float*
    else $E_2.type$

    (d) $E.type :=$ **if** $E_1.type = integer$ **and** $E_2.type = integer$ **then** *integer*
    else if $E_1.type = integer$ **and** $E_2.type = float$ **then** *float*
    else if $E_1.type = float$ **and** $E_2.type = integer$ **then** *float*
    else if $E_1.type = float$ **and** $E_2.type = float$ **then** *float*
    else *type_error*

12. **Disposition des données**. Un compilateur doit calculer la disposition des variables pour la suite de déclarations de variables suivante. Supposons que le langage de programmation interdise les réordonnancements de variables. Les conventions sur la taille et l'alignement des différents types de données sont celles qui sont adoptées à la dernière page des notes de cours du chapitre 6. Quel est le nombre d'octets de remplissage qui sont nécessaires, si on ne fait pas de gaspillage délibérément?

```
a : char;
b : double;
c : integer;
d : short;
e : long;
f : array[7] of char;
g : integer;
h : double;
```

(a) 5.

(b) 6.

(c) 8.

(d) 11.

13. **Arbres d'activation**. Soit l'implantation de la fonction '`fib`' suivante. Dessinez l'arbre d'activation dont la racine serait l'activation '`fib(6)`'. Combien y a-t-il de noeuds dans l'arbre?

```
int fib(int n)
{ return (n <= 1) ? n : fib(n-1) + fib(n-2); }
```

(a) 7.

(b) 8.

(c) 25.

(d) 36.

14. **Liaison des noms**. Considérons ce bout de programme. Lequel des énoncés à propos de la portée des déclarations est correct?

(a) La troisième déclaration de '`a`' a pour portée la section 5.

(b) La déclaration de '`b`' a pour portée les sections 2 et 3.

(c) La première déclaration de '`c`' a pour portée les sections 1, 3 et 7.

(d) La première déclaration de '`d`' a pour portée les sections 1 à 3.

```
{
    int c, d;
    ...          /* Section 1 */
    {
        int a, b, c;
        ...      /* Section 2 */
    }
    ...          /* Section 3 */
    {
        int a, d;
        ...      /* Section 4 */
        {
            int a, c;
            ...  /* Section 5 */
        }
        ...      /* Section 6 */
    }
    ...          /* Section 7 */
}
```

15. **Définitions orientées-syntaxe**. Qu'est-ce qui est synthétisé dans $S.res$ par la DOS? Faites attention au fait que tant des nombres (comme 0, 1 et 2) que des symboles numériques (comme '`0`' et '`1`') sont manipulés par la DOS.

Productions	Règles
$S \rightarrow A$	$A.m := 0;$ $S.res := A.s$
$A \rightarrow \mathtt{a}\, A_1$	$A_1.m := A.m + 1;$ $A.n := \lfloor A_1.n/2 \rfloor$ $A.b :=$ **if** $(A.n \bmod 2) = 0$ **then** '`0`' **else** '`1`' $A.s :=$ (**if** $A.n = 0$ **then** ϵ **else** $A.b) \cdot A_1.s$
$A \rightarrow \epsilon$	$A.n := A.m$ $A.b :=$ **if** $(A.n \bmod 2) = 0$ **then** '`0`' **else** '`1`' $A.s := A.b$

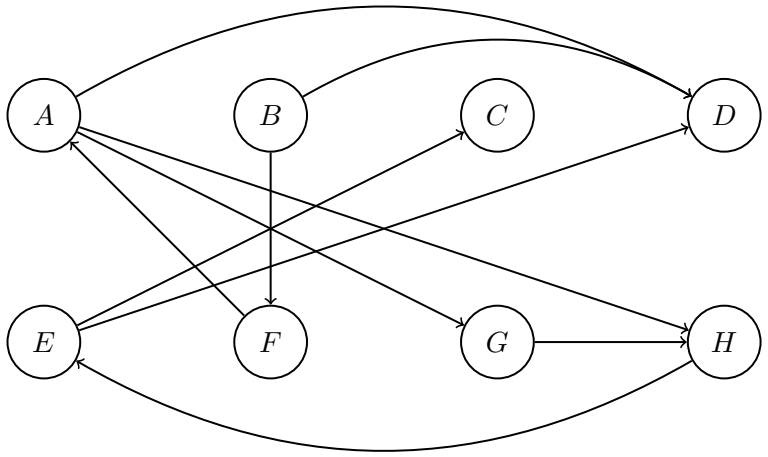
(a) Elle synthétise une chaîne de bits qui indique tous les restes (modulos) de la division de ℓ, $\ell - 1$, $\ell - 2$, ..., 1 par 2.

(b) Elle synthétise une chaîne de bits qui représente ℓ en binaire, où ℓ est la longueur de la chaîne d'entrée.

(c) Elle synthétise un nombre entier qui est la moitié de ℓ, où ℓ est la longueur de la chaîne d'entrée, en arrondissant de telle façon que cette moitié arrondie soit impaire lorsque ℓ est impaire.

(d) Elle synthétise une chaîne de bits qui encode en caractères Ascii la phrase "`J'aime IFT-3101!`".

16. **Tri topologique**. Parmi les ordonnancements proposés dans les choix de réponses, un seul est un tri topologique de ce graphe. Lequel?



(a) B, F, A, H, G, E, D, C.

(b) B, F, A, D, G, H, E, C.

(c) A, G, B, F, H, E, C, D.

(d) B, F, A, G, H, E, C, D.

17. **Définitions orientées-syntaxe**. Est-ce que les DOS des questions 3 et 15 sont L-attribuées?

(a) Les deux DOS sont L-attribuées.

(b) La DOS de la question 3 l'est mais pas celle de la question 15.

(c) La DOS de la question 3 ne l'est pas mais pas celle de la question 15 l'est.

(d) Aucune des deux DOS n'est L-attribuée.

18. **Allocation des registres**. Quel est le nombre minimal de registres qui sont nécessaires pour que toutes variables temporaires puissent être affectées à des registres? Supposez que les variables provenant du source ne peuvent être stockées dans des registres. Aussi, supposez qu'aucune variable temporaire n'est vivante à la fin du bloc de code. Les variables temporaires sont celles dont le nom commence par '`t`'.

(a) 1.

(b) 2.

(c) 3.

(d) 4.

```
1. t1 := 8 * i
2. t2 := d[t1]
3. t3 := a + 10
4. t4 := 12 * t3
5. t5 := 4 * t2
6. t6 := t5 + 1048
7. t7 := e[t6]
8. t8 := t2 + t7
9. f[t8] := t4
```

19. **Définitions orientées-syntaxe**. Considérons le langage

$$L_{19} = \left\{ \; w \in \{\mathtt{a}, \mathtt{b}\}^* \;\middle|\; |w|_{\mathtt{a}} = |w|_{\mathtt{b}} + 2020 \; \right\}.$$

Toutes les DOS proposées dans les choix de réponse sont capables de reconnaître L_{19}, sauf une. Laquelle? Encore une fois, chaque DOS synthétise un attribut $S.ok$ qui est vrai si et seulement si la chaîne d'entrée fait partie de L_{19}. Notez que la notation $|w|_c$ indique combien de fois le symbole c apparaît dans w.

(a)

Productions	Règles sémantiques
$S \to A$	$S.ok := (A.c = 0)$
$A \to \mathtt{a}\, A_1$	$A.c := A_1.c + 1$
$A \to B$	$A.c := -B.c$
$B \to \mathtt{b}\, B_1$	$B.c := B_1.c + 1$
$B \to \epsilon$	$B.c := 2020$

(b)

Productions	Règles sémantiques
$S \to A$	$S.ok := (A.c = 2020)$
$A \to \mathtt{a}\, A_1$	$A.c := A_1.c + 1$
$A \to \mathtt{b}\, A_1$	$A.c := A_1.c - 1$
$A \to \epsilon$	$A.c := 0$

(c)

Productions	Règles sémantiques	
$S \to A$	$S.ok := (A.a = A.b + 2020)$	
$A \to \mathtt{a}\, A_1$	$A.a := A_1.a + 1;$	$A.b := A_1.b$
$A \to \mathtt{b}\, A_1$	$A.a := A_1.a;$	$A.b := A_1.b + 1$
$A \to \epsilon$	$A.a := 0;$	$A.b := 0$

(d)

Productions	Règles sémantiques	
$S \to A$	$S.ok := (A.g = A.d)$	
$A \to \mathtt{a}\, A_1$	$A.g := A_1.g + 1;$	$A.d := A_1.d$
$A \to \mathtt{b}\, A_1$	$A.g := A_1.g;$	$A.d := A_1.d + 1$
$A \to \epsilon$	$A.g := 0;$	$A.d := 2020$

20. **Définitions orientées-syntaxe**. Cette question est conçue comme un puzzle logique. Il s'agit d'abord de déterminer si les divers attributs sont hérités ou synthétisés. Les attributs sont ceux qui apparaissent dans la DOS suivante.

Productions	Règles sémantiques		
$S \rightarrow A$	$A.a := 0;$	$A.b := 0;$	$S.ok := (A.c = 2020)$
$A \rightarrow \mathtt{a}\, A_1$	$A_1.a := A.a + 1;$	$A_1.b := A.b;$	$A.c := A_1.c$
$A \rightarrow \mathtt{b}\, A_1$	$A_1.a := A.a;$	$A_1.b := A.b + 1;$	$A.c := A_1.c$
$A \rightarrow \epsilon$			$A.c := A.a - A.b$

Parmi les énoncés faits dans les choix de réponses, un seul est vrai (et, donc, les trois autres sont faux). Il faut identifier l'énoncé qui est vrai.

(a) L'attribut $A.b$ est hérité et la réponse (c) est fausse.

(b) L'attribut $A.c$ est hérité si et seulement si la réponse (d) est fausse.

(c) Si la réponse (b) est vraie, alors la réponse (a) doit être vraie aussi.

(d) L'attribut $A.a$ est de la même nature (synthétisé versus hérité) que l'attribut $A.c$.

Remise des travaux

Vous devez remettre votre travail en complétant le questionnaire du TP#2 sur **monPortail**. Notez que le questionnaire ne répète pas les questions; il ne présente que les divers choix.