Graphes et outils logiques

Thibaut Balabonski @ Université Paris-Sud

Troisième partie, 23 avril 2020

Table des matières

7 Algorithmes et raisonnements récursifs (26/03)

7.1 Récurrence simple sur les entiers

Certaines fonctions sur les entiers positifs se calculent naturellement à l'aide d'équations récursives. La factorielle $n!$ par exemple est définie par les deux équations suivantes :

$$\begin{cases} 0! &= 1 \\ n! &= n \times (n-1)! \qquad \text{si } n > 0 \end{cases}$$

De telles équations peuvent être traduites en code à l'aide d'une fonction récursive, c'est-à-dire une fonction qui, lors de son calcul, s'appelle elle-même.

```
def factorielle(n):
    if n == 0:
        return 1
    else:
        return n * factorielle(n-1)
```

La plupart du temps, un tel appel récursif sert à résoudre un sous-problème (ici la factorielle de l'entier précédent) utile à la résolution du problème principal. De manière générale cependant, un tel appel récursif peut prendre un argument arbitraire. On peut également avec des fonctions récursives simuler n'importe quelle boucle **while**.

Une telle fonction récursive f utilisant $f(0)$ comme cas de base et calculant chaque $f(n+1)$ en fonction de $f(n)$ est intimement liée au principe de récurrence simple sur les entiers. Rappel : le raisonnement par récurrence permet de justifier un énoncé de la forme $\forall n \in \mathbb{N}$, $\varphi(n)$ en vérifiant d'une part que la propriété φ est vraie pour 0 et d'autre part que pour tout entier n, la validité de la propriété $\varphi(n+1)$ peut être justifiée en partant de $\varphi(n)$.

Preuve de correction d'une fonction récursive Pour vérifier que la fonction Python `factorielle` calcule bien la factorielle mathématique définie par $n! = 1 \times 2 \times \cdots \times n$, on peut démontrer par récurrence que pour tout entier $n \in \mathbb{N}$,

$$\texttt{factorielle}(n) = n!$$

Démonstration par récurrence :

- Cas de base : `factorielle(0)` vaut 1, et 0! = 1 (1 est l'élément neutre pour la multiplication), donc `factorielle(0)`$= 0!$
- Hérédité : soit n un entier naturel tel que `factorielle`$(n) = n!$ L'entier $n+1$ étant différent de 0, on a

$$\begin{aligned} \texttt{factorielle}(n+1) &= (n+1) \times \texttt{factorielle}(n) \\ &= (n+1) \times n! \\ &= (n+1)! \end{aligned}$$

7.2 Récurrence forte

Le principe de récurrence forte sur les entiers donne plus de souplesse que la récurrence simple.

Rappelons que pour démontrer $\forall n \in \mathbb{N}$, $\varphi(n)$ à l'aide d'une récurrence simple l'étape d'hérédité demande, quelquesoit n, de justifier $\varphi(n+1)$ en utilisant comme seule hypothèse $\varphi(n)$.

Avec le principe de récurrence forte, on peut lors de la justification de $\varphi(n+1)$ utiliser comme hypothèse toute formule $\varphi(k)$ telle que $k \leq n$. Cela inclut $\varphi(n)$ comme pour la récurrence simple, mais également $\varphi(n-1)$, $\varphi(n-2)$, etc. jusqu'à $\varphi(0)$.

Ainsi, pour justifier que pour tout $n \in \mathbb{N}$, la propriété $\varphi(n)$ est vraie, il suffit de démontrer que pour tout $n \in \mathbb{N}$:

$$(\forall k < n,\ \varphi(k)) \Longrightarrow \varphi(n)$$

Remarque : le cas de base n'apparaît pas explicitement dans cette formulation, mais il faut bien toujours à un moment ou un autre démontrer que $\varphi(0)$ est vraie. En effet, si l'on considère la formule précédente en 0 on obtient : $(\forall k < 0,\ \varphi(k)) \Longrightarrow \varphi(0)$. Autrement il s'agit de démontrer $\varphi(0)$ en s'aidant de l'hypothèse que φ est vraie pour tout les entiers $k \in \mathbb{N}$ strictement négatifs, autrement dit en ne s'aidant de rien.

On peut rapprocher cette récurrence généralisée d'une fonction récursive pour laquelle la décroissance du paramètre à chaque appel récursif est différente de 1. Ainsi une fonction `reste_3` calculant le reste de la division euclidienne par 3 peut être définie par les équations suivantes :

$$\begin{cases} \texttt{reste\_3}(n) & = & n & \text{si } n < 3 \\ \texttt{reste\_3}(n) & = & \texttt{reste\_3}(n-3) & \text{sinon} \end{cases}$$

On pourrait écrire une fonction correspondant à ces équations de la manière suivante :

```
def reste_3(n):
    if n < 3:
        return n
    else:
        return reste_3(n-3)
```

7.3 Récurrences multiples

Une fonction, plusieurs appels récursifs Dans une preuve par récurrence (simple ou forte), rien n'empêche d'utiliser plusieurs fois la ou les hypothèses de récurrence. De même une fonction peut effectuer plusieurs appels récursifs.

Exemple avec F_n, la suite de Fibonacci :

$$\begin{cases} F_0 & = & 0 \\ F_1 & = & 1 \\ F_{n+2} & = & F_n + F_{n+1} \end{cases}$$

qu'on peut programmer naïvement ainsi :

```
def fibo(n):
    if n < 2:
        return n
    else:
        return fibo(n-1) + fibo(n-2)
```

À noter qu'une telle fonction effectuant plusieurs appels récursifs court le risque d'effectuer un nombre d'appels exponentiel en n, comme c'est le cas ici. On préférera en l'occurrence utiliser des techniques de programmation dynamique ou de mémoïsation pour ne pas faire plusieurs fois le même travail.

Fonctions mutuellement récursives Les définitions de fonctions récursives peuvent faire intervenir plusieurs fonctions s'appelant mutuellement. Exemple avec deux fonctions `pair` et `impair` calculant la parité d'un nombre entier positif, dont chacune fait appel à l'autre.

```
def pair(n):
    if n==0:
        return True
    else:
        return impair(n-1)

def impair(n):
    if n==0
        return False
    else:
        return pair(n-1)
```

En termes de démonstration par récurrence associée on aurait ici une récurrence simple, puisque chaque appel récursif correspond à une décroissance de 1, mais avec une hypothèse de récurrence comportant plusieurs parties (une par fonction).

Ainsi démontrer la correction de ces fonctions reviendrait à démontrer par récurrence la validité de la formule $\varphi(n)$ suivante :

$$(\texttt{pair}(n) = \texttt{True} \Leftrightarrow n\%2 = 0) \quad \wedge \quad (\texttt{impair}(n) = \texttt{True} \Leftrightarrow n\%2 = 1)$$

L'initialisation revient à vérifier que $\texttt{pair}(n) = \texttt{True}$ et $\texttt{impair}(n) = \texttt{False}$. L'hérédité consiste elle, quelquesoit n, à justifier $\texttt{pair}(n+1) = \texttt{True} \Leftrightarrow (n+1)\%2 = 0$ et $\texttt{impair}(n+1) = \texttt{True} \Leftrightarrow (n+1)\%2 = 1$ à l'aide des deux hypothèses de récurrence $\texttt{pair}(n) = \texttt{True} \Leftrightarrow n\%2 = 0$ et $\texttt{impair}(n) = \texttt{True} \Leftrightarrow n\%2 = 1$.

7.4 Fonctions récursives sur des structures de données

Comme on l'a vu dans les chapitres précédents, c'est tout à fait possible ! On reviendra là-dessus dans les prochains chapitres.

8 Constructions par clôture et récurrence (02/04)

8.1 Retour sur les définitions d'ensembles

On a évoqué au début du cours deux manières de définir un ensemble :

— en fournissant tous les éléments (définition en extension : $E = \{1,2,3,4\}$), ou

— en fournissant une propriété logique caractérisant les éléments de cet ensemble (définition en compréhension : $E = \{k \in \mathbb{N} \mid 1 \leq k < 5\}$).

Dans la définition en compréhension, la formule utilisée pour caractériser les éléments était considérée comme une formule arbitraire. On peut par exemple se donner un ensemble A déjà existant, et définir un ensemble B par l'équation $B = \{k \in \mathbb{N} \mid k = 0 \vee k-2 \in A\}$. Un tel ensemble B contient 0 et tous les entiers valant 2 de plus qu'un élément de A.

Regardons maintenant cette légère variante de l'équation précédente :

$$E = \{k \in \mathbb{N} \mid k = 0 \vee k-2 \in E\}$$

Cette équation définit-elle un ensemble E ? Pour essayer de savoir ce que serait cet ensemble, on peut regarder quelques-uns des éléments qui lui appartiendraient :

— $0 \in E$, car la formule est explicite sur ce point ;
— $2 \in E$, car $2-2=0$ et $0 \in E$;
— $4 \in E$, car $4-2=2$ et $2 \in E$;
— $6 \in E$, car $6-2=4$ et $4 \in E$...

La « définition » impliquerait donc que tous les entiers pouvant être obtenus en partant de 0 et en ajoutant 2 un certain nombre de fois appartiennent à E. On pourrait en effet généraliser et montrer par récurrence que $\forall k \in \mathbb{N}$, $2k \in E$, autrement dit que l'ensemble P des entiers pairs positifs ou nuls est nécessairement inclus dans tout ensemble E vérifiant cette équation.

Pour aller un peu plus loin, on peut même démontrer que l'ensemble P des entiers pairs positifs ou nuls satisfait l'équation $P = \{k \in \mathbb{N} \mid k = 0 \vee k-2 \in P\}$ (preuve standard d'égalité entre deux ensembles : par double inclusion), et qu'il est l'unique ensemble satisfaisant cette équation (preuve par l'absurde en considérant le plus petit nombre impair appartenant à un ensemble E satisfaisant l'équation).

Une équation telle que $E = \{k \in \mathbb{N} \mid k = 0 \vee k-2 \in E\}$ peut donc dans certains cas définir un ensemble. Que dire dans ce cas des ensembles « définis » par les équations suivantes ?

$$E = \{k \in \mathbb{N} \mid k = 0 \vee k+2 \in E\}$$

$$E = \{k \in \mathbb{N} \mid k+2 \in E\}$$

$$E = \{k \in \mathbb{N} \mid k \notin E\}$$

8.2 Définitions par clôture

Systématisons cette méthode de définition d'un ensemble. Pour définir un ensemble E on se donne un ensemble de formules logiques $\varphi_1(E)$, ..., $\varphi_n(E)$, chacune de la forme

$$\forall x_1 \cdots x_l, P_1 \wedge P_2 \wedge \cdots \wedge P_k \Longrightarrow t \in E$$

appelées ***formules d'inférence***. On définit alors l'ensemble E comme le plus petit des ensembles vérifiant toutes les formules $\varphi_i(E)$.

Parmi ces formules d'inférence, on aura en particulier :
— des ***cas de base***, aussi appelés ***axiomes***, utilisés pour indiquer que certains éléments concrets appartiennent à E ; ces formules sont simplement de la forme $t \in E$, par exemple $0 \in E$ ou $5 \in E$, et
— des ***cas inductifs***, aussi appelés ***cas d'itération***, en particulier utilisés pour indiquer qui si certains éléments appartiennent à E alors d'autres éléments doivent aussi nécessairement appartenir à E ; ces formules sont de la forme $\forall x_1 \cdots x_n$, $x_1 \in E \wedge \cdots \wedge x_n \in E \Longrightarrow t \in E$, par exemple $\forall k$, $k \in E \Longrightarrow k+1 \in E$ ou $\forall k$, k', $k \in E \wedge k' \in E \Longrightarrow k+k' \in E$.

Par exemple l'ensemble P des nombres pairs positifs, dont on a vu qu'il était le plus petit des ensembles E vérifiant l'équation $E = \{k \in \mathbb{N} \mid k = 0 \vee k-2 \in E\}$, peut être défini par les règles d'inférence suivantes :
— le cas de base $0 \in E$, et
— le cas inductif $\forall k$, $k \in E \Longrightarrow k+2 \in E$.

Exemples
— Ensembles de nombres.
  — Les fomules d'inférence $0 \in E$ et $\forall n$, $n \in E \Longrightarrow n+1 \in E$ définissent l'ensemble $\mathbb{N}$ des entiers naturels.

- Les formules d'inférence $0 \in E$ et $\forall n,\ n \in E \Longrightarrow n+2 \in E$ définissent l'ensemble des entiers naturels pairs.
- Ensembles de mots. On note ϵ le mot vide et $m_1 \cdot m_2$ la concaténation de deux mots m_1 et m_2.
  - Pour tout ensemble Σ, les formules d'inférence $\epsilon \in E$ et $\forall a, m,\ a \in \Sigma \wedge m \in E \Longrightarrow a \cdot m \in E$ définissent l'ensemble Σ^* des mots sur l'alphabet Σ.
  - Les formules d'inférence $\epsilon \in D$, $\forall m,\ m \in D \Longrightarrow (\cdot m \cdot) \in D$ et $\forall m_1, m_2,\ m_1 \in D \wedge m_2 \in D \Longrightarrow m_1 \cdot m_2 \in D$ définissent l'ensemble des suites de parenthèses bien associées.
- Ensembles de graphes.
  - Les formules $(S, \emptyset) \in E$ et $\forall S, A, s, A',\ (S,A) \in E \wedge s \notin S \wedge (\forall a \in A',\ \exists s' \in S,\ \epsilon(a) = \{s, s'\}) \Longrightarrow (S \cup \{s\}, A \cup A') \in E$ définissent l'ensemble des graphes simples non orientés.
  - Les formules $(S, \emptyset) \in E$ et $\forall S, A, s, A',\ (S,A) \in E \wedge s \notin S \wedge (\forall a \in A',\ \sigma(a) = s \wedge \tau(a) \in S) \Longrightarrow (S \cup \{s\}, A \cup A') \in E$ définissent l'ensemble des graphes orientés acycliques.
  - Les formules $(\{s\}, \emptyset) \in E$ et $\forall S, A, s, a,\ (S,A) \in E \wedge (\exists s' \in S,\ \epsilon(a) = \{s, s'\}) \Longrightarrow (S \cup \{s\}, A \cup \{a\}) \in E$ définit l'ensemble des arbres.

Dérivations.

Étant donné un ensemble E_φ défini par clôture et un élément t, les principaux outils que nous ayons à disposition pour justifier l'appartenance $t \in E_\varphi$ sont les formules d'inférence. On appelle ***dérivation*** de $t \in E_\varphi$ une séquence d'utilisations de formules d'inférence dans laquelle les prémisses de chaque formule utilisée sont déjà justifiées par les conclusions des formules précédentes.

Par exemple, pour justifier que $6 \in P$, pour l'ensemble P des nombres pairs vu en introduction, on peut effectuer la dérivation suivante :

1. Par cas de base, $0 \in P$.
2. Par cas inductif et point 1, $2 \in P$.
3. Par cas inductif et point 2, $4 \in P$.
4. Par cas inductif et point 3, $6 \in P$.

Justifier une non-appartenance comme $t \notin E_\varphi$ est plus compliqué.

- On peut imaginer raisonner par l'absurde si on sait déjà qu'un certain élément t_0 n'appartient par à E_φ et qu'on sait démontrer $t \in E_\varphi \Longrightarrow t_0 \in E_\varphi$.
- Sinon il faut pouvoir raisonner sur toutes les dérivations possibles, et montrer que $t \in E_\varphi$ n'est la conclusion d'aucune. Ceci suppose au passage que E_φ ne contient que des éléments dont l'appartenance à E_φ peut être justifiée par une dérivation, ce qui est une propriété plausible mais que nous n'avons pas démontrée.

8.3 Notation alternative pour les formules d'inférence

Les formules d'inférence ont une certaine forme en commun : elles ont pour partie principale une implication d'un certain nombre de ***prémisses*** vers une ***conclusion***, toutes les variables libres de cette partie principales étant quantifiées universellement.

$$\forall x_1 \cdots x_l,\ P_1 \wedge P_2 \wedge \cdots \wedge P_k \Longrightarrow t \in E$$

La notation sous forme de ***règle d'inférence*** omet les quantificateurs universels, et sépare l'ensemble des prémisses de la conclusion par une

barre horizontale.

$$\frac{P_1 \qquad P_2 \qquad \cdots \qquad P_n}{t \in E}$$

Les conjonctions entre les prémisses sont également omises.

Les deux formules d'inférence $0 \in E$ et $\forall k,\ k \in E \Longrightarrow k+2 \in E$ définissant l'ensemble des entiers pairs correspondent aux deux règles d'inférence

$$\frac{}{0 \in E} \qquad\qquad \frac{k \in E}{k+2 \in E}$$

La notation sous forme de règles d'inférence permet également de présenter les dérivation sous la forme d'***abres de dérivation***, des arbres dont les sommets sont les utilisations de règles et les arêtes indiquent l'utilisation de la conclusion d'une règle pour justifier une prémisse d'une autre règle.

La dérivation de la propriété $6 \in P$ peut ainsi être représentée par l'arbre de dérivation

$$\dfrac{\dfrac{\dfrac{\dfrac{}{0 \in P}}{2 \in P}}{4 \in P}}{6 \in P}$$

8.4 Principe de récurrence

On considère un ensemble de formules d'inférence $\varphi_i(E)$. Par définition, l'ensemble E_φ défini par ces formules est le plus petit des ensembles E vérifiant toutes les formules $\varphi_i(E)$. Autrement dit, E_φ est inclus dans tout ensemble E vérifiant l'ensemble des formules $\varphi_i(E)$.

$$\varphi_1(E) \wedge \cdots \wedge \varphi_n(E) \Longrightarrow E_\varphi \subseteq E$$

Par définition de l'inclusion :

$$\varphi_1(E) \wedge \cdots \wedge \varphi_n(E) \Longrightarrow \forall x \in E_\varphi,\ x \in E$$

Cette formule est le ***principe de récurrence*** (aussi appelé ***principe d'induction***) relatif à E_φ, permettant de prouver des propriétés vraies pour tous les éléments de E_φ.

Dans cette définition du principe de récurrence, la propriété que l'on prouve est l'appartenance à un certain ensemble E. Grâce à ceci, pour prouver que tous les éléments $e \in E_\varphi$ vérifient une certaine propriété $P(e)$, il suffit de considérer l'ensemble $E_P = \{e \mid P(e)\}$ et de démontrer que E_P vérifie toutes les formules d'inférences définissant E_φ.

Exemple. L'ensemble $\mathbb{N}$ est défini par les formules d'inférence $0 \in \mathbb{N}$ et $\forall n,\ n \in \mathbb{N} \Longrightarrow n+1 \in \mathbb{N}$.

Notons E l'ensemble des entiers n tels que $\sum_{0 \leq k \leq n} k = n(n+1)/2$. On remarque que E vérifie aussi les deux formules d'inférences définissant $\mathbb{N}$:

- $0 \in E$
- Si $n \in E$, alors $\sum_{0 \leq k \leq n+1} k = n(n+1)/2 + n + 1 = (n(n+1) + 2(n+1))/2 = (n+1)(n+2)/2$, et ainsi $n+1 \in E$.

On en déduit $\mathbb{N} \subseteq E$, c'est-à-dire $\forall n \in \mathbb{N},\ n \in E$. Ou en développant

$$\forall n \in \mathbb{N},\ \sum_{0 \leq k \leq n} k = n(n+1)/2$$

Ce que l'on vient de faire est une preuve par récurrence du fait que la somme des entiers de 0 à n vaut $n(n+1)/2$.

8.5 Principes dérivés

On considère un ensemble E_φ défini par un ensemble de règles d'inférence. Notons

$$\frac{P_1(E) \quad \cdots \quad P_n(E)}{t \in E}$$

l'une de ces règles, et $\varphi(E)$ la formule associée.

Du principe de récurrence associé à la définition de E_φ on peut déduire deux autres outils de raisonnement.

Renforcement du principe de récurrence.

Lorsque l'on démontre par récurrence que $\forall x \in E_\varphi,\ x \in E_P$ pour un certain ensemble cible E_P, il faut vérifier entre autres la formule $\varphi(E_P)$.

Cette formule a la forme

$$\forall x_1 \cdots x_n,\ P_1(E_P) \wedge \cdots \wedge P_n(E_P) \Longrightarrow t \in E_P$$

Elle demande de démontrer $t \in E_P$ sous les hypothèses $P_i(E_P)$.

Le principe de récurrence renforcé permet de s'autoriser l'utilisation d'hypothèses additionnelles, à savoir les $P_i(E_\varphi)$. Cela revient à démontrer la formule

$$\forall x_1 \cdots x_n,\ P_1(E_\varphi) \wedge \cdots \wedge P_n(E_\varphi) \wedge P_1(E_P) \wedge \cdots \wedge P_n(E_P) \Longrightarrow t \in E_P$$

Justification. Les hypothèses additionnelles du principe de récurrence renforcé apparaissent lorsque l'on essaye de démontrer $\forall x \in E_\varphi,\ x \in E_P \cap E_\varphi$ avec le principe de récurrence ordinaire. Or les deux propriétés $\forall x \in E_\varphi,\ x \in E_P \cap E_\varphi$ et $\forall x \in E_\varphi,\ x \in E_P$ sont équivalentes.

Principe d'inversion.

L'ensemble E_φ défini par notre ensemble de règles d'inférence est le plus petit des ensembles validant chacune des règles. Autrement dit il ne contient que ce qui est directement impliqué par les règles d'inférence, et on peut à s'attendre à ce que chaque élément $t \in E_\varphi$ aie son appartenance à E_φ justifiée par une dérivation.

Supposons que la dernière règle d'inférence utilisée dans la dérivation justifiant $t \in E_\varphi$ soit la règle

$$\frac{P_1(E_\varphi) \quad \cdots \quad P_n(E_\varphi)}{t \in E_\varphi}$$

Cette règle ayant été utilisée, il est certain que les prémisses $P_i(E_\varphi)$ étaient vérifiées.

Bilan : principe d'inversion. Si on sait que la formule $t \in E_\varphi$ est vraie, alors on peut en déduire qu'il existe une règle d'inférence ayant pour conclusion $t \in E_\varphi$ et dont toutes les prémisses sont vraies.

Remarque : raisonnement par cas nécessaire. Le principe d'inversion assure la validité des prémisses de l'une des règles d'inférence, mais nous ne savons pas de laquelle. Quand on utilise ce principe il faut donc ensuite raisonner par cas sur l'ensemble des règles d'inférence, éventuellement en éliminant celles qui sont en contradiction avec $t \in E_\varphi$.

Justification. Nous avons introduit le principe d'inversion comme un principe de bon sens. Cela permet d'expliquer pourquoi on imagine que ce principe devrait être valide, mais cela ne constitue pas une preuve. La vraie justification du principe d'inversion utilise le schéma de récurrence renforcé.

8.6 Cas particulier : définition de relations par clôture

Une relation n-aire est un ensemble de n-uplets. En définissant par clôture un ensemble de n-uplets on définit donc une relation. De plus, le principe de récurrence associé à cette définition par clôture va pouvoir être utilisé pour démontrer des propriétés vraies de tous les n-uplets vérifiant la relation.

Exemple de relation binaire : accessibilité dans un graphe. Soit un graphe $G = (S, A)$. Les conditions

- pour tout sommet $s \in S$, $R(s,s)$, et
- pour tous sommets s_1, s_2, s_3, si $R(s_1, s_2)$ et s'il existe une arête de s_2 vers s_3 alors $R(s_1, s_3)$,

définissent une relation R telle que $R(s,s')$ si et seulement s'il existe un chemin de s vers s'.

Note : cette même relation peut également être définie par les conditions alternatives

- pour tout sommet $s \in S$, $R(s,s)$,
- pour toute arête a d'extrémités s et s', $R(s,s')$, et
- pour tous sommets s_1, s_2, s_3, si $R(s_1, s_2)$ et $R(s_2, s_3)$ alors $R(s_1, s_3)$.

8.7 Cas particulier : définition de fonctions par clôture

Une fonction est une relation binaire ayant la propriété d'être fonctionnelle (pour chaque élément de l'ensemble d'entrée il existe au plus une image dans l'ensemble de sortie). On peut donc définir une fonction par clôture en définissant par clôture la relation binaire sous-jacente. Le principe de récurrence associé permet de démontrer des propriétés sur les valeurs prises par la fonction.

Exemples

- La factorielle $n!$ est définie par : $0! = 1$ et $(n+1)! = (n+1)(n!)$. La relation binaire sous-jacente F telle que $F(a,b)$ si et seulement si $a! = b$ est donc définie par les conditions :
  - $F(0,1)$, et
  - pour tous entiers n, m, si $F(n,m)$ alors $F(n+1,(n+1)m)$.
- La suite de Fibonacci f_n est définie par : $f_0 = 0$, $f_1 = 1$ et $f_{n+2} = f_{n+1} + f_n$. La relation binaire sous-jacente F telle que $F(a,b)$ si et seulement si $f_a = b$ est donc définie par les conditions :
  - $F(0,0)$,
  - $F(1,1)$, et
  - pour tous entiers n, m_1, m_2, si $F(n,m_1)$ et $F(n+1,m_2)$ alors $F(n+2, m_1+m_2)$.

9 Termes structurés (09/04)

9.1 Alphabets et mots

La notion de mot en informatique désigne une suite finie de caractères. Il s'agit d'un élément central de la théorie des langages que vous étudiez dans d'autres cours. Avec une vision plus algorithmique, les mots peuvent aussi décrire des structures de données séquentielles comme des tableaux ou des listes.

Définitions Un ***alphabet*** est un ensemble Σ dont les éléments sont appelés des ***caractères***. Un ***mot*** sur l'alphabet Σ est une suite finie de caractères. On peut représenter un mot m par la suite $a_0 a_1 \cdots a_{n-1}$ de ses caractères. Plus formellement on peut définir un mot par sa ***longueur*** n (le nombre de caractères dans la suite) et par une fonction $[0, n[\longrightarrow \Sigma$ associant chaque indice de 0 inclus à n exclu au caractère présent à cet indice dans la séquence. On note $|m|$ la taille du mot m, et $m[i]$ le caractère d'indice i de m.

On note Σ^* l'ensemble des mots sur Σ et Σ^n l'ensemble des mots de longueur n.

Mots particuliers et opérations Le ***mot vide***, généralement noté ϵ, est l'unique suite de longueur 0.

Le ***mot singleton*** d'un certain caractère $a \in \Sigma$, simplement noté a, est la suite de longueur 1 formée du seul caractère a.

La ***concaténation*** de deux mots m_1 et m_2, notée $m_1 \cdot m_2$ ou simplement $m_1 m_2$, est la suite formée en faisant se suivre les suites m_1 et m_2. L'opération de concaténation est associative et admet ϵ comme élément neutre.

La ***tête*** d'un mot m, notée $\mathsf{hd}(m)$, est le premier de ses caractères (ce n'est défini que si $m \neq \epsilon$).

La ***queue*** d'un mot m, notée $\mathsf{tl}(m)$, est la suite formée en retirant le premier caractère de m (ce n'est défini que si $m \neq \epsilon$).

9.2 Récurrences sur la structure des mots

L'ensemble Σ^* des mots sur Σ vérifie un certain nombre de règles d'inférence, dont

$$\text{(A)}\ \frac{}{\epsilon \in \Sigma^*} \qquad \text{(B)}\ \frac{a \in \Sigma}{a \in \Sigma^*} \qquad \text{(C)}\ \frac{a \in \Sigma \qquad m \in \Sigma^*}{am \in \Sigma^*} \qquad \text{(D)}\ \frac{a \in \Sigma \qquad m \in \Sigma^*}{ma \in \Sigma^*}$$

$$\text{(E)}\ \frac{m_1 \in \Sigma^* \qquad m_2 \in \Sigma^*}{m_1 m_2 \in \Sigma^*}$$

Principes d'induction Plusieurs sous-ensembles de ces règles suffisent à définir l'ensemble des mots, par exemple $\{(A),(C)\}$ ou $\{(A),(B),(E)\}$, et chacun de ces systèmes donne un principe d'induction.

— Pour le système $\{(A),(C)\}$ le principe d'induction est :

> Pour tout prédicat P sur les mots, si
>
> (A) $P(\epsilon)$
>
> (C) pour tous $a \in \Sigma$ et $m \in \Sigma^*$, si $P(m)$ alors $P(am)$
>
> alors pour tout $m \in \Sigma^*$ on a $P(m)$.

— Pour le système $\{(A),(B),(E)\}$ le principe d'induction est :

> Pour tout prédicat P sur les mots, si
>
> (A) $P(\epsilon)$
>
> (B) pour tout $a \in \Sigma$ on a $P(a)$
>
> (E) pour tous $m_1 \in \Sigma^*$ et $m_2 \in \Sigma^*$, si $P(m_1)$ et $P(m_2)$ alors $P(m_1 m_2)$
>
> alors pour tout $m \in \Sigma^*$ on a $P(m)$.

Définition de fonctions On peut définir des fonctions par récurrence sur les mots en suivant la structure de ces principes d'induction. Par exemple, pour compter le nombre d'occurrences du caractère a dans un mot m :

— Version $\{(A),(C)\}$:

$$\begin{aligned} c(a,\epsilon) &= 0 \\ c(a,bm) &= \begin{cases} 1+c(m) & \text{si } a=b \\ c(m) & \text{sinon} \end{cases} \end{aligned}$$

— Version $\{(A),(B),(E)\}$:

$$\begin{aligned} c(a,\epsilon) &= 0 \\ c(a,b) &= \begin{cases} 1 & \text{si } a=b \\ 0 & \text{sinon} \end{cases} \\ c(a,m_1m_2) &= c(m_1)+c(m_2) \end{aligned}$$

Pour retourner un mot m :

— Version $\{(A),(C)\}$:

$$\begin{aligned} r(\epsilon) &= \epsilon \\ r(am) &= r(m)a \end{aligned}$$

— Version $\{(A),(B),(E)\}$:

$$\begin{aligned} r(\epsilon) &= \epsilon \\ r(a) &= a \\ r(m_1m_2) &= r(m_2)r(m_1) \end{aligned}$$

Ces techniques toutefois marchent mal si les mots considérés ont une structure particulière. Considérons le mot $1+2\times3$ sur l'alphabet $\Sigma = \mathbb{N} \cup \{+,\times,(,)\}$ représentant une expression arithmétique : nos systèmes ne permettent pas d'écrire une fonction calculant le résultat de cette expression.

9.3 Termes

Les mots possédant une structure particulière peuvent être définis par leurs propres règles d'inférence. Tentative pour l'ensemble $\mathbb{A}$ des expressions arithmétiques :

$$\frac{n \in \mathbb{N}}{n \in \mathbb{A}} \qquad \frac{a_1 \in \mathbb{A} \qquad a_2 \in \mathbb{A}}{a_1 + a_2 \in \mathbb{A}} \qquad \frac{a_1 \in \mathbb{A} \qquad a_2 \in \mathbb{A}}{a_1 \times a_2 \in \mathbb{A}}$$

On a cependant dans ce cas plusieurs dérivations possibles pour le même mot $1+2\times3$:

$$\dfrac{\dfrac{}{1 \in \mathbb{A}} \qquad \dfrac{\dfrac{}{2 \in \mathbb{A}} \qquad \dfrac{}{3 \in \mathbb{A}}}{2\times3 \in \mathbb{A}}}{1+2\times3 \in \mathbb{A}} \qquad\qquad \dfrac{\dfrac{\dfrac{}{1 \in \mathbb{A}} \qquad \dfrac{}{2 \in \mathbb{A}}}{1+2 \in \mathbb{A}} \qquad \dfrac{}{3 \in \mathbb{A}}}{1+2\times3 \in \mathbb{A}}$$

Et ces dérivations ne correspondent pas à la même interprétation : la première suit la structure correcte $1+(2\times 3)$ alors que la deuxième suit l'interprétation incorrecte $(1+2)\times 3$.

La notion de terme va donner une approche systématique pour représenter de tels mots structurés sans ambiguïté, et permettre en particulier la définition par récurrence de propriétés ou de fonctions sur ces mots.

Signatures et termes Une ***signature*** est un ensemble Σ dont chaque élément est associé à un nombre entier naturel appelé ***arité***. Les ***termes*** sur la signature Σ sont des mots sur l'alphabet $\Sigma \cup \{(\ ),\}$, dont l'ensemble $T(\Sigma)$ est défini par des règles d'inférence, chaque symbole de Σ donnant une règle. Si c est un symbole de Σ d'arité zéro, la règle associée est un cas de base :

$$\frac{}{c \in T(\Sigma)}$$

Si f est un symbole de Σ d'arité n non nulle, la règle est :

$$\frac{t_1 \in T(\Sigma) \qquad \cdots \qquad t_n \in T(\Sigma)}{f(t_1,\ldots,t_n) \in T(\Sigma)}$$

Exemple : expressions arithmétiques Les expressions arithmétiques précédentes peuvent donc être redéfinies comme des termes sur une signature contenant les nombres entiers (symboles dont l'arité est 0) et les symboles + et × d'arité 2. Les règles d'inférences deviendraient :

$$\frac{n \in \mathbb{N}}{n \in \mathbb{A}} \qquad \frac{a_1 \in \mathbb{A} \qquad a_2 \in \mathbb{A}}{+(a_1,a_2) \in \mathbb{A}} \qquad \frac{a_1 \in \mathbb{A} \qquad a_2 \in \mathbb{A}}{\times(a_1,a_2) \in \mathbb{A}}$$

Toute l'ambiguïté habituellement associée aux opérateurs arithmétiques disparaît, l'expression $1+2\times 3$ précédente étant explicitée soit en le terme $+(1,\times(2,3))$ soit en le terme $\times(+(1,2),3)$.

Termes avec sortes La structure de termes peut également être utilisée pour représenter des structures de données. La structure de liste chaînée par exemple est constituée de deux éléments principaux :

- les cellules, comportant un élément et un pointeur vers la cellule suivante
- la liste vide, dénotant une absence de cellule

On pourrait représenter une telle liste comme un terme à l'aide de deux symboles : un symbole Nil d'arité nulle pour la liste vide, et un symbole Cell d'arité 2 pour les cellules. La liste comportant les éléments 1, 2, et 4 dans cet ordre pourrait alors s'écrire $\mathsf{Cell}(1,\mathsf{Cell}(2,\mathsf{Cell}(3,\mathsf{Nil})))$.

Petite subtilité par rapport à la situation d'origine : dans un terme $\mathsf{Cell}(e,t)$ les arguments e et t n'ont pas la même nature. Ici, e est un élément de la liste, par exemple un entier, tandis que t est une liste (autrement dit un terme). On peut enrichir la notion d'arité en ne donnant pas seulement le nombre des arguments attendus mais aussi la nature, appelée ***sorte***, de chacun de ces arguments.

Pour l'exemple des listes chaînées :

- Le symbole Nil constitue directement une liste. En notant liste la sorte « liste chaînée » on pourra résumer son arité avec la notation Nil : liste.

— Le symbole Cell combine deux éléments pour former une liste, le premier élément étant un entier et le deuxième une autre liste. En notant $\mathbb{N}$ la sorte « entier », on pourra résumer l'arité de ce symbole avec la notation $\mathsf{Cell} : \mathbb{N} \times \mathsf{liste} \longrightarrow \mathsf{liste}$.

Exemple : arbres binaires Une structure arbre d'arbre binaire dans laquelle les nœuds internes sont étiquetés par des valeurs entières peut être représenté par des termes sur la signature $\{\mathsf{feuille}, \mathsf{nœud}\}$ avec les arités suivantes :

— $\mathsf{feuille} : \mathsf{arbre}$
Le symbole feuille est un arbre un soi, son arité est nulle.

— $\mathsf{nœud} : \mathsf{arbre} \times \mathbb{N} \times \mathsf{arbre} \longrightarrow \mathsf{arbre}$
Le symbole nœud a une arité de 3, et a besoin de deux sous-arbres et d'un entier pour former un nouvel arbre.

Exemple d'arbre : $\mathsf{nœud}(\mathsf{feuille}, 3, \mathsf{nœud}(\mathsf{nœud}(\mathsf{feuille}, 1, \mathsf{feuille}), 2, \mathsf{feuille}))$.

9.4 Définition de fonctions et de prédicats sur les termes

L'ensemble $T(\Sigma)$ des termes sur une signature Σ étant défini par clôture à partir de règles d'inférence, ces mêmes règles nous donnent également un schéma pour définir des fonctions et des prédicats s'appliquant aux éléments de $T(\Sigma)$.

Pour définir une telle fonction $F : T(\Sigma) \longrightarrow B$, il suffit de prévoir un cas par symbole de la signature Σ :

— pour chaque symbole c d'arité nulle, donner l'élément $b \in B$ tel que $F(c) = b$;

— pour chaque symbole f d'arité n, donner une équation exprimant $F(f(t_1, \ldots, t_n))$ en fonction des éléments t_1 à t_n, l'équation pouvant également utiliser $F(t_i)$ pour les t_i qui sont bien des termes.

Exemple : valeur d'une expression arithmétique Pour calculer la valeur $\mathsf{eval}(a)$ d'une expression arithmétique représentée par un terme a, on peut donner les trois équations suivantes, qui effectuent les opérations représentées par les symboles binaires + et ×.

$$\begin{aligned} \mathsf{eval}(n) &= n \\ \mathsf{eval}(+(a_1, a_2)) &= \mathsf{eval}(a_1) + \mathsf{eval}(a_2) \\ \mathsf{eval}(\times(a_1, a_2)) &= \mathsf{eval}(a_1) \times \mathsf{eval}(a_2) \end{aligned}$$

Exemple : hauteur et taille d'un arbre binaire Pour calculer la hauteur $\mathsf{h}(a)$ ou le nombre de feuilles $\mathsf{f}(a)$ d'un arbre binaire a, on donne les deux équations suivantes.

$$\begin{aligned} \mathsf{h}(\mathsf{feuille}) &= 0 \\ \mathsf{h}(\mathsf{nœud}(a_1, n, a_2)) &= 1 + \max(\mathsf{h}(a_1), \mathsf{h}(a_2)) \end{aligned}$$

$$\begin{aligned} \mathsf{f}(\mathsf{feuille}) &= 1 \\ \mathsf{f}(\mathsf{nœud}(a_1, n, a_2)) &= \mathsf{f}(a_1) + \mathsf{f}(a_2) \end{aligned}$$

Exemple : arbre équilibré Pour définir un prédicat Eq exprimant qu'un arbre a est équilibré, c'est-à-dire qu'en chaque nœud de a la hauteur des sous-arbres gauche et droit diffère au plus de un, on donne les deux règles d'inférences suivantes.

$$\frac{}{\mathsf{Eq}(\mathsf{feuille})} \qquad \frac{\mathsf{Eq}(a_1) \qquad \mathsf{Eq}(a_2) \qquad |\mathsf{h}(a_1) - \mathsf{h}(a_2)| \leq 1}{\mathsf{Eq}(\mathsf{nœud}(a_1, n, a_2))}$$

Exercices

— Définition d'une fonction de comptage des sommets d'un arbre binaire.
— Définition d'un prédicat pour l'appartenance d'un élément à un arbre.
— Définition d'un prédicat caractérisant les arbres binaires de recherche.
— Définition d'une fonction cherchant efficacement un élément dans un arbre binaire de recherche.
— Définition d'un prédicat caractérisant une liste chaînée triée.
— Définition d'une fonction insérant un élément dans une liste chaînée triée.

9.5 Principe de récurrence sur les termes

L'ensemble $T(\Sigma)$ des termes sur une signature Σ étant défini par clôture à partir de règles d'inférence, il bénéficie du principe de récurrence : toute propriété P satisfaisant les mêmes formules d'inférence sera vraie pour tous les termes.

Exemple : expressions arithmétiques Pour tout prédicat P, si

(a) pour tout $n \in \mathbb{N}$ on a $P(n)$,

(b) pour tous a_1, a_2 tels que $P(a_1)$ et $P(a_2)$ on a $P(+(a_1,a_2))$, et

(c) pour tous a_1, a_2 tels que $P(a_1)$ et $P(a_2)$ on a $P(\times(a_1,a_2))$,

alors pour toute expression arithmétique $a \in \mathbb{A}$ on a $P(a)$.

Proposition. Pour toute expression arithmétique $a \in \mathbb{A}$, en notant $c(a)$ le nombre de constantes dans a et $o(a)$ le nombre d'opérateurs dans a, on a $c(a) = o(a) + 1$.

Preuve par récurrence. Notons $P(a) \equiv c(a) = o(a) + 1$.

(a) soit $n \in \mathbb{N}$, on a $c(n) = 1$ et $o(n) = 0$, donc $P(n)$;

(b) soient $a_1, a_2 \in \mathbb{A}$ telles que $P(a_1)$ et $P(a_2)$, on a $c(+(a_1,a_2)) = c(a_1) + c(a_2) = o(a_1) + 1 + o(a_2) + 1 = o(+(a_1,a_2)) + 1$, et donc $P(+(a_1,a_2))$;

(c) soient $a_1, a_2 \in \mathbb{A}$ telles que $P(a_1)$ et $P(a_2)$, on a $c(\times(a_1,a_2)) = c(a_1) + c(a_2) = o(a_1) + 1 + o(a_2) + 1 = o(\times(a_1,a_2)) + 1$, et donc $P(+(a_1,a_2))$.

Donc pour tout $a \in \mathbb{A}$ on a $c(a) = o(a) + 1$.

Exemple : arbres binaires Pour tout prédicat P, si

(a) $P(\text{feuille})$, et

(b) pour tout entier n et tous arbres a_1, a_2 tels que $P(a_1)$ et $P(a_2)$ on a $P(\text{nœud}(a_1, n, a_2))$,

alors pour tout arbre binaire a on a $P(a)$.

Proposition. Pour tout arbre binaire a, en notant $f(a)$ le nombre de feuilles dans a et $s(a)$ le nombre total de sommets de a (nœuds et feuilles), on a $s(a) = 2f(a) - 1$.

Preuve par récurrence. Notons $P(a) \equiv s(a) = 2f(a) - 1$.

(a) $s(\text{feuille}) = 1$ et $f(\text{feuille}) = 1$, donc $P(\text{feuille})$;

(b) soient $n \in \mathbb{N}$ et a_1, a_2 deux arbres tels que $P(a_1)$ et $P(a_2)$, alors $s(\text{nœud}(a_1, n, a_2)) = 1 + s(a_1) + s(a_2) = 1 + 2f(a_1) - 1 + 2f(a_2) - 1 = 2(f(a_1) + f(a_2)) - 1 = 2f(\text{nœud}(a_1, n, a_2)) - 1$.

Donc pour tout arbre binaire a on a $s(a) = 2f(a) - 1$. (remarque : cette propriété est la même que la précédente sur les expressions arithmétiques)

Exercices

- — Correction de la fonction de recherche d'un élément dans un arbre binaire de recherche.
- — Préservation du caractère trié pour une fonction d'insertion d'un élément dans une liste chaînée triée.

10 Relations d'ordre et problèmes de terminaison (23/04)

10.1 Terminaison

Deux raisons pour lesquelles l'exécution d'un programme pourrait ne pas s'arrêter :

- — une boucle while
- — un appel récursif

Pour justifier qu'un programme aboutit toujours, il faut pouvoir argumenter que quelque chose progresse à chaque tour de boucle ou à chaque appel récursif, et que cette progression est de nature telle qu'elle ne peut pas se poursuivre indéfiniment.

Exemples

- — Une boucle while qui s'exécute tant que la variable a est inférieure à une borne n, et dans laquelle la valeur de a augmente de au moins 1 à chaque tour : on progresse car à chaque tour la valeur de a est plus proche de la valeur de n, et qu'il n'est pas possible de faire une infinité d'incréments sans dépasser n.
- — Une fonction récursive avec un paramètre n entier positif, tel que chaque appel récursif s'effectue avec une valeur strictement plus petite pour le paramètre : on progresse car à chaque appel la valeur du paramètre se rapproche un peu plus de 0, et qu'il n'est pas possible de faire une infinité de telles étapes tout en restant positif.

Contre-exemple : paradoxe de Zenon

- — À chaque itération, une valeur réelle est divisée par 2. On peut imaginer que l'on progresse vers 0, car à chaque tour la valeur considérée est strictement plus petite. Cependant, aucun nombre fini d'étapes ne permettra d'atteindre 0.

10.2 Ordres

Relation d'ordre Étant donné un ensemble E, une ***relation d'ordre*** sur E est une relation binaire R sur $E \times E$ (on parle de relation binaire ***homogène***) qui est :

- — réflexive ($\forall x$, xRx)
- — anti-symétrique ($\forall x, y$, $(xRy \wedge yRx) \implies x = y$)
- — transitive ($\forall x, y, z$, $(xRy \wedge yRz) \implies xRz$)

Un ***ordre total*** est un ordre pour lequel tous deux éléments sont comparables (dans un sens ou dans l'autre) : $\forall x, y$, $xRy \vee yRx$.

Exemples

- — Relation d'ordre usuelle $\leq$ sur un ensemble de nombres.
- — Relation d'inclusion $\subseteq$ sur les parties d'un ensemble A.
- — Relation de divisibilité $|$ sur les nombres entiers.

10.3 Maximum/minimum, maximaux/minimaux

On considère un ensemble E et un ordre $\leq$ sur E. Étant donnés un sous-ensemble $A \subseteq E$ et un élément $x \in A$, on dit que :

- x est le ***minimum*** de A si x est plus petit que tous les éléments de A ($\forall y \in A,\ x \leq y$),
- x est le ***maximum*** de A si x est plus grand que tous les éléments de A ($\forall y \in A,\ y \leq x$).

Note : une partie A n'admet pas nécessairement de minimum, mais dans le cas où un tel élément existe il est unique (de même pour le maximum).

Étant donnés un sous-ensemble $A \subseteq E$ et un élément $x \in A$, on dit que :

- x est un élément ***minimal*** de A s'il n'existe pas dans A d'élément plus petit que x ($\forall y \in A,\ y \leq x \implies y = x$),
- x est un élément ***maximal*** de A s'il n'existe pas dans A d'élément plus grand que x ($\forall y \in A,\ x \leq y \implies y = x$),

Note : minimal n'est pas la même chose que *minimum*, et un élément minimal de A n'est pas nécessairement unique (de même pour maximal/-maximum).

Exercices

- Le minimum, s'il existe, est unique.
- Le minimum, s'il existe, est l'unique élément minimal.
- Si l'ordre $\leq$ est total, les conditions « être le minimum de A » et « être un élément minimal de A » deviennent équivalentes.

10.4 Majorants/minorants, bornes

On considère un ensemble E, un ordre $\leq$ sur E et une partie $A \subseteq E$.

- Un élément $x \in E$ est un ***minorant*** de A s'il est plus petit que tous les éléments de A ($\forall y \in A,\ x \leq y$).
- Un élément $x \in E$ est un ***majorant*** de A s'il est plus grand que tous les éléments de A ($\forall y \in A,\ y \leq x$).
- La ***borne inférieure*** de A est, s'il existe, le maximum des minorants de A.
- La ***borne supérieure*** de A est, s'il existe, le minimum des majorants de A.

Note : les majorants, minorants et bornes de A n'existe pas forcément, et dans le cas où ils existent n'appartiennent pas nécessairement à A.

10.5 Ordres bien fondés

La notion d'ordre permet de donner du sens à la notion de « progression » évoquée dans notre problème de justification de l'arrêt d'un algorithme : on considérera avoir progressé dès lors que l'on obtiendra quelque chose de *strictement plus petit* vis-à-vis de l'ordre choisi. En revanche, tous les ordres n'empêchent pas une telle progression de se poursuivre indéfiniment. Cette dernière propriété caractérise les ordres dits « bien fondés ».

Ordre bien fondé : définition Un ordre $\leq$ sur un ensemble E est ***bien fondé*** s'il n'existe pas de suite infinie strictement décroissante pour $\leq$. Autrement dit, en notant $<$ l'ordre strict associé à $\leq$, il ne peut pas exister de suite $(x_k)_{k \in \mathbb{N}}$ telle que $\forall k,\ x_{k+1} < x_k$.

Cette propriété traduit directement la notion d'arrêt cherchée.

Caractérisation alternative Un ordre $\leq$ sur un ensemble E est bien fondé si et seulement toute partie non vide de E admet un élément minimal. Autrement écrit :

$$\forall A \subseteq E,\ A \neq \emptyset \Longrightarrow (\exists a \in A,\ \forall x \in A,\ x \leq a \Longrightarrow x = a)$$

Preuve

— Supposons $(E, \leq)$ bien fondé. Soit A une partie non vide de E.
Raisonnement par l'absurde. Supposons que A n'admette pas d'élément minimum. Autrement dit, $\forall a \in A,\ \exists a' \in A,\ a' < a$. Comme A est non vide, il existe au moins un élément $a_0 \in A$. Comme $\leq$ est bien fondé, il n'existe pas de suite infinie strictement décroissante à partir de a_0. Soit $(a_k)_{k\in[0,N]}$ une suite strictement décroissante d'éléments de A à partir de a_0, qui soit la plus longue possible. Comme $a_N \in A$ et A n'admet pas d'élément minimal, il existe $a_{N+1} \in A$ avec $a_{N+1} < a_N$. Donc $(a_k)_{k\in[0,N+1]}$ est une suite strictement décroissante dans A strictement plus longue que la précédente. Contradiction, donc A doit admettre un élément minimal.

— Supposons que toute partie non vide A de E admette un élément minimal.
Raisonnement par l'absurde. Soit $(x_k)_{k\in\mathbb{N}}$ une suite infinie strictement décroissante dans E. On note A l'ensemble des valeurs de cette suite. Cet ensemble A est non vide (il contient par exemple x_0), et admet donc un élément minimal x_k. Or $x_{k+1} < x_k$ avec $x_{k+1} \in A$, ce qui contredit la minimalité de x_k. Donc il ne peut pas exister de suite infinie strictement décroissante dans E, et l'ordre $\leq$ est bien fondé.

Exemples

— Ordre usuel $\leq$ sur $\mathbb{N}$.
— Divisibilité $|$ sur $\mathbb{Z}$.
— Inclusion $\subseteq$ sur les parties d'un ensemble *fini*.

Contre-exemples

— Ordre usuel $\leq$ sur $\mathbb{Z}$: on peut descendre indéfiniment dans les négatifs.
— Ordre usuel $\leq$ sur $\mathbb{R}^+$: on peut s'approcher indéfiniment de 0 sans jamais l'atteindre.
— Inclusion $\subseteq$ sur les parties d'un ensemble infini : on peut définir une suite infinie d'ensembles qui ont toujours moins d'éléments mais restent infinis, comme la suite $([k, \infty[)_{k\in\mathbb{N}}$.

10.6 Récurrence bien fondée

On considère un ensemble E, avec un ordre bien fondé $\leq$. En notant $<$ l'ordre strict associé à $\leq$ (par définition, $x < y$ si et seulement si $x \leq y \wedge x \neq y$), on a le nouveau principe de récurrence suivant.

Pour tout prédicat P sur les éléments de E, si

1. pour tout $e \in E$, $(\forall x \in E,\ x < e \Longrightarrow P(x))$ implique $P(e)$,

alors pour tout élément $e \in E$ on a $P(e)$.

Autrement dit, si l'on peut déduire $P(e)$ dès lors que l'on suppose la propriété vraie pour tout les élément strictement inférieurs à e, et ceci pour chaque e, alors la propriété P est vraie pour tous les éléments de E. C'est

la même idée que le principe de récurrence forte sur les entiers (chapitre 7 de ces notes de cours).

Justification Supposons que pour tout $e \in E$, $(\forall x \in E,\ x < e \implies P(x))$ implique $P(e)$, notons A l'ensemble des éléments de $e \in E$ tels que $P(e)$ ne soit pas vraie et montrons que cet ensemble est vide en *raisonnant par l'absurde*.

Supposons cet ensemble A non vide. Comme $\leq$ est bien fondé et A non vide, il existe un élément minimal a de A. Soit $x \in E$ tel que $x < a$. Comme a est minimal dans A, nous savons que $x \notin A$, et donc que $P(x)$ est vraie. Ceci étant vrai pour pour tous les $x < a$, on déduit $P(a)$. Contradiction avec l'appartenance de a à A. Donc l'ensemble A doit être vide, et tous les éléments de E vérifient P.

10.7 Combinaison d'ordres

Comment comparer deux paires d'entiers ? Comment jugeons-nous les propositions suivantes ?

— $(1,2) < (3,4)$?
— $(1,3) < (2,4)$?
— $(1,5) < (2,3)$?
— $(2,3) < (1,5)$?

De manière plus générale, étant donnés deux ensembles A et B, chacun avec un ordre (on pourra les noter respectivement $\leq_A$ et $\leq_B$), on cherche à ordonner les paires de $A \times B$.

Ordre produit cartésien L'***ordre produit*** sur $A \times B$ est défini par $(a_1, b_1) \leq (a_2, b_2)$ si et seulement si (a_1, b_1) est plus petit sur les deux composantes à la fois : $a_1 \leq_A a_2 \wedge b_1 \leq_B b_2$.

Par exemple :
— $(1,2) \leq (3,4)$
— $(1,3) \leq (2,4)$
— $(1,5)$ et $(2,3)$ sont incomparables

Note : l'ordre produit n'est donc pas total.

Ordre produit lexicographique Le ***produit lexicographique*** des deux ordres $\leq_A$ et $\leq_B$ consiste à comparer d'abord la première composante, puis à ne tenir compte de la deuxième composante qu'en cas d'égalité sur la première : on a $(a_1, b_1) \leq (a_2, b_2)$ si et seulement si $a_1 <_A a_2 \vee (a_1 = a_2 \wedge b_1 \leq_B b_2)$.

C'est selon ce principe que l'on compare deux mots dans le dictionnaire (l'ordre lexicographique est aussi appelé ***ordre du dictionnaire***).

— $(1,2) \leq (3,4)$
— $(1,3) \leq (2,4)$
— $(1,5) \leq (2,3)$

Note : l'ordre lexicographique est total.

Exercices

— L'ordre lexicographique est total.
— Si $\leq_A$ et $\leq_B$ sont des ordres bien fondés, alors les ordres produit cartésien et produit lexicographique sont tous deux bien fondés également.

11 Treillis, algèbre booléenne, théorème de point fixe (30/04)

11.1 Structure de treillis

Un ***treillis*** est un ensemble ordonné $(E, \leq)$ tel que pour tous deux éléments x, y de E, l'ensemble $\{x, y\}$ admet dans E :

— une borne inférieure, notée $x \sqcap y$,

— une borne supérieure, notée $x \sqcup y$.

Exemples

— Dans le cas d'un ordre total (comme l'ordre usuel sur les entiers) :

  — $a \sqcap b = \min(a, b)$

  — $a \sqcup b = \max(a, b)$

— Dans le cas de l'ordre d'inclusion sur l'ensemble des parties d'un ensemble E :

  — $A \sqcap B = A \cap B$

  — $A \sqcup B = A \cup B$

— Dans le cas de l'ordre de divisibilité sur $\mathbb{N}$:

  — $a \sqcap b = \text{pgcd}(a, b)$

  — $a \sqcup b = \text{ppcm}(a, b)$

Propriétés

— $x \sqcap y = y \sqcap x$ (commutativité)

— $x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z$ (associativité)

— $x \sqcap x = x$ (idempotence)

— commutativité, associativité et idempotence aussi pour $\sqcup$

— $x \sqcap (x \sqcup y) = x$ (loi d'absorption)

— $x \sqcup (x \sqcap y) = x$ (loi d'absorption)

Exercice

— Dans un treillis, tous ensemble fini d'éléments admet une borne inférieure et une borne supérieure.

11.2 Algèbre de Boole

Un treillis $(E, \leq)$ peut également être :

— ***borné*** s'il admet un minimum (noté $\bot$) et un maximum (noté $\top$) ; ces éléments vérifient de plus dans ce cas les propriétés :

  — $x \sqcap \bot = \bot$

  — $x \sqcap \top = x$

  — $x \sqcup \bot = x$

  — $x \sqcup \top = \top$

— ***distributif*** si les deux propriétés suivantes sont vérifiées :

  — $x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$

  — $x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$

— ***complémenté*** s'il est borné et s'il existe une fonction $x \mapsto \overline{x}$ (appelée ***complément***) telle que :

  — $x \sqcap \overline{x} = \bot$

  — $x \sqcup \overline{x} = \top$

Un treillis à la fois borné, distributif et complet s'appelle une ***algèbre de Boole***.

Exemples

— Ensemble des parties d'un ensemble E avec l'ordre d'inclusion.
— Valeurs booléennes vrai et faux, avec faux $\leq$ vrai.

Propriétés Dans une algèbre de Boole, le complément est unique (c'est-à-dire : il n'existe qu'une fonction avec les propriétés demandées), il vérifie les lois de de Morgan :

— $\overline{\overline{x}} = x$,
— $\overline{x \sqcap y} = \overline{x} \sqcup \overline{y}$,
— $\overline{x \sqcup y} = \overline{x} \sqcap \overline{y}$,

et il inverse l'ordre des éléments :

— $x \leq y$ si et seulement si $\overline{y} \leq \overline{x}$.

11.3 Points fixes

Treillis complet Un treillis $(E, \leq)$ est ***complet*** si toute partie X de E admet :

— une borne inférieure, notée $\bigsqcap X$,
— une borne supérieure, notée $\bigsqcup X$.

Note : ce qui est important ici est que les bornes supérieure et inférieure existent même pour les parties infinies de E.

Fonction monotone Étant donnés deux ensembles ordonnés $(A, \leq_A)$ et $(B, \leq_B)$, une fonction $f : A \longrightarrow B$ est ***monotone*** si elle préserve l'ordre des éléments :

$$\forall a_1, a_2 \in A,\ a_1 \leq_A a_2 \Longrightarrow f(a_1) \leq_B f(a_2)$$

Théorème de point fixe Si $(E, \leq)$ est un treillis complet et $f : E \longrightarrow E$ une fonction monotone sur E, alors l'ensemble $\{e \in E \mid f(e) = e\}$, appelé ensemble des ***points fixes*** de E, est non vide et admet un minimum et un maximum.

Fonction continue Étant donnés deux treillis complets $(A, \leq_A)$ et $(B, \leq_B$), une fonction $f : A \longrightarrow B$ est ***continue*** si elle préserve les bornes supérieure et inférieure de chaque ensemble, c'est-à-dire si pour toute partie $X \subseteq A$, l'ensemble $f(X) \subseteq B$ des images des éléments de X vérifie :

— $f(\bigsqcap X) = \bigsqcap(f(X))$
— $f(\bigsqcup X) = \bigsqcup(f(X))$

Caractérisation du plus petit point fixe Si $(E, \leq)$ est un treillis complet et $f : E \longrightarrow E$ une fonction continue sur E, alors le plus petit point fixe de E est donné par :

$$\bigsqcap\{f^k(\bot) \mid k \in \mathbb{N}\}$$

(sachant que l'itération f^k de la fonction f est définie par $f^0(x) = x$ et $f^{k+1}(x) = f(f^k(x))$)

Calcul du plus petit point fixe dans un treillis fini Si $(E, \leq)$ est un treillis complet fini et $f : E \longrightarrow E$ une fonction continue sur E, alors l'algorithme suivant termine et renvoie le plus petit point fixe de E :

1. Définir $p = \bot$.
2. Tant que $f(p) \neq p$,
   (a) $p := f(p)$.
3. Renvoyer p.

12 SAT : algorithmes de résolution de contraintes (07/05)

12.1 Formules logiques vues comme des termes

L'ensemble des formules de la logique propositionnelle (chapitre 1 !) peut être vu comme un ensemble des termes sur une signature contenant les symboles suivants :

— les symboles $\top$ et $\bot$, d'arité 0,
— un ensemble $\{A, B, C, ...\}$ de symboles d'arité 0 (les variables propositionnelles),
— un symbole $\neg$ d'arité 1,
— des symboles $\wedge$, $\vee$ et $\Longrightarrow$ d'arité 2.

Cette vision permet de parler des formules en tant que telles et de leurs propriétés, ou encore de parler de définir des fonctions s'appliquant aux formules.

Exemple : principe de récurrence sur les formules Soit P un propriété sur les formules. Si :

1. $P(\bot)$,
2. $P(\top)$,
3. pour toute variable A, $P(A)$,
4. pour toute formule φ, si $P(\varphi)$ alors $P(\neg\varphi)$,
5. pour toutes formules φ_1 et φ_2, si $P(\varphi_1)$ et $P(\varphi_2)$ alors $P(\varphi_1 \wedge \varphi_2)$,
6. pour toutes formules φ_1 et φ_2, si $P(\varphi_1)$ et $P(\varphi_2)$ alors $P(\varphi_1 \vee \varphi_2)$,
7. pour toutes formules φ_1 et φ_2, si $P(\varphi_1)$ et $P(\varphi_2)$ alors $P(\varphi_1 \Longrightarrow \varphi_2)$,

alors la propriété $P(\varphi)$ est vérifiée pour tout formule φ.

Satisfiabilité (rappel) Une ***interprétation*** est une fonction associant à chaque variable propositionnelle une valeur de vérité. Étant donnée une formule φ et une interprétation I de ses variables propositionnelle, on peut calculer une valeur de vérité de φ pour I. Une formule φ est :

— ***valide*** si toute interprétation rend la formule vraie,
— ***satisfiable*** si au moins une interprétation rend la formule vraie,
— ***contradictoire*** si aucune interprétation ne rend la formule vraie.

Pour savoir si une formule est satisfiable on peut construire sa table de vérité, dont les lignes sont en correspondance avec l'ensemble des interprétations imaginables de ses variable propositionnelles, et chercher si au moins une des lignes indique Vrai. Souci : le nombre de lignes est exponentiel en le nombre de variables propositionnelles.

12.2 Formes normales

Pour explorer plus efficacement les différentes interprétations des variables d'une formule φ et décider plus rapidement de sa satisfiabilité, on peut transformer φ en une formule logiquement équivalente mais de forme plus simple.

Forme normale négative Formule dans laquelle :

— le symbole d'implication n'apparaît pas,
— les négations n'apparaissent qu'au niveau des variables propositionnelles

On se débarasse des implications avec l'équivalence $A \Longrightarrow B \equiv \neg A \vee B$, et on repousse les négations au niveau des variables avec les lois de de Morgan.

En suivant la structure récursive des formules, on peut ainsi écrire des équations décrivant cette transformation. On notera $\mathsf{nnf}(\varphi)$ la forme normale négative de la formule φ, et $\overline{\mathsf{nnf}}(\varphi)$ la forme normale négative de la formule $\neg\varphi$.

$$
\begin{array}{rcl}
\mathsf{nnf}(\bot) & = & \bot \\
\mathsf{nnf}(\top) & = & \top \\
\mathsf{nnf}(A) & = & A \\
\mathsf{nnf}(\neg\varphi) & = & \overline{\mathsf{nnf}}(\varphi) \\
\mathsf{nnf}(\varphi_1 \wedge \varphi_2) & = & \mathsf{nnf}(\varphi_1) \wedge \mathsf{nnf}(\varphi_2) \\
\mathsf{nnf}(\varphi_1 \vee \varphi_2) & = & \mathsf{nnf}(\varphi_1) \vee \mathsf{nnf}(\varphi_2) \\
\mathsf{nnf}(\varphi_1 \Longrightarrow \varphi_2) & = & \overline{\mathsf{nnf}}(\varphi_1) \vee \mathsf{nnf}(\varphi_2)
\end{array}
\qquad
\begin{array}{rcl}
\overline{\mathsf{nnf}}(\bot) & = & \top \\
\overline{\mathsf{nnf}}(\top) & = & \bot \\
\overline{\mathsf{nnf}}(A) & = & \neg A \\
\overline{\mathsf{nnf}}(\neg\varphi) & = & \mathsf{nnf}(\varphi) \\
\overline{\mathsf{nnf}}(\varphi_1 \wedge \varphi_2) & = & \overline{\mathsf{nnf}}(\varphi_1) \vee \overline{\mathsf{nnf}}(\varphi_2) \\
\overline{\mathsf{nnf}}(\varphi_1 \vee \varphi_2) & = & \overline{\mathsf{nnf}}(\varphi_1) \wedge \overline{\mathsf{nnf}}(\varphi_2) \\
\overline{\mathsf{nnf}}(\varphi_1 \Longrightarrow \varphi_2) & = & \mathsf{nnf}(\varphi_1) \wedge \overline{\mathsf{nnf}}(\varphi_2)
\end{array}
$$

Formes normales disjonctive et conjonctive On appelle ***littéral*** une formule qui est soit une variable propositionnelle (A), soit la négation d'une variable propositionnelle ($\neg A$). On s'intéresse aux formes particulières suivantes :

- Une ***conjonction élémentaire*** est une conjonction de littéraux.
  Exemple : $A \wedge B \wedge \neg C$
- Une ***forme disjonctive*** est une disjonction de conjonctions élémentaires.
  Exemple : $(A \wedge B \wedge \neg C) \vee (\neg B \wedge C) \vee (\neg A)$
- Une ***clause*** est une disjonction de littéraux.
  Exemple : $A \vee \neg B \vee C$
- Une ***forme conjonctive*** est une conjonction de clauses.
  Exemple : $(A \vee B \vee \neg C) \wedge (\neg B \vee C) \wedge (\neg A)$

Mise sous forme disjonctive Il est facile de trouver une formule en forme disjonctive équivalente à une formule φ donnée, en consultant la table de vérité de φ :

- chaque ligne donne une conjonction de littéraux (contenant A si la variable A est à Vrai sur cette ligne, $\neg A$ si la variable A est à Faux),
- on prend la disjonction des lignes pour laquelle la formule vaut Vrai.

Souci : la formule obtenue est de taille exponentielle (et le calcul de la table de vérité nous donnait déjà toutes les informations que nous voulions).

Codage de Tseitin Pour obtenir une formule en forme conjonctive équivalent à une formule φ donnée, on peut recourir à l'astuce suivante :

- créer une nouvelle variable propositionnelle pour chaque symbole présent dans la formule, c'est-à-dire pour chaque nœud de l'arbre représentant la formule,
- exprimer les conditions auxquelles chacune de ces nouvelles variables sont vraies en fonction des variables représentant les sous-termes immédiats.

Ainsi, pour une formule $(A \wedge B) \vee (C \Longrightarrow D)$, on introduit trois nouvelles variables X_1, X_2, X_3 telles que :

— X_1 représente la validité de la formule $A \wedge B$, on a donc l'équivalence $X_1 \iff (A \wedge B)$, qui peut aussi être représentée par la conjonction des trois clauses $\neg X_1 \vee A$, $\neg X_1 \vee B$ et $\neg A \vee \neg B \vee X_1$;
— X_2 représente la validité de la formule $C \Longrightarrow D$, et peut être traduite en trois clauses comme la précédente ;
— X_3 représente la validité de la formule $X_1 \vee X_2$, et peut encore être traduite en trois clauses ;
— On ajoute une clause formée de l'unique littéral X_3, pour signifier qu'on cherche à satisfaire la formule complète.

Cette transformation produit une formule dont le nombre de variables comparable à la taille de la formule d'origine, le nombre de clauses est environ trois fois cette taille, et chacune des clauses est petite (entre un et trois littéraux).

12.3 Algorithme DPLL

L'algorithme DPLL (Davis/Putman/Loveland/Logemann) explore efficacement l'ensemble des interprétations possibles pour une formule en forme conjonctive. Le principe est d'essayer des interprétations partielles (c'est-à-dire concernant seulement une partie des variables), et d'élaguer certaines parties de l'arbre d'exploration.

Une exploration en cours est représentée par une paire $I >> \Delta$ où :
— I est une interprétation pour une partie des variables propositionnelles,
— Δ est un ensemble de clauses restant à satisfaire.

On a deux cas particuliers de fin d'une branche de l'exploration, c'est-à-dire d'une feuille dans l'arbre d'exploration :
— $I >> \emptyset$: toutes les clauses sont satisfaites, la formule est satisfiable.
— $I >> \Delta, C$, où tous les littéraux de la clause C sont invalidés par l'interprétation I : aucune interprétation contenant I ne peut satisfaire la formule.

Quand une clause C est satisfaite, c'est-à-dire que l'un de ses littéraux est validé par l'interprétation I, on peut la supprimer. On passe donc de $I >> \Delta, C$ à $I >> \Delta$.

Quand nous ne sommes pas dans l'un de ces cas, on peut choisir une variable A pas encore interprétée par I et poursuivre l'exploration dans deux branches, donnant respectivement les valeurs Vrai et Faux à cette variable. On a dans ce cas un nœud dans l'arbre d'exploration, dont les fils sont :

1. $I, A \mapsto \text{Vrai} >> \Delta$
2. $I, A \mapsto \text{Faux} >> \Delta$

En partant de l'ensemble des clauses de la formule et de l'interprétation vide, on obtient un arbre d'exploration. La formule est satisfiable si au moins une des feuilles a donné ce verdict. À l'inverse, la formule n'est pas satisfiable si aucune des feuilles n'est satisfiable.

Accélération avec les clauses implicatives Une clause C est appelée une ***clause implicative*** sous une interprétation partielle I si l'interprétation I invalide tous les littéraux de I sauf un, et ne donne pas de valeur au dernier littéral.

On sait alors quelle valeur donner à la variable du dernier littéral :
— Vrai si le littéral est de la forme A,
— Faux si le littéral est de la forme $\neg A$.

Ce faisant on satisfait la clause C observée (on peut l'éliminer), et surtout on évite l'exploration de tout une partie de l'arbre. On continue ainsi en cascade tant que notre ensemble Δ contient des clauses implicatives, en limitant d'autant le nombre de choix arbitraires à faire et donc de branches à explorer.

Apprentissage Les clauses implicatives forcent certains choix. On peut représenter dans un graphe l'ensemble des choix qui ont été ainsi forcés et des clauses qui y ont participé. L'observation de ce graphe permet ensuite, lorsque l'exploration aboutit à une feuille non satisfiable, de chercher les raisons premières de cet échec et de déduire une nouvelle clause appelée ***clause de conflit***. Cette nouvelle clause sera ajoutée à l'ensemble des clauses à satisfaire dans les autres branches et aura pour effet d'éviter de retomber dans les mêmes pièges.

Ainsi, si une fausse route similaire se présente à nouveau dans une autre branche elle sera cette fois évitée : l'algorithme *apprend* quelque chose du problème qu'il essaie de résoudre. C'est l'efficacité de ce dernier mécanisme qui fait que, bien que l'exploration des interprétations soit en théorie exponentiel, les solveurs actuels soient capables de résoudre des problèmes faisant intervenir des millions de variables propositionnelles.