Examen de mi parcours

Vous trouverez ici des éléments de corrections et commentaires de l'examen : n'hésitez pas à les lire en attendant d'avoir accès à vos notes.

Les 3 paragraphes suivants étant dédiées à la possibilité de consulter sa copie, elle ne concerne (pour ma part) que les étudiants de Grenoble ; (une autre façon de faire sera sûrement proposée par Valentin pour Valence).

Vos notes seront mises à disposition (de manière "anonymisée") sur caseine : le 29 Mars (dans un document à part). Si vous voyiez une incohérence du type "il est inscrit ABS alors que vous étiez là à l'examen", manifestez le au plus tôt.

Après cette date, si vous souhaitez consulter votre copie, vous aurez jusqu'au 5 avril pour m'envoyer un mail : vincent.fagnon@univ-grenoble-alpes.fr
Je répondrais le 6 avril pour vous proposer une date de consultation de votre copie (date qui sera commune pour toutes les personnes s'étant manifestées par mail).

ps : Je répondrais aux questions *"pourquoi je n'ai pas eu les points à cet exercice ?"* Si Seulement Si vous avez lu les éléments de correction présentées dans la suite de ce pdf.

L1-IMA : Modélisation des structures informatiques
Aspects formels

Exam/Partiel – 14 mars 2023 – durée 1 h 30

Sont interdits : les documents, les ordinateurs, les téléphones (incluant smartphone, tablettes,... tout ce qui contient un dispositif électronique).
Il sera tenu compte de la qualité de la rédaction et de la clarté de la présentation. Les exercices sont indépendants.

Éléments de réponse : Note a l'attention du lecteur.
L'exercice 1 ressemble aux TP et à la feuille TD2. Il se peut qu'en TP, vous ayez réussi à passer les jeux de tests avec votre code, que vous vous soyez inspiré de celui ci pour répondre aux questions de cet exercice, et que malgré tout, vous n'ayez pas eu les points pour avoir réussi l'exercice 1. Ce n'est pas parce que vous passez les jeux de test du TP que votre code est juste !
Les formules de l'exercice 2 sont des formules présentes dans la feuille de TD3. Néanmoins le questions sont formulées différemment.
L'exercice 3. Questions de cours... Vous êtes 180 inscrits dans la promo, mais n'êtes que 40 grand max à venir en CM... Mais si vous estimez que vous n'avez pas besoin d'aller en CM (ni en TD, ni en TP) pour obtenir votre diplôme...
L'exercice 4 : Le but n'étant pas que vous passiez une heure d'examen à écrire une preuve, on vous en donnait une, et on vous demandait de repérer les fautes. Et comprendre le fonctionnement d'une preuve par récurrence... En cours, on vous explique quels sont les façons d'écrire une preuve. Mais pourquoi les "preuves" qu'on vous présentent prouvent quelque chose ? Un exercice où il fallait prendre du recul, et comprendre le lien entre "écrire un algorithme qui fait quelque chose" et écrire une preuve qui montre "qu'il est possible de faire quelque chose".

Exercice 1 : Algorithmes (Écriture et Analyse) ($\sim 35min$)

Vous disposez d'une fonction *Séparer*(T) qui prend en entrée une formule bien formée T = (R connecteur S), et retourne un triplet (R,C,S), avec R la sous formule de gauche, S la sous formule de droite, et C le connecteur entre R et S. Elle s'utilise de la manière suivante :

```
(R,C,S) = Séparer ((A ∧ B))
print R          # affiche la sous formule A. (ici, R== la sous formule A),
print C          # affiche le caractère "∧". (ici, C=="∧")
print S          # affiche la sous formule B.

(R,C,S) = Séparer ((¬ B))
print R          # affiche "vide". (ici, R=="vide")
print C          # affiche le caractère "¬"
print S          # affiche la sous formule B.

(R,C,S) = Séparer ((x)) : avec x une variable propositionnelle
print R          # cela affiche "vide", (R=="vide")"
print C          # il n'y a pas de connecteur C : C=="vide" : cela affiche "vide"
print S          # affiche "x".
```

1. Écrire la fonction *contient_négation*(), qui prend en entrée une formule bien formée T, qui retourne 1 si l'opérateur ¬ se trouve dans une des sous formules de T, 0 sinon. *(Par convention, on considère que T est une sous formule de T.)*

Exemples de ce que doit retourner *contient_négation*() si on suppose que A, B, C ne contiennent pas d'opérateurs ¬ :
contient_négation((A ∧ (B ∧ C))) = 0
contient_négation((A ∧ (B ∧ ¬ C))) = 1
contient_négation((¬ C)) = 1

Éléments de réponse : Vu dans pas mal de copies.

```
contient_négation(T)
  (R,C,S) = Separer(T)
  if C=="¬"
    return 1
  else
    return 0
```

Prenons la formule $A \vee (B \wedge \neg D)$. C = "∧", donc vous rentrez pas dans le IF, donc vous retournez 0. Alors qu'il y a un ¬ dans la sous formule de droite.

Éléments de réponse : On vous donne la fonction Séparer. (f1,f2,f3)= Séparer(T). Vous êtes pas mal à avoir utilisé les fonctions du TP à la place de Séparer. Pourquoi pas. Mais dans ce cas, il fallait les utiliser correctement (et ne pas confondre nb operandes et nboperateurs). Séparer() ici présente était plus facile à utiliser.
Vu dans pas mal de copies : *if nb_opérandes(T) == 1 : (if opérateur == "∨" : return 0, else return 1)*
Si nb opérandes ==1, ça veut dire que R est vide. Donc soit l'opérateur est ¬, soit l'opérateur et vide. Il ne peut donc jamais être égal a ∨. "if opérateur == "∨" : return 0" n'a donc aucun sens. Si on le code, effectivement en TP, ça fonctionne, vous ne rentrez jamais dans le if, vous alle toujours dans le else, et votre code passe les jeux de test. Mais vous montrez en une ligne que vous ne comprenez pas ce que votre code fait.

Éléments de réponse : Une façon de faire

```
contient_négation(T)
  if T==vide
    return 0 # c'est pas necessaire, mais comme on a pas dit ce qu'il se passait si Separer("vide")
  R,C,S=Séparer(T)
  if C==vide
    return 0 # ce if permet de retourner 0 dans les cas de base.
  return contient_négation(R) ∨ contient_négation(S) ∨ (C=="¬")
  # On peut aussi faire :
  if C=="¬"
    return 1
  else
    return max(contient_négation(R), contient_négation(S))
  # Et pas contient_négation(R) + contient_négation(S). si jamais y'a un non des deux cotés,
  votre fonction retourne la valeur 2, ça correspond pas a ce qu'on veut.
```

Éléments de réponse : Si vous voulez à tout pris faire de l'itératif... Pour accéder aux sous formules de T, il faut ré-appliquer séparer sur la sous formule de droite, et sous la sous formule de gauche. Il s'agit de sous formules, pas d'une liste (structure python) où vous pouvez faire un banal if $return(\neg \in T)$. Dans l'idée, si a l'étape 1, vous devez faire séparer sur R et sur S, il y a des chances que vous deviez faire séparer sur 4 enfants l'étape d'après (la sous formule gauche de R, la sous formule droite de R, pareil pour S ...). La bonne façon de faire c'est le récursif, mais si vous voulez à tout pris faire de l'itératif, ça se passe comme ça (et faut rajouter des cas si T vide) :

```
contient_négation(T)
    liste_a_traiter = T          # une liste des sous formules encore à traiter
    while liste_a_traiter n'est pas vide
        (f1,f2,f3) = Separer(premier_element_de_la_liste)
        if f2 == "vide"
            # On est dans le cas où f3 est soit vide, soit une variable, soit une constante
            On ne fait rien
        if f2 == "¬"
            tab # On a trouvé un non, ça sert a rien de continuer a chercher.
            return 1
        else on ajoute f1 (=R) et f3 (=S) dans la liste_a_traiter
    # Si on a fini de tout traiter (la liste a traiter est vide) c'est qu'on a pas trouvé de non, donc
    return 0
```

2. Écrire la fonction *nbr_ou*(), qui prend en entrée une formule bien formée T, qui retourne le nombre d'opérateurs $\vee$ présents dans T.

Exemples de ce que retourne *nbr_ou*() en supposant que A, B, C ne contiennent pas d'opérateurs $\vee$
nbr_ou((A $\vee$ (B $\vee$ C))) = 2
nbr_ou((A$\wedge$ (B $\vee$ $\neg$ C))) = 1 ;
nbr_ou($\neg$ C)) = 0

Éléments de réponse : Même combat. Sauf que là il faut faire des +.

```
nbr_ou(T)
    If (T==vide) : return 0
    (R,C,S)=Séparer(T)
    If (C=="vide") : return 0
    return (nbr_ou(R)+nbr_ou(S)+ (C=="∨"))          #le dernier = 1 si vrai, 0 sinon
```

3. On souhaite maintenant écrire la fonction *fct_c*(T) qui retourne :
   - le nombre de "ou" présents dans T si il n'y a pas de ¬ dans T.
   - la valeur "18" si il y a un ¬ dans T.

   (a) Écrire la fonction *fct_c* en réutilisant vos deux fonctions précédentes.

Éléments de réponse : *fct_c*(T)
if(*contient_négation*(T)) retourne 18
else retourne *nbr_ou*(T)

Éléments de réponse :

```
fct_c(T) if contient_négation(T)
    return 18
  else
    nbr_ou(T)
```

   (b) **(difficile)** Vous aurez remarqué que votre fonction précédente parcours récursivement 2 fois l'"arbre" de la formule T. Proposer une solution qui fasse ce que l'on attend de la fonction *fct_c* mais qui ne ferait qu'un seul parcours récursif de T. *(Vous n'êtes pas obligés de tout faire au sein d'une unique fonction)*

Éléments de réponse : Ce qui est compliqué ici, c'est si vous appelez fct_c sur R, et qu'il vous donne la valeur 18, est ce que ça veut dire que R contient 18 "ou", ou alors que R contient un "not" ? Et donc, est ce que ça veut dire que vous (fct(T)), vous devez retourner 18+un nombre de ou, ou alors retourner 18 car il y a un "not" dans R.... Une façon de faire en récursif (une fonction d'encapsulation) :

```
fct_c(T)
  a = fct_bis(T)
  if a==-1
    retourne 18
  else
    retourne a
```

```
fct_bis(T)
  if T==vide
    retourne 0
  R,C,S=Séparer(T)
  a=fct_bis(R)
  b=fct_bis(S)
  if (C == ¬) ∨ (a == −1) ∨ (b == −1)
    retourne -1
  else
    retourne a + b + (0, 1)[C == ∨]
```

Exercice 2 : Formules ((Vrai $\vee$ Faux) $\wedge$ Justifier) ($\sim 10min$)

Pour chaque proposition suivante, vous devez dire si la proposition est vraie ou fausse, et justifier brièvement votre réponse. *(Une phrase courte et claire vaut mieux qu'un discours long et incompréhensible.)*

1. L'évaluation de la formule suivante dépend de l'interprétation de r : $\big((p \to q) \wedge (r \to (s \vee r))\big) \wedge \big((\neg p) \to (\neg q)\big)$

   Éléments de réponse : Qu'est ce que veut dire cette question en langage plus courant : est ce que ça change quelque chose que r soit égal a 1 ou que r soit égal à 0 pour cette formule ?
   Dire : "regardez sur cet exemple ça ne change rien" : ça ne suffit pas. Il faut montrer que quelque soit l'exemple, la valeur de r ne change rien.
   Donc soit vous faites une table de vérité, et vous montrez que la valeur de la formule pour lignes r=0 et r=1 sont les mêmes QUELQUE SOIENT les valeurs de p et q et s (ça fait beaucoup de lignes à faire dans la table de vérité (16) !
   Sinon, vous prenez un peu plus le temps de regarder ce qu'il se passe, et vous voyez que vous pouvez faire un peu de substitution. $(r- > (s \vee r)) <=> (r- > s \vee r- > r) <=> r- > s \vee \top <=> \top$
   La formule devient : $\big((p \to q) \wedge (\top) \wedge \big((\neg p) \to (\neg q)\big)$: et voila, il n'y a plus de r dans la formule. Donc l'évaluation de la formule ne dépend pas de l'interprétation de r. La réponse était donc Faux.

2. Ces deux formules sont équivalentes :

   — (x $\wedge$ y) $\to$ z
   — x $\to$ (y $\to$ z)

   Éléments de réponse : Plusieurs façons de faire ici, pareil qu'au dessus, une table de verité et vous montrez que les deux formules ont les memes valeurs de vérité. Ou alors par substitution. (pour vous donner l'intuition) Si on essaye de faire des tables de vérité sans s'embeter à tracer le tableau, le seul cas negatif pour l'implication est $1- > 0$ (d'un truc juste on ne peut pas déduire un truc faux.) Donc pour la premiere formule, c'est quand x et y = 1, et z=0.
   Pour la seconde, il faut x=1, et il faudrait que $y- > z$ = 0, qui n'arrive que quand y=1 et z=0.
   On remarque que la premiere formule est fausse SSI x=1,y=1,x=0, ce qui est aussi le cas de la seconde. Vrai.

   Ps : pour faire le lien avec la partie du cours sur les Relations, supposons une relation R. soit a est en relation avec b, soit il ne l'est pas. On va dire aRb = x avec x = 0 ou 1 (on sait pas si a et b sont en relation). idem pour b et c : y = bRc, et idem pour a et c :z=aRc. Et bien on peut dire que la relation est transitive SSI Pour tout x,y,z : (x $\wedge$ y) $\to$ z ; et c'est pareil de dire que R est transitive SSI pour tout x,y,z : x $\to$ (y $\to$ z).

3. Ces deux formules sont équivalentes :
   — $\big(p \wedge (q \vee \neg q)\big) \wedge \big((p \to q) \wedge (q \to p)\big)$
   — $p \wedge (p \leftrightarrow q)$

   Éléments de réponse : On peut remarquer que $(q \vee \neg q) <=> \top$; donc on peut remplacer $p \wedge (q \vee \neg q)\big)$ par p dans la première formule. Et puis $\big((p \to q) \wedge (q \to p)\big)$, c'est équivalent à $(p <=> q)$. Bon bah voila. les deux formules sont équivalentes. Vous pouviez aussi passer par une table de vérité et comparer les colonnes, mais c'est plus long.

Exercice 3 : Questions de cours et définitions ($\sim 20min$)

1. Énoncer les lois de distributivité de De Morgan

   Éléments de réponse : C'était la distributivité des NON. C'était d'ailleurs (presque) dit dans l'exercice 4, cas inductif, (d).
   $(\neg(A \wedge B)) <=> (\neg A \vee \neg B)$
   $(\neg(A \vee B)) <=> (\neg A \wedge \neg B)$

2. Soit P une formule propositionnelle construite à partir des variables r et s. Soit ($r = 1$, $s = 1$) un modèle de P. Que pouvez vous dire de ($r = 0$, $s = 0$) ?

   Éléments de réponse : On ne peut rien en déduire. Vous ne connaissez pas la tête de la formule P. Imaginez que P = $r \vee s$, dans ce cas 00 est un contre modèle. Mais si P = $r \vee \neg s$, 00 est un modèle. si $p = r- > s$ c'est un modèle... On ne peut rien dire sans connaître P.

3. Voici une formule : $\big(\neg(p \wedge (q \vee q))\big) \rightarrow \big(\neg(\neg p \vee q))\big)$
   (a) Donnez un modèle de cette formule
   (b) Donnez un contre modèle de cette formule
   (c) Donnez une forme disjonctive de cette formule
   (Rappel de cours : une forme disjonctive est une disjonction de clauses conjonctives)
   (d) En déduire une forme conjonctive (détaillez les étapes de calculs).

   Éléments de réponse : On peut faire la table de vérité :
   p q F
   0 0 0
   0 1 0
   1 0 1
   1 1 1
   Donc les modèles sont p=1 q=0 ET p=1 q=1, les contre modèles sont donc les deux autres.
   Une forme disjonctive, c'est donc des $\wedge$ de clauses où il y a des $\vee$ (c'est pas grave si vous avez confondu). Une façon de faire, vous bourrinez la formule, avec des substitutions ect ect. Sinon vous regardez la table de vérite. La formule est vraie dans le cas où p=1 q=0 ou dans le cas où p=1 q=1. ok. Donc elle est équivalente à $(p \wedge q) \vee (p \wedge \neg q)$ (c'est vrai dans les mêmes cas, faux sinon.
   Pour en déduire l'autre forme, soit méthode bourrin, vous développez. Soit vous voyez que $(p \wedge q) \vee (p \wedge \neg q) <=> p$.

4. Voici une autre formule : $\big((\top \vee q) <=> (p \wedge \top)\big) \rightarrow \big(\neg\big((\neg p \wedge p) \wedge q\big)\big)$
   (a) Donnez un modèle de cette formule.
   (b) Donnez un contre modèle de cette formule
   (c) Donnez une forme conjonctive OU disjonctive de cette formule.

   Éléments de réponse : Résolution : Si on developpe la formule.
   $\big((\top \vee q) <=> (p \wedge \top)\big) \rightarrow \big(\neg\big((\neg p \wedge p) \wedge q\big)\big)$
   $\big(\top <=> p\big) \rightarrow \big(\neg\big(\bot \wedge q\big)\big)$
   $p \rightarrow \top$
   $\top$
   N'importe quelle affectation de variable est un modele (du point cadeau là, vous n'aviez qu'a répondre un truc au pif). Il n'y a pas de contre modèle. Une forme conjonctive ou dijsonctive ? $\top, p \vee \not{p}$, peu importe tant que vous respectiez les règles des formes clausales vues en cours. ps : et $\top$ c'est une constante qui vaut toujours 1.... pas une variable à mettre a 0 et 1 dans une table de vérité.... faut venir en cours !

Exercice 4 : Preuve (Comprendre et Corriger) ($\sim 25min$)

Un étudiant est confronté en examen à la question suivante :

> Supposons que vous avez à disposition un algorithme $\mathcal{A}$ qui, lorsqu'on lui donne une formule bien formée T composée uniquement de connecteurs $\in \{ \neg , \vee \}$ retourne une formule bien formée équivalente, T', qui n'est composée quant à elle que de connecteurs logiques $\in \{ \neg , \rightarrow \}$.
>
> Démontrer qu'il est possible de transformer toute formule bien formée (composée des opérateurs vus en cours) en une formule équivalente composée uniquement d'opérateurs $\in \{\neg , \rightarrow\}$.

Cet étudiant rédige sur sa copie la preuve suivante. Malheureusement, elle n'est pas correcte. La voici :

> On considère une formule bien formée F.
>
> 1. Cas de base :
>   (a) Soit F est vide : F ne contient donc aucun connecteur logique. Comme $\emptyset \in \{\neg, \rightarrow\}$; il n'y a rien a faire, F est déjà de la forme qu'on veut.
>   (b) Soit F est une variable propositionnelle. On doit alors appliquer l'algorithme $\mathcal{A}$ sur F qui nous donne une formule F', avec F' une fbf equivalente à F. Donc il existe bien une façon de transformer F en une formule de la forme qu'on veut.
> 2. Cas Inductif :
>   (a) F est de la forme R $\rightarrow$ S : $\rightarrow$ appartient à l'ensemble des connecteurs "autorisés" dans la forme que l'on souhaite, donc il est possible de transformer F dans la forme que l'on veut si et seulement si il est possible de transformer R dans la forme que l'on veut ET si il est possible de transformer S dans la forme que l'on veut.
>   (b) F est de la forme $\neg$ S : On a le droit d'utiliser e connecteur $\neg$ dans la forme qu'on désire obtenir, donc F est déjà de la forme qu'on veut.
>   (c) F est de la forme R $\vee$ S : même si dans les sous formules R et S, il y a peut etre des connecteurs qui n'appartiennent pas à $\{\neg , \vee\}$ je peux utiliser l'algorithme $\mathcal{A}$ en faisant un changement de variables (pour "cacher les connecteurs interdits"). Le $\vee$ entre R et S est donc transformé par $\mathcal{A}$ en des $\neg$ et des $\rightarrow$ entre des R et des S on ne sait pas comment. Mais il suffit de dire qu'il est possible de transformer F dans la forme que l'on veut si et seulement si il est possible de transformer R dans la forme que l'on veut ET si il est possible de transformer S dans la forme que l'on veut.
>   (d) F est de la forme R $\wedge$ S : On peut appliquer De Morgan pour transformer R $\wedge$ S en une formule qui dépend de R, S, et qui fait apparaître les connecteurs $\neg$ et $\vee$, qu'on peut gérer dans le cas inductif précédent.
> 3. Conclusion : A partir de tous ces cas inductifs, on a montré qu'on est capable de transformer la formule qui nous interesse si et seulement si on est capable de transformer une sous formule "plus petite". De plus on est capables de transformer les cas de base. Donc on peut transformer une formule aussi petite qu'on veut.

1. **(difficile)** Pourquoi la preuve est-elle incorrecte ? Que faudrait-il modifier pour que la preuve soit valide. *Vous pouvez justifier vos réponses si vous en ressentez le besoin.*

Éléments de réponse : Regardons la conclusion en premier. Vous, vous avez l'habitude de l'inverse dans une preuve par reccurence : on part de n=0, et on dit que si on sait résoudre le probleme de taille n, alors on sait résoudre de taille n+1. Ici, on regarde plutôt l'aspect : on sait résoudre un problème de taille SSI on sait résoudre un problème de taille n-1, et qu'on sait résoudre un problème de taille n=0.
Ce qui est poposé par l'étudiant : si pour résoudre une formule d'une certaine taille, je peux soit le faire directement, soit réussir à transformer le problème en "résoudre une (ou plusieurs) sous formule(s) de taille plus petite", et que je peux résoudre une formule de taille 1 (un cas de base), alors je peux résoudre le problème peut importe la taille de ma formule ($>$1).
Qu'est ce qu'on pouvait dire ? (liste non exaustive)
- on ne traite pas le cas du $R <=> S$
- le cas de base : F est une constante n'est pas traité
- dans le cas de base "variable propositionnelle", il n'y a pas de connecteur, donc pas besoin d'appliquer l'algo A pour faire disparaitre les connecteurs interdits.
- dans le cas inductif $\neg S$, ok on a le droit de garder le NON, mais il faut verifier S (comme au dessus)
- on peut se questionner sur la validité de l'algorithme.
- ce n'était pas une erreur de dire "on ne sait pas comment" dans le cas $R \vee S$. On ne sait pas ce que fait l'Algorithme A. Peut etre que A transforme $R \vee S$ en $\neg R- > S$, peut etre qu'il la transforme en $(R- > R)- > (\neg R- > S)$ on ne sait pas comment il le fait, mais on sait qu'il fait disparaître le OU.
- ...