NFP120 - Examen 1ère session 2013

Durée 3h – Tout document autorisé

1 Logique

Sémantique des formules

Ex. 1 — (1pt) La formule propositionnelle suivante est-elle valide ? Justifiez.

$$x \vee (\neg x \wedge (y \vee \neg y)$$

Solution (Ex. 1) — Faire une table de vérité de la formule et montrer que toute les lignes donnent *Vrai*.

Ex. 2 — (1pt) À l'aide des tables de vérité, montrez que $x \to y$ est la conséquence logique de y et $\neg x$ (ce qui s'écrit $y, \neg x \models x \to y$).

Solution (Ex. 2) — Faire la table de vérité de la formule et montrer que sur chaque ligne où y est vrai $x \to y$ est vrai aussi.

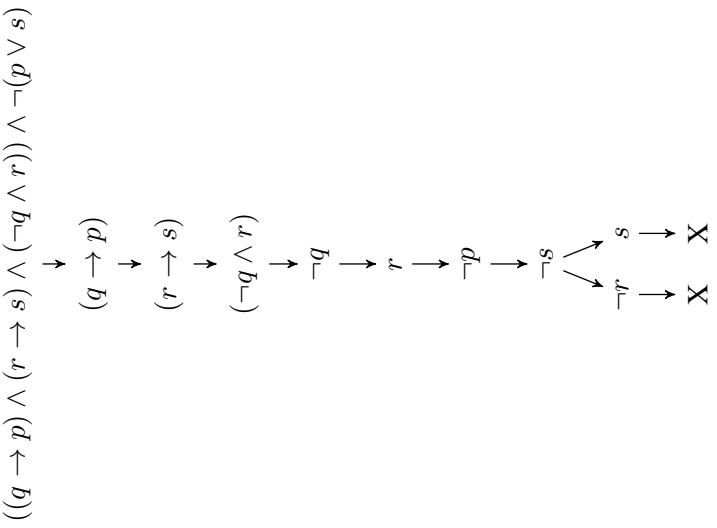
Tableaux sémantiques

Vous trouverez un résumé des règles de résolution des tableaux dans l'annexe A.

Ex. 3 — (2pt) Démontrez à l'aide de la méthode des tableaux la propriété suivante :

$$((q \to p) \wedge (r \to s) \wedge (\neg q \wedge r)) \vdash (p \vee s)$$

Solution (Ex. 3) —



2 Sémantique et Logique de Hoare

Vous trouverez un résumé des règles de Hoare dans l'annexe, ainsi qu'une partie des règles de sémantique vues en cours.

Prédicats sur les tableaux

Exemple : le prédicat *toutazero*(t, i, j), signifiant que les cases i à j (i et j compris) du tableau t sont égales à zéro, peut être défini par la formule logique suivante : *toutazero*$(t, i, j) = \forall k, i \leq k \leq j \rightarrow t[i] = 0$.

Ex. 4 — (1pt) Définissez de la même manière le prédicat *permut*(t, t', i, k) signifiant que les cases i à k (i et k compris) des tableaux t et t' sont les mêmes à une permutation près. *On supposera que le tableau n'a pas deux cases identiques*.

Solution (Ex. 4) —

$$\forall a, i \leq a \leq k \rightarrow \exists a', i \leq a' \leq k \wedge t[a] = t[a']$$

Ex. 5 — (1pt) Définissez de la même manière le prédicat *permut2*(t, t', i, j, k, l) signifiant que les cases i à j de t et les cases k à l de t' sont les mêmes à une permutation près. *On supposera que le tableau n'a pas deux cases identiques*.

Solution (Ex. 5) —

$$\forall a, i \leq a \leq j \rightarrow \exists a', k \leq a' \leq l \wedge t[a] = t[a']$$

Ex. 6 — (5pt (question bonus difficile, à essayer uniquement si vous avez le temps)) Comment définir les prédicats des deux questions précédentes sans supposer que la tableau n'a pas deux cases identiques ?

On supposera dans la suite, même si vous n'avez pas répondu à cette question, que les prédicats permut *et* permut2 *sont définis et corrects même pour des tableaux avec cases identiques.*

Invariant A

On rappelle que $!n$ désigne la factorielle de l'entier n, autrement dit : $!n = 1 \times 2 \ldots \times n$. Cas particulier : $!0 = 1$. On sait entre autre que pour $n \geq 1$, $!n = n \times !(n-1)$.

Soit la boucle A ci-contre.

A :
```
while i <= n do
  res := res*i;
  i := i+1
done
```

Ex. 7 — (0,5pt) Donnez un variant de cette boucle.

Solution (Ex. 7) — $n - i$

Ex. 8 — (1,5pt) Donnez un invariant de cette boucle permettant de démontrer le triplet de Hoare suivant :

$$\{\texttt{i} = 1 \wedge res = 1 \wedge n \geq 0\}\ A\ \{res = !n\}.$$

Solution (Ex. 8) — $res = !(i-1) \wedge 0 < i \leq n+1$

Ex. 9 — (2pt) Donnez l'arbre d'exécution sémantique de cette boucle pour l'environnement de départ $\sigma = [\texttt{i} = 2; \texttt{n} = 2; \texttt{res} = 1]$ (qu'on notera $[2;2;1]$ pour gagner de la place).

Solution (Ex. 9) —

$$\dfrac{\dfrac{}{\langle [2;2;1], \texttt{i<=n}\rangle \to \texttt{true}} \quad \dfrac{\dfrac{\langle [2;2;1], \texttt{res*i}\rangle \to 2}{\langle [2;2;1], \texttt{res:=res*i}\rangle \rightsquigarrow [2;2;2]} \quad \dfrac{\langle [2;2;2], \texttt{i+1}\rangle \to 3}{\langle [2;2;2], \texttt{i:=i+1}\rangle \rightsquigarrow [3;2;2]}}{\langle [2;2;1], \texttt{res:=res*i; i:=i+1}\rangle \rightsquigarrow [3;2;2]} \quad \dfrac{}{\langle [3;2;2], \texttt{i<=n}\rangle \to \texttt{false}}}{\langle [2;2;1], A\rangle \rightsquigarrow [3;2;2]}$$

Ex. 10 — (2,5pt) Donnez l'arbre de preuve (correction partielle, ne mettez pas le variant) du triplet de Hoare de la question précédente : $\{\texttt{i} = 1 \wedge res = 1 \wedge n \geq 0\}\ A\ \{res = !n\}$.

Solution (Ex. 10) —

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{res = !(i-1) \wedge 0 < i \leq n+1 \wedge i \leq n\}\texttt{res:=res*i}\{res = !(i-1) \wedge 0 < i \leq n+1 \wedge i \leq n\}}\ \text{AFF} \quad \dfrac{\dfrac{}{\{res_0 = !(i-1) \wedge 0 < i \leq n+1 \wedge i \leq n \wedge res = res_0 * i\}\texttt{i:=i+1}\{res_0 = !(i_0-1) \wedge i_0 \leq n+1 \wedge i_0 \leq n \wedge res = res_0 * i_0 \wedge i = i_0+1\}}\ \text{AFF}}{\{res = !(i-1) \wedge 0 < i \leq n+1\}}\ \text{CONSEQ}}{\{res = !(i-1) \wedge 0 < i \leq n+1 \wedge i \leq n\}\texttt{res:=res*i;i:=i+1}\{res = !(i-1) \wedge 0 < i \leq n+1\}}\ \text{SEQ}}{\{res = !(i-1) \wedge 0 < i \leq n+1\}\ A\ \{res = !(i-1) \wedge 0 < i \leq n+1 \wedge i > n\}}\ \text{WHILE}}{\{\texttt{i} = 1 \wedge res = 1 \wedge n \geq 0\}\ A\ \{res = !n\}}\ \text{CONSEQ}$$

Invariant C

Soit le programme C suivant, qui correspond à la boucle interne du tri par insertion : on cherche à insérer une valeur `k` dans un morceau de tableau déjà trié. On parcourt le tableau de droite à gauche en décalant les cases vers la droite jusqu'à trouver l'emplacement pour `k`. Le booléen `fini` permet de sortir lorsqu'on a trouvé l'emplacement (il vaut **`false`** initialement). Prenez le temps de comprendre comment la boucle fonctionne.

C :

```
while(j>0 and not fini) do
  if t[j-1] <= k then
      fini := true
  else
    begin
      t[j] := t[j-1];
      j = j - 1
    end
done;
t[j] := k
```

Ex. 11 — (0,5pt) Donnez un variant de cette boucle.

Solution (Ex. 11) — `j`

Ex. 12 — (1pt) Donnez un triplet de Hoare de la forme `{P}` C `{Q}` exprimant la propriété suivante : si la valeur de `k` contient initialement la valeur (initiale) de `t[j]` alors le tableau (entre 0 et j) contient à la fin de l'algorithme une permutation de ses valeurs initiales.
On notera t_0 le tableau initial, j_0 et k_0 les valeur initiale de `j` et `k` et on pourra utiliser le prédicat de l'exercice 6 (même si vous n'avez pas répondu à l'exercice).

Solution (Ex. 12) —

$$\{k = t[j] \wedge t = t_0 \wedge j = j_0\} C \{\textit{permut}(t_0, t, 0, j_0)\}$$

Ex. 13 — (2,5pt) Donnez un invariant de la boucle suffisant pour en déduire la post-condition `Q` à la sortie de la boucle.

Solution (Ex. 13) —

$$0 \leq j \leq j_0 \wedge \forall k, j < k \leq j_0 \rightarrow t[k] = t_0[k-1] \wedge \forall k, 0 \leq k < j \rightarrow t[k] = t_0[k]$$

Ex. 14 — (1pt) Donnez un triplet de Hoare de la forme `{P'}` C `{Q'}` exprimant l'autre propriété de ce programme, à savoir que si le tableau est trié entre 0 et $j_0 - 1$ au début, il est trié entre 0 et j_0 à la fin.
On note j_0 la valeur de `j` au début de l'algorithme et on s'autorise l'utilisation des prédicats définis dans l'exercice 6 (même si vous n'avez pas répondu à l'exercice).

Solution (Ex. 14) —

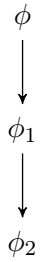
$$\{fini = \textbf{false} \wedge \textit{trié}(t, 0, j-1) \wedge j = j_0\} C \{\textit{trié}(t, 0, j_0)\}$$

Ex. 15 — (2,5pt) Donnez un invariant de la boucle suffisant pour en déduire la post-condition à la sortie du programme. Attention il y aune subtilité : n'oubliez pas que l'invariant doit être vrai *même si la boucle est utilisée zéro fois*. À défaut n'hésitez pas à mettre un invariant « un peu faux » en expliquant pourquoi il est faux.

Solution (Ex. 15) — Pour être complètement correct il faut distinguer le premier tour des suivants. Au premier tour seul les $j_0 - 1$ cases sont triées, dès le deuxième tour le j_0 premières cases son triées car la case j_0 contient $t_0[j_0 - 1]$.

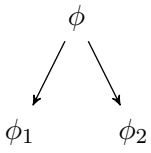
$$
\begin{array}{ll}
 & 0 \leq j \leq j_0 \\
\wedge & j = j_0 \rightarrow \textit{trié}(t, 0, j_0 - 1) \\
\wedge & j < j_0 \rightarrow \textit{trié}(t, 0, j_0) \\
\wedge & \forall l. j \leq l \leq j_0 \rightarrow k < t[l]
\end{array}
$$

A Annexe : Règle de la méthode des tableaux



ϕ	ϕ_1	ϕ_2
$\neg\neg\phi_1$	ϕ_1	
$\phi_1 \wedge \phi_2$	ϕ_1	ϕ_2
$\neg(\phi_1 \vee \phi_2)$	$\neg\phi_1$	$\neg\phi_2$
$\neg(\phi_1 \rightarrow \phi_2)$	ϕ_1	$\neg\phi_2$
$\phi_1 \leftrightarrow \phi_2$	$\phi_1 \rightarrow \phi_2$	$\phi_2 \rightarrow \phi_1$
$\exists x, (\phi_1(x))$	$\phi_1(a_i)$	
$\forall x, (\phi_1(x))$	$\phi_1(c)$	
$\neg\exists x, (\phi_1(x))$	$\neg\phi_1(c)$	
$\neg\forall x, (\phi_1(x))$	$\neg\phi_1(a_i)$	

FIGURE 1 – Les règles de type « et », qui ne créent pas d'embranchement. a_i désigne une nouvelle constante « fraîche » et c désigne une constante existante.



ϕ	ϕ_1	ϕ_2
$\phi_1 \vee \phi_2$	ϕ_1	ϕ_2
$\neg(\phi_1 \wedge \phi_2)$	$\neg\phi_1$	$\neg\phi_2$
$\phi_1 \rightarrow \phi_2$	$\neg\phi_1$	ϕ_2
$\neg(\phi_1 \leftrightarrow \phi_2)$	$\neg(\phi_1 \rightarrow \phi_2)$	$\neg(\phi_2 \rightarrow \phi_1)$

FIGURE 2 – Les règles de type "ou", qui créent un embranchement

On s'autorise à appliquer plusieurs fois la règles sur le même quantificateur si la formule commence par plusieurs application du même quantificateur. Par exemple si la formule est de la forme $\forall x, \forall y$ on s'autorise à instancier x et y en une seule fois. De même si la formule est de la forme $\exists x, \exists y$ on s'autorise à donner un nom frais à x et à y en une fois (bien sûr les deux noms doivent être différents).

B Récapitulatif des règles de Hoare

$$\textsc{Aff}\ \frac{}{\{P\}\,\mathtt{x:=}E\,\{P\mathtt{[x \leftarrow x_0]} \wedge \mathtt{x=}E\mathtt{[x \leftarrow x_0]}\}}$$

$$\textsc{Conseq}\ \frac{P \Rightarrow P' \quad \{P'\}\ C\ \{Q'\} \quad Q' \Rightarrow Q}{\{P\}\ C\ \{Q\}} \qquad \textsc{Seq}\ \frac{\{P\}\ \mathtt{C_1}\ \{Q\} \qquad \{Q\}\ \mathtt{C_2}\ \{R\}}{\{P\}\ \mathtt{C_1;C_2}\ \{R\}}$$

$$\textsc{Cond}\ \frac{\{P\wedge B\}\ I_1\ \{Q\} \quad \{P\wedge\neg B\}\ I_2\ \{Q\}}{\{P\}\ \mathbf{if}\ B\ \mathbf{then}\ I_1\ \mathbf{else}\ I_2\ \{Q\}} \qquad \textsc{While}\ \frac{\{P\wedge B\}\ C\ \{P\}}{\{P\}\ \mathbf{while}\ B\ \mathbf{do}\ C\ \mathbf{done}\ \{P\wedge\neg B\}}$$

correction totale : $$\textsc{WhileT}\ \frac{\langle P \wedge B \wedge E = n \wedge E \geq 0\rangle\ C\ \langle\ P \wedge E < n \wedge E \geq 0\ \rangle}{\langle P\rangle\ \mathbf{while}\ B\ \mathbf{do}\ C\ \mathbf{done}\ \langle P\wedge\neg B\rangle}$$

C Sémantique opérationnelle à grands pas

Expressions

$$\frac{\sigma(\mathtt{x}) = v}{\langle \sigma, \mathtt{x} \rangle \to v} \qquad \frac{\langle \sigma, e \rangle \to v}{\langle \sigma, \mathtt{not}\ \mathtt{e} \rangle \to \neg v}$$

$$\frac{\langle \sigma, e_1 \rangle \to v_1 \qquad \langle \sigma, e_2 \rangle \to v_2}{\langle \sigma, e_1 \mathtt{+} e_2 \rangle \to v_1 + v_2} \qquad \frac{\langle \sigma, e_1 \rangle \to v_1 \qquad \langle \sigma, e_2 \rangle \to v_2}{\langle \sigma, e_1 \mathtt{-} e_2 \rangle \to v_1 - v_2}$$

$$\ldots$$

Instructions

$$\textsc{Aff}\ \frac{\langle \sigma, e \rangle \to v}{\langle \sigma, \mathtt{x:=e} \rangle \rightsquigarrow \sigma[x \leftarrow v]} \qquad \frac{\langle \sigma, p_1 \rangle \rightsquigarrow \sigma' \qquad \langle \sigma', p_2 \rangle \rightsquigarrow \sigma''}{\langle \sigma, p_1 \,\mathtt{;}\, p_2 \rangle \rightsquigarrow \sigma''}\ \textsc{Seq}$$

$$\textsc{If}_1\ \frac{\langle \sigma, e \rangle \to \mathtt{true} \qquad \langle \sigma, p_1 \rangle \rightsquigarrow \sigma'}{\langle \sigma, \mathtt{if}\ e\ \mathtt{then}\ p_1\ \mathtt{else}\ p_2 \rangle \rightsquigarrow \sigma'} \qquad \frac{\langle \sigma, e \rangle \to \mathtt{false} \qquad \langle \sigma, p_2 \rangle \rightsquigarrow \sigma'}{\langle \sigma, \mathtt{if}\ e\ \mathtt{then}\ p_1\ \mathtt{else}\ p_2 \rangle \rightsquigarrow \sigma'}\ \textsc{If}_2$$

$$\textsc{W}_1\ \frac{\langle \sigma, e \rangle \to \mathtt{true} \qquad \langle \sigma, p \rangle \rightsquigarrow \sigma' \qquad \langle \sigma', \mathtt{while}\ e\ \mathtt{do}\ p\ \mathtt{done} \rangle \rightsquigarrow \sigma''}{\langle \sigma, \mathtt{while}\ e\ \mathtt{do}\ p\ \mathtt{done} \rangle \rightsquigarrow \sigma''} \qquad \frac{\langle \sigma, e \rangle \to \mathtt{false}}{\langle \sigma, \mathtt{while}\ e\ \mathtt{do}\ p\ \mathtt{done} \rangle \rightsquigarrow \sigma}\ \textsc{W}_2$$