Introduction à la théorie de l'informatique

Pierre Geurts

Version du 8 novembre 2013

E-mail : p.geurts@ulg.ac.be
URL : http://www.montefiore.ulg.ac.be/~geurts/iti.html
Bureau : R 73 a (Montefiore)
Téléphone : 04.366.48.15 — 04.366.99.64

Organisation du cours

Tous les vendredis de 9h à 12h30, local 2/93

Cours théorique (1h30-2h, Pierre Geurts) suivi d'une séance d'exercices (1h30-2h, Simon Liénardy, Antonio Sutera)

Examen écrit uniquement (en janvier et septembre). Très probablement à livre ouvert.

Objectifs et plan du cours

Objectifs :

- Vous former aux outils mathématiques et aux modes de raisonnement les plus utilisés en informatique.
- Introduire certains domaines et résultats importants de l'informatique théorique

L'accent sera mis sur l'exploitation pratique des concepts vus au cours plus que sur la restitution de la matière théorique

Trois parties principales :

- Partie 1 : Techniques de preuves et analyse formelle de programmes *(±5 cours)*
- Partie 2 : Théorie des graphes *(±3 cours)*
- Partie 3 : Combinatoire et analyse de complexité *(±5 cours)*

Notes de cours

Transparents disponibles en ligne :
http://www.montefiore.ulg.ac.be/~geurts/iti.html

Principaux ouvrages de référence :

- **Mathematics for Computer Science. Eric Lehman, Tom Leighton et Albert Meyer, 2012.** http://courses.csail.mit.edu/6.042/spring12/mcs.pdf.
- Discrete Mathematics with Applications. Susanna E. Epp, 2010 (4ème édition).
- Discrete Mathematics and Its Applications. Kenneth Rosen, MCGraw-Hill, 2012 (7ème édition).
- Notes de cours de Guy McCusker et Matthew Hennessy, University of Sussex. http://www.informatics.sussex.ac.uk/courses/semantics/current/GuysNotes.pdf
- David A. Schmidt, Programming Language Semantics. http://people.cis.ksu.edu/~schmidt/705s12/Lectures/chapter.pdf
- R. Sedgewick et P. Flagolet, *Analysis of Algorithms*, Addison-Wesley, 1995. http://aofa.cs.princeton.edu/.

Des pointeurs supplémentaires seront fournis au fur et à mesure du cours.

Partie 1

Techniques de preuves et analyse formelle de programme

- Chapitre 1 : Preuves
- Chapitre 2 : Machines d'état et correction de programmes
- Chapitre 3 : Définitions récursives et induction structurelle
- Chapitre 4 : Introduction à la syntaxe et à la sémantique des langages de programmation

Objectifs :

- Vous former à l'écriture de preuves et au raisonnement inductif
- Utiliser ces outils pour vérifier des propriétés de programmes (et de structures de données)

Partie 2

Introduction à la théorie des graphes

Chapitre 5 :

1. Définitions
2. Connexité
3. Coloriage de graphes
4. Graphes bipartis et appariement
5. Réseaux de communication

Objectifs :

- Introduire les notions essentielles autour des graphes
- Vous former à l'écriture de preuves relatives aux graphes

Partie 3

Outils pour l'analyse d'algorithmes

- Chapitre 6 : Sommations et comportements asymptotiques
- Chapitre 7 : Récurrences
- Chapitre 8 : Fonctions génératrices

Objectifs :

- Vous fournir les principaux outils mathématiques nécessaires à l'analyse d'algorithmes, c'est-à-dire évaluer leur coût, en terme de temps de calcul ou d'utilisation de la mémoire.

Chapitre 1

Techniques de preuves

Plan

1. Définitions
2. Techniques de preuves simples
3. Principe du bon ordre
4. Induction
5. Principe d'invariant

Lectures conseillées : MCS : chapitres 1, 2, et 5.

Définition : Une *démonstration* est une vérification d'une *proposition* par une séquence de *déductions logiques* à partir d'un ensemble d'*axiomes*.

Propositions

Définition : Une *proposition* est un énoncé qui est *soit vrai*, *soit faux*.

Exemples :

- $2 + 3 = 5$. Proposition vraie.
- $(\forall n \in \mathbb{N})$ $n^2 + n + 41$ est un nombre premier. Proposition fausse : pour $n = 40$, on a $n^2 + n + 41 = 40^2 + 40 + 41 = 41^2$.
- (Conjecture d'Euler, 1769) $a^4 + b^4 + c^4 = d^4$ n'a pas de solution quand $a, b, c, d \in \mathbb{N}^+$. Proposition fausse (Elkies, 1988). Contre-exemple : $a = 95800, b = 217519, c = 414560, d = 422481$.
- $(\exists a, b, c, d \in \mathbb{N}^+)$ $a^4 + b^4 + c^4 = d^4$. Proposition vraie.

- $(\forall n \in \mathbb{Z})$ $(n \geq 2) \Rightarrow (n^2 \geq 4)$. Proposition vraie.
- $1 = 0 \Rightarrow (\forall n \in \mathbb{N})$ $n^2 + n + 41$ est un nombre premier. Proposition vraie.
- $(\forall n \in \mathbb{Z})$ $(n \geq 2) \Leftrightarrow (n^2 \geq 4)$. Proposition fausse.

Prédicats

Définition : Une proposition dont la valeur de vérité dépend de la valeur d'une ou plusieurs variables

Exemples :

- "n est un carré parfait" : vrai pour $n = 4$ mais faux pour $n = 10$
- Souvent noté $P(n) =$"n est un carré parfait". $P(4)$ est vrai. $P(5)$ est faux.

Axiomes

- Définition : Un *axiome* est une proposition qui est *supposée vraie*.
- Exemple : $(\forall a, b, c \in \mathbb{Z})\ (a = b \text{ et } b = c) \Rightarrow (a = c)$.
- Un ensemble d'axiomes est *consistant* s'il n'existe pas de proposition dont on peut démontrer qu'elle est *à la fois vraie et fausse*.
- Un ensemble d'axiomes est *complet* si, pour toute proposition, il est possible de démontrer qu'elle est vraie ou fausse.
- Théorème d'incomplétude de Gödel (1931) : tout ensemble consistant d'axiomes pour l'arithmétique sur les entiers est nécessairement incomplet.
- Dans ce cours, on considérera comme axiomes les notions des mathématiques de base.

Autres types de proposition

- Un *théorème* est une proposition qui peut être démontrée
- Un *lemme* est une proposition préliminaire utile pour faire la démonstration d'autres propositions plus importantes
- Un *corrolaire* est une proposition qui peut se déduire d'un théorème en quelques étapes logiques
- Une *conjecture* est une proposition pour laquelle on ne connaît pas encore de démonstration mais que l'on soupçonne d'être vraie, en l'absence de contre-exemple. Exemple : tout entier pair strictement plus grand que 2 est la somme de deux nombres premiers (Conjecture de Golbach).

Déductions logiques

- Définition : Les *règles de déductions logiques*, ou *règles d'inférence*, sont des règles permettant de combiner des axiomes et des propositions vraies pour établir de nouvelles propositions vraies.
- Exemple : $\boxed{\dfrac{P \qquad P \Rightarrow Q}{Q}}$ (modus ponens).

  Le modus ponens est fortement lié à la proposition $(P \wedge (P \Rightarrow Q)) \Rightarrow Q$, qui est une *tautologie*.
  (= une proposition qui est toujours vraie quelles que soient les valeurs de ses variables)

Exemples de démonstrations

Théorème : La proposition suivante est une tautologie :

$$(X \Rightarrow Y) \Leftrightarrow (\neg Y \Rightarrow \neg X).$$

Démonstration : Montrons que $(X \Rightarrow Y)$ est logiquement équivalent à sa *contraposée* $(\neg Y \Rightarrow \neg X)$, quelles que soient les valeurs booléennes des variables X et Y.

X	Y	$X \Rightarrow Y$	$\neg Y \Rightarrow \neg X$
V	V	V	V
V	F	F	F
F	V	V	V
F	F	V	V

La proposition $(X \Rightarrow Y) \Leftrightarrow (\neg Y \Rightarrow \neg X)$ est donc vraie dans tous les cas, ce qui implique qu'elle est une tautologie. □

Les deux règles suivantes sont donc des règles d'inférence.

$$\frac{P \Rightarrow Q}{\neg Q \Rightarrow \neg P} \qquad \frac{\neg Q \Rightarrow \neg P}{P \Rightarrow Q.}$$

Schémas de preuves classiques

Quelques schémas de preuves classiques :

- Implication
- Equivalence (si et seulement si)
- Preuve par cas
- Preuve par contradiction (ou par l'absurde)
- Principe du bon ordre
- Induction faible
- Induction forte
- Principe d'invariant

Une preuve complexe requière la plupart du temps la combinaison de plusieurs de ces techniques

Implications

Deux méthodes pour prouver que P implique Q (noté $P \Rightarrow Q$)

1. Supposez que P est vrai et montrez que Q vrai en découle.

Théorème : Si $0 \leq x \leq 2$, alors $-x^3 + 4x + 1 > 0$
Démonstration :

- Supposons que $0 \leq x \leq 2$.
- Alors x, $2 - x$ et $2 + x$ sont tous non négatifs et leur produit l'est également.
- Ajouter 1 à ce produit donne un nombre strictement positif :

$$x(2 - x)(2 + x) + 1 > 0$$

- En multipliant les termes, on obtient :

$$-x^3 + 4x + 1 > 0$$

□

Implications

Deux méthode pour prouver que P implique Q (noté $P \Rightarrow Q$)

2. On démontre la *contraposée*. Régle d'inférence : $\dfrac{\neg Q \Rightarrow \neg P}{P \Rightarrow Q}$

Théorème : Si r est irrationnel, alors $\sqrt{r}$ est aussi irrationnel
Démonstration :

- Par contraposition, il suffit de démontrer que si $\sqrt{r}$ est rationnel, alors r est rationnel.
- Supposons que $\sqrt{r}$ est rationnel. Il existe alors des entiers m et n tels que :
$$\sqrt{r} = \frac{m}{n}$$
- En mettant au carré les deux côtés de l'équation, on obtient :
$$r = \frac{m^2}{n^2}$$
- Etant donné que m^2 et n^2 sont des entiers, on a montré que r était rationnel. □

Si et seulement si

Deux méthodes :

1. Chaînage de si et seulement si
$$\frac{P \Leftrightarrow R, R \Leftrightarrow Q}{P \Leftrightarrow Q}$$
2. Preuve des implications dans les deux sens
$$\frac{P \Rightarrow Q, Q \Rightarrow P}{P \Leftrightarrow Q}$$

Si et seulement si : exemple

Théorème : $(\forall a \in \mathbb{Z})$ (a est pair) $\Leftrightarrow$ (a^2 est pair).

Démonstration : Soit a un entier quelconque.

$\boxed{a \text{ est pair} \Rightarrow a^2 \text{ est pair}}$ Supposons que a soit pair. On a donc $a = 2b$, avec $b \in \mathbb{Z}$. Dès lors, on obtient $a^2 = (2b)^2 = 4b^2 = 2(2b^2)$. Le nombre a^2 est donc pair.

$\boxed{a^2 \text{ est pair} \Rightarrow a \text{ est pair}}$ Par contraposition, il suffit de démontrer que a est impair $\Rightarrow$ a^2 est impair. Supposons que a soit impair. On a donc $a = 2b + 1$, avec $b \in \mathbb{Z}$. Dès lors, on obtient $a^2 = (2b + 1)^2 = 4b^2 + 4b + 1 = 2(2b^2 + 2b) + 1$. Le nombre a^2 est donc impair. □

Preuve par cas

Décomposer la preuve en différents cas qui seront traités séparément

Exemple :
Théorème : Toute ensemble de 6 personnes inclut un groupe de 3 personnes qui se sont déjà rencontrées ou un groupe de 3 étrangers.

Démonstration La preuve se fait par une analyse de cas. Soit x une des 6 personnes. Il y a deux cas possibles :

- Parmi les 5 autres personnes du groupe, au moins 3 connaissent x.
- Parmi les 5 autres personnes du groupe, au moins 3 ne connaissent pas x.

En effet, si on divise le groupe des 5 personnes en deux groupes, ceux qui connaissent x et ceux qui ne le connaissent pas, un des deux groupes doit contenir au moins la moitié des 5 personnes.

Preuve par cas

On va traiter les deux cas séparément.
Cas 1 : Au moins 3 personnes connaissent x. Ce cas peut se diviser en deux sous-cas :

- **Cas 1.1 :** Aucune de ces 3 personnes ne se connaissent et ils forment donc un groupe de 3 étrangers
- **Cas 1.2 :** Deux personnes parmi les 3 se connaissent et avec x, ils forment un groupe de 3 personnes qui se connaissent.

Dans les deux cas, le théorème est vérifié et donc il l'est dans le cas 1.
Cas 2 : Au moins 3 personnes ne connaissent pas x. Ce cas peut aussi se diviser en deux sous-cas :

- **Cas 1.1 :** Ces 3 personnes se connaissent.
- **Cas 1.2 :** Deux personnes parmi les 3 ne se connaissent pas et avec x, ils forment un groupe de 3 étrangers.

Dans les deux cas, le théorème est vérifié et donc il l'est dans le cas 2.
Le théorème est donc vrai dans tous les cas. □

Démonstrations par l'absurde

Principe :

- On veut démontrer qu'une proposition P est vraie.
- On suppose que $\neg P$ est vraie, et on montre que cette hypothèse conduit à une *contradiction*.
- Ainsi, $\neg P$ est fausse, ce qui implique que P est vraie.

Règle d'inférence correspondante :

$$\frac{\neg P \Rightarrow \text{faux}}{P}$$

Exemple

Théorème : $\sqrt{2} \in \mathbb{R}\backslash\mathbb{Q}$.

Démonstration : Par l'absurde, supposons que $\sqrt{2} \in \mathbb{Q}$. On a donc

$$\sqrt{2} = \frac{a}{b},$$

où $a, b \in \mathbb{Z}$, $b \neq 0$ et où cette fraction est réduite (a et b n'ont pas de facteur commun). Cela implique $2 = \frac{a^2}{b^2}$, et donc

$$2b^2 = a^2.$$

Par conséquent, le nombre a^2 est pair, ce qui implique que a est lui-même pair.

Il existe donc $a' \in \mathbb{Z}$ tel que $a = 2a'$. On a donc $a^2 = 4a'^2$. Donc, on a $2b^2 = 4a'^2$, ce qui implique que

$$b^2 = 2a'^2.$$

Dès lors, b^2 est pair, et donc b est lui-même pair. Il existe donc $b' \in \mathbb{Z}$ tel que $b = 2b'$. La fraction

$$\frac{a}{b} = \frac{2a'}{2b'}$$

n'est donc pas réduite. C'est une contradiction. Par conséquent, l'hypothèse selon laquelle $\sqrt{2} \in \mathbb{Q}$ est fausse. Donc, on a $\sqrt{2} \in \mathbb{R}\backslash\mathbb{Q}$. □

Écrire de bonnes démonstrations

En plus d'être logiquement correcte, une bonne démonstration doit être *claire*.

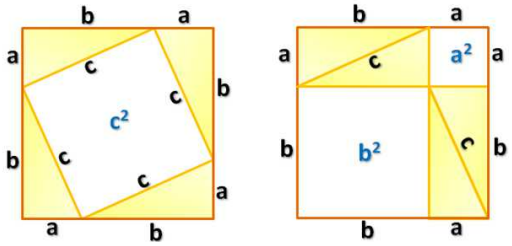
Conseils pour l'écriture de bonnes démonstrations :

- Expliquez la manière dont vous allez procéder (par l'absurde, contraposition, induction . . .) ;
- Donnez une explication séquentielle ;
- Expliquez votre raisonnement (passages d'une étape à l'autre, arithmétique, induction, . . .) ;
- N'utilisez pas trop de symboles ; utiliser du texte lorsque c'est possible ;
- Simplifiez ;

- Introduisez des notations judicieusement, en prenant soin de définir leur signification ;
- Si la démonstration est trop longue, structurez-la (par exemple établissez à l'aide de *lemmes* les faits dont vous aurez souvent besoin) ;
- N'essayez pas de camoufler les passages que vous avez du mal à justifier ;
- Terminez en expliquant à quelles conclusions on peut arriver.

Une preuve sans mot

Théorème de Pythagore : Dans un triangle rectangle, le carré de la longueur de l'hypothénuse est égal à la somme des carrés des longueurs des deux autres côtés.
Démonstration :



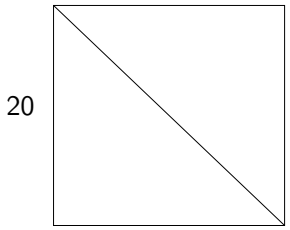
Source :
`http://mathandmultimedia.com/2012/06/01/mathematical-proof-without-words/`

Un faux théorème

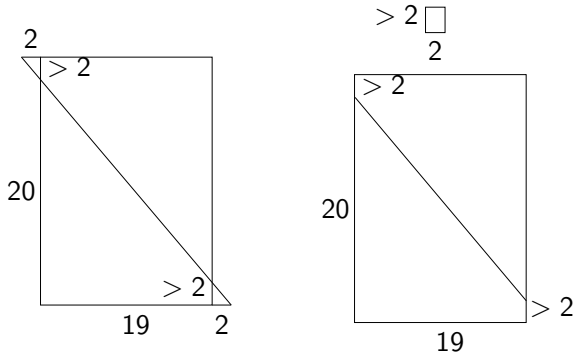
Quelle est l'erreur dans la démonstration suivante ?

Faux théorème : $420 > 422$.

Démonstration erronée : Démonstration géométrique. Soit un rectangle de dimension 20×21. Son aire vaut donc 420.



Découpage + glissement de 2 unités vers la gauche :



- Aire du petit rectangle : > 4.
- Aire du grand rectangle : $> (20 + 2) \times 19 = 418$.
- $\Rightarrow$ Aire totale : > 422. Par conservation d'aire, on a donc $420 > 422$. □

Plan

Principe du bon ordre

Le principe du bon ordre (sur les naturels) s'énonce comme suit :

> *tout ensemble non vide d'entiers non-négatifs possède un plus petit élément.*

Principe évident mais à la base de nombreuses preuves.

Principe du bon ordre

Exemple dans une précédente preuve :

> *...$\sqrt{2}$ peut s'écrire $\frac{a}{b}$ où on suppose que a et b n'ont pas de facteur commun...*

Montrons que c'est toujours possible par le principe du bon ordre :

- Soit l'ensemble suivant : $C = \{a \in \mathbb{N} | \exists b \in \mathbb{N} : \sqrt{2} = \frac{a}{b}\}$.
- Par le principe du bon ordre, il existe un plus petit élément a' dans C. Soit $b' \in \mathbb{N}$ tel que $\sqrt{2} = \frac{a'}{b'}$.
- Supposons que a' et b' aient un facteur commun $c' > 1$. On a alors $\sqrt{2} = \frac{a'/c'}{b'/c'}$ et donc a'/c' appartient à C.
- Or $a'/c' < a'$, ce qui amène une contradiction puisque a' est le plus petit élément de C.
- On a donc $\sqrt{2} = \frac{a'}{b'}$ où a' et b' n'ont pas de facteur commun.

Schéma de preuve par le principe du bon ordre

Pour prouver qu'un prédicat $P(n)$ est vrai pour tout $n \in \mathbb{N}$:

- Définir l'ensemble $C = \{n \in \mathbb{N} | P(n) \text{ est faux}\}$.
- Supposer que C est non vide comme base pour une preuve par contradiction
- Par le principe du bon ordre, il y a un plus petit élément, n, dans C
- Atteindre une contradiction, souvent en utilisant n pour trouver un autre élément de C plus petit que n.
- Conclure que C doit être vide et donc que $P(n)$ est vrai pour tout n.

Exemple

Théorème : Pour tout $n \in \mathbb{N}$, on a

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}.$$

Démonstration : Par l'absurde. Supposons que le théorème soit faux et définissons C comme suit :

$$C = \{n \in \mathbb{N} | \sum_{i=1}^{n} i \neq \frac{n(n+1)}{2}\}.$$

Si le théorème est faux, C est non vide et par le principe du bon ordre, il contient un élément minimum. Soit c cet élément.
Par définition de c, le théorème est vrai pour tout $n < c$. Or, il est vrai pour $n = 0$ et donc on a $c > 0$. $c - 1$ est donc un entier non négatif et comme $c - 1 < c$, le théorème est vrai pour $c - 1$.

On en déduit que :

$$\sum_{i=1}^{c-1} i = \frac{(c-1)c}{2}.$$

En ajoutant c des deux côtés, on obtient

$$\sum_{i=1}^{c-1} i + c = \sum_{i=1}^{c} i = \frac{(c-1)c}{2} + c = \frac{c(c+1)}{2},$$

ce qui veut dire que le théorème est vérifié pour c.

On arrive donc à une contradiction, ce qui nous permet de conclure que le théorème est vrai pour tout $n \in \mathbb{N}$. □

Un autre exemple

Théorème : Tout nombre entier positif plus grand que 1 peut s'écrire comme un produit de nombres premiers

Démonstration :

- La preuve utilise le principe du bon ordre.
- Soit C l'ensemble des entiers supérieurs à 1 qui ne peuvent pas être factorisés comme un produit de premiers. Supposons que C soit non vide et montrons que nous arrivons à une contradiction.
- Soit n le plus petit élément de C, par le principe du bon ordre. n ne peut pas être premier, car un premier est un produit de premiers (de taille 1) et donc n serait alors dans C.
- n est donc le produit de deux entiers a et b tels que $1 < a, b < n$.

- Puisque a et b sont plus petits que n, ils ne peuvent pas appartenir à C et donc a peut s'écrire comme un produit de premiers $p_1 p_2 \dots p_k$ et b comme un produit de premiers $q_1 q_2 \dots q_l$.
- En conséquence, $n = p_1 \dots p_k q_1 \dots q_l$ peut s'écrire comme un produit de premiers, ce qui contredit $n \in C$. □

Ensembles bien ordonnés

On peut généraliser le principe du bon ordre à d'autres ensembles.

Définition : Un ensemble est *bien ordonné* si tous ses sous-ensembles possèdent un élément minimal.

Théorème : Pour tout entier non négatif, n, l'ensemble des entiers supérieurs ou égaux à $-n$ est bien ordonné

Démonstration : Soit S un sous-ensemble non vide d'entiers $\geq -n$, montrons que S possède un élément minimum. Ajoutons n à chaque élément de S et appelons ce nouvelle ensemble $S + n$. $S + n$ est une sous-ensemble non vide d'entiers non-négatifs, donc, par le principe du bon ordre, il a un élément minimum m. Il est facile de se convaincre que $m - n$ est l'élément minimum de S. □

Un ensemble bien ordonné plus complexe

Soit l'ensemble $To1$ des fractions suivantes :

$$\frac{0}{1}, \frac{1}{2}, \frac{2}{3}, \dots, \frac{n}{n+1}, \dots$$

$To1$ est bien ordonné, le minimum d'un sous-ensemble étant la fraction de ce sous-ensemble avec le numérateur le plus petit (qui existe par le principe du bon ordre).

Soit $\mathbb{N} + To1$ l'ensemble de tous les nombres $n + f$ où n est un entier non négatif et f est un élément de $To1$.

Théorème : $\mathbb{N} + To1$ est bien ordonné.

Démonstration :
Soit un sous-ensemble non vide, S, de $\mathbb{N} + To1$. Soit l'ensemble de tous les entiers non négatifs, n, tels que $n + f$ est dans S pour un $f \in To1$. Cet ensemble est une sous-ensemble non vide d'entiers non négatifs et par le principe du bon ordre, il a donc un élément minimum. Notons le n_S.

Soit l'ensemble des fractions f tels que $n_S + f$ soit dans S. Par définition de n_S, cet ensemble est un sous-ensemble non vide de $To1$ et $To1$ étant bien ordonné, il possède donc un élément minimum. Soit f_S cet élément.

Étant donné que toute fraction de $To1$ est inférieure à 1 et non négative, il est facile de vérifier que $n_S + f_S$ est l'élément minimum de S. □

Plan

Principe d'induction

Principe d'induction :

Soit $P(n)$ un prédicat. Si

- $P(0)$ est vrai, et si
- pour tout $n \in \mathbb{N}$, $P(n)$ implique $P(n+1)$,

alors $P(n)$ est vrai pour tout $n \in \mathbb{N}$.

$$\frac{P(0), \forall n : P(n) \Rightarrow P(n+1)}{\forall n : P(n)}$$

Variante :

Soit $P(n)$ un prédicat et $k \in \mathbb{N}$. Si

- $P(k)$ est vrai, et si
- pour tout $n \geq k$, $P(n)$ implique $P(n+1)$,

alors $P(n)$ est vrai pour tout $n \geq k$.

$$\frac{P(k), \forall n \geq k : P(n) \Rightarrow P(n+1)}{\forall n \geq k : P(n)}$$

Un modèle pour les démonstrations par induction

1. Annoncer que la démonstration utilise une induction ;
2. Définir un prédicat approprié $P(n)$;
3. Démontrer que $P(0)$ est vrai (**"cas de base"**) ;
4. Démontrer que $P(n)$ implique $P(n+1)$ pour tout $n \in \mathbb{N}$ (**"'cas inductif"**) ;
5. Invoquer l'induction (cette étape est souvent implicite).

Illustration

Théorème : Pour tout $n \in \mathbb{N}$, on a

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}.$$

Démonstration :

La démonstration fonctionne par induction.

Soit $P(n)$ le prédicat qui est vrai si et seulement si $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.

Cas de base : $P(0)$ est vrai car $\sum_{i=1}^{0} i = 0 = \frac{0(0+1)}{2}$.

Cas inductif : Supposons que $P(n)$ soit vrai, où n est un nombre naturel quelconque, et démontrons que cette hypothèse implique la validité de $P(n+1)$. On a

$$\sum_{i=1}^{n+1} i = \left(\sum_{i=1}^{n} i\right) + (n+1).$$

Comme $P(n)$ (l'"hypothèse d'induction") est vraie, cette expression est égale à $\frac{n(n+1)}{2} + (n+1)$. On obtient donc

$$\sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2}.$$

Dès lors, par induction, $P(n)$ est vrai quel que soit le nombre naturel n, et le théorème est démontré. □

Un théorème de divisibilité

Définition : Un nombre entier a *divise* un nombre entier b si b est un multiple de a. Lorsque a divise b, on écrit $a \mid b$.

Exemple : On a $3 \mid (5^3 - 5)$ car $5^3 - 5 = 120$ est un multiple de 3.

On souhaite démontrer par induction que, quel que soit $n \in \mathbb{N}$, on a $3 \mid (n^3 - n)$.

Soit $P(n)$ le prédicat "$3 \mid (n^3 - n)$".

Le cas de base $P(0)$ est immédiat. Pour démontrer le cas inductif, il faut supposer que $3 \mid (n^3 - n)$ et en déduire que $3 \mid ((n+1)^3 - (n+1))$.

On a

$$\begin{aligned}(n+1)^3 - (n+1) &= n^3 + 3n^2 + 2n \\ &= (n^3 - n) + (3n^2 + 3n).\end{aligned}$$

Comme 3 divise $(n^3 - n)$ par hypothèse d'induction, et que $3n^2 + 3n$ est un multiple de 3, la somme $(n^3 - n) + (3n^2 + 3n)$ est un multiple de 3.

Réorganisons ce raisonnement dans une démonstration claire.

Théorème : $(\forall n \in \mathbb{N})\ 3 \mid (n^3 - n)$.

Démonstration :

- La démonstration fonctionne par induction.
- Soit $P(n)$ la proposition $3 \mid (n^3 - n)$.
- *Cas de base :* $P(0)$ est vrai car $3 \mid (0^3 - 0)$.
- *Cas inductif :* Supposons que $P(n)$ soit vrai, où $n \in \mathbb{N}$. On a

$$\begin{aligned}3 \mid (n^3 - n) &\Rightarrow 3 \mid ((n^3 - n) + 3(n^2 + n)) \\ &\Rightarrow 3 \mid (n^3 + 3n^2 + 3n + 1 - n - 1) \\ &\Rightarrow 3 \mid ((n+1)^3 - (n+1)).\end{aligned}$$

  Première implication : $3(n^2 + n)$ est divisible par 3.
  Autres implications : réécriture de l'expression de droite.
  On a prouvé que $P(n)$ implique $P(n+1)$ pour tout $n \in \mathbb{N}$.
- Dès lors, par induction, $P(n)$ est vrai quel que soit $n \in \mathbb{N}$, et le théorème et démontré. □

Une démonstration par induction erronée

Faux théorème : Tous les chevaux ont la même couleur.

Démonstration erronée : *(Où est l'erreur ?)*

- La démonstration fonctionne par induction.
- $P(n)$: "pour tout ensemble de n chevaux, tous ces chevaux ont la même couleur".
- *Cas de base :* $P(1)$ est vrai car tous les chevaux dans un ensemble de 1 cheval ont la même couleur.
- *Cas inductif :* Supposons que $P(n)$ soit vrai. Soit un ensemble de $n+1$ chevaux :

$$c_1, c_2, \ldots, c_n, c_{n+1}.$$

Par hypothèse, les n premiers chevaux ont la même couleur. Il en est de même pour les n derniers :

$$\underbrace{c_1, c_2, \ldots, c_n}_{\text{même couleur}}, c_{n+1}.$$

$$c_1, \underbrace{c_2, \ldots, c_n, c_{n+1}}_{\text{même couleur}}.$$

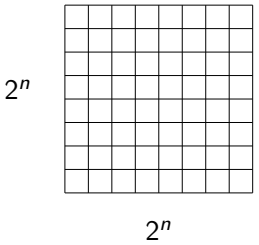
Dès lors, les chevaux $c_1, c_2, \ldots, c_{n+1}$ ont la même couleur, i.e., $P(n+1)$ est vrai. Donc, $P(n)$ implique $P(n+1)$.

- Par induction, $P(n)$ est vrai pour tout $n \geq 1$. Le théorème est un cas particulier de ce résultat : celui où n vaut le nombre total de chevaux dans le monde. □

Dallage

On souhaite créer une terrasse de dimension $2^n \times 2^n$ à la place de la pelouse située au centre du bâtiment B28.

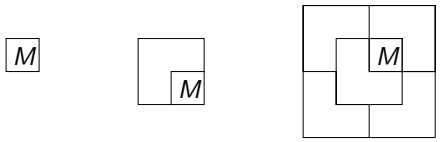
Photo : ©ULg - M. Houet



Contraintes :

- Sur un des emplacements situés au centre de la terrasse, on doit ériger une statue de Georges Montefiore (M).
- Tous les autres emplacements doivent être couverts par des dalles en "L", sans que ces dalles ne se recouvrent.

Remarque : Pour $n = 0$, $n = 1$ et $n = 2$, un dallage existe :



On demande de démontrer qu'un tel dallage existe quelle que soit la valeur $n \in \mathbb{N}$.

Problème : Choisir $P(n) =$ "il existe un dallage d'une terrasse $2^n \times 2^n$ avec M au centre" n'est pas adéquat : un dallage pour une terrasse de dimension $2^n \times 2^n$ ne permet pas de construire facilement un dallage pour une terrasse de dimension $2^{n+1} \times 2^{n+1}$.

Solution : Choisir une hypothèse d'induction *plus générale*.

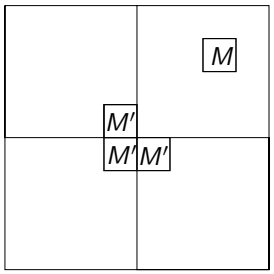
$P(n) =$ "Pour *tout* emplacement de M sur une terrasse de dimension $2^n \times 2^n$, il y a une possibilité de dallage pour le reste de la terrasse."

Théorème : Pour tout $n \in \mathbb{N}$, il existe un dallage d'une terrasse de dimension $2^n \times 2^n$ avec M au centre.

Démonstration :

- La démonstration fonctionne par induction.
- Soit $P(n) =$ "Pour *tout* emplacement de M sur une terrasse de dimension $2^n \times 2^n$, il y a une possibilité de dallage pour le reste de la terrasse."
- *Cas de base :* $P(0)$ est vrai car M couvre toute la terrasse.
- *Cas inductif :* Supposons que $P(n)$ soit vrai pour un $n \in \mathbb{N}$.
  Soit une terrasse de dimension $2^{n+1} \times 2^{n+1}$, et supposons que M se trouve sur un quelconque emplacement de celle-ci.

Divisons la terrasse en 4 quadrants, chacun de dimension $2^n \times 2^n$. Un d'entre-eux contient M. Plaçons un M temporaire (M' sur le schéma) sur chacun des 3 emplacements centraux situés dans les 3 autres quadrants.



Par l'hypothèse d'induction, chacun des 4 quadrants admet un dallage. Remplacer les 3 emplacements de M' par une dalle en "L" permet de terminer le travail. Donc $P(n)$ implique $P(n+1)$ pour tout $n \in \mathbb{N}$.

- Par induction, $P(n)$ est vrai pour tout $n \in \mathbb{N}$. Le théorème en est un cas particulier. □

Induction forte

Principe d'induction forte :

Soit $P(n)$ un prédicat. Si

- $P(0)$ est vrai, et si
- pour tout $n \in \mathbb{N}$, $P(0) \wedge P(1) \wedge \cdots \wedge P(n)$ implique $P(n+1)$,

alors $P(n)$ est vrai pour tout $n \in \mathbb{N}$.

$$\frac{P(0), \forall n : (\forall k \leq n : P(k)) \Rightarrow P(n+1)}{\forall n : P(n)}$$

Variante :

Soit $P(n)$ un prédicat, et soit $k \in \mathbb{N}$. Si

- $P(k)$ est vrai, et si
- pour tout $n \geq k$, $P(k) \wedge P(k+1) \wedge \cdots \wedge P(n)$ implique $P(n+1)$,

alors $P(n)$ est vrai pour tout $n \geq k$.

Application : jeu de dépilage

Règles du jeu :

- On commence avec une pile de n boîtes.
- A chaque étape, on divise une pile en deux piles non vides.
- Le jeu s'arrête lorsque l'on obtient n piles, chacune contenant une seule pile.
- Une division où l'on transforme une pile de hauteur $a+b$ en deux piles d'hauteurs a et b permet d'obtenir ab points.

Exemple :

hauteurs des piles										score
10										
5	5									25 points
5	3	2								6
4	3	2	1							4
2	3	2	1	2						4
2	2	2	1	2	1					2
1	2	2	1	2	1	1				1
1	1	2	1	2	1	1	1			1
1	1	1	1	2	1	1	1	1		1
1	1	1	1	1	1	1	1	1	1	1
									score total =	45 points

Est-il possible de trouver une meilleure stratégie ?

Théorème : Toute manière de dépiler n blocs conduit à un score de $n(n-1)/2$ points.

Démonstration :

- La démonstration fonctionne par induction forte.
- Soit $P(n) =$ "Toute manière de dépiler n blocs conduit à un score de $n(n-1)/2$ points".
- *Cas de base :* $P(1)$ est vrai car une pile de 1 bloc est déjà dépilée. Le score est donc de $0 = 1(1-1)/2$.
- *Cas inductif :* Supposons que $P(1)$, $P(2)$, ..., $P(n)$ soient vrais, avec $n \geq 1$, et supposons que nous disposions d'une pile de $n+1$ blocs.
  - Premier mouvement : divise la pile initiale en deux piles de tailles k et $n+1-k$, avec $1 \leq k < n+1$.

- On obtient :

$$\begin{aligned}
\text{s. total} &= \text{score du premier mouvement} \\
&\quad + \text{score du dépliage de } k \text{ blocs} \\
&\quad + \text{score du dépliage de } n+1-k \text{ blocs} \\
&= k(n+1-k) + \frac{k(k-1)}{2} + \frac{(n+1-k)(n-k)}{2} \\
&= \frac{2kn + 2k - 2k^2 + k^2 - k + n^2 - kn + n - k - kn + k^2}{2} \\
&= \frac{(n+1)n}{2}
\end{aligned}$$

- La conjonction $P(1) \wedge P(2) \wedge \cdots \wedge P(n)$ implique donc $P(n+1)$ quel que soit $n \geq 1$.
- Par induction forte, on a donc $P(n)$ pour tout $n \geq 1$. □

Factorisation en nombres premiers

Redémontrons le théorème suivant par induction forte :

Théorème : Tout nombre entier positif plus grand que 1 peut s'écrire comme un produit de nombres premiers

Démonstration :

- La démonstration fonctionne par induction forte.
- $P(n) =$ "n s'écrit comme un produit de nombres premiers".
- *Cas de base :* $P(1)$ est vrai car il s'écrit comme le produit d'un ensemble vide de nombres premiers.
- *Cas inductif :* Supposons $P(1) \wedge P(2) \wedge \cdots \wedge P(n)$.
  - Si $n+1$ est premier, $P(n+1)$ est vrai.
  - Sinon, $n+1 = ab$, avec $2 \leq a, b \leq n$.
  - Par induction, a et b sont des produits de nombres premiers. Donc, $P(n+1)$ est vrai.
- Par induction, $P(n)$ est vrai pour tout $n \in \mathbb{N}_0$. □

Remarques :

- Tout théorème qui peut être démontré par induction forte peut aussi être démontré par induction simple.
  - Supposons que $P(0)$ et $\forall n : (\forall k \leq n : P(k)) \Rightarrow P(n+1)$ soient vraies.
  - On peut montrer par induction simple que

$$Q(n) = "\forall 0 \leq k \leq n : P(k)"$$

    est vraie pour tout n.
  - Et donc en déduire que $P(n)$ est vraie pour tout n.
- Utiliser l'induction forte rend parfois les preuves plus simples.
- Cependant, si $P(n)$ permet de démontrer facilement que $P(n+1)$ est vrai, alors, par soucis de simplicité, il est préférable d'utiliser l'induction simple.

Equivalence entre bon ordre et induction

Tout ce qui peut se démontrer par induction peut également se démontrer par le principe du bon ordre.

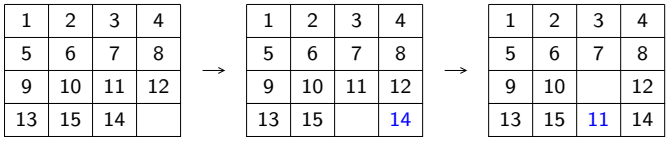
Si on suppose qu'un de ces principes est un axiome, on peut en déduire l'autre.

Théorème : Soit un prédicat $P(n)$. Si $P(0)$ est vraie et $P(n) \Rightarrow P(n+1)$ est vraie pour tout $n \geq 0$, alors $P(n)$ est vraie pour tout $n \geq 0$.
Démonstration :

- Supposons $P(0)$ et $\forall n \geq 0 : P(n) \Rightarrow P(n+1)$ vraies.
- Soit l'ensemble $C = \{n | P(n) \text{ fausse}\}$.
- Si le théorème est faux, C est non vide et par le principe du bon ordre, il contient un plus petit élément. Soit m cet élément.
- Puisque $P(0)$ est vraie, on a $m > 0$ et donc $m - 1 \geq 0$.
- $P(m-1) \Rightarrow P(m)$ est vraie et donc $P(m)$ doit être vraie, ce qui contredit le choix de m.
- $P(n)$ est donc vraie pour tout $n \geq 0$. □

Plan

L'énigme du Taquin (Sam Lloyd, ±1870)

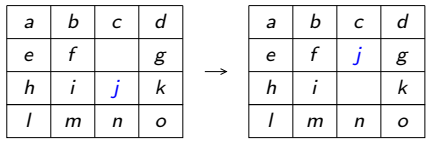


Existe-t-il une séquence de mouvements qui permet d'échanger les pièces 15 et 14 de la configuration de gauche, sans modifier l'emplacement des autres pièces ?

Principe de la démonstration :

- Nous allons établir un propriété de la grille qui est toujours vraie, quelle que soit la façon dont les pièces sont déplacées.
- Nous montrerons ensuite que la grille recherchée viole cette propriété et n'est donc pas atteignable

- Deux types de mouvements : mouvement de ligne et mouvement de colonne.
- Lemme 1 : Un mouvement de ligne ne modifie pas l'ordre des pièces.
  Démonstration : C'est immédiat. □
- Lemme 2 : Un mouvement de colonne modifie l'ordre relatif d'exactement 3 paires de pièces.



  Démonstration : Faire glisser une pièce vers le bas la déplace après les 3 pièces suivantes. Faire glisser une pièce vers le haut la déplace avant les 3 pièces précédentes. □

- Lemme 3 : Un mouvement de ligne ne modifie jamais la parité du nombre d'inversions. Un mouvement de colonne modifie toujours la parité du nombre d'inversions.

  Démonstration : Par le lemme 1, un mouvement de ligne ne modifie pas l'ordre des pièces. En particulier, il ne modifie pas le nombre d'inversions.

  Par le lemme 2, un mouvement de colonne modifie l'ordre relatif d'exactement 3 paires de pièces. Donc, un nombre pair d'inversions devient impair, et vice-versa. □

- Lemme 4 : Dans toute configuration accessible à partir de la configuration ci-dessous, la parité du nombre d'inversions est différente de la parité du numéro de la ligne contenant la case vide.

ligne 1	1	2	3	4
ligne 2	5	6	7	8
ligne 3	9	10	11	12
ligne 4	13	15	14	

Démonstration :

- La démonstration fonctionne par induction.
- Soit $P(n)$ = "Après n mouvements, la parité du nombre d'inversions est différente de la parité du numéro de la ligne contenant la case vide".
- *Cas de base :* $P(0)$ est vrai, car, initialement, le nombre d'inversions vaut 1, tandis que le numéro de la ligne contenant la case vide vaut 4.

- *Cas inductif :* Supposons que $P(n)$ soit vrai pour un $n \in \mathbb{N}$.
  - Si le mouvement $n+1$ est un mouvement de ligne, alors $P(n+1)$ est vrai car, la ligne contenant la case vide n'a pas changé, et par le lemme 3 la parité du nombre d'inversions n'est pas modifiée.
  - Si le mouvement $n+1$ est un mouvement de colonne, alors, par le lemme 3, la parité du nombre total d'inversions a été modifiée. De plus, la parité du numéro de la ligne contenant la case vide a été modifiée également. Donc, $P(n+1)$ est vrai.
- Dès lors, $P(n)$ implique $P(n+1)$ pour tout $n \in \mathbb{N}$.
- Par induction, $P(n)$ est vrai pour tout $n \in \mathbb{N}$. □

- Théorème : Aucune séquence de mouvements ne permet d'obtenir la configuration de droite à partir de la configuration de gauche :

1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Démonstration : Dans la configuration de droite, le nombre total d'inversions est de 0, tandis que la case vide est dans la ligne 4. Par le lemme 4, cette configuration n'est pas accessible. □

Chapitre 2

Machines d'état et correction de programmes

Plan

Lectures conseillées :

- MCS : 5.4, 7.2.
- Leslie Lamport. Computation and state machines. 2008 `http://research.microsoft.com/en-us/um/people/lamport/pubs/state-machine.pdf`

Principe d'invariant

La démonstration du taquin est basée sur le principe d'invariant.

Idée générale :

- On étudie un processus qui évolue étape par étape
- On souhaite montrer qu'une propriété est vérifiée à chaque étape du processus. On appelle ce type de propriété un invariant

La vérification de l'invariant se fait par induction sur le nombre d'étapes du processus.

Un processus qui évolue étape par étape peut se modéliser par une machine d'état.

Machine d'état

Un machine d'état est un modèle abstrait d'un processus évoluant pas à pas.

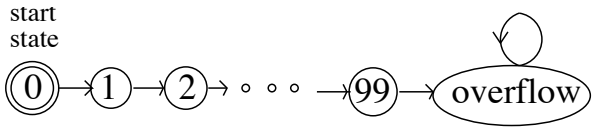
Définition : Une *machine d'état* est un triplet $(\mathcal{Q}, \mathcal{Q}_0, \delta)$ où :

- $\mathcal{Q}$ est un ensemble (non vide) d'états (potentiellement infini),
- $\mathcal{Q}_0 \subseteq \mathcal{Q}$ est l'ensemble (non vide) des états initiaux,
- $\delta \subseteq \mathcal{Q} \times \mathcal{Q}$ est l'ensemble des transitions permises.

Une machine d'état peut être interprétée comme un graphe dirigé dont les sommets sont les états et les arêtes (dirigées) sont les transitions.

Dans la suite, on notera $q \to r$ une transition (q, r).

Exemples



Un compteur :

- $\mathcal{Q} = \{0, 1, \ldots, 99, \mathit{overflow}\}$
- $\mathcal{Q}_0 = \{0\}$
- $\delta = \{n \to n+1 | 0 \leq n < 99\} \bigcup \{99 \to \mathit{overflow}, \mathit{overflow} \to \mathit{overflow}\}$

Le taquin :

- $\mathcal{Q}$ =l'ensemble des configurations possibles du taquin, $|\mathcal{Q}| = 16!$
- $\mathcal{Q}_0 = \{$la configuration initiale$\}$
- $\delta = \{\langle$config1, config2$\rangle |$ config2 peut s'obtenir à partir de config1 en un mouvement$\}$

Exécution et déterminisme

Définition : Une *exécution* ou *trajectoire* d'une machine d'état est une séquence (potentiellement infinie) d'états $s_0, s_1, s_2 \ldots$ telle que :

- $s_0 \in \mathcal{Q}_0$
- $(s_i, s_{i+1}) \in \delta, \forall i \geq 0$

Définition : Une machine d'état est *déterministe* si et seulement si il n'y a qu'un état initial ($|\mathcal{Q}_0| = 1$) et la relation δ est une fonction, c'est-à-dire si pour tout état $s \in \mathcal{Q}$, il y a au plus un état $t \in \mathcal{Q}$ tel que $(s, t) \in \delta$.

Exemples :

- Le compteur est déterministe. Une seule exécution possible :

$$0, 1, 2, \ldots, 99, \textit{overflow}, \textit{overflow} \ldots$$

- Le taquin ne l'est pas (plusieurs mouvements possibles par état).

Atteignabilité et invariant

Définition : Un état r est *atteignable* s'il est contenu dans une exécution de longueur finie.

Définition : Un *invariant* est un prédicat $P(s)$ défini sur $\mathcal{Q}$ qui est vrai pour tous les états *atteignables*.

Exemple : le prédicat "la parité du nombre d'inversions est différente de la parité du numéro de la ligne contenant la case vide" est un invariant pour le taquin.

Démonstration d'un invariant

Théorème : Soit une machine d'état $\mathcal{M} = (\mathcal{Q}, \mathcal{Q}_0, \delta)$. Si P est un prédicat sur $\mathcal{Q}$ tel que :

- $P(s)$ est vrai pour tout $s \in \mathcal{Q}_0$
- et $P(s) \Rightarrow P(s')$ est vrai pour tout $(s, s') \in \delta$,

alors P est un invariant pour $\mathcal{M}$.

Démonstration :

- La preuve se fait par induction sur la longueur d'exécution.
- Soit $Q(n)$ le prédicat suivant : [1]

  "$P(s)$ est vrai pour tout s contenu dans une exécution de longueur n".

- Cas de base : $Q(1)$ est vrai puisque toute exécution de longueur 1 contient seulement un état initial et $P(s)$ est vrai pour tout $s \in \mathcal{Q}_0$.

1. Ne pas confondre P et Q !

- Cas inductif :
  - Supposons $Q(n)$ vrai et montrons que $Q(n+1)$ est vrai également.
  - Soit r' un état contenu dans une exécution de longueur $n+1$. Il suffit de montrer que $P(r')$ est vrai.
  - Si r' est le premier état de l'exécution, alors $P(r')$ est vérifié.
  - Sinon, soit r l'état précédant r' dans l'exécution. r est contenu dans une exécution de longueur n et donc, par hypothèse inductive, $P(r)$ est vrai. Puisque $(r, r') \in \delta$, $P(r)$ implique que $P(r')$ est vrai.
- $P(s)$ est donc vrai pour tout état s contenu dans une exécution de longueur finie, ce qui correspond aux états atteignables de la machine.
- P est donc un invariant pour $\mathcal{M}$. □

Invariant inductif

- Définition : Un invariant qui peut se démontrer par le théorème précédent s'appelle un *invariant inductif*.
- Tous les invariants ne peuvent pas se démontrer de cette manière.

- Pour démontrer un invariant $P(s)$, il suffit de trouver un invariant (inductif) $Q(s)$ tel que :
  - $Q(s)$ est vrai pour tout $s \in \mathcal{Q}_0$
  - $Q(s) \Rightarrow Q(s')$ est vrai pour tout $(s, s') \in \delta$,
  - $Q(s) \Rightarrow P(s)$ est vrai pour tout état s atteignable

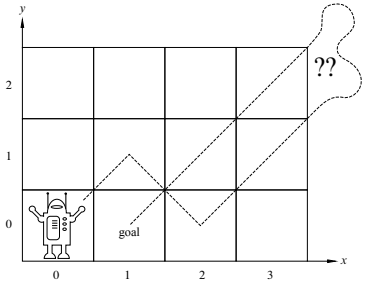
- Exemple : Dans l'exemple du taquin :
  - $P(n)$ ="Après n mouvements, la grille cible n'est pas atteinte" est un invariant mais pas un invariant inductif ($P(n+1)$ ne peut pas se déduire de $P(n)$).
  - $Q(n)$ ="Après n mouvements, la parité du nombre d'inversions est différente de la parité du numéro de la ligne contenant la case vide" est un invariant inductif (tel que pour tout n, $Q(n) \Rightarrow P(n)$).

Application

Soit un robot se déplaçant sur une grille entière à deux dimensions :

- L'état du robot est spécifié par les coordonnées entières (x, y) représentant sa position courante sur la grille. $\mathcal{Q} = \mathbb{Z} \times \mathbb{Z}$
- Le robot se trouve initialement à l'origine $\mathcal{Q}_0 = \{(0, 0)\}$
- A chaque pas, le robot peut se déplacer uniquement en diagonale $\delta = \{(m, n) \rightarrow (m \pm 1, n \pm 1) | (m, n) \in \mathcal{Q}\}$

$\Rightarrow$ On cherche à déterminer si le robot peut atteindre l'état $(1, 0)$.



Théorème : La somme des coordonnées de tout état atteignable par le robot est pair.
Démonstration :

- Soit le prédicat $Q((m, n))$ ="$m + n$ est pair". Montrons que $Q((m, n))$ est un invariant inductif.
- Cas de base : $Q((0, 0))$ est vrai.
- Cas inductif :
  - Nous devons montrer que $Q((m, n)) \Rightarrow Q((m', n'))$ est vrai pour toute transition $(m, n) \rightarrow (m', n') \in \delta$.
  - Toute transition étant de la forme $(m, n) \rightarrow (m \pm 1, n \pm 1)$, la somme des coordonnées est augmentée de 0, 2 ou -2 à chaque transition.
  - Ajouter 0, 2 ou -2 à un nombre pair donne un nombre pair.
- Par le théorème d'invariant, on peut conclure que P est un invariant, ce qui démontre le théorème. □

Corrolaire : Le robot n'atteindra jamais la position $(1, 0)$.
Démonstration : C'est une conséquence directe du théorème puisque $1 + 0$ est impair. □

Remarque : L'invariant qu'on veut montrer est $P((m, n))$ ="$(m, n) \neq (1, 0)$" qui n'est pas inductif. On montre que c'est une conséquence de l'invariant inductif $Q((m, n))$ ="$m + n$ est pair".

Exercice : Réécrivez la démonstration du problème du taquin en vous appuyant sur le théorème d'invariant.

Le problème des cruches

Soient deux cruches non graduées et initialement vides, de contenances respectives 3 et 6 litres.

Seules les trois actions suivantes sont possibles :

- remplissage d'une cruche via la fontaine,
- vidage d'une cruche dans la fontaine,
- transvasement d'une cruche vers l'autre jusqu'à ce que l'une soit remplie ou que l'autre soit vide.

Question : Est-il possible d'obtenir exactement 4 litres dans la grande cruche ?

Modélisation par une machine d'état

- Un état est défini par la quantité d'eau dans chacune des cruches

$$\mathcal{Q} = \{(a, b) | 0 \leq a \leq 3, 0 \leq b \leq 6, a, b \in \mathbb{R}\}$$

- Initialement, les cruches sont vides

$$\mathcal{Q}_0 = \{(0, 0)\}$$

- L'ensemble des transitions est l'union de
  - $(a, b) \to (3, b)$ (remplissage petite cruche)
  - $(a, b) \to (a, 6)$ (remplissage grande cruche)
  - $(a, b) \to (0, b)$ (vidage petite cruche)
  - $(a, b) \to (a, 0)$ (vidage grande cruche)
  - $(a, b) \to (\max(0, a - (6 - b)), \min(a + b, 6))$ (transvasement petite vers grande cruche)
  - $(a, b) \to (\min(a + b, 3), \max(0, b - (3 - a)))$ (transvasement grande vers petite cruche)

(cette machine est non déterministe)

Réponse à l'énigme

Théorème : Il n'est pas possible de remplir une des deux cruches avec exactement 4 litres

Démonstration :

- Soit le prédicat $P((a, b)) =$" a et b sont des multiples de 3".
- Il suffit de montrer que $P((a, b))$ est un invariant pour la machine d'état :
  - $P((0, 0))$ est vrai
  - "$P((a, b)) \Rightarrow P((c, d))$ pour tout $(a, b) \to (c, d) \in \delta$" se montre en traitant au cas par cas les 6 types de transition

    (Laissé comme exercice)
- 4 n'étant pas un multiple de 3, le théorème est vérifié. □

Programme informatique et machine d'état

- Un programme informatique (itératif) peut être modélisé par une machine d'état (déterministe)
- Inversément, un programme informatique peut être vu comme une manière de décrire une machine d'état.

- L'état de la machine est défini par les valeurs des variables utilisées par le programme [2].
- Les transitions sont définies par les instructions du programme.

2. plus, si nécessaire, certaines variables "cachées" telles que le numéro de l'instruction, le contenu de la pile d'appel des fonctions, etc.

Exemple

Soit le programme suivant (en notation pseudo-code) :

```
1  x = a ; y = b
2  while x ≠ y
3      if x > y
4          x = x − y
5      elseif x < y
6          y = y − x
```

La machine d'état $(\mathcal{Q}, \mathcal{Q}_o, \delta)$ correspondante est définie par :

- $\mathcal{Q} = \{(x, y) | x, y \in \mathbb{Z}\}$
- $\mathcal{Q}_0 = \{(a, b)\}$
- $\delta =$

$$(x, y) \rightarrow \begin{cases} (x - y, y) & \text{si } x > y \\ (x, y - x) & \text{si } x < y \end{cases}$$

Etats terminaux

Définition : Un état d'une machine d'état est *terminal* s'il n'existe aucune transition partant de cet état.

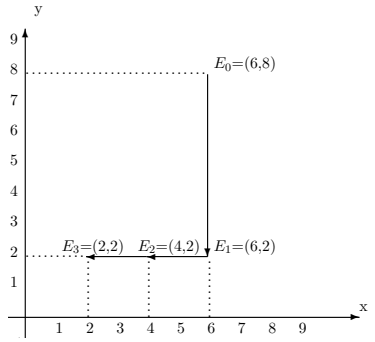
Dans le contexte de l'étude de programmes informatiques, on n'étudiera que deux types d'exécutions :

- Les exécutions de longueur infinie
- Les exécutions de longueur finie dont le dernier état est un état terminal

Les premières correspondent à des boucles infinies, les secondes à des programmes qui se terminent.

La nature de l'exécution varie en général en fonction de l'état initial.

Exemples d'exécution de la machine du transparent 93



(De Marneffe)

- En partant de $(6, 8)$, la machine finit par s'arrêter en $(2, 2)$ (plus de transition possible)
- En partant de $(-4, -2)$, l'exécution (infinie) est $(-4, -2)$, $(-4, 2)$, $(-4, 6)$, $(-4, 10)$,. . ..
- En partant de $(4, 0)$, la machine reste dans cet état et ne s'arrête jamais.

Précondition et postcondition

Parmi les exécutions de la machine qui se terminent, seules certaines sont intéressantes.

Elles sont définies par deux prédicats :

- **Postcondition :** prédicat qui doit être vérifié par les états terminaux solutions du problème posé.
  - Exemple : $Post((x, y))$ ="x est le plus grand commun diviseur de a et b"
- **Précondition :** définit un ensemble d'états[3] qui, étant les états initiaux de la machine, donneront des exécutions finies dont les états terminaux vérifient la postcondition.
  - Exemple : $Pre((x, y))$ ="$x > 0$ et $y > 0$".

Remarque : on appelle *précondition la plus faible* le prédicat qui définit l'ensemble de *tous* les états initiaux qui mèneront à un état terminal vérifiant la postcondition.

3. ceux qui vérifient le prédicat

Correction de programmes

Un programme est *totalement correct* s'il vérifie deux propriétés :

- **Correction partielle :** Si le programme atteint un état terminal à partir d'un état initial vérifiant la précondition, alors cet état satisfait la postcondition.
- **Terminaison :** Le programme atteint toujours un état terminal à partir d'un état initial vérifiant la précondition

L'un n'implique pas l'autre. Le programme suivant est seulement partiellement correct :

```
1  while x ≠ y
2        y = y + 1
```

$Pre((x, y)) =$ "$x, y \in \mathbb{Z}$"

$Post((x, y)) =$ "$x = y$"

La correction partielle se montre généralement par le principe d'invariant. La terminaison se montre par le principe du bon ordre.

Exponentiation

```
1  x = a; y = 1; z = b
2  while z ≠ 0
3        r = REMAINDER(z, 2)
4        z = QUOTIENT(z, 2)
5        if r = 1
6              y = xy
7        x = x^2
```

$Pre((x, y, z)) =$ "$x \in \mathbb{R}, z \in \mathbb{Z}$"

$Post((x, y, z)) =$ "$y = a^b$"

(REMAINDER$(q, r) = q \bmod r$ et QUOTIENT$(q, r) = \lfloor \frac{q}{r} \rfloor$)

Machine d'état :

- $\mathcal{Q} = \mathbb{R} \times \mathbb{R} \times \mathbb{N}$
- $\mathcal{Q}_0 = \{(a, 1, b)\}$
- $\delta =$

$$(x, y, z) \to \begin{cases} (x^2, y, \text{QUOTIENT}(z, 2)) & \text{si } z \neq 0 \text{ et z pair} \\ (x^2, xy, \text{QUOTIENT}(z, 2)) & \text{si } z \neq 0 \text{ et z impair} \end{cases}$$

Exponentiation : correction partielle

Théorème : Si l'algorithme se termine, on a $y = a^b$ pour $a \in \mathbb{R}$ et $b \in \mathbb{Z}$.
Démonstration :
Soit le prédicat :

$$P((x, y, z)) = \text{“}z \in \mathbb{N} \text{ et } yx^z = a^b\text{”}.$$

Montrons que P est un invariant inductif pour la machine d'état.

Cas de base : $P((a, 1, b))$ est vérifié puisque $1 \cdot a^b = a^b$.

Cas inductif : Soit une transition $(x, y, z) \to (x', y', z') \in \delta$. Supposons que $P((x, y, z))$ est vérifié et montrons que $P((x', y', z'))$ est vérifié, c'est-à-dire

$$z' \in \mathbb{N} \text{ et } y'x'^{z'} = a^b.$$

Puisqu'il y a une transition depuis (x, y, z), on a $z \neq 0$ et puisque $z \in \mathbb{Z}$, on peut considérer deux cas :

- Cas 1 : z est pair. On a alors $x' = x^2, y' = y, z' = z/2$. Donc, $z' \in \mathbb{Z}$ et
$$y'x'^{z'} = y(x^2)^{z/2} = yx^{2z/2} = yx^z = a^b$$
(car $P((x, y, z))$ est vérifié).
- Cas 2 : z est impair. On a alors $x' = x^2, y' = xy, z' = (z - 1)/2$. Donc, $z' \in \mathbb{Z}$ et
$$y'x'^{z'} = xy(x^2)^{(z-1)/2} = yx^{1+2(z-1)/2} = yx^z = a^b.$$

Par le théorème d'invariant, on conclut que P est vérifié pour tout état atteignable.

Etant donné le gardien de la boucle, les seuls états terminaux atteignables sont ceux pour lesquels $z = 0$. Donc, si un état terminal est atteint, par l'invariant, on a $y = yx^0 = a^b$. □

Interprétation

La preuve précédente suit le schéma de démonstration du transp. 85 :

- On veut montrer que le prédicat :

$$\begin{aligned} Q((x,y,z)) &= \text{"}(x,y,z) \text{ terminal} \Rightarrow Post((x,y,z))\text{"} \\ &= \text{"}z = 0 \Rightarrow y = a^b \text{"} \end{aligned}$$

  est un invariant pour la machine d'état
- Q n'étant pas un invariant inductif, on passe par un invariant inductif :

$$P((x,y,z)) = \text{“}z \in \mathbb{N} \text{ et } yx^z = a^b\text{”},$$

  qui est tel que pour tout état atteignable $(x,y,z) \in \mathcal{Q}$:

$$P((x,y,z)) \Rightarrow Q((x,y,z))$$

Exponentiation : terminaison

Théorème : L'algorithme d'exponentiation se termine toujours en un état où $z = 0$.
Démonstration (pédantique) :

- Après chaque transition, z devient strictement plus petit (par l'instruction $z = \text{QUOTIENT}(z, 2)$).
- Or par l'invariant, z est un entier naturel. Soit $S \subseteq \mathbb{N}$ l'ensemble des valeurs de z aux états atteignables par la machine.
- Par le principe du bon ordre, S possède une valeur minimale z_0.
- Vu la décroissance stricte de z, la machine finira toujours par atteindre cette valeur et s'arrêtera (par l'absurde, si ce n'était pas le cas, une transition à partir de cet état diminuerait encore z et donc z_0 ne serait pas le minimum).
- La seule manière de s'arrêter est d'avoir $z = 0$. On a donc $z_0 = 0$. □

Exercices

Montrez par induction forte que le nombre de transition de la machine d'état de l'algorithme d'exponentiation s'arrêtera après $\lceil \log_2 n \rceil + 1$ transitions à partir d'un état où $z = n \neq 0 \in \mathbb{N}$.

Montrer que l'algorithme du transparent 93 est totalement correct pour les précondition et postcondition suivantes :

$$Pre((x,y)) = \text{"}x > 0 \text{ et } y > 0\text{"}$$

$$Post((x,y)) = \text{"}x \text{ est le plus grand commun diviseur de } a \text{ et } b\text{"}$$

Variables dérivées

Définition : Une *variable dérivée* pour une machine $\mathcal{M} = (\mathcal{Q}, \mathcal{Q}_O, \delta)$ est une fonction $f : \mathcal{Q} \to S$ où S est un ensemble quelconque (typiquement $\mathbb{N}$).

Une variable dérivée est aussi appelée un variant ou une fonction potentielle.

Exemple : La coordonnée x du robot ou le nombre d'inversions du taquin,

Définition : Une variable dérivée $f : \mathcal{Q} \to \mathbb{R}$ est *strictement décroissante* si et seulement si $q \to q' \in \delta$ implique $f(q') < f(q)$. Elle est *faiblement décroissante* si et seulement si $q \to q' \in \delta$ implique $f(q') \leq f(q)$.

Variables dérivées et terminaison

Théorème : Si $f : \mathcal{Q} \to \mathbb{N}$ est une variable dérivée strictement décroissante pour une machine d'état, alors le nombre de transitions d'une exécution démarrant dans un état q_0 est au plus $f(q_0)$.
Démonstration : Par contradiction. Supposons qu'il existe une exécution $q_0, q_1, \ldots, q_t, \ldots$ où $t > f(q_0)$. Alors $f(q_t) \leq f(q_0) - t < 0$, puisque la valeur de f décroît d'au moins 1 à chaque transition, ce qui contredit que $f(q) \in \mathbb{N}$. □

Remarques :

- Une fonction faiblement décroissante n'assure évidement pas la terminaison.
- La fonction f est parfois appelée *fonction de terminaison* pour la machine d'état.
- Le théorème peut se généraliser à des variables dérivées prenant leur valeur dans un ensemble bien ordonné.

Un exemple plus compliqué

Soit la machine d'état suivante :

- $\mathcal{Q} \in \mathbb{N} \times \mathbb{N}$
- $\mathcal{Q}_0 = \{(a, b)\}$ avec $a, b \in \mathbb{N}$
- $\delta = \{(x, y) \to (x - 1, y) | \forall x, y \in \mathbb{N} : x > 0\} \bigcup \{(x, y) \to (z, y - 1) | \forall x, y, z \in \mathbb{N} : y > 0, z \geq x\}$

qui modélise par exemple les déplacements d'un robot dans un espace discret à deux dimensions.

Théorème : Quel que soit son état initial, la machine finira toujours par atteindre l'état $(0, 0)$.

Démonstration :

- Soit la variable dérivée $v : \mathcal{Q} \to \mathbb{N} + To1$:

$$v((x, y)) = y + \frac{x}{x + 1}.$$

- Chaque transition $(x, y) \to (x', y')$ est telle que $v((x', y')) < v((x, y))$ et donc v est une variable strictement décroissante.
- L'ensemble $\mathbb{N} + To1$ étant bien ordonné (voir transp. 43), v finira par atteindre son minimum.
- Le seul état sans transition étant $(0, 0)$, la machine ne peut s'arrêter que dans cet état. □

Vérification formelle de programmes

- La méthode d'invariant et le principe du bon ordre permettent :
  - de prouver formellement la correction et la terminaison de programmes
  - de vérifier que certaines propriétés sont vérifiées tout au long de l'exécution d'un programme
- Cette vérification est cruciale dans de nombreuses applications critiques de l'informatique (domaines médical, spatial, sécurité, etc.)

- Idéalement, on aimerait que ces vérifications formelles puissent se faire de manière automatique
- Des outils d'aide à la vérification existent mais ces outils ne peuvent automatiquement :
  - inventer les invariants de boucle
  - prouver un invariant donné dans les cas non triviaux
  - prouver dans tous les cas la terminaison des programmes

Invariant difficile à trouver

Que fait le programme suivant ?

```
ALGO(A)
1  i = 1
2  j = A.length
3  while i < j
4       if A[i] > A[j]
5            SWAP(A[i], A[i + 1])
6            SWAP(A[i], A[j])
7            i = i + 1
8       else j = j − 1
```


Terminaison difficile à prouver

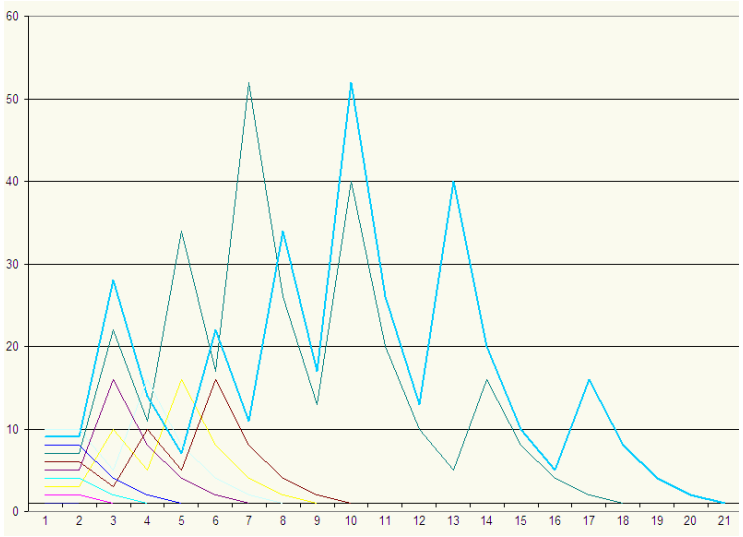
Conjecture de Syracuse : Le programme suivant :

```
ALGO(n)
1  while n ≠ 1
2       if n pair
3            n = n/2
4       else n = 3n + 1
```

se termine pour tout $n \in \mathbb{N}^+$.

On a montré expérimentalement que la conjecture était vraie pour tout $n < 1,25 \cdot 2^{62}$ mais à ce jour elle n'a toujours pas été prouvée.

Exemples d'exécutions



Source : `animalerienumerique.blogspot.be`

Problème de l'arrêt

En théorie de la calculabilité, le problème de l'arrêt consiste à déterminer si un programme informatique se termine ou non.

Ce problème n'est pas décidable : on peut montrer qu'il n'est pas possible d'écrire un programme informatique prenant comme entrée un programme quelconque et renvoyant VRAI si le programme s'arrête, FAUX sinon.

On en donnera ici qu'un preuve informelle. Voir le cours "Introduction à la calculabilité" pour une preuve plus rigoureuse.

Problème de l'arrêt

Théorème : Il n'existe pas d'algorithme CheckHalt qui prend en entrée un programme P et des données D et renvoie "halts" si P s'arrête après un nombre fini d'étapes quand il est appliqué aux données D, "loops forever" sinon.

Démonstration :

- Par contradiction. Supposons qu'un tel algorithme CheckHalt existe.
- Un programme P peut lui-même être considéré comme une entrée (une chaîne de caractères). On peut donc appliquer CheckHalt sur l'entrée (P, P).
- Soit l'algorithme Test prenant en entrée un programme P et défini comme suit :

```
Test(P)
1  if CheckHalt(P, P) == "halts"
2        "exécuter une boucle infinie"
3  else
4        return
```


- Appliquons maintenant Test à lui-même. Deux résultats possibles :
  - Test(Test) se termine : la valeur de CheckHalt(Test, Test) est alors "halts" et donc Test(Test) boucle à l'infini.
  - Test(Test) boucle à l'infini : CheckHalt(Test, Test) renvoie "loops forever" et donc Test(Test) se termine.
- Cette analyse montre que Test(Test) boucle et aussi se termine.
- Cette contradiction montre qu'il n'est pas possible d'écrire une fonction CheckHalt. □

Résumé

- Une machine d'état permet de modéliser un processus calculatoire.
- La technique d'invariant inductif permet de prouver des propriétés d'une machine d'état.
- Un programme informatique peut être modélisé par une machine d'état.
- La correction partielle se prouve par invariant et la terminaison par le principe du bon ordre.

- Les machine d'état sont très utilisées en informatique (compilateur, théorie de la calculabilité, protocoles réseaux, processus de décision etc.).
- Beaucoup de variantes existent (automates, machine de turing, probabilistes etc.).

Chapitre 3

Définitions récursives et induction structurelle

Plan

Lectures conseillées :

- MCS : chapitre 6

Définition récursive

Une *définition récursive* explique comment construire un élément d'un type donné à partir d'élément du même type plus simple.

Un *type de données récursif* R est défini par :

- des règles de base qui affirment que des éléments appartiennent à R
- des règles inductives de construction de nouveaux éléments de R à partir de ceux déjà construits
- Une règle (souvent implicite) qui dit que seuls les éléments obtenus par les deux règles précédentes sont dans R.

Exemples : Soit l'ensemble E des entiers pairs. E peut être défini comme suit :

- Cas de base : $0 \in E$,
- Cas récursifs :
  1. si $n \in E$, alors $n+2 \in E$,
  2. si $n \in E$, alors $-n \in E$.

Chaîne de caractères

Définition : Soit A un ensemble non vide appelé *alphabet*, dont les éléments sont appelés des caractères, lettres ou symboles. L'ensemble A^* des chaînes (*string*) sur l'alphabet A est défini récursivement comme suit :

- Cas de base : La chaîne vide, λ, est dans A^*.
- Cas récursif : Si a $\in A$ et $x \in A^*$, alors a$x \in A^*$.
  (où ax désigne la chaîne x au début de laquelle on a ajouté le caractère a).

Exemple : Si $A = \{0, 1\}$, λ, 1, 1011 et 00101 appartiennent à A^*.

Définition sur un ensemble défini récursivement

Des fonctions ou opérations peuvent être définies, récursivement, sur les éléments d'un ensemble défini de manière récursive.

Définition : La *longeur* $|x|$ d'une chaîne est *définie récursivement* sur base de la définition de $x \in A^*$:

- Cas de base : $|\lambda| ::= 0$.[4]
- Cas récursif : Si $x \in A^*$ et a $\in A$, $|$a$x| ::= 1 + |x|$.

Exemple : $|1011| = 1 + |011| = 2 + |11| = 3 + |1| = 4 + |\lambda| = 4$

Définition : Soit deux chaînes x et $y \in A^*$, la concaténation $x \cdot y$ de x et y est définie récursivement sur base de la définition de $x \in A^*$:

- Cas de base : $\lambda \cdot y ::= y$
- Cas récursif : Si $x \in A^*$ et a $\in A$, a$x \cdot y ::=$ a$(x \cdot y)$.

On notera dans la suite $x \cdot y$ par xy.
Exemple : $01 \cdot 010 = 0(1 \cdot 010) = 0(1(\lambda \cdot 010)) = 0(1(010)) = 01010$.

4. ::= signifie "égal par définition".

Preuve sur un ensemble défini récursivement

Définition : Soit $A = \{0, 1\}$ et soit $S \subset A^*$ l'ensemble défini récursivement comme suit :

- Cas de base : $\lambda \in S$, où λ est la chaîne vide
- Cas récursifs :
  1. Si $x \in S$, alors $0x1 \in S$,
  2. Si $x \in S$, alors $1x0 \in S$,
  3. Si $x, y \in S$, alors $xy \in S$, où xy désigne la concaténation de x et y.

Théorème : Tout élément s dans S contient le même nombre de 0 et de 1.

(Exercice : Montrez que toute chaîne $s \in A^*$ contenant le même nombre de 0 et de 1 appartient à S. En déduire que S est l'ensemble de toutes les chaînes de A^* contenant autant de 0 que de 1.)

Démonstration : Soit le prédicat $P(n)$ vrai si toute chaîne s de S obtenue après n applications des régles récursives contient un nombre identique de 0 et de 1. Montrons que $P(n)$ est vrai pour tout $n \in \mathbb{N}$ par *induction forte*.
Cas de base ($n = 0$) : Le seul élément qui peut être obtenu après 0 applications des règles récursives est la chaîne vide qui contient un nombre identique de 0 et 1.
Cas inductif ($n \geq 0$) : Supposons $P(0), \ldots, P(n)$ vérifiés et montrons que $P(n+1)$ l'est également. Soit s une séquence obtenue après $n+1$ applications des règles. Il y a 3 cas possibles :

- Cas 1 : la $n+1$ règle est la règle 1. On a $s = 0x1$, où x est obtenu après n applications des règles et contient donc le même nombre de 0 et de 1 (par hypothèse inductive). s contient donc également le même nombre de 0 et de 1 (un de plus que x).
- Cas 2 : la $n+1$ régle est la règle 2. Ce cas est symétrique au cas précédent.
- Cas 3 : la $n+1$ règle est la règle 3. $s = xy$ où x et y sont dans S et ont été obtenus par au plus n applications des règles. x et y contenant un nombre identique de 0 et de 1, c'est aussi le cas de $s = xy$. □

Induction structurelle

Le schéma de la démonstration précédente est applicable à tous les types de données définis récursivement. On peut se passer de l'induction sur le nombre d'applications des règles en invoquant le *principe d'induction structurelle*.

Principe d'induction structurelle : Soit un prédicat P défini sur un type de données R défini récursivement. Pour montrer que P est vrai pour tout élément de R, il suffit de montrer

- que P est vrai pour chaque élément de base, $b \in R$ et
- que P est vrai pour tout élément résultant de l'application d'une règle récursive, en supposant qu'il est vrai pour les éléments combinés par cette règle.

Le principe d'induction structurelle peut se montrer par induction forte sur le nombre d'applications des règles récursives.

Exemple

Pour montrer qu'un prédicat P est vrai pour tout élément de S (transp. 121), il faut montrer :

- que $P(\lambda)$ est vrai et
- que $P(0x1)$, $P(1x0)$ et $P(xy)$ sont vrais dès que $P(x)$ et $P(y)$ sont vrais.

Réécriture de la démonstration du transp. 122 :
Soit $P(s) =$"s contient autant de 0 que de 1". Montrons pas induction structurelle que $P(s)$ est vrai pour tout $s \in S$.

Cas de base : $P(\lambda)$ est vrai trivialement.
Cas inductifs : Si x et y contiennent autant de 1 que de 0, c'est aussi le cas de $1x0$, $0x1$ et xy.

Par le principe d'induction structurelle, on en conclut que $P(s)$ est vrai pour tout $s \in S$. □

Un exemple (un peu) plus compliqué

Définition : Soit un sous-ensemble de chaînes de caractères sur l'alphabet $\{M, I, U\}$ défini de manière récursive comme suit, et appelé le système MIU :

- Cas de base : MI appartient au système MIU
- Cas récursifs :
  1. Si xI est dans le système MIU, où x est une chaîne, alors xIU est dans le système MIU.
  2. Si Mx est dans le système MIU, où x est une chaîne, alors Mxx est le système MIU.
  3. Si xIIIy est dans le système MIU, où x et y sont deux chaînes (potentiellement vides), alors xUy est dans le système MIU.
  4. Si xUUy est dans le système MIU, où x et y sont deux chaînes (potentiellement vides), alors xUy est dans le système MIU.

MUIU appartient-il au système MIU ? Qu'en est-il de MU ?

Source : Douglas Hofstadter, *Gödel, Escher, Bach* (New York : Basics Books), 1979.

Appartenance

Théorème : MUIU appartient au système MIU.

Démonstration :
Montrons qu'on peut dériver MUIU du cas de base MI en utilisant les règles récursives :

- Par le cas de base, MI appartient au système MIU.
- En utilisant la règle 2, on en déduit que MII $\in$ au système MIU.
- En utilisant la règle 2, on en déduit que MIIII $\in$ au système MIU.
- En utilisant la règle 3, on en déduit que MUI $\in$ au système MIU.
- En utilisant la règle 1, on en déduit que MUIU $\in$ au système MIU.

□

Prédicat (invariant)

Théorème : Le nombre de I dans une chaîne s appartenant au système MIU n'est jamais un multiple de 3.
Démonstration : Soit $P(s)$ ="le nombre de I dans s n'est pas un multiple de 3". Montrons par induction structurelle que $P(s)$ est vrai pour toute chaîne du système MIU.
Cas de base : Le nombre de I dans MI n'est pas une multiple de 3.
Cas récursifs :

- Règle 1 : Par hypothèse inductive, on suppose que le nombre de I dans xI n'est pas un multiple de 3. Le nombre de I dans xIU n'est donc pas non plus un multiple de 3.
- Règle 2 : Par hypothèse inductive, on peut supposer que le nombre de I dans Mx est soit $3k+1$, soit $3k+2$ pour un k. Le nombre de I dans Mxx est donc soit $6k+2 = 3(2k)+2$, soit $6k+4 = 3(2k+1)+1$. Aucun des deux n'est donc un multiple de 3.

- Règle 3 : Par hypothèse inductive, on peut supposer que le nombre de I dans xIIIy est soit $3k+1$, soit $3k+2$ pour un k. Le nombre de I dans xUy est soit $3(k-1)+1$, soit $3(k-1)+2$, aucun n'étant un multiple de 3.
- Règle 4 : Par hypothèse inductive, on suppose que le nombre de I dans xUUy n'est pas un multiple de 3. Le nombre de I dans xUy n'est donc pas non plus un multiple de 3.

Par induction structurelle, on peut en conclure que $P(s)$ est vérifié pour tout s du système MIU. □

Corrolaire : MU n'appartient pas au système MIU.
Démonstration : C'est une conséquence directe du théorème précédent puisque MU contient zéro I qui est un multiple de 3. □

Modélisation par une machine d'état

Soit la machine d'état $\mathcal{M} = (\mathcal{Q}, \mathcal{Q}_0, \delta)$ définie comme suit :

- $\mathcal{Q} = A^*$ où $A = \{\mathsf{M}, \mathsf{I}, \mathsf{U}\}$,
- $\mathcal{Q} = \{\mathsf{MI}\}$,
-

$$\begin{aligned}\delta = & \{x\mathsf{I} \to x\mathsf{IU} | x \in A^*\} \\ \bigcup & \{\mathsf{M}x \to \mathsf{M}xx | x \in A^*\} \\ \bigcup & \{x\mathsf{III}y \to x\mathsf{U}y | x, y \in A^*\} \\ \bigcup & \{x\mathsf{UU}y \to x\mathsf{U}y | x, y \in A^*\}\end{aligned}$$

L'ensemble des états atteignables par $\mathcal{M}$ correspond exactement au système MIU.

Le prédicat $P(s)$ ="le nombre de I dans s n'est pas un multiple de 3" est un invariant inductif de la machine d'état. $Q(s)$ ="$s \neq$ MU" découle directement de $P(s)$.

Ambiguité d'une définition récursive

Pour prouver qu'un élément r appartient à un ensemble défini de manière récursive R, il suffit de trouver un ordre d'applications des règles à partir d'un ou plusieurs éléments de base qui permet de générer r.

Exemple : $0110 \in S$ car $\lambda \to 01$ (règle 1), $\lambda \to 10$ (règle 2), et $01, 10 \to 0110$ (règle 3).

S'il existe plusieurs dérivations d'un même élément, on dira que la définition est *ambiguë*.

Exemple : La définition de S est ambiguë. Par exemple, 1010 peut être dérivé de deux manières :

- $\lambda \to 10$ (règle 2), puis $10, 10 \to 1010$ (règle 3)
- $\lambda \to 01$ (règle 1), puis $01 \to 1010$ (règle 2)

Ambiguité d'une définition récursive

Une fonction définie sur base d'une définition ambiguë peut être mal définie.

Définition : Soit la fonction $f : S \to \mathbb{N}$ calculant le nombre d'applications de la règle 3 pour générer s :

- Cas de base : $f(\lambda) = 0$
- Cas récursifs :
  1. $f(0x1) = f(x)$
  2. $f(1x0) = f(x)$
  3. $f(xy) = 1 + f(x) + f(y)$

La définition de f mène à une contradiction :

$$\begin{aligned}f(1010) &= 1 + f(10) + f(10) = 1 + f(\lambda) + f(\lambda) = 1 \\ &= f(01) = f(\lambda) = 0\end{aligned}$$

(Exercice : trouver une définition récursive de S non ambiguë)

Plan

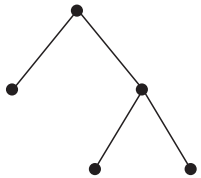
Arbres

On peut définir récursivement une structure d'arbre binaire entier.

Définition : L'ensemble $bTree$ des arbres binaires entiers est défini comme suit :

- Cas de base : La feuille **leaf** appartient à $bTree$.
- Cas récursif : Soit $T_1, T_2 \in bTree$, alors **branch**$(T1, T2) \in bTree$.

Exemple : **branch**(**leaf**, **branch**(**leaf**, **leaf**)) représente l'arbre ci-dessous.



Définition récursive sur un arbre

Définitions : La fonction leaves : $bTree \to \mathbb{N}$ compte le nombre de feuilles dans un arbre :

- Cas de base : leaves(**leaf**) ::= 1
- Cas récursif : leaves(**branch**(T_1, T_2)) ::= leaves(T_1) + leaves(T_2)

La fonction branches : $bTree \to \mathbb{N}$ compte le nombre de branches dans un arbre :

- Cas de base : branches(**leaf**) ::= 0
- Cas récursif :
  branches(**branch**(T_1, T_2)) ::= branches(T_1) + branches(T_2) + 1

Théorème : Pour tout arbre $T \in bTree$,

$$\text{leaves}(T) = \text{branches}(T) + 1.$$

Démonstration : Par induction structurelle sur T :

- Cas de base : Par définition, leaves(**leaf**) = 1 = 1 + branches(**leaf**).
- Cas récursif : Par définiton,

$$\text{leaves}(\textbf{branch}(T_1, T_2)) = \text{leaves}(T_1) + \text{leaves}(T_2).$$

Par hypothèse inductive, on a :

$$\text{leaves}(T_1) = \text{branches}(T_1) + 1, \ \text{leaves}(T_2) = \text{branches}(T_2) + 1$$

et donc

$$\text{leaves}(\textbf{branch}(T_1, T_2)) = \text{branches}(T_1) + 1 + \text{branches}(T_2) + 1.$$

Par définition,

$$\text{branches}(\textbf{branch}(T_1, T_2)) = \text{branches}(T_1) + \text{branches}(T_2) + 1$$

et donc :

$$\text{leaves}(\textbf{branch}(T_1, T_2)) = \text{branches}(\textbf{branch}(T_1, T_2)) + 1.$$

□

Exercice

Définitions : La fonction nodes : $bTree \to \mathbb{N}$ calcule le nombre total de nœuds de l'arbre :

- Cas de base : nodes(**leaf**) ::= 1
- Cas récursif : nodes(**branch**(T_1, T_2)) ::= nodes(T_1) + nodes(T_2) + 1

La fonction height : $bTree \to \mathbb{N}$ calcule la hauteur d'un arbre comme suit :

- Cas de base : height(**leaf**) ::= 0
- Cas récursif :
  height(**branch**(T_1, T_2)) ::= 1 + max(height(T_1), height(T_2))

Démontrez le théorème suivant :

Théorème : Pour tout arbre $T \in bTree$,

$$\text{nodes}(T) \leq 2^{\text{height}(T)+1} - 1.$$

Fonctions récursives sur les entiers non-négatifs

Définition : L'ensemble $\mathbb{N}$ est un type de données défini récursivement comme suit :

- Cas de base : $0 \in \mathbb{N}$.
- Cas récursif : si $n \in \mathbb{N}$, alors le successeur, $n+1$, de n est dans $\mathbb{N}$.

L'induction ordinaire est donc équivalente à l'induction structurelle sur la définition récursive de $\mathbb{N}$.

Cette définition récursive permet aussi de définir des fonctions récursives sur les naturels.

Exemple : Fonction factorielle : $n!$

- $\text{fac}(0) ::= 1$.
- $\text{fac}(n+1) ::= (n+1) \cdot \text{fac}(n)$ pour $n \geq 0$.

Autres exemples

Nombres de Fibonacci :

- $F_0 ::= 0$,
- $F_1 ::= 1$,
- $F_i ::= F_{i-1} + F_{i-2}$ pour $i \geq 2$.

Notation $\sum$:

- $\sum_{i=1}^{0} f(i) ::= 0$,
- $\sum_{i=1}^{n+1} ::= \sum_{i=1}^{n} f(i) + f(n+1)$, pour $n \geq 0$.

Définition croisée :

- $f(0) ::= 1$, $g(0) ::= 1$,
- $f(n+1) ::= f(n) + g(n)$, pour $n \geq 0$,
- $g(n+1) ::= f(n) \times g(n)$, pour $n \geq 0$

Définitions mal formées

Des problèmes peuvent apparaître lorsque la fonction ne suit pas la définition du type de données récursifs :

$$f_1(n) ::= \begin{cases} 0 & \text{si } n = 1, \\ f_1(n+1) & \text{sinon.} \end{cases}$$ (ne définit pas $f_1(2)$ de manière unique)

$$f_2(n) ::= \begin{cases} 0, & \text{si } n \text{ est divisible par 2,} \\ 1, & \text{si } n \text{ est divisible par 3,} \\ 2, & \text{sinon.} \end{cases}$$ ($f_2(6) = 0$ ou $f_2(6) = 1$)

$$f_3(n) ::= \begin{cases} 1 & \text{si } n = 1 \\ 1 + f_3(n/2), & \text{si } n \text{ est pair} \\ f_3(3n-1) & \text{si } n \text{ est impair et } n > 1. \end{cases}$$ ($f_3(5)$ n'est pas défini)

Trois fonctions bien définies

Suite de Syracuse :

$$f(n) ::= \begin{cases} 1 & \text{si } n \leq 1, \\ f(n/2) & \text{si } n > 1 \text{ est pair,} \\ f(3n+1) & \text{si } n > 1 \text{ est impair.} \end{cases}$$

Conjecture : f(n)=1 pour tout n. (Voir transparent 110)

Fonction d'Ackermann :

$$A(m,n) ::= \begin{cases} 2n, & \text{si } m = 0 \text{ ou } n \leq 1, \\ A(m-1, A(m, n-1)), & \text{sinon.} \end{cases}$$

Bien définie. Croissante extrêmement rapide : $A(4,4) \simeq 2^{2^{2^{65536}}}$

Fonction de McCarthy :

$$M(n) ::= \begin{cases} n - 10, & \text{si } n > 100, \\ M(M(n+11)) & \text{si } n \leq 100. \end{cases}$$

$M(n) = 91$ pour tout $n \leq 101$, $n - 10$ sinon.

Expressions arithmétiques

Définition : Soit *Aexp* l'ensemble des expressions arithmétiques définies sur une seule variable x. *Aexp* peut être défini récursivement de la manière suivante :

- Cas de base :
  - x est dans *Aexp*.
  - $\forall k \in \mathbb{N}$, $\mathtt{k} \in Aexp$. [5]
- Cas récursifs : si $e, f \in Aexp$, alors :
  - $[e + f] \in Aexp$, (somme)
  - $[e * f] \in Aexp$, (produit)
  - $-[e] \in Aexp$. (négation)

Note : toutes les sous-expressions sont entre crochets et les exposants ne sont pas permis. $3x^2 + 2x + 1$ s'écrira dans *Aexp* :

$$[[3 * [x * x]] + [[2 * x] + 1]].$$

5. $\mathtt{k}$ désigne une chaîne de caractère codant pour un nombre, et k désigne le nombre correspondant.

Evaluation d'une expression

On peut définir l'évaluation comme une fonction définie récursivement sur la structure de *Aexp*.

Définition : La fonction d'évaluation, eval : $Aexp \times \mathbb{Z} \to \mathbb{Z}$ est définie récursivement sur les expressions $e \in Aexp$ comme suit. Soit n un entier :

- Cas de base :
  - $\text{eval}(x, n) ::= n$,
  - $\text{eval}(\mathtt{k}, n) ::= k$.
- Cas récursifs :
  - $\text{eval}([e_1 + e_2], n) ::= \text{eval}(e_1, n) + \text{eval}(e_2, n)$,
  - $\text{eval}([e_1 * e_2], n) ::= \text{eval}(e_1, n) \cdot \text{eval}(e_2, n)$,
  - $\text{eval}(-[e_1], n) ::= -\text{eval}(e_1, n)$.

Exemple :

$$\begin{aligned} \text{eval}([3 + [x * x]], 2) &= \text{eval}(3, 2) + \text{eval}([x * x], 2) \\ &= 3 + \text{eval}([x * x], 2) \\ &= 3 + (\text{eval}(x, 2) \cdot \text{eval}(x, 2)) \\ &= 3 + (2 \cdot 2) \\ &= 7. \end{aligned}$$

Substitution

La fonction $\text{subst}(f, e)$ remplace toutes les occurrences de x dans e par f.

Définition : La fonction subst : $Aexp \times Aexp \to Aexp$ est définie récursivement sur les expressions $e \in Aexp$ comme suit. Soit $f \in Aexp$.

- Cas de base :
  - $\text{subst}(f, x) ::= f$,
  - $\text{subst}(f, \mathtt{k}) ::= \mathtt{k}$.
- Cas récursifs :
  - $\text{subst}(f, [e_1 + e_2]) ::= [\text{subst}(f, e_1) + \text{subst}(f, e_2)]$,
  - $\text{subst}(f, [e_1 * e_2]) ::= [\text{subst}(f, e_1) * \text{subst}(f, e_2)]$,
  - $\text{subst}(f, -[e_1]) ::= -[\text{subst}(f, e_1)]$.

Exercice : montrez que

$$subst([3 * x], [x * [x + -[1]]]) = [[3 * x] * [[3 * x] + -[1]]]$$

.

Modèle d'évaluation avec substitution

Supposons qu'on veuille évaluer l'expression $e' ::= \text{subst}(f, e)$ pour une valeur de $x = n$.

Deux possibilités :

- Calculer $\text{eval}(\text{subst}(f, e), n)$, c'est-à-dire remplacer x par f dans e et puis évaluer l'expression résultante pour $x = n$.
  (Modèle de substitution)
- Calculer $\text{eval}(e, \text{eval}(f, n))$, c'est-à-dire évaluer f pour $x = n$ et ensuite e pour x fixé à la valeur obtenue pour f.
  (Modèle d'environnement)

La deuxième approche est généralement plus efficace (d'autant plus que f est complexe et qu'il y a de x dans e).

Exercice : tester les deux approches pour $f = [3 * x]$, $e = [x * [x + -[1]]]$, $n = 2$.

Modèle d'évaluation avec substitution

En terme de langage de programmation :

- Modèle de substitution :

```
x=2;
return (3*x)*((3*x)+-(1))
```

  ⇒ 3 multiplications

- Modèle d'environnement :

```
x=2
f=3*x
return f*(f+-(1))
```

  ⇒ 2 multiplications

Equivalence des deux approches

Théorème : Pour toutes expressions $e, f \in Aexp$ et $n \in \mathbb{Z}$,

$$\text{eval}(\text{subst}(f, e), n) = \text{eval}(e, \text{eval}(f, n)).$$

Démonstration : La preuve se fait par induction structurelle sur e.

- Cas de base :
  - $e = x$: les membres de gauche et droite valent tous deux $\text{eval}(f, n)$.
  - $e = \mathtt{k}$: par la définition de eval et subst, les deux membres valent $\mathtt{k}$.
- Cas récursifs :
  - $e = [e_1 + e_2]$: Par hypothèse d'induction structurelle, on suppose que pour tout $f \in Aexp$, $n \in \mathbb{Z}$, et $i \in \{1, 2\}$

$$\text{eval}(\text{subst}(f, e_i), n) = \text{eval}(e_i, \text{eval}(f, n)).$$

    Montrons que

$$\text{eval}(\text{subst}(f, [e_1 + e_2]), n) = \text{eval}([e_1 + e_2], \text{eval}(f, n)).$$

Par la définition de subst, le membre de gauche vaut :

$$\text{eval}([\text{subst}(f, e_1) + \text{subst}(f, e_2)], n),$$

qui, par la définition de eval, est égal à :

$$\text{eval}(\text{subst}(f, e_1), n) + \text{eval}(\text{subst}(f, e_2), n).$$

Par hypothèse inductive, cette expression est égale à

$$\text{eval}(e_1, \text{eval}(f, n)) + \text{eval}(e_2, \text{eval}(f, n)).$$

Par définition de eval, cette dernière expression est égale au membre de droite, ce qui prouve le théorème dans ce cas.

- $e = [e_1 * e_2]$, $e = -[e_1]$ sont traités de manière similaire.

Tous les cas de base et récursifs étant traités, le théorème est prouvé par induction structurelle. □

Résumé

On a vu :

- comment définir des ensembles de manière récursive
- comment définir des fonctions sur les éléments d'un ensemble défini récursivement
- comment prouver l'appartenance d'un élément à un ensemble
- comment prouver des propriétés sur ces ensembles par le principe d'induction structurelle
- qu'il existe des définitions récursives ambiguës

Note : Dans tous nos exemples, on pourrait se passer de l'induction structurelle et s'en sortir avec l'induction ordinaire. Cependant :

- L'induction structurelle simplifie les preuves
- En théorie, l'induction structurelle est plus puissante que l'induction ordinaire

Applications

Les applications de ces principes sont innombrables en informatique :

- Définition de structure de données récursives (arbres, chaînes de caractère, listes, etc.)
- Calcul de fonctions et vérification de propriétés sur ces structures
- Définition de la syntaxe et de la sémantique de langages de programmation (voir le chapitre suivant)
- Fonctions récursives sur $\mathbb{N}$: dénombrement d'ensembles et analyse de complexité (voir la troisième partie du cours)

Chapitre 4

Introduction à la syntaxe et à la sémantique des langages de programmation

Plan

Lectures conseillées :

- Notes de cours de Guy McCusker et Matthew Hennessy, University of Sussex. `http://www.informatics.sussex.ac.uk/courses/semantics/current/GuysNotes.pdf`
- David A. Schmidt, Programming Language Semantics. `http://people.cis.ksu.edu/~schmidt/705s12/Lectures/chapter.pdf`

Syntaxe et sémantique

La **syntaxe** d'un langage de programmation détermine ce qui constitue un programme valide du point de vue de la **forme**/du texte.

Quelles séquences de caractères constituent des programmes ?

La **sémantique** définit la signification d'un programme, c'est-à-dire ce qu'il calcule, comment il le calcule.

Que signifie ce programme ? Quand est-ce que deux programmes sont équivalents ? Est-ce que ce programme satisfait aux spécifications ?

Dans ce chapitre, on se focalisera sur la sémantique de langages de programmation très simples.

Sémantique formelle

Pourquoi définir la sémantique de manière formelle (plutôt que de manière informelle) ?

- Implémentation : correction des compilateurs, optimisation, analyse statique, etc.
- Vérification : permet de raisonner sur les programmes et leurs propriétés, que ce soit de manière automatique ou à la main.
- Design de langage : permet de détecter des ambiguités et d'organiser les choses de manière plus claire.

Plan

Avant de s'attaquer à un langage de programmation, on va se servir des expression arithmétiques (sans variable) pour définir les principaux concepts.

Expressions arithmétiques sur $\mathbb{N}$: syntaxe

L'ensemble des expressions *Exp* est défini récursivement comme suit :

- Cas de base : $\forall n \in \mathbb{N}$, $\mathtt{n} \in \mathit{Exp}$
- Cas récursifs : Si E_1 et $E_2 \in \mathit{Exp}$:
  - $(E_1 + E_2) \in \mathit{Exp}$
  - $(E_1 \times E_2) \in \mathit{Exp}$

On simplifie cette définition par une expression du type :

$$E \in \mathit{Exp} ::= \mathtt{n} | (E + E) | (E \times E)$$

où :

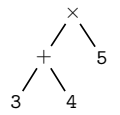
- "|" signifie "ou".
- $E + E$ signifie "une expression suivie de '+' suivi d'une expression"
- $\mathtt{n}$, '+', '$\times$' sont des *symboles terminaux* (cas de base). E est un *symbole non terminal* (cas récursif).

Cette notation s'appelle la *forme de Backus-Naur* (BNF)

Syntaxe concrète versus syntaxe abstraite

On distingue deux types de syntaxe :

- Syntaxe concrète : programme représenté par la suite de caractères tapés par l'utilisateur
  - Exemple : $3 + 4 \times 5$
- Syntaxe abstraite : programme représenté par une structure basée sur la définition récursive, typiquement un arbre
  - Exemple :



Passer de la syntaxe concrète à la syntaxe abstraite s'appelle *l'analyse syntaxique* du programme.

Dans ce chapitre, on supposera toujours que la syntaxe abstraite est connue (pas d'ambiguïté sur la manière dont l'expression est obtenue).

Sémantique

Essentiellement deux types de sémantiques :

- **Sémantique dénotationnelle :** la signification d'un programme est déterminée par une fonction, la *dénotation*, associant une valeur à un programme.
- **Sémantique opérationnelle :** la signification d'un programme est déterminée par le suite des *états* de la machine qui l'exécute.

Sémantique dénotationnelle

On définit une fonction, appelée *dénotation*, prenant en argument un programme (et d'autres valeurs potentielles) et renvoyant la valeur résultant de l'exécution du programme.

Seule importe la valeur finale résultant de l'exécution et pas la manière de l'obtenir. Les deux programmes suivants sont identiques du point de vue de la sémantique dénotationnelle :

$x = 0;$
while $x < 10$
x=x+1 ;

$x = 10;$

Le domaine des valeurs résultats de l'exécution du programme est appelé le *domaine sémantique* du programme.

Exemple : expressions arithmétiques

Le domaine sémantique est $\mathbb{N}$.

La dénotation est une fonction eval : $Exp \to \mathbb{N}$ définie récursivement :

- Cas de base : $\text{eval}(\mathtt{n}) = n$ où n est le naturel associé au numéral $\mathtt{n}$.
- Cas récursifs :
  - $\text{eval}(E_1 + E_2) ::= \text{eval}(E_1) + \text{eval}(E_2)$
  - $\text{eval}(E_1 \times E_2) ::= \text{eval}(E_1) \cdot \text{eval}(E_2)$

Exemple :

$$\begin{aligned} \text{eval}((1 + (2 + 3))) &= \text{eval}(1) + \text{eval}(2 + 3) \\ &= 1 + \text{eval}(2 + 3) \\ &= 1 + (2 + 3) \\ &= 6 \end{aligned}$$

Sémantique opérationnelle

La signification d'un programme est définie par une modélisation des étapes de son exécution.

Les programmes suivants sont très différents d'une point de vue sémantique opérationnelle :

$x = 0;$
while $x < 10$
x=x+1 ;

$x = 10;$

L'exécution d'un programme peut être modélisée par une *machine d'état*.

Expressions arithmétiques

Idée générale : chaque étape de l'évaluation d'une expression consiste à simplifier le terme le plus à gauche (par convention). Exemple :

$$((1+2)+(4+5)) \to (3+(4+5)) \to (3+9) \to 12$$

Une machine d'état pour l'évaluation des expressions arithmétiques (addition uniquement) :

- $\mathcal{Q} = Exp$, l'ensemble des expressions
- $\mathcal{Q}_0 = \{E\}$, où $E \in Exp$ est l'expression à évaluer.
- δ est défini récursivement :
  - Cas de base : $\forall n_1, n_2 \in \mathbb{N} : (\mathtt{n}_1 + \mathtt{n}_2) \to \mathtt{n}_3 \in \delta$, où $n_3 = n_1 + n_2$
  - Cas récursif :
    - Si $E_2 \in Exp$ et $E_1 \to E_1' \in \delta$, alors $(E_1 + E_2) \to (E_1' + E_2) \in \delta$.
    - Si $E \to E' \in \delta$, alors $(\mathtt{n} + E) \to (\mathtt{n} + E') \in \delta$.

(Exercice : ajoutez la multiplication et simplifiez le terme à droite d'abord)

Règles déductives

Les transitions sont généralement définies sous la forme de régles déductives.

$$\frac{E_1 \to E_1'}{(E_1 + E_2) \to (E_1' + E_2)} \text{ (S-LEFT)}$$

$$\frac{E \to E'}{(\mathtt{n} + E) \to (\mathtt{n} + E')} \text{ (S-RIGHT)}$$

$$\frac{}{(\mathtt{n}_1 + \mathtt{n}_2) \to \mathtt{n}_3} \text{ (S-ADD)} \quad (\text{où } n_3 = n_1 + n_2)$$

Par exemple, la règle (S-LEFT) signifie que pour évaluer une addition, il faut d'abord évaluer le premier terme.

Exemple

Evaluation de $(3 + (2 + 1))$:

- Par l'axiome (S-ADD), on a $(2 + 1) \to 3$.
- Par (S-RIGHT), $(3 + (2 + 1)) \to (3 + 3)$
- Par (S-ADD), on a $(3 + 3) \to 6$.
- Finalement :

$$(3 + (2 + 1)) \to (3 + 3) \to 6$$

La sémantique fixe l'ordre d'évaluation. On a :

$$((1 + 2) + (3 + 4)) \to (3 + (3 + 4))$$

et pas

$$((1 + 2) + (3 + 4)) \to ((1 + 2) + 7)$$

(Exercice : Ecrivez tous les $E \in Exp$ tels que $E \to ((1 + 2) + 7)$)

Notations

Introduisons quelques notations.

Définition : On a $E \to^* E'$ si et seulement si soit $E = E'$, soit il existe une séquence finie :

$$E \to E_1 \to E_2 \to \ldots E_k \to E'.$$

($\Leftrightarrow E'$ est atteignable à partir de E).

Définition : La relation $\to^k$ pour tout $k \in \mathbb{N}$ est définie récursivement sur $\mathbb{N}$:

- Cas de base : $E \to^0 E$ pour tout $E \in Exp$
- Cas récursif : $E \to^{k+1} E'$ s'il existe E'' tel que :
  - $E \to^k E''$ et
  - $E'' \to E'$.

Définition : Soit une expression $E \in Exp$. n est la réponse finale de E si $E \to^* \mathtt{n}$.

Déterminisme et normalisation

Propriété de la sémantique dénotationnelle :

- Toute expression possède une réponse finale *(normalisation)*.
- Cette réponse est unique et ne peut être évaluée que d'une et une seule manière *(déterminisme)*.

Formellement : pour tout $E \in Exp$:

- Si $E \to E_1$ et $E \to E_2$, alors $E_1 = E_2$. (*déterminisme*)
- Si $E \to^* \mathtt{n}$ et $E \to^* n'$, alors $n = n'$. (*déterminisme*)
- Il existe n tel que $E \to^* \mathtt{n}$. (*normalisation*).

Déterminisme

Théorème : Si $E \to E_1$ et $E \to E_2$, alors $E_1 = E_2$.

Démonstation : Prouvons par induction structurelle le prédicat $P(E)$ ="Pour tout F_1, F_2, si $E \to F_1$ et $E \to F_2$, alors $F_1 = F_2$".
Cas de base : $P(\mathtt{n})$ est vrai trivialement puisqu'il n'existe aucune transition à partie d'un numéral.
Cas inductif : $E = (E_1 + E_2)$. Supposons $P(E_1)$ et $P(E_2)$ vrais et montrons que $P(E_1 + E_2)$ est vrai.
Supposons que $E_1 + E_2 \to F_1$ et $E_1 + E_2 \to F_2$. On doit montrer que $F_1 = F_2$.
Deux cas possibles :

- E_1 est un numéral $\mathtt{n}_1$. Il y a alors deux façons de dériver $\mathtt{n}_1 + E_2 \to F_1$:
  - En appliquant la règle :
    $$\frac{E_2 \to G_1}{(\mathtt{n}_1 + E_2) \to (\mathtt{n}_1 + G_1)}\ \texttt{(S-RIGHT)}$$
    Dans ce cas $F_1 = \mathtt{n}_1 + G_1$, pour un G_1 tel que $E_2 \to G_1$. Puisqu'il y a une dérivation $E_2 \to G_1$, E_2 n'est pas un numéral. Toute dérivation à partir de E_2 doit donc utiliser `(S-RIGHT)`. On peut donc montrer similairement que $F_2 = \mathtt{n}_1 + G_2$, pour un G_2 tel que $E_2 \to G_2$.
    Or par hypothèse inductive, on a $G_1 = G_2$ et donc $F_1 = F_2$.

  - En appliquant la règle `(S-ADD)`. Dans ce cas, E_2 est un numéral $\mathtt{n}_2$ et donc F_1 est aussi un numéral $\mathtt{n}_3$, avec $n_1 + n_2 = n_3$. Considérons maintenant la dérivation $n_1 + E_2 \to F_2$. Puisque $E_2 = \mathtt{n}_2$, la seule règle possible est `(S-ADD)` et donc, $F_2 = \mathtt{n}_3$, ce qui montre $F_1 = F_2$.
- E_1 n'est pas un numéral. La seule règle possible est `(S-LEFT)` qui donne un règle de la forme :
  $$\frac{E_1 \to G_1}{(E_1 + E_2) \to (G_1 + E_2)}\ \texttt{(S-LEFT)}$$
  F_1 est donc de la forme $G_1 + E_1$ pour un G_1 tel que $E_1 \to G_1$. De même, F_2 doit aussi être de la forme $G_2 + E_1$ pour un G_2 tel que $E_1 \to G_2$.
  On utilisant $P(E_1)$, on obtient $G_1 = G_2$ et donc $F_1 = F_2$.

□

Déterminisme

Corrolaire : Pour tout naturel k et expression $E \in Exp$, si $E \to^k E_1$ et $E \to^k E_2$, alors $E_1 = E_2$.

Démonstation : La démonstration fonctionne par induction sur $\mathbb{N}$.
Soit le prédicat $P(k)$ ="$E \to^k E_1$ et $E \to^k E_2$ implique $E_1 = E_2$".

Cas de base : Supposons que $E \to^0 E_1$ et $E \to^0 E_2$. On a trivialement $E_1 = E_2 = E$.

Cas inductif : Supposons $P(k)$ vrai et montrons $P(k+1)$. Par définition de $\to^{k+1}$, il existe E_1' et E_2' tels que

$$E \to^k E_1' \to E_1$$

$$E \to^k E_2' \to E_2$$

Par hypothèse inductive, on a $E_1' = E_2'$. Par le théorème précédent (déterminisme de $\to$), on en déduit que $E_1 = E_2$.

□

Déterminisme

Théorème : Si $E \to^* \mathtt{n}$ et $E \to^* \mathtt{n}'$, alors $\mathtt{n} = \mathtt{n}'$.

Démonstration : $E \to^* \mathtt{n}$ signifie qu'il existe $k \in \mathbb{N}$ tel que $E \to^k \mathtt{n}$. De même, il existe $k' \in \mathbb{N}$ tel que $E \to^{k'} \mathtt{n}'$.
Soit $k \leq k'$, soit $k \geq k'$. Supposons $k \leq k'$ (le cas $k \geq k'$ est symétrique). La dérivation est de la forme :

$$E \to^k \mathtt{n}$$
$$E \to^k E' \to^{(k'-k)} \mathtt{n}'$$

pour un E'. Par le corrolaire précédent, E' doit être égal à $\mathtt{n}$. Selon les règles du transparent 162, il n'y a pas de transition à partir d'un numéral. La réduction $E' \to^{(k'-k)} \mathtt{n}'$ doit donc être $\mathtt{n} \to^0 \mathtt{n}'$. Et donc $\mathtt{n} = \mathtt{n}'$.

□

Normalisation

Théorème : Pour tout E, il existe un n tel que $E \to^* n$.

Démonstation : La démonstration fonctionne par induction structurelle sur E.
Cas de base : $E = \mathtt{n}$. On a trivialement $\mathtt{n} \to^* \mathtt{n}$.

Cas inductif : $E = (E_1 + E_2)$. Par hypothèse inductive, il existe n_1 et n_2 tels que $E_1 \to^* \mathtt{n}_1$ et $E_2 \to^* \mathtt{n}_2$. Pour chaque étape de la dérivation :

$$E_1 \to E_1' \to E_1'' \ldots \to \mathtt{n}_1,$$

appliquer la régle de réduction de l'argument de gauche donne :

$$(E_1 + E_2) \to (E_1' + E_2) \to (E_1'' + E_2) \ldots \to (\mathtt{n}_1 + E_2).$$

En appliquant ensuite la règle de réduction de l'argument de droite, on peut déduire :

$$(\mathtt{n}_1 + E_2) \to^* (\mathtt{n}_1 + \mathtt{n}_2) \to \mathtt{n}_3,$$

où $n_3 = n_1 + n_2$. D'où $(E_1 + E_2) \to^* \mathtt{n}_3$.

□

Déterminisme et normalisation

Corrolaire : Pour tout expression E, il y a exactement un n tel que $E \to^* \mathtt{n}$.

Démonstration : C'est une conséquence directe des théorèmes précédents.

□

Correspondance

On peut montrer que les deux sémantiques sont équivalentes.

Théorème : Pour tout expression E, $\text{eval}(E) = n$ si et seulement si $E \to^* \mathtt{n}$.

Démonstration : Montrons par induction structurelle sur E que $P(E)$="$\text{eval}(E) = n \Leftrightarrow E \to^* \mathtt{n}$" est vrai pour tout E.

Cas de base : E est un numéral $\mathtt{n}$. Dans ce cas, on a clairement $\text{eval}(E) = n$ et $E \to^* \mathtt{n}$ et donc $P(E)$ est vérifié.

Cas inductif : $E = (E_1 + E_2)$. Supposons que $P(E_1)$ et $P(E_2)$ soient vérifiés et montrons que $P(E)$ est vérifié.

Montrons d'abord que $(E_1 + E_2) \to^* \mathtt{n}_3 \Rightarrow \text{eval}((E_1 + E_2)) = n_3$.

> Si $(E_1 + E_2) \to^* \mathtt{n}_3$, il existe $\mathtt{n}_1$ et $\mathtt{n}_2$ tels que :
> $(E_1 + E_2) \to^* (\mathtt{n}_1 + E_2) \to^* (\mathtt{n}_1 + \mathtt{n}_2) \to \mathtt{n}_3$ et $n_1 + n_2 = n_3$. Par hypothèse inductive, on a $\text{eval}(E_1) = n_1$ et $\text{eval}(E_2) = n_2$. Par définition de eval, on a donc
> $\text{eval}((E_1 + E_2)) = \text{eval}(E_1) + \text{eval}(E_2) = n_1 + n_2 = n_3$.

Montrons que $\text{eval}((E_1 + E_2)) = n_3 \Rightarrow (E_1 + E_2) \to^* \mathtt{n}_3$:

> Soit n_1 et n_2 tels que $\text{eval}(E_1) = n_1$ et $\text{eval}(E_2) = n_2$. Par définition de eval, on a $\text{eval}((E_1 + E_2)) = n_1 + n_2 = n_3$. Par hypothèse inductive, on a $E_1 \to^* \mathtt{n}_1$ et $E_2 \to^* \mathtt{n}_2$ et donc $(E_1 + E_2) \to^* (\mathtt{n}_1 + E_2) \to^* (\mathtt{n}_1 + \mathtt{n}_2) \to \mathtt{n}_3$, qui implique $(E_1 + E_2) \to^* \mathtt{n}_3$.

□

Résumé : expression arithmétique

Jusqu'ici, nous avons :

- défini la syntaxe du langage des expressions arithmétiques par une forme BNF,
- défini sa sémantique dénotationnelle par une dénotation définie récursivement sur la syntaxe,
- défini sa sémantique opérationnelle par une machine d'état,
- prouvé le déterminisme et la normalisation de la sémantique opérationnelle,
- prouvé la correspondance entre les sémantiques dénotationnelle et opérationnelle.

Plan

Dans cette section, on va :

- définir la syntaxe d'un langage de programmation simple
- donner un sémantique opérationnelle
- esquisser la démonstration de son déterminisme
- donner une sémantique dénotationnelle

Un langage de programmation simple

Soit le langage dont la syntaxe (en forme BNF) est donnée ci-dessous.

$$
\begin{array}{rcl}
B \in \mathit{Bool} & ::= & \mathtt{true} \mid \mathtt{false} \mid E = E \mid E < E \mid \ldots \\
 & & \mid B \& B \mid \neg B \mid \ldots \\
E \in \mathit{Exp} & ::= & x \mid \mathtt{n} \mid (E + E) \mid \ldots \\
C \in \mathit{Com} & ::= & x := E \mid \mathtt{if}\ B\ \mathtt{then}\ C\ \mathtt{else}\ C \\
 & & \mid C ; C \mid \mathtt{skip} \mid \mathtt{while}\ B\ \mathtt{do}\ C
\end{array}
$$

Un langage de programmation simple

Remarques :

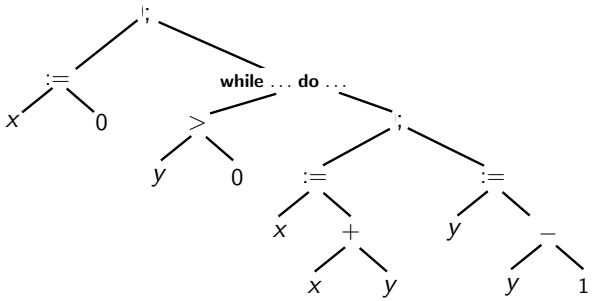
- *Exp* contient les expressions arithmétiques, *Bool* les expressions à valeurs booléennes, *Com* est l'ensemble des programmes.
- Ces trois ensembles sont définies de manière récursive et croisée.
- $\mathtt{n}$ et x sont des méta-variables de la définition. $\mathtt{n}$ désigne le numéral correspondant au naturel $n \in \mathbb{N}$. x est un identifiant de variable quelconque.
- La sémantique est la sémantique habituelle des langages impératifs.
- Comme pour les expressions, on travaillera toujours sur base de la syntaxe abstraite.

Exemple

Un programme :

$x := 0;$
while $y > 0$ **do**
  $x := x + y;$
  $y := y - 1$

L'arbre syntaxique correspondant :



Sémantique opérationnelle : état

L'état de la machine évaluant un programme doit inclure :

- Le programme P ($\in \mathit{Com} \cup \mathit{Exp} \cup \mathit{Bool}$) restant à exécuter
- L'état de la mémoire contenant les valeurs assignées aux variables

L'état de la mémoire est modélisé par une fonction partielle s associant une valeur numérique $s(x) \in \mathbb{N}$ à un nombre fini d'identifiants x.

Soit s un état de la mémoire. L'état de la mémoire noté $s[x \mapsto n]$ est défini par :

$$s[x \mapsto n](y) = \begin{cases} \mathtt{n} & \text{si } y = x, \\ s(y) & \text{sinon.} \end{cases}$$

La sémantique opérationnelle définit des transitions du type :

$$\langle P, s \rangle \to \langle P', s' \rangle.$$

Sémantique opérationnelle : expressions

Comme précédemment, sauf pour la première règle :

$$\frac{}{\langle x, s \rangle \to \langle \mathtt{n}, s \rangle} \text{ (W-EXP.NUM)} \quad (\text{où } s(x) = \mathtt{n})$$

$$\frac{}{\langle (\mathtt{n}_1 + \mathtt{n}_2), s \rangle \to \langle \mathtt{n}_3, s \rangle} \text{ (W-EXP.ADD)} \quad (\text{où } n_3 = n_1 + n_2)$$

$$\frac{\langle E_1, s \rangle \to \langle E_1', s' \rangle}{\langle (E_1 + E_2), s \rangle \to \langle (E_1' + E_2, s' \rangle} \text{ (W-EXP.LEFT)}$$

...

Note : ces règles n'ont pas d'effet sur la mémoire

(*Exercice : ajouter les autres règles pour les expression et les règles pour les booléens*)

Sémantique opérationnelle : assignation

Evaluation de la valeur à assigner :

$$\frac{\langle E, s\rangle \to \langle E', s'\rangle}{\langle x := E, s\rangle \to \langle x := E', s'\rangle}\ \text{(W-ASS.EXP)}$$

Assignation proprement dite :

$$\frac{}{\langle x := \mathtt{n}, s\rangle \to \langle \mathtt{skip}, s[x \mapsto \mathtt{n}]\rangle}\ \text{(W-ASS.NUM)}$$

Sémantique opérationnelle : composition

Exécution de l'instruction à gauche :

$$\frac{\langle C_1, s\rangle \to \langle C_1', s'\rangle}{\langle C_1; C_2, s\rangle \to \langle C_1'; C_2, s'\rangle}\ \text{(W-SEQ.LEFT)}$$

Passage à la seconde commande, une fois la première exécutée :

$$\frac{}{\langle \mathtt{skip}; C_2, s\rangle \to \langle C_2, s\rangle}\ \text{(W-SEQ.RIGHT)}$$

Sémantique opérationnelle : conditions

On évalue d'abord le gardien :

$$\frac{\langle B, s\rangle \to \langle B', s'\rangle}{\langle \mathtt{if}\ B\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2, s\rangle \to \langle \mathtt{if}\ B'\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2, s'\rangle}\ \text{(W-COND.B)}$$

En fonction de sa valeur, on évalue la première ou la deuxième commande :

$$\frac{}{\langle \mathtt{if\ true\ then}\ C_1\ \mathtt{else}\ C_2, s\rangle \to \langle C_1, s\rangle}\ \text{(W-COND.TRUE)}$$

$$\frac{}{\langle \mathtt{if\ false\ then}\ C_1\ \mathtt{else}\ C_2, s\rangle \to \langle C_2, s\rangle}\ \text{(W-COND.FALSE)}$$

Sémantique opérationnelle : while

Une première version incorrecte :

$$\frac{\langle B, s\rangle \to \langle B', s'\rangle}{\langle \mathtt{while}\ B\ \mathtt{do}\ C, s\rangle \to \langle \mathtt{while}\ B'\ \mathtt{do}\ C, s'\rangle}$$

$$\frac{}{\langle \mathtt{while\ false\ do}\ C, s\rangle \to \langle \mathtt{skip}, s\rangle}$$

$$\frac{}{\langle \mathtt{while\ true\ do}\ C, s\rangle \to ?}$$

Une version correcte :

$$\frac{}{\langle \mathtt{while}\ B\ \mathtt{do}\ C, s\rangle \to \langle \mathtt{if}\ B\ \mathtt{then}(C; \mathtt{while}\ B\ \mathtt{do}\ C)\ \mathtt{else\ skip}, s\rangle}\ \text{(W-WHILE)}$$

(On recopie la boucle while à l'intérieur d'une condition)

Synthèse de la sémantique opérationnelle

$$\frac{}{\langle x, s\rangle \to \langle \mathtt{n}, s\rangle}\ \texttt{(W-EXP.NUM)} \qquad \frac{\langle E, s\rangle \to \langle E', s'\rangle}{\langle x := E, s\rangle \to \langle x := E', s'\rangle}\ \texttt{(W-ASS.EXP)}$$

$$\frac{}{\langle (\mathtt{n}_1 + \mathtt{n}_2), s\rangle \to \langle \mathtt{n}_3, s\rangle}\ \texttt{(W-EXP.ADD)} \qquad \frac{}{\langle x := \mathtt{n}, s\rangle \to \langle \mathtt{skip}, s[x \mapsto \mathtt{n}]\rangle}\ \texttt{(W-ASS.NUM)}$$

$$\frac{\langle E_1, s\rangle \to \langle E_1', s'\rangle}{\langle (E_1 + E_2), s\rangle \to \langle (E_1' + E_2, s'\rangle}\ \texttt{(W-EXP.LEFT)} \qquad \frac{\langle C_1, s\rangle \to \langle C_1', s'\rangle}{\langle C_1; C_2, s\rangle \to \langle C_1'; C_2, s'\rangle}\ \texttt{(W-SEQ.LEFT)}$$

$$\frac{}{\langle \mathtt{skip}; C_2, s\rangle \to \langle C_2, s\rangle}\ \texttt{(W-SEQ.SKIP)}$$

$$\frac{\langle B, s\rangle \to \langle B', s'\rangle}{\langle \mathtt{if}\ B\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2, s\rangle \to \langle \mathtt{if}\ B'\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2, s'\rangle}\ \texttt{(W-COND.B)}$$

$$\frac{}{\langle \mathtt{if\ true\ then}\ C_1\ \mathtt{else}\ C_2, s\rangle \to \langle C_1, s\rangle}\ \texttt{(W-COND.TRUE)}$$

$$\frac{}{\langle \mathtt{if\ false\ then}\ C_1\ \mathtt{else}\ C_2, s\rangle \to \langle C_2, s\rangle}\ \texttt{(W-COND.FALSE)}$$

$$\frac{}{\langle \mathtt{while}\ B\ \mathtt{do}\ C, s\rangle \to \langle \mathtt{if}\ B\ \mathtt{then}(C; \mathtt{while}\ B\ \mathtt{do}\ C)\ \mathtt{else\ skip}, s\rangle}\ \texttt{(W-WHILE)}$$

Illustration

Soit le programme P suivant calculant la factorielle de x dans a :

$y := x; a := 1;$
while $y > 0$ **do**
$(a := a \times y;$
$y := y - 1)$

Soit s l'état initial de la mémoire tel que $s(x) = 3, s(y) = 2, s(a) = 9$. Montrez que

$$\langle P, s\rangle \to \langle \mathtt{skip}, s'\rangle$$

où $s'(a) = 6$.

Illustration

Si on note l'état de la mémoire $s_{i,j,k}$ pour $x \mapsto i$, $y \mapsto j$, et $a \mapsto k$, l'exécution de la machine d'état est la suivante (où P' désigne la boucle while) :

$$\begin{aligned}
& \langle y := x, a := 1; P', s_{3,2,9}\rangle \\
\to\ & \langle y := 3; a := 1; P', s_{3,2,9}\rangle \\
\to\ & \langle \mathtt{skip}; a := 1; P', s_{3,3,9}\rangle \\
\to\ & \langle a := 1; P', s_{3,3,9}\rangle \\
\to\ & \langle \mathtt{skip}; P', s_{3,3,1}\rangle \\
\to\ & \langle P', s_{3,3,1}\rangle \\
& \dots \\
\to\ & \langle \mathtt{skip}, s_{3,0,6}\rangle
\end{aligned}$$

Réponse finale et normalisation

Pour un programme P et un état de la mémoire s, la *réponse finale* associée à l'exécution de P à partir de s est l'état s' tel que :

$$\langle P, s\rangle \to^* \langle \mathtt{skip}, s'\rangle$$

Cet état s' n'existe cependant pas pour tout P et s.

Exemples :

- Soit s associant x à 3 et indéfini pour tout autre identifiant :

$$\langle y := y + 1, s\rangle \to ?$$

- Il n'y a pas d'états s et s' tels que :

$$\langle \mathtt{while\ true\ do\ skip}, s\rangle \to^* \langle \mathtt{skip}, s'\rangle$$

Déterminisme

Lemme (déterminisme) : Pour toute configuration $\langle P, s\rangle$, il y a au plus une configuration $\langle P', s'\rangle$ telle que $\langle P, s\rangle \to \langle P', s'\rangle$.
Démonstration : La démonstration fonctionne par induction structurelle. Soit le prédicat : $Q(P)$="Si $\langle P, s\rangle \to \langle P', s'\rangle$ et $\langle P, s\rangle \to \langle P'', s''\rangle$, alors $\langle P', s'\rangle = \langle P'', s''\rangle$".

- Cas : `skip`, `n`, `true`, `false` : il n'y a pas de transition possible à partir d'un de ces cas de base. Le prédictat est donc vrai.
- Cas : $C_1; C_2$. Supposons $Q(C_1)$ et $Q(C_2)$ vrais et montrons que $Q(C_1; C_2)$ est vrai.
  Soit C', s', C'', s'' tels que

$$\langle C_1; C_2, s\rangle \to \langle C', s'\rangle,$$

$$\langle C_1; C_2, s\rangle \to \langle C'', s''\rangle.$$

En examinant les règles, les seules possibilités sont :

- $C_1 =$ `skip` : La seule règle possible est `(W-SEQ.SKIP)`. On a donc $\langle C', s'\rangle = \langle C_2, s\rangle = \langle C'', s''\rangle$
- C_1 n'est pas une valeur (booléen, numéral, ou `skip`) : la règle appliquée est dans les deux cas `(W-SEQ.LEFT)`. Il existe alors C_1' et C_1'' tels que $\langle C_1, s\rangle \to \langle C_1', s'\rangle$ et $\langle C_1, s\rangle \to \langle C_1'', s''\rangle$ et donc $C' = C_1'; C_2$ et $C'' = C_1''; C_2$. Par hypothèse inductive, on a $\langle C_1', s'\rangle = \langle C_1'', s''\rangle$ et donc $\langle C', s'\rangle = \langle C_1'; C_2, s'\rangle = \langle C_1''; C_2, s''\rangle = \langle C'', s''\rangle$.

- Cas : ...

□

Exercice : complétez la démonstration pour les autres commandes et les booléens.

Boucle infinie

Théorème : Pour aucun s, il n'existe de s' tel que :

$$\langle \texttt{while true do skip}, s\rangle \to^* \langle \texttt{skip}, s'\rangle$$

Démonstration : Calculons les trois premières étapes de l'évaluation de la boucle :

$$\begin{aligned} & \langle \texttt{while true do skip}, s\rangle \\ \to\ & \langle \texttt{if true then (skip; while true do skip) else skip}, s\rangle \\ \to\ & \langle \texttt{skip; while true do skip}, s\rangle \\ \to\ & \langle \texttt{while true do skip}, s\rangle \end{aligned}$$

Par contradiction, supposons qu'il existe s' tel que

$$\langle \texttt{while true do skip}, s\rangle \to^* \langle \texttt{skip}, s'\rangle$$

et soit n le nombre d'étapes nécessaires à cette évaluation.

Vu le déterminisme de la machine, n est bien défini et les 3 premières étapes de cette évaluation sont les trois étapes reprises ci-dessus.
Les $n - 3$ étapes restantes de l'évaluation montrent que

$$\langle \texttt{while true do skip}, s\rangle \to \langle \texttt{skip}, s'\rangle,$$

ce qui n'est pas possible puisque cette évaluation requière n étapes.

□

Effets de bord

Dans notre langage, le choix de l'ordre d'évaluation des arguments des opérateurs ($+$, &, etc.) n'a pas d'importance.

Dans des langages plus complexes, l'ordre peut avoir de l'importance.

Exemples :

- Dans l'expression suivante, il est essentiel de connaître l'ordre d'évaluation des arguments du $+$:

$$(x := x + 1; \texttt{return}(x)) + (x := x \times 2; \texttt{return}(x))$$

- Selon l'implémentation du &, l'expression suivante aura on non une valeur :

$$\texttt{false\&(while true do skip; return(true))}$$

Evaluation du & avec court-circuit

$$\frac{B_1 \to B_1'}{(B_1 \& B_2) \to (B_1' \& B_2)}$$

$$\frac{}{(\texttt{false}\& B_2) \to \texttt{false}} \qquad \frac{}{(\texttt{true}\& B_2) \to B_2}$$

Sémantique dénotationnelle

Rappel : la dénotation est une fonction associant une valeur du domaine sémantique à un programme.

Pour évaluer un programme, on doit pouvoir évaluer aussi les expressions arithmétique et booléennes et l'évaluation dépend de l'état initial de la mémoire.

On va définir trois fonctions de dénotation :

$$\begin{aligned} \text{eval}_{Com} &: Com \times \Sigma \to \Sigma_\perp, \\ \text{eval}_{Exp} &: Exp \times \Sigma \to \mathbb{N}_\perp, \\ \text{eval}_{Bool} &: Bool \times \Sigma \to \{\texttt{true}, \texttt{false}, \perp\}, \end{aligned}$$

où

- Σ est l'ensemble des configurations possibles de la mémoire,
- Le symbole $\perp$ (*bottom*) permet de représenter une valeur *indéfinie* (boucle infinie ou variable non définie)
- $\Sigma_\perp = \Sigma \cup \{\perp\}$, $\mathbb{N}_\perp = \mathbb{N} \cup \{\perp\}$.

Sémantique dénotationnelle

Exemples :

- $\text{eval}_{Com}(x := 0, s) = s[x \mapsto 0]$
- $\text{eval}_{Exp}(x + 3, s) = s(x) + 3$ si $s(x)$ est défini, $\perp$ sinon.

Quels sont les domaines sémantiques associés ?

- *Com* : l'ensemble des fonctions totales de Σ vers $\Sigma_\perp$
- *Exp* : l'ensemble des fonctions totales de Σ vers $\mathbb{N}_\perp$
- *Bool* : l'ensemble des fonctions totales de Σ vers $\{\texttt{true}, \texttt{false}, \perp\}$

En effet, on a :

$$Com \times \Sigma \to \Sigma_\perp = Com \to (\Sigma \to \Sigma_\perp)$$

$$Exp \times \Sigma \to \mathbb{N}_\perp = Exp \to (\Sigma \to \mathbb{N}_\perp)$$

$$Bool \times \Sigma \to \{\texttt{true}, \texttt{false}, \perp\} = Bool \to (\Sigma \to \{\texttt{true}, \texttt{false}, \perp\})$$

Expressions et booléens

Les fonctions eval_{Exp} et eval_{Bool} sont définies (récursivement) comme les fonctions eval aux transparents 142 et 159.

Seules modifications :

- Recherche de la valeur d'une variable :

$$\text{eval}_{Exp}(x, s) = \begin{cases} s(x) & \text{si } s(x) \text{ est défini} \\ \bot & \text{sinon} \end{cases}$$

- Il faut faire remonter le $\bot$:

$$\text{eval}_{Exp}(E_1 + E_2, s) = \begin{cases} \text{eval}_{exp}(E_1) + \text{eval}_{exp}(E_2) & \text{si eval}_{exp}(E_1) \text{ et eval}_{exp}(E_2) \neq \bot \\ \bot & \text{sinon} \end{cases}$$

(Exercice : définissez complètement eval_{Exp} et eval_{Bool}.)

Commandes : assignation, `skip`, composition

La fonction eval_{Com} est définie également récursivement sur l'ensemble *Com*.

$$C \in Com \quad ::= \quad x := E \mid \texttt{if } B \texttt{ then } C \texttt{ else } C \mid C; C \mid \texttt{skip} \mid \texttt{while } B \texttt{ do } C$$

- Assignation :

$$\text{eval}_{Com}(x := E, s) = \begin{cases} s[x \mapsto \text{eval}_{Exp}(E, s)]] & \text{si eval}_{Exp}(E, s) \neq \bot \\ \bot & \text{sinon} \end{cases}$$

- `skip` :

$$\text{eval}_{Com}(\texttt{skip}, s) = s$$

- Composition de commandes :

$$\text{eval}_{Com}(C_1; C_2, s) = \begin{cases} \bot & \text{si eval}_{Com}(C_1, s) = \bot \\ \text{eval}_{Com}(C_2, \text{eval}_{Com}(C_1, s)) & \text{sinon} \end{cases}$$

Exemple

Montrez que $\text{eval}_{Com}(x := 0; x := x + 1, s) = s[x \mapsto 1]$.

$$\begin{aligned} & \text{eval}_{Com}(x := 0; y := x + 1) \\ = \; & \text{eval}_{Com}(y := x + 1, \text{eval}_{Com}(x := 0, s)) \\ = \; & \text{eval}_{Com}(x := x + 1, s[x \mapsto 0]) \\ = \; & s[x \mapsto \text{eval}_{Exp}(x + 1, s[x \mapsto 0])] \\ = \; & s[x \mapsto \text{eval}_{Exp}(x, s[x \mapsto 0]) + \text{eval}_{Exp}(1, s[x \mapsto 0])] \\ = \; & s[x \mapsto s[x \mapsto 0](x) + 1] \\ = \; & s[x \mapsto 0 + 1] \\ = \; & s[x \mapsto 1] \end{aligned}$$

Exercices : Montrez que pour tout $s \in \Sigma$:

- $eval_{Com}(x := y; y := x, s) = eval_{Com}(x := y, s)$
- $eval_{Com}(x := z; y := z, s) = eval_{Com}(y := z; x := z, s)$
- $eval_{Com}(C_1; (C_2; C_3), s) = eval_{Com}((C_1; C_2); C_3, s)$

Commandes : condition, while

- Condition :

$$\text{eval}_{Com}(\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2, s) = \begin{cases} \text{eval}_{Com}(C_1, s) & \text{si eval}_{Bool}(B, s) = \texttt{true} \\ \text{eval}_{Com}(C_2, s) & \text{si eval}_{Bool}(B, s) = \texttt{false} \\ \bot & \text{sinon} \end{cases}$$

- While : Une première idée :

$$\begin{aligned} & \text{eval}_{Com}(\texttt{while } B \texttt{ do } C, s) \\ = \; & \text{eval}_{Com}(\texttt{if } B \texttt{ then } (C; \texttt{while } B \texttt{ do } C) \texttt{ else skip}, s) \\ = \; & \ldots \text{eval}_{Com}(\texttt{while } B \texttt{ do } C, s') \ldots \end{aligned}$$

On se mord la queue...

Sémantique du `while`

Définition : Soit `diverge` un programme qui rentre tout de suite dans une boucle infinie, c'est-à-dire tel que pour tout $s \in \Sigma$:

$$\text{eval}_{com}(\texttt{diverge}, s) = \bot.$$

Soit un programme `while` B `do` C. Définissons (récursivement) les programmes suivants :

$$\begin{aligned} C_0 &= \texttt{diverge} \\ C_1 &= \texttt{if } B \texttt{ then } (C; \texttt{diverge}) \texttt{ else skip} \\ &\vdots \\ C_{n+1} &= \texttt{if } B \texttt{ then } (C; C_n) \texttt{ else skip}. \end{aligned}$$

Approximations du `while`

Propriétés des C_n : Etant donné un état initial s :

- Si, partant de s, la boucle doit être exécutée moins que n fois, alors C_n permet d'arriver au même état final que la boucle
- Si plus de n exécutions sont nécessaires, alors C_n donne comme état final $\bot$ qui est potentiellement différent de l'état final de la boucle
- Si $\text{eval}_{Com}(C_n, s) \neq \bot$, C_n a le même effet que la boucle sur s.

Quand n croît, C_n est identique à la boucle pour de plus en plus d'états initiaux $s \in \Sigma$.

La séquence $C_0, C_1, C_2, \ldots$ constitue une séquence d'approximateurs de qualité croissante de la boucle `while`.

Sémantique du `while`

En conséquence de la discussion du transparent précédent, la sémantique du `while` peut être définie par :

$$\text{eval}_{Com}(\texttt{while } B \texttt{ do } C, s) = \begin{cases} s' & \text{si } \text{eval}_{Com}(C_k, s) = s' \text{ pour un } k \\ \bot & \text{sinon} \end{cases}$$

Etant donné le théorème ci-dessous, cette expression définit bien une fonction (il n'y a pas plusieurs $s' \neq \bot$ tels que $\text{eval}_{Com}(C_k, s) = s'$ pour un C_k).

Théorème : Pour tout $n \in \mathbb{N}$ et tout $s \in \Sigma$, si $\text{eval}_{Com}(C_n, s) \neq \bot$, alors $\text{eval}_{Com}(C_{n+1}, s) = \text{eval}_{Com}(C_n, s)$.
Démonstration : Par induction sur n en se basant sur la définition de eval_{Com}. *(Exercice).*

Application

Utilisons la sémantique pour démontrer que le gardien d'une boucle est toujours faux après l'exécution de la boucle.

Lemme : Pour tout $n \in \mathbb{N}$ et $s \in \Sigma$, si $\text{eval}_{Com}(C_n, s) = s' \neq \bot$, alors $\text{eval}_{Bool}(B, s') = \texttt{false}$.

Démonstration : La preuve fonctionne par induction sur n.

Cas de base : $C_0 = \texttt{diverge}$, il n'y a donc pas de s' tel que $\text{eval}_{Com}(C_n, s) = s'$ et donc rien à prouver.

Cas inductif : Considérons le cas $n + 1$. Par définition,

$$C_{n+1} = \texttt{if } B \texttt{ then } (C; C_n) \texttt{ else skip}.$$

et donc par définition de eval_{com} :

$$\text{eval}_{Com}(C_{n+1}, s) = \begin{cases} \text{eval}_{Com}((C; C_n), s) & \text{si } \text{eval}_{Bool}(B, s) = \texttt{true} \\ s & \text{si } \text{eval}_{Bool}(B, s) = \texttt{false} \\ \bot & \text{sinon} \end{cases}$$

Si $\text{eval}_{Com}(C_{n+1}, s) = s'$, il y a deux cas possibles :

- $\text{eval}_{Bool}(B, s) = \texttt{false}$. Dans ce cas, $s = s'$ et donc $\text{eval}_{Bool}(B, s') = \texttt{false}$.
- $\text{eval}_{Bool}(B, s) = \texttt{true}$. Dans ce cas $s' = \text{eval}_{Com}((C; C_n), s)$.
  Par définition de eval_{Com} d'une composition, on a $s' = \text{eval}_{Com}(C_n, s'')$ avec $s'' = \text{eval}_{Com}(C, s)$.
  Par hypothèse inductive, on a donc $\text{eval}_{Bool}(B, s') = \texttt{false}$

□

Théorème : Si $\text{eval}_{Com}(\texttt{while } B \texttt{ do } C, s) = s' \neq \perp$, alors $\text{eval}_{Bool}(B, s') = \texttt{false}$.

Démonstration : Par la définition de la sémantique du `while`, si $\text{eval}_{Com}(\texttt{while } B \texttt{ do } C, s) = s'$, alors $s' = \text{eval}_{Com}(C_n, s)$ pour un n. Par le lemme précédent, on a donc bien $\text{eval}_{Bool}(B, s') = \texttt{false}$. □

Résumé

Ce qu'on a vu :

- Notion de syntaxe et sémantique d'un langage
- Sémantique opérationnelle (modélisation par une machine d'état)
- Sémantique dénotationnelle (définition récursive sur la syntaxe)
- Preuves de déterminisme de la sémantique opérationnelle
- Preuves de certaines propriétés des langages sur base de la sémantique

Au delà de cours

Syntaxe :

- Les langages qu'on a vu ici sont appelés des langages hors contexte. On peut définir des langages plus complexes, dépendant du contexte.
- La dérivation de la syntaxe abstraite à partir de la syntaxe concrète est non triviale pour la plupart des langages

Sémantique :

- Il existe d'autres types de sémantique. Par exemple :
  - Axiomatique : l'effet du programme est déterminé par la transformation de prédicats définis sur les variables (méthode d'invariant, triplet de Hoare, vue au chapitre 2).
  - Sémantique naturelle (ou opérationnelle à grand pas) : entre la sémantique opérationnelle et la sémantique dénotationnelle.
- Autres questions importantes en sémantique : prise en compte des fonctions, analyse de type...

(voir les cours INFO0016 (calculabilité) et INFO0085 (compilateurs))

Chapitre 5

Théorie des graphes

Plan

Sources : MCS, chapitres 10 et 11.

Introduction

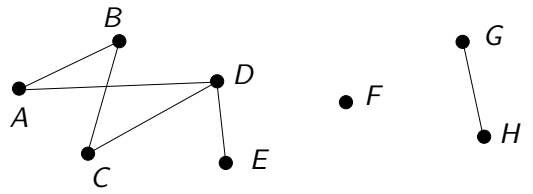
Définition : Un *graphe* est une paire $G = (V, E)$ où

- V est un ensemble fini mais non vide de *sommets*,
- E est un ensemble d'*arêtes*, chacune d'entre-elles étant un ensemble de deux sommets.

Remarques : Les sommets sont parfois appelés des *nœuds* et les arêtes des *arcs*.

Exemple :

- $V = \{A, B, C, D, E, F, G, H\}$
- $E = \{\{A, B\}, \{A, D\}, \{B, C\}, \{C, D\}, \{D, E\}, \{G, H\}\}$

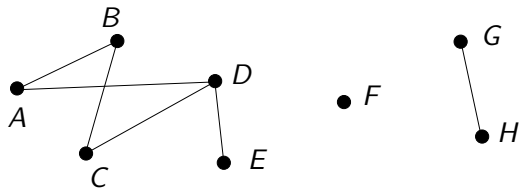


Notation : L'arête $\{A, B\} = \{B, A\}$ pourra être dénotée par A—B ou par B—A.

Définitions : Dans un graphe $G = (V, E)$,

- deux sommets $A, B \in V$ sont *adjacents* s'ils sont reliés par une arête, i.e, si A—$B \in E$;
- une arête A—B est *incidente* aux sommets A et B ;
- le *degré* d'un sommet est le nombre d'arêtes qui lui sont incidentes.

Exemple :



- A et B sont adjacents ;
- l'arête B—C est incidente à B et à C ;
- le degré de A est 2 ;
- le degré de D est 3 ;
- le degré de F est 0 ;
- le degré de G est 1.

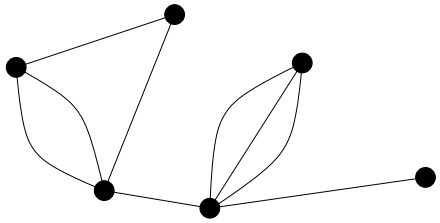
Sous-graphes

Définition : Soit un graphe $G = (V, E)$. Le graphe $G' = (V', E')$ est un *sous-graphe* de G si les conditions suivantes sont réunies :

- $V' \subseteq V$ et $V' \neq \emptyset$;
- $E' \subseteq E$;
- les sommets composant les arêtes de E' doivent appartenir à V'.

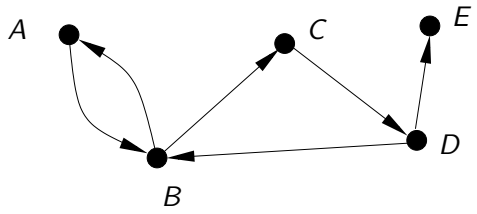
Extensions des graphes

Multigraphes : Une paire de sommets peut être connectée par *plus d'une arête*.



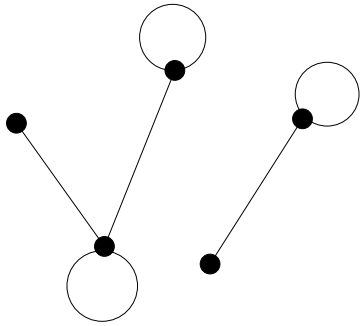
Graphes dirigés :

- Les arêtes sont des paires *ordonnées* de sommets.
- Une arête partant du sommet A et allant au sommet B est dénotée $A \longrightarrow B$.
- Le *degré intérieur* d'un sommet est le nombre d'arêtes arrivant à ce sommet.
- Le *degré extérieur* d'un sommet est le nombre d'arêtes sortant de ce sommet.



Le degré intérieur de D est 1 ; son degré extérieur est 2.

Boucles : On peut autoriser qu'un graphe contienne des boucles, c'est-à-dire qu'une arête ait pour extrémités le même sommet.



Notes :

- Des combinaisons sont possibles.
- Sauf lorsque cela sera spécifié expressément, un graphe sera toujours "simple" :
  - arêtes non dirigées,
  - pas de boucles,
  - au plus une arête entre deux sommets.

Applications

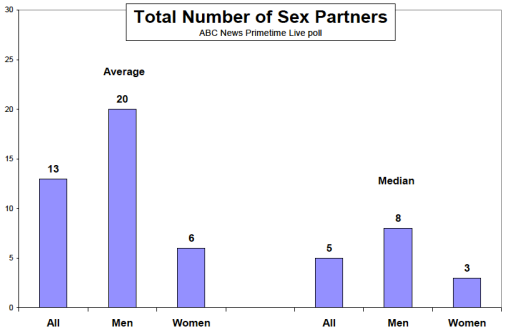
Les graphes peuvent être utilisés pour modéliser une grande variété de problèmes.

Exemples :

- cartes routières,
- connexions aériennes,
- WWW,
- réseaux sociaux,
- structures de données,
- etc.

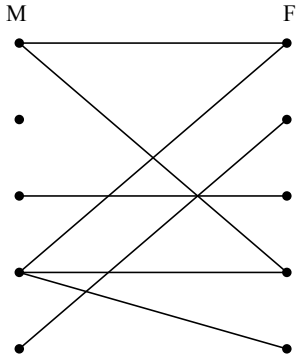
Un étude sur les comportements sexuels aux USA

- Une étude de la chaîne américaine ABC News a mené au résultat suivant :



- Ces résultats vous paraissent-ils sérieux ?

Le graphe des relations



- $G = (V, E)$ où V est divisé en deux sous-ensembles, les hommes M et les femmes F. Il y a une arête entre un homme et une femme s'ils ont eu une relation.
- Le graphe résultant est *biparti* (voir plus loin).

- Toute arête a exactement une extrémité dans M et une extrémité dans F. On a donc :

$$\sum_{x \in M} \deg(x) = \sum_{x \in F} \deg(x) = |E|$$

- En divisant les deux membres par $|M| \cdot |F|$, on obtient :

$$\frac{\sum_{x \in M} \deg(x)}{|M|} \cdot \frac{1}{|F|} = \frac{\sum_{x \in F} \deg(x)}{|F|} \cdot \frac{1}{|M|}$$

- qui donne directement :

$$\text{Avg. deg in } M = \frac{|F|}{|M|} \cdot \text{Avg. deg in } F$$

- Le rapport entre les degrés moyens ne dépend donc que du rapport entre les nombres d'hommes et de femmes dans la population
- D'après l'étude, on aurait :

$$20 = \frac{|F|}{|M|} \cdot 6,$$

c'est-à-dire 3 fois plus de femmes que d'hommes dans la population, ce qui est impossible.

Lemme des "poignées de main"

Lemme : La somme des degrés des sommets d'un graphe est égale à deux fois le nombre d'arêtes :

$$\sum_{x \in V} \deg(x) = 2 \cdot |E|$$

Démonstration : Chaque arête ajoute deux à la somme des degrés, un pour chacun de ses extrémités. □

Conséquences :

- Tout graphe a un nombre pair de sommets de degré impair.
- Exemple : il n'existe pas de graphe avec trois sommets de degrés respectivement 2,2, et 1.

Isomorphisme

Définition : Un *isomorphisme* entre des graphes $G = (V_G, E_G)$ et $H = (V_H, E_H)$ est une bijection $f : V_G \to V_H$ telle que :

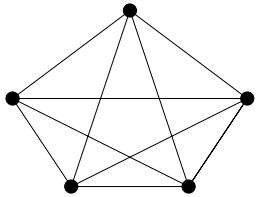
$$u—v \in E_G \text{ si et seulement si } f(u)—f(v) \in E_H$$

Deux graphes sont *isomorphes* quand il y a un isomorphisme entre eux.

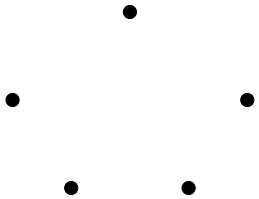
Des graphes isomorphes partagent la plupart de leurs propriétés : nombre de sommets, arêtes, patterns de degrés de sommets, etc.

Quelques graphes particuliers

- K_n, le graphe complet contenant n sommets :

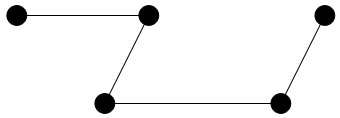


- Le *graphe vide* contenant n sommets :



- Un *chemin* est un graphe $G = (V, E)$ où
  - $V = \{v_1, v_2, \ldots, v_n\}$,
  - $E = \{v_1—v_2, v_2—v_3, \ldots, v_{n-1}—v_n\}$,
  - $n \geq 1$,
  - les sommets $v_1, v_2, \ldots, v_n$ sont tous distincts,
  - les sommets v_1 et v_n sont appelés les *extrémités* du chemin.

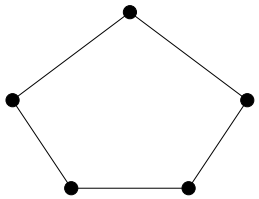
La *longueur* d'un chemin contenant n sommets est $n - 1$.



Soit $G = (V, E)$ un graphe, et $u, v \in V$. On dit qu'il *existe un chemin de u à v dans G* s'il existe un sous-graphe de G qui est un chemin à extrémités u et v.

- Un *cycle* est un graphe $G = (V, E)$ où
  - $V = \{v_1, v_2, \ldots, v_n\}$,
  - $E = \{v_1—v_2, v_2—v_3, \ldots, v_{n-1}—v_n, v_n—v_1\}$,
  - $n \geq 3$,
  - les sommets $v_1, v_2, \ldots, v_n$ sont tous distincts.

La *longueur* d'un cycle contenant n sommets est n.



Soit $G = (V, E)$ un graphe. Un cycle dans G est un sous-graphe de G qui est isomorphe à un cycle pour une longueur $n \geq 3$.

Parcours

Définition : Un *parcours* d'un graphe $G = (V, E)$ est une séquence de sommets et d'arêtes de la forme suivante :

$$v_0 \; v_0\text{—}v_1 \; v_1 \; v_1\text{—}v_2 \; v_2 \; \ldots \; v_{n-1} \; v_{n-1}\text{—}v_n \; v_n$$

où v_i—$v_{i+1} \in E$ pour $i = \{0, 1, \ldots, n-1\}$.

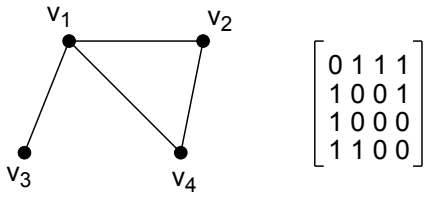
Si $v_0 = v_n$, alors le parcours est dit *fermé*. La longueur du parcours est égale au nombre d'arêtes n.

Remarques :

- Aussi appelé une *promenade*.
- Il existe un parcours entre deux sommets si et seulement si il existe un chemin entre ces sommets.
- Théorème : Le plus court parcours entre deux sommets est un chemin.

Matrices d'adjacence

- Un graphe $G = (V, E)$ avec pour sommets $V = \{v_1, v_2, \ldots, v_n\}$ peut être représenté par une matrice d'adjacence de taille $n \times n$. L'élément (i, j) de cette matrice vaut 1 si v_i—$v_j \in E$, 0 sinon.
- Par exemple :



Théorème : Soit G un graphe dirigé (potentiellement avec des boucles) avec pour sommets $v_1, v_2, \ldots, v_n$ et M sa matrice d'adjacence. $(M^k)_{ij}$ est égal au nombre de parcours de longueur k de v_i à v_j.

Démonstration :

- La démonstration fonctionne par induction sur k.
- Soit $P(k) =$ "$(M^k)_{ij}$ est égal au nombre de parcours de longueur k de v_i à v_j".
- *Cas de base ($k = 1$) :* Par définition de la matrice d'adjacence, $(M^1)_{ij} = M_{i,j} = 1$ s'il y a un chemin de longueur 1 entre v_i et v_j, 0 sinon.
- *Cas inductif :*
  - Supposons $P(k)$ vrai pour un $k \geq 1$.
  - Tout parcours de longueur $k+1$ entre v_i et v_j est constitué d'une chemin de longueur k de v_i à un certain sommet v_m suivi d'une arête v_m—v_j.

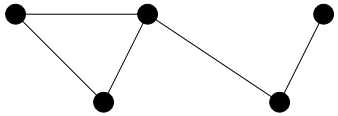
  - Le nombre de parcours de longueur $k+1$ entre v_i et v_j est donc égal à :
    $$(M^k)_{i1}M_{1j} + (M^k)_{i2}M_{2j} + \ldots + (M^k)_{in}M_{nj},$$
    qui est précisément la valeur de $(M^{k+1})_{ij}$.
  - Par induction, $P(k)$ est vrai pour $k \geq 1$. □

Application : La longueur du plus court chemin entre deux sommets v_i et v_j est la plus petite valeur de k tel que $M^k_{ij} \neq 0$.

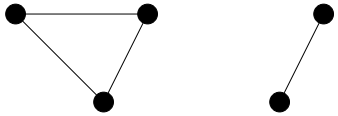
Graphes connexes

Définition : Un graphe $G = (V, E)$ est *connexe* si pour toute paire de sommets $u, v \in V$, il existe un chemin à extrémités u et v dans G.

Exemple de graphe connexe :



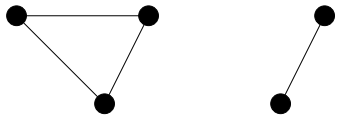
Exemple de graphe non connexe :



Composantes connexes

Définition : Soit $G = (V, E)$ un graphe. Une *composante connexe* de G est un sous-graphe connexe maximal, c'est-à-dire un sous-graphe connexe tel que l'ajout de tout sommet supplémentaire rend le sous-graphe non connexe.

Exemple :



Ce graphe contient 2 composantes connexes.

Théorème : Tout graphe $G = (V, E)$ contient au moins $|V| - |E|$ composantes connexes.

Démonstration :

- La démonstration fonctionne par induction sur le nombre d'arêtes.
- Soit $P(n)$ = "Tout graphe $G = (V, E)$ avec $|E| = n$ contient au moins $|V| - n$ composantes connexes".
- *Cas de base :* Dans un graphe sans arête, tout sommet est une composante connexe. Il y en a donc exactement $|V|$.
- *Cas inductif :*
  - Supposons que, pour un $n \in \mathbb{N}$, tout graphe contenant n arêtes possède au moins $|V| - n$ composantes connexes.
  - Soit $G = (V, E)$ un graphe contenant $n + 1$ arêtes.

  - Enlevons une arête arbitraire u—v de G.
  - Soit G' le sous-graphe résultant.
  - Par hypothèse d'induction, G' possède au moins $|V| - n$ composantes connexes.
  - Ajoutons l'arête u—v pour réobtenir le graphe G.
  - Si u et v étaient dans la même composante connexe de G', alors G a le même nombre de composantes connexes que G'.
  - Sinon, les composantes connexes dans lesquelles se trouvaient u et v dans G' se voient fusionnées, tandis que les autres composantes connexes restent inchangées. G a donc au moins $|V| - n - 1 = |V| - (n + 1)$ composantes connexes.
- Par induction, $P(n)$ est vrai pour tout $n \in \mathbb{N}$. □

Corollaire : Tout graphe connexe contenant n sommets possède au moins $n - 1$ arêtes.

Remarques à propos de la démonstration :

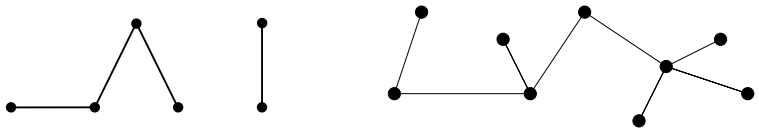
- L'induction sur le nombre d'arêtes ou sur le nombre de nœuds marchent souvent (mais pas toujours !) pour faire une démonstration sur des graphes
- On est parti d'un graphe de $k+1$ arêtes, on a retiré une arête arbitraire qu'on a ensuite remise dans le graphe.
- Cette approche est à préférer à une approche consistant à partir d'un graphe de k arêtes et à lui ajouter une arête supplémentaire *(Voir problème 11.30 de MCS)*

Arbres et forêts

Définition : Un graphe est *acyclique* si chacun de ses sous-graphes n'est pas un cycle.

Définition : Un graphe acyclique est appelé une *forêt*. Un graphe acyclique connexe est appelé un *arbre*.
Toute composante connexe d'une forêt est un arbre.

Exemple :



Définition : Dans un arbre, une *feuille* est un sommet de degré 1. (Dans l'exemple de droite, il y a 5 feuilles.)

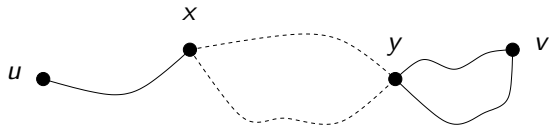
Théorème : Soit $T = (V, E)$ un arbre. Entre chaque paire de sommets, il y a un chemin unique.

Démonstration :

Existence : Le graphe étant connexe, il y a au moins un chemin entre chaque paire de sommets.

Unicité :

- Par l'absurde, supposons qu'il existe deux chemins différents entre les sommets u et v.
- En commencant par u, soit x le premier sommet à partir duquel les chemins divergent, et soit y le sommet suivant qu'ils partagent.



- Il existe deux chemins entre x et y sans arête commune. Ils définissent un cycle.
- C'est une contradiction car les arbres sont acycliques. □

Théorème : Soit $T = (V, E)$ un arbre.

1. Tout sous-graphe connexe de T est un arbre.
2. Ajouter une arête crée un cycle.
3. Retirer une arête rend le graphe non connexe.

Démonstration :

1. Un cycle d'un sous-graphe est cycle du graphe complet. Donc, un sous-graphe d'un graphe acyclique est acyclique. S'il est connexe, c'est un arbre par définition.
2. Une arête supplémentaire u—v entre le chemin unique à extrémités u et v crée un cycle.
3. - Supposons que l'on retire une arête u—v.
   - Le chemin unique entre u et v devait être u—v.
   - Il n'existe donc plus de chemin entre u et v.
   - Par conséquent, le graphe est devenu non connexe. □

Théorème : Soit $T = (V, E)$ un arbre contenant au moins 2 sommets. T contient au moins 2 feuilles.

Démonstration :

- Soit $v_1, \ldots, v_m$ la séquence de sommets d'un plus long chemin dans T.
- T contient au moins 2 sommets et est connexe. Donc, T contient au moins une arête.
- Il ne peut pas y avoir d'arête v_1—v_i, pour $2 < i \leq m$, sinon la séquence $v_1, \ldots, v_i$ formerait un cycle.
- Il ne peut pas y avoir d'arête u—v_1 où u n'est pas dans le chemin, sinon on pourrait allonger ce chemin.
- Seule arête incidente à v_1 : v_1—v_2. v_1 est donc une feuille.
- Un argument symétrique permet de montrer que v_m est une deuxième feuille. □

Théorème : Soit $T = (V, E)$ un arbre. On a $|V| = |E| + 1$.

Démonstration :

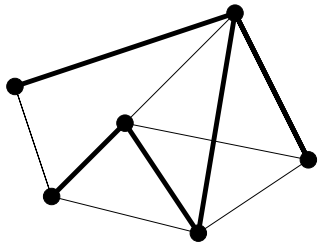
- La démonstration fonctionne par induction sur $|V|$.
- *Cas de base :* Si $|V| = 1$, on a $|E| + 1 = 0 + 1 = 1$.
- *Cas inductif :*
  - Supposons que le théorème soit vrai pour tout arbre contenant n sommets.
  - Soit $T = (V, E)$ un arbre contenant $n + 1$ sommets.
  - Soit v une feuille quelconque (elle existe car T contient au moins 1 arête, donc 2 sommets).
  - Soit $T' = (V', E')$ l'arbre obtenu en retirant v et son arête incidente.
  - On a $|V'| = |E'| + 1$.
  - En réinsérant v et son arête incidente, on obtient $|V| = |E| + 1$. □

Lemme : Un graphe $G = (V, E)$ est un arbre si et seulement si G est une forêt et $|V| = |E| + 1$.

Arbres couvrants

Définition : Soit $G = (V, E)$ un graphe. Un arbre $T = (V', E')$ est un *arbre couvrant* de G si $V' = V$ et $E' \subseteq E$.

Exemple :



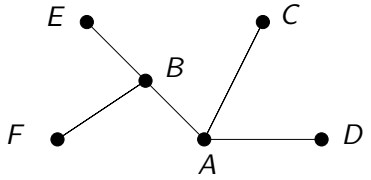
Théorème : Tout graphe connexe $G = (V, E)$ contient un arbre couvrant.

Démonstration :

- Par l'absurde, supposons que tout sous-graphe connexe $T = (V, E')$ de G soit nécessairement cyclique.
- Soit $T = (V, E')$ un de ces sous-graphes qui possède le plus petit nombre possible d'arêtes.
- T admet un cycle : v_0—v_1, v_1—$v_2, \ldots, v_n$—v_0.
- Supposons que l'on retire l'arête v_n—v_0. Soit x, y une paire de sommets quelconque.
  - Si x et y étaient connectés par un parcours ne contenant pas v_n—v_0, ils le restent.
  - Sinon, ils restent connectés par un parcours contenant le reste du cycle.
- Contradiction : T avait le plus petit nombre possible d'arêtes.
- Donc, T est acyclique. □

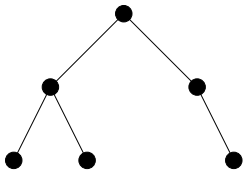
Arbres particuliers

- Arbre avec racine :
  - Un *arbre avec racine* est un arbre dans lequel un sommet est identifié comme étant la *racine*.
  - Soit u—v une arête d'un arbre avec racine telle que u est plus proche de la racine que v. Le sommet u est le *père* de v, et le sommet v est le *fils* de u.
  - Exemple :



Si A est la racine, alors E et F sont les fils de B, et A est le père de B, de C et de D.

- Un *arbre binaire* est un arbre avec racine dans lequel tout sommet a au plus 2 fils.



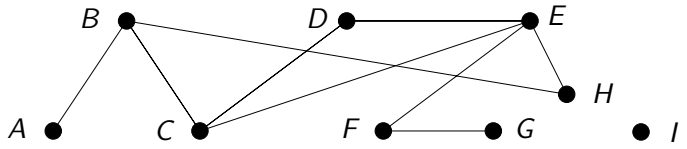
- Un arbre binaire est *ordonné* si les fils d'un sommet sont distingués : on les appelle *fils à gauche* et *fils à droite*.

Distance et diamètre

Définition :

- La *distance* entre deux sommets d'un graphe est la longueur du plus court chemin entre eux.
- Si un tel chemin n'existe pas, la distance entre les deux sommets est dite "infinie".

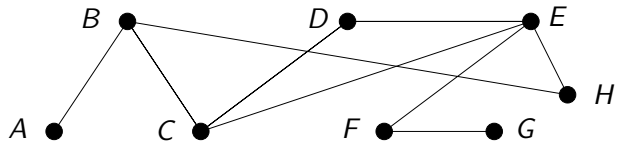
Exemple :



- la distance entre G et C est 3,
- la distance entre A et lui-même est 0,
- la distance entre I et n'importe quel autre sommet est infinie.

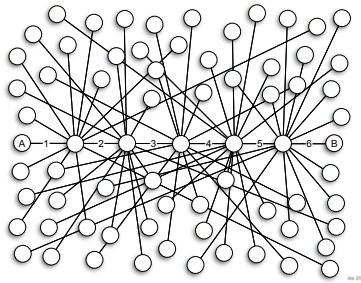
Définition : Le *diamètre* d'un graphe est la distance entre les deux sommets les plus éloignés.

Exemple :



- Les sommets qui sont les plus distants sont A et G.
- Leur distance est de 5.
- Le graphe a donc un diamètre de 5.

Six degrés de séparation



(wikipedia)

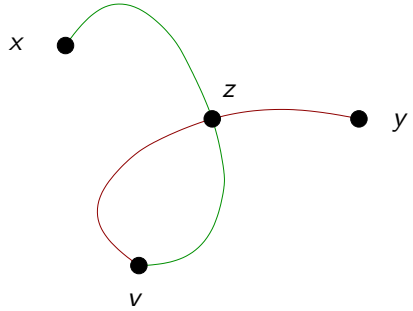
- Théorie selon laquelle deux personnes quelconques sur la planète peuvent être reliées au travers d'une chaîne d'au plus 6 relations.
- Ou de manière équivalente, le diamètre du réseau social global est au plus 6.
- Le distance moyenne a été mesuré à 6.5 (sur base de msn, facebook, etc.).

Théorème : Soit v un sommet arbitraire d'un graphe G. Si tout sommet est distant de v d'une distance au plus égale à d, alors le diamètre de G est borné par $2d$.

Démonstration :

- Soient x et y deux sommets quelconques de G.
- Il existe un chemin π_1 de longueur au plus égale à d entre x et v.
- Il existe un chemin π_2 de longueur au plus égale à d entre v et y.
- Soit z le sommet qui se trouve à la fois sur π_1 et sur π_2, et qui est le plus proche possible de x. (Un tel z existe toujours car, au pire, il pourrait être v.)
- On obtient un chemin entre x et y de longueur au plus égale à $2d$ en joignant le segment de x et z à celui entre z et y. □

Illustration :



Coloriage de graphes

Problème : l'apparitorat de la faculté doit mettre au point l'horaire des examens de la session de juin.

Contraintes :

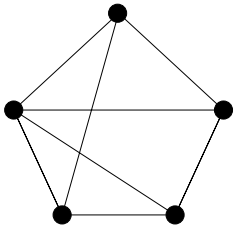
- Un étudiant ne peut pas participer à deux examens en même temps.
- La période d'examens doit être la plus courte possible.

Question : De quelle manière la théorie des graphes peut-elle nous aider à modéliser ce problème ?

Solution : Soit un graphe avec

- un sommet par cours,
- une arête entre deux sommets si au moins un des étudiants suit les deux cours.

Exemple :

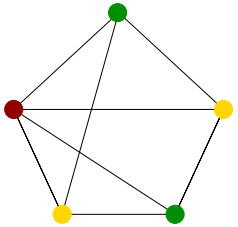


Associons une couleur à chaque plage horaire :

- lundi matin
- lundi après-midi
- mardi matin
- ...

Il est possible d'organiser l'examen sur *n plages horaires* si et seulement si il est possible de *colorier* les sommets du graphe à l'aide de *n couleurs* de telle manière que pour toute paire de sommets adjacents, ces sommets soient coloriés différemment.

Exemple :



Autres applications : allocation de registres, allocation de fréquences de station radio, coloriage de cartes...

k-coloriages

Définition : Un graphe G est *k-coloriable* s'il existe un ensemble C de k couleurs tel que chaque sommet puisse être colorié avec une couleur $c \in C$ sans que deux sommets adjacents ne partagent la même couleur.

Remarque : Tout graphe k-coloriable est nécessairement $(k+1)$-coloriable.

Définition : Le *nombre chromatique* $\chi(G)$ d'un graphe G est le plus petit nombre de couleurs nécessaires pour colorier le graphe G.

Un graphe est k-coloriable ssi $\chi(G) \leq k$.

Théorème : Soit $k \in \mathbb{N}_0$, et soit G un graphe dont chaque sommet est au plus de degré k. Le graphe G est $(k+1)$-coloriable.

Démonstration :

- La démonstration fonctionne par induction sur le nombre n de sommets de G.
- Soit $P(n) =$ "Tout graphe à n sommets de degrés au plus égaux à k est $(k+1)$-coloriable".
- *Cas de base :* $P(1)$ est vrai, car tout graphe à 1 sommet est 1-coloriable.

- *Cas inductif :* Supposons que $P(n)$ soit vrai.
  - Soit G un graphe à $n+1$ sommets, chacun de degré au plus égal à k.
  - Retirons de G un sommet v arbitraire, ainsi que ses arêtes incidentes. Soit G' le graphe résultant.
  - G' est $(k+1)$-coloriable.
  - Ajoutons le sommet v et ses arêtes incidentes.
  - Le degré de v est au plus égal à k, et $k+1$ couleurs sont disponibles.
  - Associons à v une couleur différente de tous ses sommets adjacents.
  - G est donc $(k+1)$-coloriable.
- Par induction, $P(n)$ est vrai pour tout $n \in \mathbb{N}_0$. □

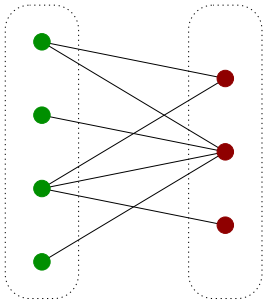
Graphes bipartis

Définition : Un graphe *biparti* est un graphe dont les sommets peuvent être divisés en deux sous-ensembles disjoints $L(G)$ et $R(G)$ tels que toute arête a une extrémité dans $L(G)$ et l'autre extrémité dans $R(G)$.

Lemme : Un graphe G est *biparti* ssi il est 2-coloriable.

Propriété : Soit G un graphe biparti. Si deux sommets u, v de G sont adjacents, alors dans tout 2-coloriage de G, un des deux sommets sera colorié avec une couleur, et l'autre sommet sera colorié avec la couleur restante.

Tout graphe biparti peut donc être représenté d'une façon similaire à la suivante :

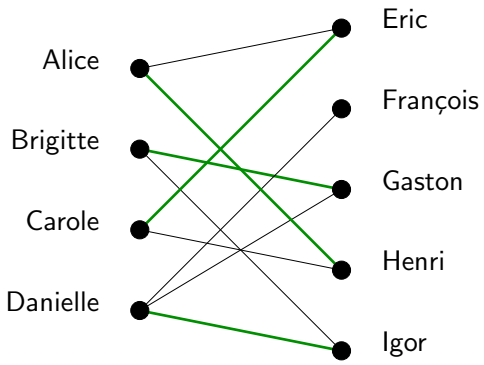


Théorème : Un graphe est biparti si et seulement s'il ne contient aucun cycle de longueur impaire.

Théorème de Hall

Données : Une groupe contient un certain nombre de filles et de garçons. Chaque fille aime certains garçons.

Problème : Sous quelles conditions chaque fille peut-elle être mariée à un garçon qu'elle aime ?



Définition : L'*ensemble des garçons aimés par un ensemble de filles* est l'ensemble des garçons aimés par au moins une de ces filles.

Condition de mariage : *Tout sous-ensemble des filles aime au moins un ensemble de garçons aussi grand.*

Exemple : Il est impossible de trouver une correspondance si un ensemble de 4 filles aime un ensemble composé de seulement 3 garçons.

Théorème : Il est possible de trouver une correspondance entre un ensemble F de filles et un ensemble G de garçons si et seulement si la condition de mariage est satisfaite.

Démonstration :

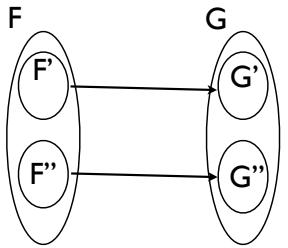
$\boxed{\Rightarrow}$

- Considérons une correspondance possible.
- Soit F' un sous-ensemble quelconque de F.
- Chaque fille $f \in F'$ aime au moins le garçon avec lequel elle est mariée.
- Donc, la condition de mariage est satisfaite.

$\boxed{\Leftarrow}$

- Supposons que la condition de mariage soit satisfaite, et montrons qu'il existe une correspondance.
- La démonstration fonctionne par induction forte sur $|F|$.
- *Cas de base :* Si $|F| = 1$, une correspondance existe.
- *Cas inductif :* Supposons $|F| \geq 2$. Deux cas possibles :
  - Tout sous-ensemble propre des filles aime un ensemble de garçons *strictement plus grand*.
    - Dans ce cas, on peut marier une fille quelconque avec un garçon qu'elle aime.
    - La condition de mariage est satisfaite pour les personnes restantes.
    - On peut trouver une correspondance par induction.

- Il existe un ensemble de filles $F' \subset F$ qui aime un ensemble de garçons $G' \subseteq G$ tel que $|G'| = |F'|$.
  - On peut marier les filles de F' avec les garçons de G' par induction.
  - Montrons que la condition de mariage est satisfaite pour les garçons et filles restantes.
  - Soit un sous-ensemble quelconque $F'' \subseteq (F \setminus F')$. Soit G'' l'ensemble des garçons de $G \setminus G'$ aimés par F''.
  - Il faut montrer que $|F''| \leq |G''|$.
  - Comme $F' \cup F''$ aime $G' \cup G''$, on a $|F' \cup F''| \leq |G' \cup G''|$.
  - Comme $|F'| = |G'|$, on a $|F''| \leq |G''|$. □



Un énoncé formel

Définition : Soit S un sous-ensemble des sommets d'un graphe. $N(S)$ est défini par le nombre de sommets n'appartenant pas à S, mais adjacents à au moins un sommet de S.

Théorème : Soit $G = (L \cup R, E)$ un graphe biparti tel que toute arête a une extrémité dans L et l'autre extrémité dans R. Il existe une correspondance pour les sommets de L si et seulement si $|S| \leq |N(S)|$ pour tout $S \subseteq L$.

Problème des mariages stables

Données : Un groupe contient un nombre identique N d'hommes et de femmes. Chacun des hommes et des femmes détermine un ordre de préférence sur les personnes de sexe opposé.

Homme	Préférence
1	A B C
2	A C B
3	C A B

Femme	Préférence
A	3 1 2
B	3 2 1
C	1 2 3

Problème : Marier les hommes et les femmes de manière à satisfaire *au mieux* les préférences de chacun.

Problème des mariages stables

Définitions : Un ensemble de mariages est *instable* s'il y a un homme et une femme qui se préfèrent à leurs époux respectifs. On appelle un tel couple un couple *fripon*. Un ensemble de mariages est stable s'il ne contient pas de couple fripon.

Exemple : Si 1 est marié à A, 3 à C et 2 à B, alors 2 et C forment un couple fripon et donc l'ensemble de mariages est instable.

Reformulation du problème : Marier tous les hommes et femmes de manière à obtenir un ensemble de mariages stable.

Notes :

- Problème défini en 1962 par Gale et Shapley.
- De nombreuses variantes existent
- Applications nombreuses : assignation des internes à des hopitaux, agences de rencontres, assignation de trafic sur des serveurs...

Un algorithme

Solution : Répéter le processus suivant :

- Chaque homme se présente à la femme en tête de sa liste de préférence courante (si cette liste est non vide).
- Chaque femme demande à être retirée de la liste de tous les hommes qui se présentent à elle excepté leur favori parmi ceux-ci.

L'itération s'arrête lorsque chaque femme a au plus un homme qui se présente à elle et c'est avec cet homme qu'elle se marie.

Exemple : Sur le problème précédent, on arrive à l'ensemble de mariage $\{1-B, 2-C, 3-A\}$

Montrons que cet algorithme :

- se termine
- est partiellement correct : S'il se termine, tous les hommes et femmes sont en couple et l'ensemble des mariages obtenu est stable

Terminaison

Théorème : L'algorithme se termine après au plus N^2 itérations.

Démonstration : Soit la somme f des tailles des listes de préférences des hommes. On a :

- $f \in \mathbb{N}$.
- Initialement, f est égale à N^2.
- A chaque itération, au moins un homme supprime une femme de sa liste (et aucun n'ajoute de femme dans sa liste). f est donc strictement décroissante.

Par le théorème du transparent 105, on en conclut que l'algorithme se termine après au plus N^2 itérations. □

Correction partielle

Définition : Soit P le prédicat : pour toute femme w et tout homme m, si w a été supprimée de la liste de m, alors w a un prétendant qu'elle préfère à m.

Lemme : P est un invariant de l'algorithme du transparent 270.

Démonstration : Par induction sur le nombre d'itérations.
Cas de base : Initialement, aucune femme n'a été supprimée des listes de préférences des hommes. Le prédicat est donc vrai.
Cas inductif : Supposons P vérifié après d itérations et soit une femme w supprimée de la liste de m après l'itération $d+1$. Deux cas possibles :

- Cas 1 : w a été supprimée à l'itération $d+1$. Alors, w a un prétendant qu'elle a préféré à m à l'itération $d+1$.
- Cas 2 : w a été supprimée avant l'itération $d+1$. Puisque P est vrai après l'itération d, w a un prétendant qu'elle préfère à m après l'itération d. Elle a donc le même prétendant ou un meilleur après l'itération $d+1$.

Dans les deux cas, P est vrai après l'itération $d+1$ et est donc un invariant de l'algorithme. □

Correction partielle

Théorème : Tous les hommes et les femmes sont en couple lorsque l'algorithme se termine.

Démonstration :

- La démonstration fonctionne par contradiction.
- Supposons qu'une personne ne soit pas mariée.
- Vu qu'il y a autant d'hommes que de femmes et qu'un mariage implique exactement un homme et une femme, il y a au moins un homme, m, et une femme, w, qui ne sont pas mariés.
- Puisque m n'est pas marié, sa liste doit être vide. Par l'invariant, chaque femme a donc un prétendant qu'elle préfère à m.
- Comme l'algorithme s'est terminé, chaque femme a du être mariée à ce prétendant, ce qui contredit le fait que w n'est pas mariée.

□

Correction partielle

Théorème : L'algorithme produit un ensemble de mariages stable.

Démonstration : Soit un homme m et une femme w qui ne sont *pas* mariés l'un à l'autre en sortie de l'algorithme. Montrons que m et w ne forment pas une couple fripon. Il y a deux cas possibles :

- w n'est pas sur la liste de m. Par l'invariant, w a un prétendant (qui est son mari) qu'elle préfère à m et donc ils ne forment pas un couple fripon.
- w est sur la liste de m. Puisque m n'est pas marié à w, il s'est présenté (seul) face à une autre femme qu'il préfère donc à w.

□

Corrolaire : Quelles que soient les préférences, il est toujours possible de trouver un ensemble de mariages stable.

Equité

L'algorithme est-il équitable ? Non !

Définitions :

- Etant donné des listes de préférences, une personne p_1 est un partenaire possible de p_2 s'il existe un ensemble stable de mariages où p_1 et p_2 sont mariés.
- Le partenaire *optimal* d'une personne est celui qu'elle préfère parmi ses partenaires possibles. Son *pire* partenaire est celui qu'elle aime le moins parmi tous les partenaire possibles.

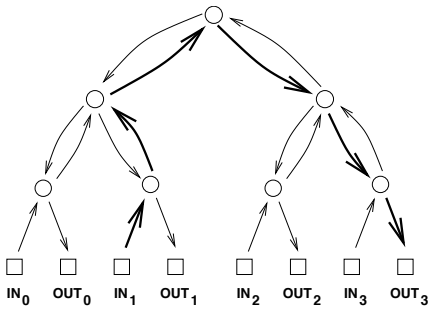
Théorème : L'algorithme du transparent 270 marie tous les hommes à leur partenaire optimale et toutes les femmes à leur pire partenaire.

Démonstration : Le preuve fonctionne par contradiction. Laissée à titre d'exercice.

Réseaux de communication

On souhaite envoyer des paquets sur un réseau informatique entre des nœuds d'entrée et des nœuds de sortie (ordinateurs, téléphones, etc.) connectés entre eux par des commutateurs ("switches") chargés de dispatcher les paquets sur le réseau.

Le réseau peut être représenté par un graphe dirigé.



Réseaux de communications

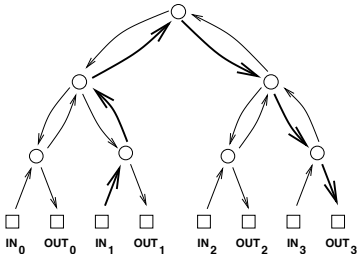
Hypothèses de travail :

- Les paquets transmis entre nœuds sont de taille constante
- La transmission d'un paquet sur un lien se fait en un temps constant, indépendant du lien
- Il y a exactement autant de nœuds d'entrée que de nœuds de sortie et chaque nœud d'entrée cherche à transmettre un paquet à un nœud de sortie distinct.
- Le nombre de nœuds est une puissance exacte de 2.

Objectif : on cherche à mettre au point un réseau de manière à minimiser :

- Le nombre de commutateurs et leur taille (nombre d'entrées et de sorties)
- Le diamètre du réseau
- Les congestions (le nombre de paquets passant par un même commutateur

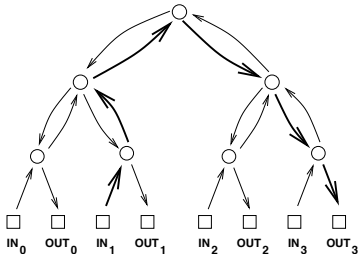
Nombre et taille des commutateurs



Taille maximale des commutateurs : 3×3 (3 entrées, 3 sorties)

Nombre de commutateurs : $2N - 1$ (pour N nœuds d'entrée)

Diamètre



Définition : Le diamètre d'un réseau de communication est défini comme la longueur maximale d'un plus court chemin entre une entrée et une sortie.

Exemple : Dans l'exemple, cette distance correspond à la distance entre l'entrée 0 et la sortie 3, c'est-à-dire 6. En général, le diamètre d'un arbre binaire complet à N entrées/sorties est $2 \log_2 N + 2$

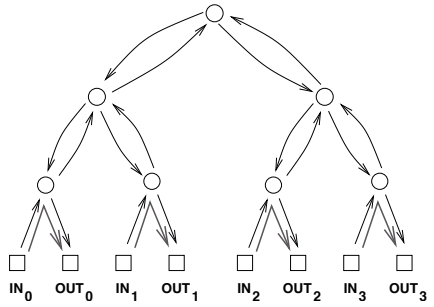
Routage

Définitions :

- Un problème de *routage* est défini par une permutation $\pi : \{1, \ldots, N\} \to \{1, \ldots, N\}$ des nœuds, où $\pi(i)$ est le nœud de sortie auquel le nœud i souhaite envoyer un paquet.
- Un *routage* P, solution d'un problème de routage π, est un ensemble de parcours P_i de chaque entrée i vers sa sortie $\pi(i)$.

Exemple :

- Soit le problème de routage $\pi(i) = i$ pour l'arbre binaire complet.
- Un routage possible est obtenu en prenant comme parcours P_i le chemin de l'entrée i vers la sortie i passant directement pas le commutateur qui les relie.

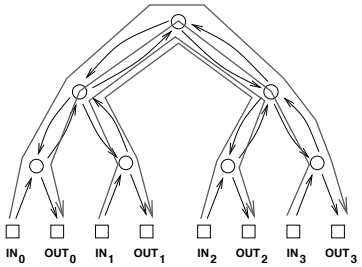


Congestion

Définition :

- La *congestion* d'un routage P est le nombre maximum de parcours dans P qui traversent un même commutateur.
- Etant donné une permutation π, un routage est optimal par rapport aux congestions s'il minimise la congestion parmi tous les routages qui résolvent π.
- La *congestion d'un réseau* est la congestion maximale sur toutes les routages optimaux.

Exemple : Dans le cas de l'arbre binaire complet, la congestion maximale est N, qui correspond à $\pi(i) = (N-1) - i$.

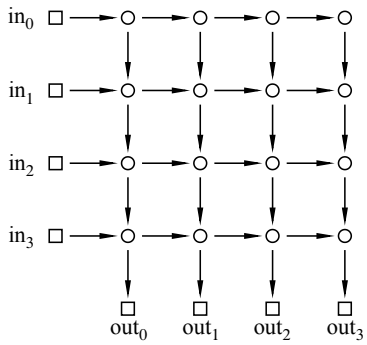


Synthèse sur les arbres binaires complets

Réseau	Diamètre	Taille commut.	#commut.	congestion
Arbre binaire complet	$2\log_2 N + 2$	3×3	$2N - 1$	N

- Le nombre de commutateurs est proche de l'optimum étant donné leur taille. Le diamètre est bon également.
- On ne peut pas faire pire en matière de congestion.

Tableau à deux dimensions



Nombre et taille des commutateurs : N^2 commutateurs de taille 2×2

Diamètre : $2N$

Tableau à deux dimensions

Théorème : La congestion d'un tableau à N entrées est 2.

Démonstration :

Montrons d'abord que la congestion est au plus 2 :

- Soit π une permutation. Une solution P pour π est l'ensemble des chemins P_i où P_i se déplace à droite à partir de l'entrée i jusqu'à la colonne $\pi(i)$ et puis descend jusqu'à la sortie $\pi(i)$.
- Le commutateur à la ligne i et la colonne j transmet au plus 2 paquets : celui qui vient de l'entrée i et celui qui va vers la sortie j.

Montrons que la congestion est au moins 2 :

- Quel que soit π tel que $\pi(0) = 0$ et $\pi(N-1) = N-1$ deux paquets doivent passer par le commutateur en bas à gauche

□

Tableau à deux dimensions

Réseau	Diamètre	Taille commut.	#commut.	congestion
Arbre binaire complet	$2\log_2 N + 2$	3×3	$2N - 1$	N
Tableau à 2 dim.	$2N$	2×2	N^2	2

- La congestion est minimale
- Le nombre de commutateurs est prohibitif

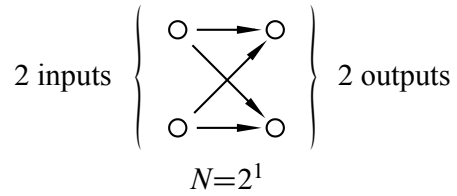
Peut-on faire mieux que ces deux réseaux ?

Réseau de Benés

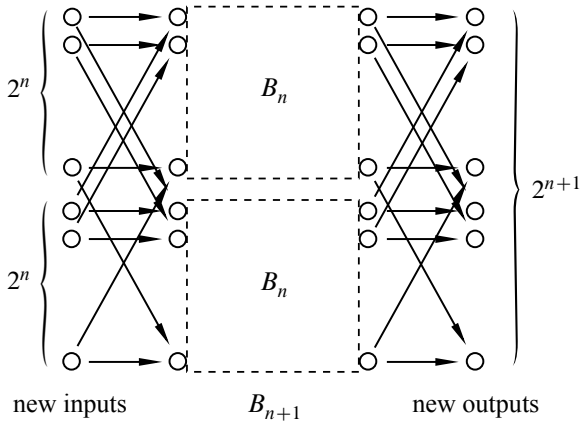
Le réseau de Benés est bon selon tous les critères.

B_n est le réseau de Benés avec $N = 2^n$ entrées et sorties. On le définit de manière récursive :

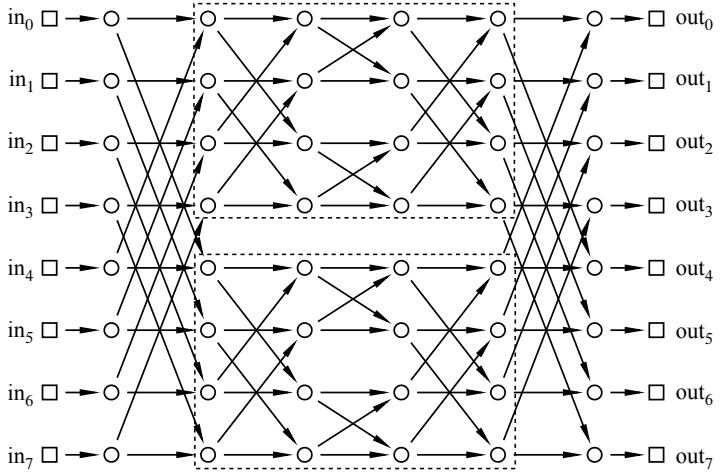
- Cas de base : B_0 est donné par :



- Cas récursif : B_{n+1}



Exemple : B_3



Propriétés : commutateurs et diamètre

Le nombre de commutateurs (2×2) pour N entrées, $C(N)$, est donné par :

$$C(N) = \begin{cases} 4 & \text{si } N = 2, \\ 2C(N/2) + 2N & \text{si } N > 2 \end{cases}$$

$$\Rightarrow C(N) = 2N \log_2 N$$

Diamètre :

- Tous les chemins entre entrées et sorties ont la même longueur donnée par $L(N) + 2$ où :

$$L(N) = \begin{cases} 1 & \text{si } N = 2, \\ L(N/2) + 2 & \text{si } N > 2 \end{cases}$$

$$\Rightarrow L(N) = 2 \log_2 N - 1$$

- Le diamètre est $2 \log_2 N + 1$.

(Exercice : prouvez les formes analytiques de $C(N)$ et $L(N)$ par induction)

Propriété : congestion

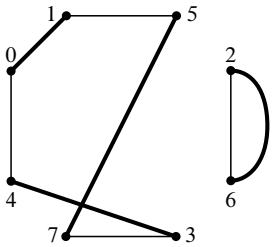
Théorème : La congestion du réseau B_n est 1.

Démonstration : La preuve fonctionne par induction sur n où $N = 2^n$.
$P(n) =$"la congestion de B_n est 1".
Cas de base ($n = 1$) : Etant donné B_1, il n'y a qu'un seul routage et sa congestion est 1.
Cas inductif : Supposons que la congestion de B_n soit 1 et montrons que la congestion de B_{n+1} est aussi 1.
Soit π une permutation arbitraire de $\{0, 1, \ldots, N-1\}$. Soit G le graphe dont les sommets sont les numéros d'entrée 0,1,..., N-1 et dont les arêtes sont l'union des deux ensembles :

- $E_1 = \{u—v| \ |u—v| = N/2\}$, et
- $E_2 = \{u—w| \ |\pi(u)—\pi(v)| = N/2\}$, et

(G n'est pas nécessairement simple)

Exemple pour B_3 et $\pi = \{1, 5, 4, 7, 3, 6, 0, 2\}$.



E_1 connecte les entrées qui ne peuvent pas être envoyées vers le même sous-réseau, haut ou bas, sous peine d'avoir une congestion supérieure à 1.
E_2 connecte les entrées dont les sorties correspondantes ne doivent pas être atteintes à partir du même sous-réseau, haut ou bas, sous peine d'avoir une congestion supérieure à 1.

S"il existe un 2-coloriage de G, le routage qui envoie les paquets d'une couleur vers le sous-réseau haut (de type B_n) et les paquets de l'autre couleur vers le sous-réseau bas (de type B_n également) donne une congestion de 1.

C'est un conséquence de la définition du graphe G et de l'hypothèse inductive :

- Les paquets n'entrent pas en collision à l'entrée et à la sortie (par définition de G)
- Il n'y a pas non plus de collision dans le passage au travers des sous-réseaux B_n haut et bas (par hypothèse inductive)

Il suffit donc de montrer que le graphe G est 2-coloriable quel que soit la permutation π.

Chaque nœud de G est incident à exactement une arête de E_1 et une arête de E_2 et a donc un degré 2. Le graphe est donc un ensemble de cycles disjoints.
Pour montrer qu'il est 2-coloriable, il faut montrer que chacun de ses cycles est de longueur paire :

- Si on avait un cycle de longueur impaire, il devrait y avoir dans ce cycle soit deux arêtes successives de E_1, soit deux arêtes successives de E_2.
- Le nœud incident à ces deux arêtes aurait donc soit deux arêtes dans E_1, soit deux arêtes dans E_2.
- Ce n'est pas possible vu la définition de G.

On en déduit donc que G est 2-coloriable et donc que la congestion de B_n est 1.

□

Note : le théorème donne un algorithme pour trouver un routage optimal pour une permutation donnée.

Réseau de Bénes

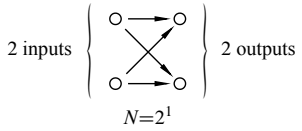
Réseau	Diamètre	Taille commut.	#commut.	congestion
Arbre binaire complet	$2\log_2 N + 2$	3×3	$2N - 1$	N
Tableau à 2 dim.	$2N$	2×2	N^2	2
Réseau de bénes	$2\log N + 1$	2×2	$2N \log N$	1

Combine les avantages des deux autres réseaux :

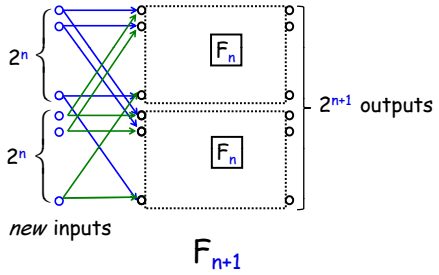
- Nombre de commutateurs et diamètre faibles
- Pas de congestion

Réseau butterfly

Cas de base : F_1



Cas inductif : F_{n+1}



Exercice : étudiez les propriétés de ce réseau.

Théorie des graphes

On a seulement effleuré quelques sujets importants en théorie des graphes.

Autres thématiques importantes :

- Planarité (voir chapitre 12 de MCS)
- Algorithmique sur les graphes (voir cours de SDA et techniques de programmation)
- Problème de flots
- Cycles eulériens et hamiltoniens
- Coupes de graphe
- ...

Partie 3

Outils pour l'analyse d'algorithmes

Analyse d'algorithmes

Dans cette partie du cours, on va voir des outils permettant *d'analyser des algorithmes* :

- c'est-à-dire d'évaluer leur coût, en termes de temps de calcul, nombres d'opérations, ou encore utilisation de la mémoire.

Matière :

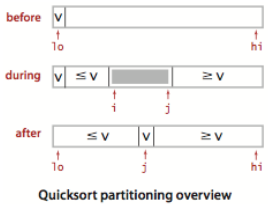
- Sommations et notations asymptotiques
- Récurrences
- Fonctions génératrices (pour la résolution de récurrence et le dénombrement)

Sources :

- MCS
- R. Sedgewick et P. Flagolet, *Analysis of Algorithms*, Addison-Wesley, 1995. `http://aofa.cs.princeton.edu/`.
- J. L. Gross, *Combinatorial methods with computer applications*, Chapman & Hall, 2008.

Exemple introductif : quicksort

```
PARTITION(A, lo, hi)
1  i = lo ; j = hi + 1 ; v = A[lo]
2  while (true)
3      repeat i = i + 1 until A[i] >= v
4      repeat j = j − 1 until A[j] <= v
5      if (i >= j)
6          break
7      swap(A[i], A[j])
8  swap(A[lo], A[j]) ;
9  return j
```



Quicksort partitioning overview

`http://algs4.cs.princeton.edu`

```
QUICKSORT(A, lo, hi)
1  if begin < end
2      q = PARTITION(A, lo, hi)
3      QUICKSORT(A, lo, q − 1)
4      QUICKSORT(A, q + 1, hi)
```


Approche scientifique pour l'analyse d'algorithmes

Pour analyser un algorithme : *Sur papier*

- Identifier une *opération abstraite* au cœur de l'algorithme.
- Développer un *modèle des entrées* de l'algorithme.
- Déterminer la fréquence d'exécution C_N de l'opération pour une entrée de taille N.
- Faire l'hypothèse que le coût de l'algorithme est $\sim aC_N$ où a est une constante.

Validation du modèle : *Sur ordinateur*

- Développer un générateur d'entrées selon le modèle
- Calculer a en exécutant l'algorithme pour des entrées larges
- Vérifier le résultat sur des entrées encore plus larges
- Valider le modèle d'entrée en testant l'algorithme sur une application réelle.

Quicksort

```
PARTITION(A, lo, hi)
1  i = lo ; j = hi + 1 ;v = A[lo]
2  while (true)
3       repeat i = i + 1 until A[i] >= v
4       repeat j = j − 1 until A[j] <= v
5       if (i >= j)
6            break
7       swap(A[i], A[j])
8  swap(A[lo], A[j]) ;
9  return j
```

```
QUICKSORT(A, lo, hi)
1  if begin < end
2       q = PARTITION(A, lo, hi)
3       QUICKSORT(A, lo, q − 1)
4       QUICKSORT(A, q + 1, hi)
```

- Opération de base : comparaison
- Modèle d'entrée :
  - Tableau A ordonné aléatoirement
  - Toutes les valeurs de A sont différentes
- Hypothèse : temps de calcul est $\sim aC_N$ où a est une constante et C_N est le nombre de comparaisons.

Modèle mathématique

Etant donné le modèle :

- Nombre de comparaisons pour le partitionnement : $N+1$
- Probabilité que le pivot soit à la position k : $1/N$
- Tailles des sous-tableaux dans ce cas-là : $k-1$ et $N-k$
- Les sous-tableaux sont aussi triés aléatoirement

Le nombre *moyen* de comparaisons utilisées par le quicksort est donné par la récurrence suivante :

$$C_N = N + 1 + \sum_{k=1}^{N} \frac{1}{N}(C_{k-1} + C_{N-k})$$

Essayons de dériver une formulation analytique de cette récurrence (voir *chapitre 7*).

Formulation analytique

$$C_N = N + 1 + \sum_{k=1}^{N} \frac{1}{N}(C_{k-1} + C_{N-k})$$

Par symétrie :

$$C_N = N + 1 + \frac{2}{N}\sum_{k=1}^{N} C_{k-1}$$

En multipliant par N :

$$NC_N = N(N+1) + 2\sum_{k=1}^{N} C_{k-1}$$

En soustrayant la même formule pour $N-1$:

$$NC_N - (N-1)C_{N-1} = 2N + 2C_{N-1}$$

En rassemblant les termes :

$$NC_N = (N+1)C_{N-1} + 2N$$

On divise par $N(N+1)$:

$$\frac{C_N}{N+1} = \frac{C_{N-1}}{N} + \frac{2}{N+1}$$

Téléscopage :

$$\begin{aligned} \frac{C_N}{N+1} &= \frac{C_{N-1}}{N} + \frac{2}{N+1} = \frac{C_{N-2}}{N-1} + \frac{2}{N} + \frac{2}{N+1} \\ &= \frac{C_1}{2} + \frac{2}{3} + \ldots + \frac{2}{N} + \frac{2}{N+1} \end{aligned}$$

Simplification :

$$C_N = 2(N+1)\sum_{k=1}^{N} \frac{1}{k} - 2N$$

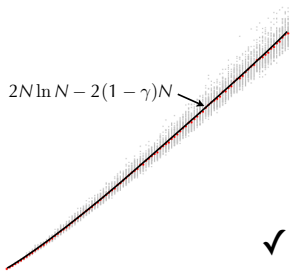
Approximation de la somme (*voir chapitre 6*)

$$C_N \sim 2N \ln N - 2(1-\gamma)N,$$

où $\gamma = 0.57721$.

Validation du résultat

Comparaison du modèle avec les valeurs réelles mesurées :



- 1 point gris= 1 essai sur un tableau aléatoire
- 1 point rouge=moyenne pour chaque N

`http://aofa.cs.princeton.edu/`

Limitations de l'approche scientifique

Le modèle peut ne pas être réaliste

- Pour le quicksort, on peut randomiser le tableau d'entrée avant d'appliquer l'algorithme pour se mettre dans les conditions du modèle d'entrée

Les maths peuvent être trop difficiles

- L'objectif des prochains cours est de vous donner quelques outils de base pour faire ce genre d'analyse.

Chapitre 6

Sommations et comportements asymptotiques

Plan

Sources : MCS (chapitre 13), Gross (chapitre 3).

Sommations

Définition : Soit une suite $x_i (i \in \mathbb{Z})$. La sommation $\sum_{i=a}^{b} x_i$ pour $a, b \in \mathbb{Z}$ est définie récursivement par :

- $\sum_{i=a}^{b} x_i = 0$ si $b < a$ *(cas de base)*
- $\sum_{i=a}^{b} x_i = x_a$ si $b = a$ *(cas de base)*
- $\sum_{i=a}^{b} x_i = \left(\sum_{i=a}^{b-1} x_i\right) + x_b$ si $b > a$ *(cas inductif)*

Définition : Soit une suite de réels x_i, $i \in \mathbb{N}$, la *série* de *terme général* x_i est la suite de *sommes partielles*

$$\sum_{i=0}^{n} x_i \ (n \in \mathbb{N}).$$

Etant donné une série, on notera S_n la n-ème somme partielle $\sum_{i=0}^{n} x_i$.
La suite des sommes partielles peut être définie récursivement :

- $S_0 = x_0$
- $S_n = S_{n-1} + x_n$ pour $n > 0$

Sommations

Les sommations apparaissent fréquemment dans le cadre de l'analyse d'algorithme et de la résolution de récurrences.

Objectif de ce chapitre : dériver des solutions analytiques à des sommations, et en particulier aux éléments d'une suite de somme partielle.

- Définition : Une *solution analytique* est une expression mathématique qui peut être évaluée à l'aide d'un nombre constant d'opérations de base (addition, multiplication, exponentiation, etc.).

But : simplifier l'évaluation des sommations pour prédire/étudier les performances d'un algorithme.

Preuve d'une solution analytique

Une solution analytique se prouve généralement facilement par induction.

Exemple : Série géométrique :

Théorème : Pour tous $n \geq 1$ et $z \neq 1$, on a

$$\sum_{i=0}^{n-1} z^i = \frac{1 - z^n}{1 - z}.$$

Démonstration : La preuve fonctionne par induction.
$P(n) =$ " $\sum_{i=0}^{n-1} z^i = \frac{1-z^n}{1-z}$ "
Cas de base $(n = 1)$: $P(1)$ est vérifié
Cas inductif $(n > 1)$: Si $P(n)$ est vérifié, on peut écrire :

$$\sum_{i=0}^{n} z^i = \sum_{i=0}^{n-1} z^i + z^n = \frac{1 - z^n}{1 - z} + z^n = \frac{1 - z^n + z^n - z^{n+1}}{1 - z} = \frac{1 - z^{n+1}}{1 - z}$$

□

(Exercice : montrer que $\sum_{i=0}^{n} i^2 = \frac{(2n+1)(n+1)n}{6}$*)*

Sommes infinies

Définition : $\sum_{i=0}^{\infty} z_i = \lim_{n \to \infty} \sum_{i=0}^{n} z_i.$

Théorème : Si $|z| < 1$, alors $\sum_{i=0}^{\infty} z^i = \frac{1}{1 - z}$.

Démonstration :

$$\begin{aligned} \sum_{i=0}^{\infty} z^i &= \lim_{n \to \infty} \sum_{i=0}^{n} z^i \\ &= \lim_{n \to \infty} \frac{1 - z^{n+1}}{1 - z} \\ &= \frac{1}{1 - z}. \end{aligned}$$

□

Trouver une solution analytique

Si prouver une solution analytique est aisé, imaginer cette solution l'est moins.

Différentes techniques génériques existent :

- Dériver cette solution de la solution analytique d'une autre série (par exemple par dérivation ou intégration),
- Méthode de perturbation,
- Par identification paramétrique.
- ...

Variantes des séries géométriques

Théorème : Pour tous $n \geq 0$ et $z \neq 1$, on a

$$\sum_{i=0}^{n} iz^i = \frac{z-(n+1)z^{n+1}+nz^{n+2}}{(1-z)^2}.$$

Démonstration : On a

$$\sum_{i=0}^{n} iz^i = z \cdot \sum_{i=0}^{n} iz^{i-1} = z \cdot \left(\frac{d}{dz}\sum_{i=0}^{n} z^i\right) = z \cdot \left(\frac{d}{dz}\frac{1-z^{n+1}}{1-z}\right).$$

En développant, on obtient

$$\begin{aligned} & z \cdot \left(\frac{d}{dz}\frac{1-z^{n+1}}{1-z}\right) \\ = \quad & z \cdot \left(\frac{-(n+1)z^n(1-z)-(-1)(1-z^{n+1})}{(1-z)^2}\right) \end{aligned}$$

$$\begin{aligned} = \quad & z \cdot \left(\frac{-(n+1)z^n+(n+1)z^{n+1}+1-z^{n+1}}{(1-z)^2}\right) \\ = \quad & z \cdot \left(\frac{1-(n+1)z^n+nz^{n+1}}{(1-z)^2}\right) \\ = \quad & \frac{z-(n+1)z^{n+1}+nz^{n+2}}{(1-z)^2}. \end{aligned}$$

□

Corollaire : Si $|z| < 1$, alors $\sum_{i=0}^{\infty} iz^i = \frac{z}{(1-z)^2}$.

Autre variante : En intégrant les deux côtés de $\sum_{i=0}^{\infty} z^i = \frac{1}{1-z}$ (de 0 à x), on peut obtenir :

$$\sum_{j=1}^{\infty} \frac{x^j}{j} = -\ln(1-x).$$

Méthode de perturbation

Soit S_n la n-ème somme partielle de la série de terme général x_i. Par définition, on a

$$S_n + x_{n+1} = x_0 + \sum_{i=1}^{n+1} x_k \ (= S_{n+1})$$

Si on peut exprimer le membre de droite comme une fonction de S_n, on peut obtenir une solution analytique en résolvant l'équation pour S_n.

Exemple : Pour la série géométrique $S_n = \sum_{i=0}^{n-1} z^i$:

$$S_{n+1} = S_n + z^n = z^0 + \sum_{i=1}^{n} z^i = 1 + z\sum_{i=0}^{n-1} z^i = 1 + zS_n$$

D'où, on tire immédiatement :

$$S_n = \frac{1-z^n}{1-z}$$

Un autre exemple

Problème[6] : Dériver une solution analytique de $S_n = \sum_{k=0}^{n} k2^k$.

Solution : Par la méthode de perturbation :

$$\begin{aligned} S_n + (n+1)2^{n+1} &= 0 \cdot 2^0 + \sum_{k=1}^{n+1} k2^k = \sum_{k=1}^{n+1} k2^k \\ &= \sum_{k=0}^{n} (k+1)2^{k+1} \\ &= \sum_{k=0}^{n} k2^{k+1} + \sum_{k=0}^{n} 2^{k+1} \\ &= 2\sum_{k=0}^{n} k2^k + 2\sum_{k=0}^{n} 2^k \\ &= 2S_n + 2(2^{n+1} - 1) \\ \Rightarrow S_n &= (n-1)2^{n+1} + 2 \end{aligned}$$

6. Cette somme apparaît dans l'analyse du tri par tas (voir INFO0902).

Perturbation indirecte

Parfois, ça ne marche pas directement.

Exemple : $S_n = \sum_{k=0}^{n} k^2$

$$\begin{aligned} S_n + (n+1)^2 &= 0^2 + \sum_{k=1}^{n+1} k^2 \\ &= \sum_{k=0}^{n} (k+1)^2 = \sum_{k=0}^{n} (k^2 + 2k + 1) \\ &= \sum_{k=0}^{n} k^2 + 2\sum_{k=0}^{n} k + \sum_{k=0}^{n} 1 \\ &= S_n + 2\sum_{k=0}^{n} k + (n+1) \\ \Rightarrow \sum_{k=0}^{n} k &= \frac{n(n+1)}{2} \end{aligned}$$

C'est correct mais ce n'est pas ce qu'on voulait calculer.

Perturbation indirecte

Dans ce cas, appliquer la perturbation à la suite $n \cdot x_n$ peut fonctionner :

Exemple : $S_n = \sum_{k=0}^{n} k \cdot k^2 = \sum_{k=0}^{n} k^3$

$$\begin{aligned} S_n + (n+1)^3 &= 0^3 + \sum_{k=1}^{n+1} k^3 \\ &= \sum_{k=0}^{n} (k+1)^3 = \sum_{k=0}^{n} (k^3 + 3k^2 + 3k + 1) \\ &= \sum_{k=0}^{n} k^3 + 3\sum_{k=0}^{n} k^2 + 3\sum_{k=0}^{n} k + \sum_{k=0}^{n} 1 \\ &= S_n + 3\sum_{k=0}^{n} k^2 + 3\frac{n(n+1)}{2} + (n+1) \\ \Rightarrow \sum_{k=0}^{n} k^2 &= \frac{2(n+1)^3 - 3n(n+1) - 2(n+1)}{6} = \frac{(2n+1)(n+1)n}{6} \end{aligned}$$

Par identification

On fait une hypothèse sur la forme de la solution et on identifie les paramètres en prenant quelques valeurs.

Exemple : $\sum_{i=1}^{n} i^2 = \frac{(2n+1)(n+1)n}{6}$.

- Supposer que la somme est un polynôme de degré 3 (car somme∼intégration)

$$\sum_{i=1}^{n} i^2 = an^3 + bn^2 + cn + d$$

- Identifier les constantes a, b, c, d à partir de quelques valeurs de la somme
- Prouver sa validité par induction (!)

Série harmonique

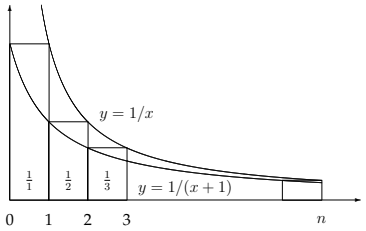
Complexité moyenne du quicksort (voir transp. 304) :

$$C_N = 2(N+1)\sum_{k=1}^{N}\frac{1}{k} - 2N$$

Définition : $H_n = \frac{1}{1} + \frac{1}{2} + \ldots + \frac{1}{n}$ est la *série harmonique*. H_n est le n-ème nombre harmonique.

La série harmonique n'a pas de solution analytique (connue). Des bornes inférieures et supérieures peuvent cependant être déterminées par intégration.

Approximation par intégration



$$\begin{aligned}\int_0^n \frac{1}{x+1}\,dx &\leq H_n \leq 1 + \int_1^n \frac{1}{x}\,dx\\ [\ln(x+1)]_0^n &\leq H_n \leq 1 + [\ln x]_1^n\\ \ln(n+1) &\leq H_n \leq 1 + \ln(n)\end{aligned}$$

Définition : Soient deux fonctions $f, g : \mathbb{R} \to \mathbb{R}$. On écrit $f(x) \sim g(x)$ ssi $\lim_{x\to\infty} f(x)/g(x) = 1$ (f et g sont *asymptotiquement équivalents*).

$$\ln(n+1) \leq H_n \leq 1 + \ln(n)$$
$$\Rightarrow H_n \sim \ln n$$

Nombres harmoniques

Une meilleure approximation existe :

$$H_n = \ln(n) + \gamma + \frac{1}{2n} + \frac{1}{12n^2} + \frac{\epsilon(n)}{120n^4},$$

où γ est la constante d'Euler ($0.57721\ldots$) et $0 \leq \epsilon(n) \leq 1$.

Complexité du quicksort :

$$\begin{aligned}C_N &= 2(N+1)H_N - 2N\\ &\sim 2N\ln N - 2(1-\gamma)N\end{aligned}$$

Méthode d'intégration

La méthode d'intégration peut être appliquée pour approximer beaucoup d'autres séries.

Définition : Une fonction $f : \mathbb{R}^+ \to \mathbb{R}^+$ est *strictement croissante* si $x < y$ implique $f(x) < f(y)$ et est *monotone croissante* si $x < y$ implique $f(x) \leq f(y)$.

Théorème : Soit une fonction $f : \mathbb{R}^+ \to \mathbb{R}^+$ monotone croissante. On a :

$$\int_1^n f(x)dx + f(1) \leq \sum_{i=1}^{n} f(i) \leq \int_1^n f(x)dx + f(n).$$

Le théorème peut être adapté trivialement aux fonctions décroissantes.

Exercice : montrez que $\frac{2}{3}n^{3/2} + \frac{1}{3} \leq \sum_{i=1}^{n}\sqrt{i} \leq \frac{2}{3}n^{3/2} + \sqrt{n} - \frac{2}{3}$.

Sommes doubles

Généralement, il suffit d'évaluer la somme intérieure et puis la somme extérieure.
Exercice : montrez que $\sum_{n=0}^{\infty}(y^n \sum_{i=0}^{n} x^i) = \frac{1}{(1-y)(1-xy)}$

Quand la somme intérieure n'a pas de solution analytique, échanger les deux sommes peut aider.
Exemple :

$$\begin{aligned}\sum_{k=1}^{n} H_k &= \sum_{k=1}^{n}\sum_{j=1}^{k}\frac{1}{j}\\ &= \sum_{j=1}^{n}\sum_{k=j}^{n}\frac{1}{j}\\ &= \dots\\ &= (n+1)H_n - n\end{aligned}$$

		j						
		1	2	3	4	5	...	n
k	1	1						
	2	1	1/2					
	3	1	1/2	1/3				
	4	1	1/2	1/3	1/4			
		...						
	n	1	1/2		...			$1/n$

Remarque sur les produits

Les mêmes techniques peuvent être utilisées pour calculer des produits en utilisant le logarithme :

$$\prod f(n) = \exp\left(\ln\left(\prod f(n)\right)\right) = exp\left(\sum \ln f(n)\right).$$

Permet de borner $n! = 1 \cdot 2 \cdot 3 \dots (n-1) \cdot n$. Par la méthode d'intégration, on a :

$$n\ln(n) - n + 1 \leq \sum_{i=1}^{n} \ln(i) \leq n\ln(n) - n + 1 + \ln(n).$$

En prenant l'exponentielle :

$$\frac{n^n}{e^n} \leq n! \leq \frac{(n+1)^{(n+1)}}{e^n}$$

Stirling's formula :

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n.$$

Plan

Notations asymptotiques

Les notations asymptotiques permettent de caractériser une fonction $f(x)$ lorsque x est très grand.

On a déjà vu la notion d'équivalence asymptotique ($\sim$) :
Définition : Soient deux fonctions $f, g : \mathbb{R} \to \mathbb{R}$. On écrit $f(x) \sim g(x)$ ssi $\lim_{x\to\infty} f(x)/g(x) = 1$ (f et g sont *asymptotiquement équivalents*).

On peut définir deux notations supplémentaires :
Définitions : Soient deux fonctions $f, g : \mathbb{R} \to \mathbb{R}$.

- On écrit $f(x) = o(g(x))$ ssi $\lim_{x\to\infty} f(x)/g(x) = 0$. On dira que f est négligeable devant g asymptotiquement.
- On écrit $f(x) = w(g(x))$ ssi $\lim_{x\to\infty} g(x)/f(x) = 0$. On dira que f domine g asymptotiquement.

Exemples :

- $2n = o(n^2)$, $2n^2 \neq o(n^2)$
- $n^2/2 = w(n)$, $n^2/2 \neq w(n^2)$

Quelques propriétés des notations $\sim$, o et w

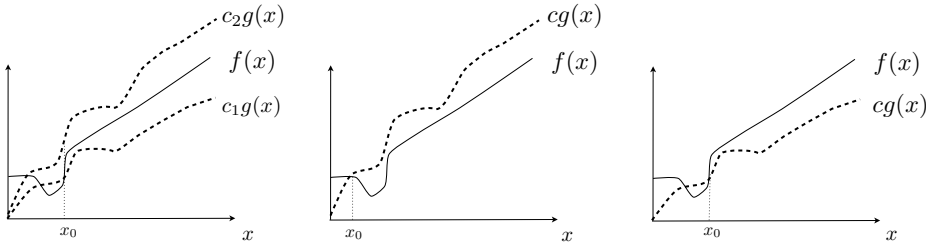
1. $f(x) = o(g(x))$ ssi **pour tout** $c > 0$, il existe un $x_0 \in \mathbb{R}$ tel que pour tout $x \geq x_0$, $|f(x)| \leq c|g(x)|$.
2. $f(x) = w(g(x))$ ssi **pour tout** $c > 0$, il existe un $x_0 \in \mathbb{R}$ tel que pour tout $x \geq x_0$, $|f(x)| \geq c|g(x)|$.
3. $f(x) = w(g(x))$ ssi $g(x) = o(f(x))$.
4. $f(x) \sim g(x)$ ssi $f(x) - g(x) = o(g(x))$.
5. $f(x) \sim g(x)$ ssi $f(x) = g(x) + h(x)$ pour une fonction $h(x) = o(g(x))$.
6. $x^a = o(x^b)$ pour tout $a < b$
7. $\log x = o(x^\epsilon)$ pour tout $\epsilon > 0$

(Exercices : démontrez ces propriétés)

Notations asymptotiques : O, Ω et Θ

Définitions : Soient $f : \mathbb{R} \to \mathbb{R}$ et $g : \mathbb{R} \to \mathbb{R}$ deux fonctions :

- On écrit $f(x) = O(g(x))$ s'il existe des constantes x_0 et $c > 0$ telles que $|f(x)| \leq c.|g(x)|$ pour tout $x \geq x_0$.
- $f(x) = \Omega(g(x))$ s'il existe des constantes x_0 et $c > 0$ telles que $|f(x)| \geq c.|g(x)|$ pour tout $x \geq x_0$.
- $f(x) = \Theta(g(x))$ s'il existe des constantes x_0, c_1 et $c_2 > 0$ telles que $c_1|g(x)| \leq |f(x)| \leq c_2|g(x)|$ pour tout $x \geq x_0$.



$f(x) = \Theta(g(x))$ $f(x) = O(g(x))$ $f(x) = \Omega(g(x))$

Quelques propriétés

1. $f(x) = \Omega(g(x)) \Leftrightarrow g(x) = O(f(x))$
2. $f(x) = \Theta(g(x)) \Leftrightarrow f(x) = O(g(x))$ et $f(x) = \Omega(g(x))$
3. $f(x) = \Theta(g(x)) \Leftrightarrow f(x) = O(g(x))$ et $g(x) = \Omega(f(x))$
4. $f(x) = o(g(x))$ ou $f \sim g \Rightarrow f(x) = O(g(x))$
5. $f(x) = O(g(x))$ et $g(x) = o(h(x)) \Rightarrow f(x) = o(h(x))$
6. $f(x) = O(g(x))$ et $g(x) = O(h(x)) \Rightarrow f(x) = O(h(x))$ (transitivité)
7. Si $f_1(x) = O(g_1(x))$ et $f_2(x) = O(g_2(x))$, alors $f_1(x) + f_2(x) = O(g_1(x) + g_2(x)) = O(\max\{g_1(x), g_2(x)\}$.
8. Si $f_1(x) = O(g_1(x))$ et $f_2(x) = O(g_2(x))$, alors $f_1(x)f_2(x) = O(g_1(x)g_2(x))$.

Les propriétés 6, 7 et 8 sont valable pour Θ, Ω, o et w.

NB : On peut faire une analogie entre ces notations et les comparateurs sur les réels : $o \to <$, $O \to \leq$, $\Theta \to =$, $\Omega \to \geq$, $w \to >$.

Démonstrations d'une relation asymptotique

Propriété : On a $5x + 100 = O(x)$.

Démonstration : On doit trouver des constantes x_0 et $c > 0$ telles que $|5x + 100| \leq cx$ pour tout $x \geq x_0$. Soient $c = 10$ et $x_0 = 20$. On a

$$|5x + 100| \leq 5x + 5x = 10x$$

pour tout $x \geq 20$. □

Propriété : On a $x = O(x^2)$.

Démonstration : On doit trouver des constantes x_0 et $c > 0$ telles que $|x| \leq c \cdot x^2$ pour tout $x \geq x_0$. Soient $c = 1$ et $x_0 = 1$. On a

$$|x| \leq 1 \cdot x^2$$

pour tout $x \geq 1$. □

Propriété : On a $x^2 \neq O(x)$.

Démonstration : Par l'absurde, supposons qu'il existe des constantes x_0 et $c > 0$ telles que

$$|x^2| \leq c \cdot x$$

pour tout $x \geq x_0$. On doit donc avoir

$$x \leq c$$

pour tout $x \geq x_0$, ce qui est impossible à satisfaire pour $x = \max(x_0, c+1)$. □

Remarques importantes

- On devrait écrire $f(x) \in O(g(x))$ plutôt que $f(x) = O(g(x))$
  - $O(g(x))$ n'est pas une fonction mais l'ensemble des fonctions $f(x)$ telles que $f(x) = O(g(x))$.
- Par abus de notation, on se permet d'écrire :

$$f(x) = g(x) + O(h(x))$$

qui signifie qu'il existe une fonction $i(x) = O(h(x))$ telle que :

$$f(x) = g(x) + i(x).$$

Exemples :

- $H_n = \ln n + \gamma + O(\frac{1}{n})$
- $2n^2 + \Theta(n) = \Theta(n^2)$
- $\sum_{i=1}^{n} O(i) = O(n^2)$ (mais $O(1) + O(2) + \ldots + O(n) = O(n^2)$ n'a pas de sens)

Chapitre 7

Récurrences

Plan

Lectures conseillées :

- MCS, chapitre 20.
- Rosen, 8.1, 8.2, 8.3
- Introduction to algorithms, Cormen et al. Chapitre 4.

Introduction

Rappel : La notation asymptotique vue dans le chapitre 6 permet d'approximer la complexité des algorithmes.

But de ce chapitre : Étudier des méthodes permettant de résoudre des *équations récurrentes.*

Motivation :

- La complexité des *algorithmes récursifs* est souvent calculable à partir d'équations récurrentes.
- Les récurrences permettent de résoudre des problèmes de *dénombrement*

Complexité d'algorithme récursif : tris

Tri rapide (*Quicksort*) : Nombre de comparaisons en moyenne (voir chapitre 6) :

$$\begin{aligned} C_1 &= 2 \\ C_N &= N+1+\sum_{k=1}^{N}\frac{1}{N}(C_{k-1}+C_{N-k}) \text{ pour } N>1 \end{aligned}$$

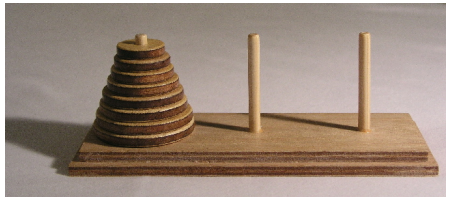
Tri par fusion (*merge sort*) : Pour trier un tableau de n éléments :

1. Diviser la liste en deux ;
2. Trier récursivement les deux sous-listes ;
3. Fusionner les deux listes triées.

Nombre de comparaison (dans le pire cas) :

$$\begin{aligned} C_1 &= 0 \\ C_N &= C_{\lceil N/2 \rceil}+C_{\lfloor N/2 \rfloor}+N-1 \text{ pour } N>1 \end{aligned}$$

Complexité d'algorithmes récursif : Tours de Hanoï



Source : `http://fr.wikipedia.org/wiki/Tours_de_Hanoï`

But : Déplacer la tour complète de la première tige vers une des deux autres tiges.

Contraintes :

- On ne peut déplacer qu'un seul disque à la fois.
- Un disque ne peut jamais être déposé sur un disque de diamètre inférieur.

Solution récursive :

- Cas de base : Déplacer une tour d'un seul disque est immédiat.
- Cas récursif : Pour déplacer une tour de $n+1$ disques de la première vers la troisième tige en connaissant une solution pour le déplacement d'une tour de n disques :
  1. Par récursion, déplacer n disques vers la deuxième tige ;
  2. Déplacer le disque restant vers la troisième tige ;
  3. Par récursion, déplacer les n disques de la deuxième tige vers la troisième tige.

Notation : Soit T_n le nombre minimum d'étapes nécessaires au déplacement d'une tour de n disques d'une tige vers une autre.

Propriété (borne supérieure) : On a $T_n \leq 2T_{n-1} + 1$.

Remarques :

- Pour déplacer une tour, il faut obligatoirement déplacer le disque du bas.
- Accéder au disque du bas nécessite de déplacer tous les autres disques vers une tige libre (au moins T_{n-1} étapes).
- Ensuite, il faut remettre en place le reste de la tour (au moins T_{n-1} étapes).

On a donc la propriété suivante :
Propriété (borne inférieure) : On a $T_n \geq 2T_{n-1} + 1$.

D'où on déduit la Récurrence :

$$\begin{aligned} T_1 &= 1 \\ T_n &= 2T_{n-1} + 1 \text{ pour } n > 1 \end{aligned}$$

Dénombrement : exemple 1

Problème : Supposons le mécanisme suivant d'engagement des professeurs à Montefiore :

- Chaque professeur :
  - est nommé à vie ;
  - est supposé immortel
  - forme chaque année exactement un étudiant qui deviendra professeur l'année suivante (exception : lors de leur première année d'enseignement, les professeurs sont trop occupés pour former un étudiant).
- Année 0 : il n'y a aucun professeur.
- Année 1 : le premier professeur (autodidacte) est formé.

Combien y aura-t-il de professeurs à la fin de la n-ème année ?

Solution : Soit F_n le nombre de professeurs à la fin de la n-ème année :

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2} \text{ pour } n > 1 \end{aligned}$$

Exercice : montrez que le nombre de séquences de n bits qui ne contiennent pas deux 0 consécutifs suit la même récurrence, avec des conditions initiales différentes

Dénombrement : exemple 2

Problème :

- Un programme informatique considère un chaîne constituée de chiffres décimaux comme un mot de passe valide si elle contient un nombre pair de chiffres 0.
- Par exemple "1230407869" et "12345" sont des mots de passe valide mais "120987045608" et "012879" ne le sont pas.
- Calculez T_n, le nombre de mots de passe valides de longueur n.

Solution : Une chaîne valide de n chiffres peut être obtenue de deux façons :

- en ajoutant un chiffre parmi $\{1, \ldots, 9\}$ au début d'une chaîne valide de longueur $n-1$
- en ajoutant un 0 au début d'une chaîne non valide de longueur $n-1$

On a donc :

$$T_n = 9T_{n-1} + (10^{n-1} - T_{n-1}) = 8T_{n-1} + 10^{n-1}$$

avec $T_0 = 1$.

Dénombrement : exemple 3

Définition : L'ensemble $\mathcal{B}$ des *arbres binaires* est défini comme suit :

- Cas de base : L'arbre vide $\emptyset$ appartient à $\mathcal{B}$.
- Cas récursif : Soit $B_1, B_2 \in \mathcal{B}$, alors **branch**$(B_1, B_2) \in bTree$.

Le nombre de nœuds $n(B)$ est défini récursivement par :

- Cas de base : $n(\emptyset) = 0$
- Cas inductif : $n(\textbf{branch}(B_1, B_2)) = 1 + n(B_1) + n(B_2)$

Théorème : Le nombre b_n d'arbres binaires ayant n nœuds est donné par :

$$\begin{aligned} b_0 &= 1 \\ b_n &= \sum_{k=0}^{n-1} b_k b_{n-1-k} \text{ pour } n > 0 \end{aligned}$$

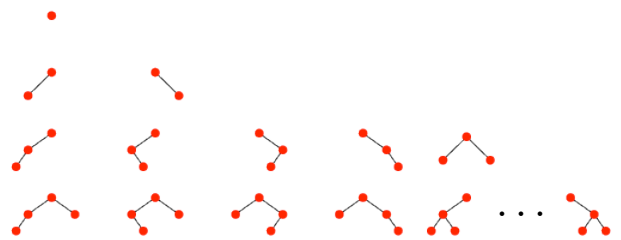
$b_0 = 1$

$b_1 = 1$

$b_2 = 2$

$b_3 = 5$

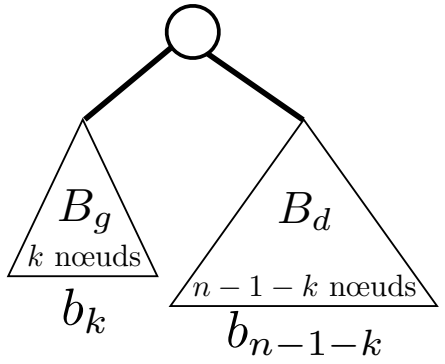
$b_4 = 14$



Démonstration : Par induction sur n.
Cas de base : $b_0 = 1$
Cas inductif :

- Un arbre binaire ayant $n > 0$ nœuds ne peut s'obtenir qu'en considérant tous les arbres binaires possibles de la forme **branch**(B_g, B_d) où B_g possède k nœuds et B_d possède $n - k - 1$ nœuds (avec $0 \leq k \leq n - 1$).
- Pour un k donné, il existe donc b_k arbres B_g possibles et b_{n-1-k} arbres B_d possibles.
- On a donc finalement :

$$b_n = \sum_{k=0}^{n-1} b_k b_{n-1-k}.$$



Nombres de Catalan

Les nombres :

$$\begin{aligned} b_0 &= 1 \\ b_n &= \sum_{k=0}^{n-1} b_k b_{n-1-k} \text{ pour } n > 0 \end{aligned}$$

sont appelés les *nombres de Catalan* (du nom d'un mathématicien belge).

Ils apparaissent dans de nombreux problèmes de dénombrement.

Exercice : Montrez que le nombre b_n de façons de parenthéser le produit de n nombres $x_0 \cdot x_1 \cdots x_n$ satisfait la même récurrence.
Exemple : $b_2 = 2 \rightarrow (x_0 \cdot x_1) \cdot x_2, x_0 \cdot (x_1 \cdot x_2)$

Classification des récurrences

Forme générale : $u_n = f(\{u_0, u_1, \ldots, u_{n-1}\}, n)$

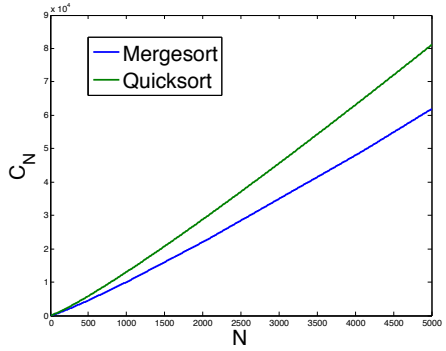
Une récurrence peut être :

- *linéaire* si f est une combinaison linéaire à coefficients *constants* ou *variables*.
  - $u_n = 3u_{n-1} + 4u_{n-2}$, $u_n = n * u_{n-2} + 1$
- *polynomiale* si f est un polynôme.
  - $b_n = \sum_{k=0}^{n-1} b_k b_{n-1-k}$
- *d'ordre k* si f dépend de $u_{n-1}, \ldots, u_{n-k}$.
- *complète* si f dépend de $u_{n-1}, \ldots, u_0$.
- *de type "diviser pour régner"* si f dépend de $u_{n/a}$, avec $a \in \mathbb{N}$ constant.
  - $u_n = 2u_{n/2}$
- *homogène* si f ne dépend que des u_i.
  - $u_n = 2u_{n/2} + u_{n/4}$
- *non homogène* si elle est de la forme $u_n = f(\{u_p | p < n\}) + g(n)$.
  - $u_n = 2u_{n-1} + n$

Calculer les valeurs d'une récurrence

On peut utiliser l'ordinateur pour calculer les valeurs d'une récurrence

Quicksort versus mergesort :



Avantages : ça s'applique à toutes les récurrences
Limitations : temps de calcul peut être élevé, comparaisons hasardeuses.

Remarque sur l'implémentation d'une récurrence

La manière d'implémenter la récurrence est importante

$$C_N = N + 1 + \frac{2}{N} \sum_{k=1}^{N} C_{k-1}$$

Implémentation récursive naïve

```
CQS(N)
1  if N == 0
2      return 0
3  else val = 0
4      for k = 1 to N
5          val = val + CQS(k − 1)
6      return N + 1 + 2 ∗ val/N
```

Nombre d'additions :

$$\begin{aligned} & C'_N = N + 2 + \sum_{k=1}^{N} C'_{k-1} \\ \Leftrightarrow \quad & C'_N = 2C'_{N-1} + 1 \\ \Leftrightarrow \quad & C'_N = 2^N - 1 \end{aligned}$$

(avec $C'_0 = 0$)

Implémentation efficace
(par programmation dynamique)

```
CQS(N)
1  if N == 0
2      return 0
3  else sum = 0
4      for k = 1 to N
5          val = k + 1 + 2 ∗ sum/k
6          sum = sum + val
7      return val
```

Nombre d'additions :

$$C'_N = 3N$$

Techniques de résolution de récurrences

On souhaite obtenir une solution analytique à une récurrence.

Plusieurs approches possibles :

- "Deviner-et-vérifier"
- "Plug-and-chug" (telescopage)
- Arbres de récursion
- Equations caractéristique (linéaire)
- Master theorem et Akra-Bazzi (diviser-pour-régner)
- Changement de variable
- Fonctions génératrices (Chapitre 8)

Méthode "Deviner-et-Vérifier"

Principes :

1. Calculer les quelques premières valeurs de T_n ;
2. Deviner une solution analytique ;
3. Démontrer qu'elle est correcte, par exemple par induction.

Application :

- $T_n = 2T_{n-1} - 1$ (tours de Hanoï) :

n	1	2	3	4	5	6	$\cdots$
T_n	1	3	7	15	31	63	$\cdots$

  $\Rightarrow$ On devine $T_n = 2^n - 1$
- On doit démontrer (par induction) que la solution est correcte.

Preuve d'une solution par induction

Théorème : $T_n = 2^n - 1$ satisfait la récurrence :

$$\begin{aligned} T_1 &= 1 \\ T_n &= 2T_{n-1} + 1 \text{ pour } n \geq 2. \end{aligned}$$

Démonstration : Par induction sur n. $P(n) =$ " $T_n = 2^n - 1$".
Cas de base : $P(1)$ est vrai car $T_1 = 1 = 2^1 - 1$.
Cas inductif : Montrons que $T_n = 2^n - 1$ (pour $n \geq 2$) est vrai dès que $T_{n-1} = 2^{n-1} - 1$ est vrai :

$$\begin{aligned} T_n &= 2T_{n-1} + 1 \\ &= 2(2^{n-1} - 1) + 1 \\ &= 2^n - 1. \end{aligned}$$

□

Piège de l'induction

Si une solution exacte ne peut pas être trouvée, on peut vouloir trouver une borne supérieure sur la solution.

Essayons de montrer que $T_n \leq 2^n$ pour la récurrence $T_n = 2T_{n-1} + 1$.

Tentative de démonstration : Par induction sur n. $P(n) =$ " $T_n \leq 2^n$".
Cas de base : $P(1)$ est vrai car $T_1 = 1 \leq 2^1$.
Cas inductif : Supposons $T_{n-1} \leq 2^{n-1}$ et montrons que $T_n \leq 2^n$ ($n \geq 2$) :

$$\begin{aligned} T_n &= 2T_{n-1} + 1 \\ &\leq 2(2^{n-1}) + 1 \\ &\not\leq 2^n \end{aligned}$$

□

La démonstration ne fonctionne pas si on n'utilise pas la borne exacte plus forte (voir aussi le transparent 369).

Méthode "Plug-and-Chug" (force brute)

(aussi appelée téléscopage ou méthode des facteurs sommants)

1. "Plug" (appliquer l'équation récurrente) et "Chug" (simplifier)

$$\begin{aligned} T_n &= 1 + 2T_{n-1} \\ &= 1 + 2(1 + 2T_{n-2}) \\ &= 1 + 2 + 4T_{n-2} \\ &= 1 + 2 + 4(1 + 2T_{n-3}) \\ &= 1 + 2 + 4 + 8T_{n-3} \\ &= \dots \end{aligned}$$

Remarque : Il faut simplifier *avec modération*.

2. Identifier et vérifier un "pattern"
   - Identification :

$$T_n = 1 + 2 + 4 + \cdots + 2^{i-1} + 2^i T_{n-i}$$

   - Vérification en développant une étape supplémentaire :

$$\begin{aligned} T_n &= 1 + 2 + 4 + \cdots + 2^{i-1} + 2^i(1 + 2T_{n-(i+1)}) \\ &= 1 + 2 + 4 + \cdots + 2^{i-1} + 2^i + 2^{i+1} T_{n-(i+1)} \end{aligned}$$

3. Exprimer le $n^{\text{ème}}$ terme en fonction des termes précédents
   En posant $i = n - 1$, on obtient

$$\begin{aligned} T_n &= 1 + 2 + 4 + \cdots + 2^{n-2} + 2^{n-1} T_1 \\ &= 1 + 2 + 4 + \cdots + 2^{n-2} + 2^{n-1} \end{aligned}$$

4. Trouver une solution analytique pour le $n^{\text{ème}}$ terme

$$\begin{aligned} T_n &= 1 + 2 + 4 + \cdots + 2^{n-2} + 2^{n-1} \\ &= \sum_{i=0}^{n-1} 2^i \\ &= \frac{1 - 2^n}{1 - 2} \\ &= 2^n - 1 \end{aligned}$$

Tri par fusion

Appliquons le "plug-and-chug" à la récurrence du tri par fusion *dans le cas où* $N = 2^n$:

$$\begin{aligned} C_1 &= 0 \\ C_N &= C_{\lceil N/2 \rceil} + C_{\lfloor N/2 \rfloor} + N - 1 = 2C_{N/2} + N - 1 \text{ pour } N > 1 \end{aligned}$$

- Pattern :

$$\begin{aligned} C_N &= 2^i C_{N/2^i} + (n - 2^{i-1}) + (n - 2^{i-2}) + \ldots + (n - 2^0) \\ &= 2^i C_{N/2^i} + i \cdot N - 2^i + 1 \end{aligned}$$

- En posant $i = n$ et en utilisant $n = \log_2 N$:

$$\begin{aligned} C_N &= 2^n C_{N/2^n} + n \cdot N - 2^n + 1 \\ &= NC_1 + N \log_2 N - N + 1 \\ &= N \log_2 N - N + 1 \end{aligned}$$

Récurrence linéaire d'ordre 1

Le plug-and-chug appliqué à une récurrence du type :

$$T_n = T_{n-1} + f(n)$$

donne directement :

$$T_n = T_0 + \sum_{k=1}^{n} f(n).$$

La récurrence n'est rien d'autre que la réécriture d'une somme (voir chapitre 6).

Si le coefficient de T_{n-1} n'est pas 1 :

$$T_n = g(n)T_{n-1} + f(n),$$

diviser gauche et droite par $h(n) = g(n) \cdot g(n-1) \cdot g(n-2) \cdots g(1)$ permet de se ramener à une récurrence de la forme :

$$\frac{T_n}{h(n)} = \frac{T_{n-1}}{h(n-1)} + \frac{f(n)}{h(n)} \Rightarrow T_n = h(n)\left(\frac{T_0}{h(0)} + \sum_{k=1}^{n} \frac{f(k)}{h(k)}\right)$$

Intéressant si le produit $h(n)$ a une forme simple

Récurrence linéaire d'ordre 1 : Exemples

Exemple 1 : $T_0 = 0,\ T_n = 2T_{n-1} + 2^n$ (pour $n > 0$)
En divisant gauche et droite par $h(n) = 2 \cdot 2 \cdots 2 = 2^n$, on obtient :

$$\frac{T_n}{2^n} = \frac{T_{n-1}}{2^{n-1}} + 1 \Rightarrow T_n = 2^n(0 + \sum_{k=1}^{n} 1) = n2^n$$

Exemple 2 : $T_0 = 0,\ T_n = (1 + \frac{1}{n})T_{n-1} + 2$ (pour $n > 0$) *(Quicksort)*
En divisant gauche et droite par :

$$h(n) = \frac{n+1}{n}\frac{n}{n-1} \cdots \frac{3}{2}\frac{2}{1} = n + 1,$$

on obtient

$$\frac{T_n}{n+1} = \frac{T_{n-1}}{n} + \frac{2}{n+1} = \sum_{k=1}^{n} \frac{2}{n+1} \Rightarrow T_n = 2(n+1)H_n - 2n$$

Arbres de récursion

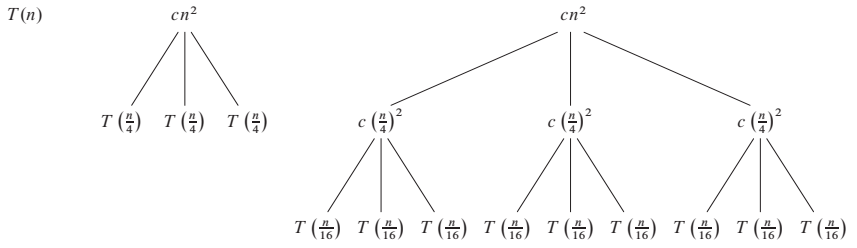
Approche graphique pour *deviner* une solution analytique (ou une borne asymptotique) à une récurrence.

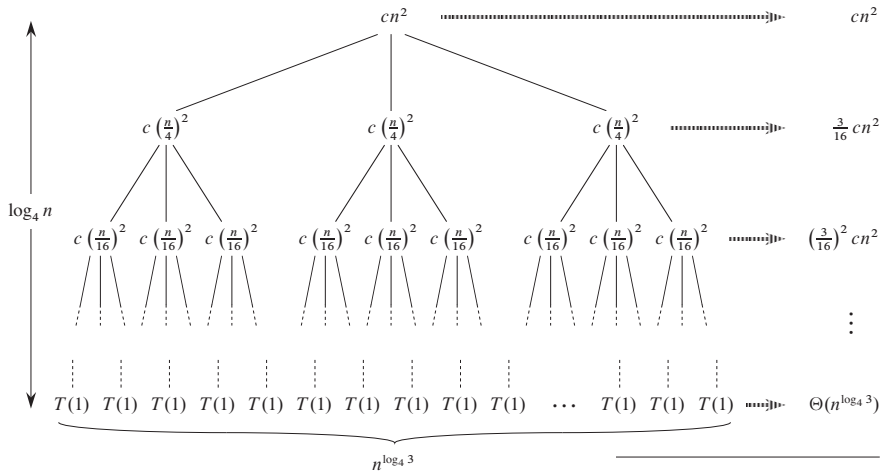
La solution devra toujours être démontrée ensuite (par induction).

Illustration sur la récurrence suivante :

- $T(1) = a$
- $T(n) = 3T(n/4) + cn^2$ (Pour $n > 1$)

(Introduction to algorithms, Cormen et al.)





- Le coût total est la somme du coût de chaque niveau de l'arbre :

$$\begin{aligned} T(n) &= cn^2 + \frac{3}{16}cn^2 + \ldots + \left(\frac{3}{16}\right)^{\log_4 n-1} cn^2 + an^{\log_4 3} \\ &= \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i cn^2 + an^{\log_4 3} \\ &< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + an^{\log_4 3} \\ &= \frac{1}{1-(3/16)} cn^2 + an^{\log_4 3} \\ &= O(n^2) \end{aligned}$$

(à vérifier par induction)

- Comme le coût de la racine est cn^2, on a aussi $T(n) = \Omega(n^2)$ et donc $T(n) = \Theta(n^2)$.

Preuve d'une borne supérieure

Théorème : Soit la récurrence :

- $T(1) = a$,
- $T(n) = 3T(n/4) + cn^2$ (Pour $n > 1$).

On a $T(n) = O(n^2)$.

Préparation de la démonstration : Soit $P(n) =$ "$T(n) \leq dn^2$". L'idée est de trouver un d tel que $P(n)$ peut se montrer par induction forte.
Cas de base : $P(1)$ est vrai si $a \leq d \cdot 1^2 = d$.
Cas inductif : Supposons $P(1), P(4), \ldots, P(n/4)$ vérifiés et trouvons une contrainte sur d telle que $P(n)$ le soit aussi :

$$\begin{aligned} T(n) &\leq 3T(n/4) + cn^2 \\ &\leq 3d(n/4)^2 + cn^2 \\ &= \frac{3}{16}dn^2 + cn^2 \\ &\leq dn^2, \end{aligned}$$

La dernière étape est valide dès que $d \geq (16/13)c$. Le théorème est donc vérifié pour autant que $d \geq \max\{(16/13)c, a\}$.

Preuve d'une borne supérieure

Ré-écriture de la preuve :

Démonstration : Soit une constante d telle que $d \geq \max\{(16/13)c, a\}$. Montrons par induction forte que $P(n) =$ "$T(n) \leq dn^2$" est vrai pour tout $n \geq 1$ (qui implique $T(n) = O(n)$).
Cas de base : $P(1)$ est vrai puisque $d \geq \max\{(16/13)c, a\} \geq a$.
Cas inductif : Supposons $P(1), P(4), \ldots, P(n/4)$ vérifiés et montrons que $P(n)$ l'est aussi :

$$\begin{aligned} T(n) &\leq 3T(n/4) + cn^2 \\ &\leq 3d(n/4)^2 + cn^2 \\ &= \frac{3}{16}dn^2 + cn^2 \\ &\leq dn^2, \end{aligned}$$

où la dernière étape découle de $d \geq max((16/13)c, a)$.

□

Induction et notation asymptotique

Théorème faux : Soit la récurrence :

- $T(1) = 1$,
- $T(n) = 2T(n/2) + n$ (pour $n > 1$).

On a $T(n) = O(n)$.
(la solution correcte est $T(n) = \Theta(n \log(n))$)

Démonstration : (par induction forte)

- Soit $P(n) =$ "$T(n) = O(n)$".
- *Cas de base :* $P(1)$ est vrai car $T(1) = 1 = O(1)$.
- *Cas inductif :* Pour $n \geq 2$, supposons $P(1), P(2), \ldots, P(n-1)$. On a :

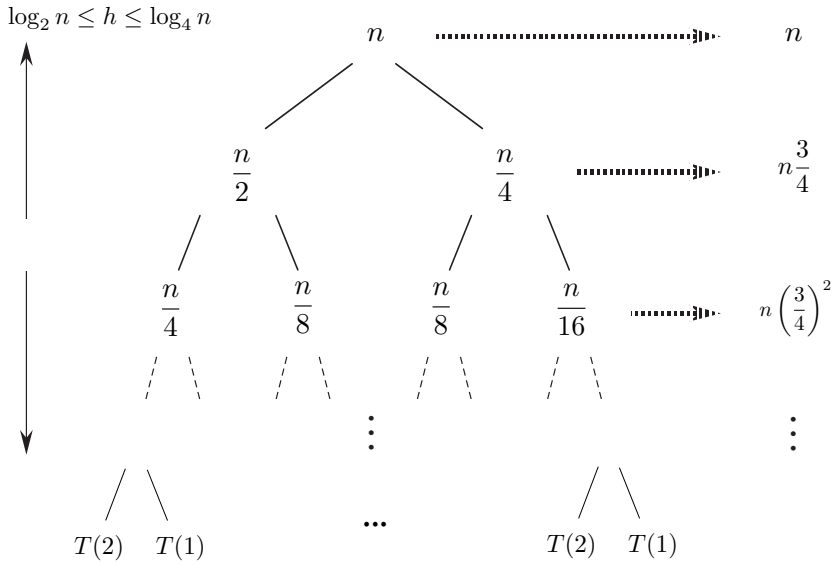
$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2O(n/2) + n \\ &= O(n) \end{aligned}$$

Où est l'erreur ?

Autre exemple :

- $T(1) = 1$
- $T(2) = 2$
- $T(n) = T(n/2) + T(n/4) + n$ (pour $n > 2$)

(On suppose que n est toujours une puissance de 2)



- On déduit de l'arbre que

$$T(n) \leq \sum_{i=0}^{\infty} n \left(\frac{3}{4}\right)^i = n \frac{1}{1 - \frac{3}{4}} = O(n)$$

- Vérification par induction forte qu'il existe un c tel que pour tout $n \geq n_0$, on a $T(n) \leq cn$

$$\begin{aligned} T(n) &= T(n/2) + T(n/4) + n \\ &\leq cn/2 + cn/4 + n \\ &= (c3/4 + 1)n \\ &\leq cn \end{aligned}$$

  $\Rightarrow$ ok pour tout $c > 4$
- Puisqu'on a aussi $T(n) = \Omega(n)$, on en déduit $T(n) = \Theta(n)$.

Synthèse

Trois approches empiriques pour trouver une solution analytique :

- "Deviner-et-vérifier"
- "Plug-and-Chug"
- Arbres de récursion

Ces approches sont génériques mais

- il n'est pas toujours aisé de trouver le pattern ;
- il faut pouvoir résoudre la somme obtenue ;
- la solution doit être vérifiée par induction.

On va voir des méthodes plus systématiques pour résoudre des récurrences particulières

- Récurrences linéaires d'ordre $k \geq 1$ à coefficients constants (solution analytique exacte)
- Récurrences "diviser-pour-régner" (borne asymptotique uniquement)

Récurrences linéaires

Définition : Une *récurrence linéaire homogène* est une récurrence de la forme

$$f(n) = a_1 f(n-1) + a_2 f(n-2) + \cdots + a_k f(n-k)$$

où $a_1, a_2, \ldots, a_k \in \mathbb{R}$ sont des constantes. La valeur $k \in \mathbb{N}_0$ est appelée l'*ordre* de la récurrence.

Définition : Une *récurrence linéaire* (générale) est une récurrence de la forme

$$f(n) = a_1 f(n-1) + a_2 f(n-2) + \cdots + a_k f(n-k) + g(n),$$

où g est une fonction ne dépendant pas de f.

Théorème : Si $f_1(n)$ et $f_2(n)$ sont solutions d'une récurrence linéaire homogène (sans tenir compte des conditions initiales), alors toute combinaison linéaire $c_1 f_1(n) + c_2 f_2(n)$ de $f_1(n)$ et $f_2(n)$ est également une solution pour tout $c_1, c_2 \in \mathbb{R}$.

Démonstration : On a $f_1(n) = \sum_{i=1}^{k} a_i f_1(n-i)$ et $f_2(n) = \sum_{i=1}^{k} a_i f_2(n-i)$.

Dès lors,

$$\begin{aligned} c_1 f_1(n) + c_2 f_2(n) &= c_1 \cdot \left(\sum_{i=1}^{k} a_i f_1(n-i) \right) + c_2 \cdot \left(\sum_{i=1}^{k} a_i f_2(n-i) \right) \\ &= \sum_{i=1}^{k} a_i (c_1 f_1(n-i) + c_2 f_2(n-i)). \end{aligned}$$

□

Exemple

Résolvons la récurrence du transparent 343 (Fibonacci) :

$$\begin{aligned} f(0) &= 0 \\ f(1) &= 1 \\ f(n) &= f(n-1) + f(n-2) \text{ pour } n > 1 \end{aligned}$$

- Une solution analytique pour une récurrence linéaire a souvent une forme exponentielle.
- On devine $f(n) = cx^n$ (c et x sont des paramètres à trouver).
-

$$\begin{aligned} & f(n) = f(n-1) + f(n-2) \\ \Rightarrow \quad & cx^n = cx^{n-1} + cx^{n-2} \\ \Rightarrow \quad & x^2 = x + 1 \\ \Rightarrow \quad & x = \frac{1 \pm \sqrt{5}}{2} \end{aligned}$$

- Les fonctions $c\left(\frac{1+\sqrt{5}}{2}\right)^n$ et $c\left(\frac{1-\sqrt{5}}{2}\right)^n$ sont des solutions de la récurrence (sans tenir compte des conditions initiales).
- Il en est de même pour toute combinaison linéaire de ces deux fonctions.
- On a donc $f(n) = c_1\left(\frac{1+\sqrt{5}}{2}\right)^n + c_2\left(\frac{1-\sqrt{5}}{2}\right)^n$.

- $f(0) = c_1\left(\frac{1+\sqrt{5}}{2}\right)^0 + c_2\left(\frac{1-\sqrt{5}}{2}\right)^0 = c_1 + c_2 = 0.$
- $f(1) = c_1\left(\frac{1+\sqrt{5}}{2}\right)^1 + c_2\left(\frac{1-\sqrt{5}}{2}\right)^1 = 1.$
- On obtient $c_1 = \frac{1}{\sqrt{5}}$ et $c_2 = \frac{-1}{\sqrt{5}}$.
- Finalement,

$$f(n) = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n.$$

Résolution des récurrences linéaires

Soit une récurrence de la forme

$$f(n) = a_1 f(n-1) + a_2 f(n-2) + \cdots + a_k f(n-k) + g(n)$$

et les conditions initiales $f(0) = b_0$, $f(1) = b_1$, etc.

Etape 1 : Trouver les racines de l'*équation caractéristique*

Définition : L'équation caractéristique est

$$x^k = a_1 x^{k-1} + a_2 x^{k-2} + \cdots + a_{k-1}x + a_k.$$

Remarque : Le terme $g(n)$ n'est pas pris en compte dans l'équation caractéristique.

Etape 2 : Trouver une *solution homogène*, sans tenir compte des conditions initiales

Il suffit d'additionner les termes suivants :

- Une racine non répétée r de l'équation caractéristique génère le terme
$$c_r r^n,$$
où c_r est une constante à déterminer plus tard.
- Une racine r avec multiplicité m de l'équation caractéristique génère les termes
$$c_{r_1} r^n, c_{r_2} n r^n, c_{r_3} n^2 r^n, \ldots, c_{r_m} n^{m-1} r^n,$$
où $c_{r_1}, c_{r_2}, \ldots, c_{r_m}$ sont des constantes à déterminer plus tard.

Etape 3 : Trouver une *solution particulière*, sans tenir compte des conditions initiales.

Une technique simple consiste à *deviner et vérifier* en essayant des solutions *ressemblant* à $g(n)$.

Exemples :

- Si $g(n)$ est un polynôme, essayer avec un polynôme de même degré, ensuite avec un polynôme de degré immédiatement supérieur, et ainsi de suite.
  Exemple : Si $g(n) = n$, essayer d'abord $f(n) = bn + c$, ensuite, $f(n) = an^2 + bn + c$, ...
- Si $g(n) = 3^n$, essayer d'abord $f(n) = c3^n$, ensuite $f(n) = bn3^n + c3^n$, $f(n) = an^2 3^n + bn3^n + c3^n$, ...

Remarque : On doit attribuer aux constantes $a, b, c, \ldots$ des valeurs satisfaisant l'équation récurrente.

Etape 4 : Former une *solution générale*, sans tenir compte des conditions initiales

Il suffit d'additionner la solution homogène et la solution particulière

Etape 5 : Déterminer les valeurs des constantes introduites à l'étape 2

- Pour chaque condition initiale, appliquer la solution générale à cette condition. On obtient une équation en fonction des constantes à déterminer.
- Résoudre le système formé par ces équations.

Exemple

Résolvons la récurrence du transparent 344 :

- $f(0) = 1$
- $f(n) = 8f(n-1) + 10^{n-1}$

Etape 1 : Trouver les racines de l'équation caractéristique

- L'équation caractéristique est $x = 8$.
- Sa seule racine est 8.

Etape 2 : Trouver une solution homogène, sans tenir compte des conditions initiales

La solution homogène est $f(n) = c8^n$.

Etape 3 : Trouver une solution particulière, sans tenir compte des conditions initiales.

- On devine que la solution est de la forme $d10^{n-1}$, où d est une constante.
- En substituant, on obtient

$$\begin{aligned} d \cdot 10^{n-1} &= 8 \cdot d \cdot 10^{n-2} + 10^{n-1} \\ 10 \cdot d &= 8 \cdot d + 10 \\ d &= 10/2 \end{aligned}$$

- On vérifie que $\frac{10}{2} \cdot 10^{n-1} = \frac{10^n}{2}$ est bien une solution particulière.

Etape 4 : Former une solution générale, sans tenir compte des conditions initiales

On obtient la solution générale

$$f(n) = c8^n + \frac{10^n}{2}.$$

Etape 5 : Déterminer les valeurs des constantes introduites à l'étape 2

$$\begin{aligned} f(0) = 1 \quad &\Rightarrow \quad c8^0 + \frac{10^0}{2} = 1 \\ &\Rightarrow \quad c = \frac{1}{2}. \end{aligned}$$

Conclusion : $f(n) = \dfrac{8^n + 10^n}{2}$.

Récurrence générale "diviser-pour-régner"

Définition : Un récurrence "diviser-pour-régner" est une récurrence de la forme :

$$T_n = \sum_{i=1}^{k} a_i T(b_j n) + g(n),$$

où $a_1, \ldots, a_k$ sont des constantes positives, $b_1, \ldots, b_k$ sont des constantes comprises entre 0 et 1 et $g(n)$ est une fonction non négative.

Exemple : $k = 1$, $a_1 = 2$, $b_1 = 1/2$ et $g(n) = n - 1$ correspond au tri par fusion

Sous certaines conditions, il est possible de trouver des bornes asymptotiques sur les récurrences de ce type.

Un premier théorème

Théorème (Master theorem) : Soient deux constantes $a \geq 1$ et $b > 1$ et une fonction $f(n) = O(n^d)$ avec $d \geq 0$. La complexité asymptotique de la récurrence suivante :

$$T(n) = aT(n/b) + f(n)$$

est :

$$T(n) = \begin{cases} O(n^d) & \text{pour } d > \log_b a \\ O(n^d \log n) & \text{pour } d = \log_b a; \\ O(n^{\log_b a}) & \text{pour } d < \log_b a. \end{cases}$$

(Introduction to algorithms, Cormen et al.)

NB : les bornes sont valables quelles que soient les conditions initiales.

Exemple d'application

- Soit la récurrence suivante :

$$T(n) = 7T(n/2) + O(n^2).$$

(Méthode de Strassen pour la multiplication de matrice)

- $T(n)$ satisfait aux conditions du théorème avec $a = 7$, $b = 2$, et $d = 2$.
- $\log_b a = \log_2 7 = 2.807... \Rightarrow d = 2 < \log_b a = 2.807...$
- Par le troisième cas du théorème, on a :

$$T(n) = O(n^{\log_b a}) = O(n^{2.807...}).$$

Un second théorème plus général

Forme générale d'une récurrence "Diviser pour régner" :

$$T(x) = \begin{cases} \text{est défini} & \text{pour } 0 \leq x \leq x_0 \\ \sum_{i=1}^{k} a_i T(b_i x) + g(x) & \text{pour } x > x_0 \end{cases}$$

avec

- $a_1, a_2, \ldots, a_k > 0$,
- $b_1, b_2, \ldots, b_k \in [0, 1[$,
- x_0 suffisamment grand,
- $|g'(x)| = O(x^c)$ pour un $c \in \mathbb{N}$.

Théorème (Akra-Bazzi) :

$$T(x) = \Theta\left(x^p \left(1 + \int_1^x \frac{g(u)}{u^{p+1}}\, \mathrm{d}u\right)\right)$$

où

- p satisfait l'équation $\sum_{i=1}^{k} a_i b_i^p = 1$

Exemple d'application

- Soit la récurrence "diviser pour régner" suivante :

$$T(x) = 2T(x/2) + 8/9T(3x/4) + x^2$$

- On a bien $|g'(x)| = |2x| = O(x)$
- Trouvons p satisfaisant :

$$2(\frac{1}{2})^p + \frac{8}{9}(\frac{3}{4})^p = 1$$

$\Rightarrow p = 2$

- Par application du théorème, on obtient :

$$\begin{aligned} T(x) &= \Theta\left(x^2\left(1 + \int_1^x \frac{u^2}{u^3} du\right)\right) \\ &= \Theta(x^2(1 + \log x)) \\ &= \Theta(x^2 \log x) \end{aligned}$$

Changement de variables

Un changement de variables permet parfois de transformer une récurrence "diviser pour régner" en une récurrence linéaire.

- Soit la récurrence du transparent 241 :

$$T(n) = 7T(n/2) + O(n^2)$$

- En posant : $n = 2^m$ et $S(m) = T(2^m)$, on obtient :

$$S(m) = T(2^m) = 7T(2^{m-1}) + O((2^m)^2) = 7S(m-1) + O(4^m)$$

Changement de variables

Un changement de variable permet de résoudre des récurrences qui semblent a priori complexes.

- Considérons la récurrence suivante :

$$T(n) = 2T(\sqrt{n}) + \log n$$

- Posons $m = \log n$. On a :

$$T(2^m) = 2T(2^{m/2}) + m.$$

- Soit $S(m) = T(2^m)$. On a :

$$S(m) = 2S(m/2) + m \Rightarrow S(m) = O(m \log m).$$

- Finalement :

$$T(n) = T(2^m) = S(m) = O(m \log m) = O(\log n \log\log n).$$

Résumé

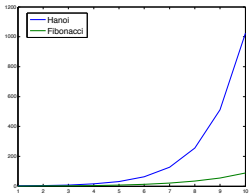
- Outils de résolution d'équations récurrentes :
  - Méthodes génériques : "Deviner-et-Vérifier", "Plug-and-Chug", arbres de récursion
  - Récurrences linéaires d'ordre k (à coefficients constants)
  - Récurrences "Diviser pour régner" : théorème "Master", théorème d'Akra-Bazzi

  Les deux dernières sont les plus systématiques.
- Le plus dur reste de traduire un problème réel en une équation récurrente.
- Exemple : Soit un type de plante qui vit éternellement, mais qui peut seulement se reproduire la première année. A quelle vitesse la population croît-elle ?

Comparaisons de récurrences : linéaires versus "d-p-r"

	Récurrence	Solution
Tours de Hanoï	$T_n = 2T_{n-1} + 1$	$T_n \sim 2^n$
Tours de Hanoï 2	$T_n = 2T_{n-1} + n$	$T_n \sim 2 \cdot 2^n$
Algo rapide	$T_n = 2T_{n/2} + 1$	$T_n \sim n$
Tri par fusion	$T_n = 2T_{n/2} + n - 1$	$T_n \sim n \log n$
Fibonacci	$T_n = T_{n-1} + T_{n-2}$	$T_n \sim (1.618\ldots)^{n+1}/\sqrt{5}$

- Récurrences "Diviser pour régner" généralement polynomiales
- Récurrences linéaires généralement exponentielles
- Générer des sous-problèmes petits est beaucoup plus important que de réduire la complexité du terme non homogène
  - Tri par fusion et Fibanocci sont exponentiellement plus rapides que les tours de Hanoï



Comparaisons de récurrences : nombre de sous-problèmes

Récurrences linéaires :

$$T_n = 2T_{n-1} + 1 \quad \Rightarrow \quad T_n = \Theta(2^n)$$
$$T_n = 3T_{n-1} + 1 \quad \Rightarrow \quad T_n = \Theta(3^n)$$

Augmentation exponentielle des temps de calcul quand on passe de 2 à 3 sous-problèmes.

Récurrence "diviser-pour-régner" :

$$\begin{aligned} T_1 &= 0 \\ T_n &= aT_{n/2} + n - 1 \end{aligned}$$

Par le master théorème, on a :

$$T_n = \begin{cases} \Theta(n) & \text{pour } a < 2 \\ \Theta(n \log_2 n) & \text{pour } a = 2 \\ \Theta(n^{\log_2 a}) & \text{pour } a > 2. \end{cases}$$

La solution est complètement différente entre $a = 1.99$ et $a = 2.01$.

Chapitre 8

Fonctions génératrices

Plan

Lectures conseillées :

- MCS, Chapitre 15
- R. Sedgewick et P. Flagolet, *Analysis of Algorithms*, Addison-Wesley, 1995. `http://aofa.cs.princeton.edu/`.
- Cours "Analyse de structures de données et d'algorithmes" de C. Lavault `http://lipn.univ-paris13.fr/~lavault/Polys/Polymath.pdf`

Introduction

Les fonctions génératrices forment un lien entre l'analyse mathématique des fonctions à valeurs réelles, et les problèmes portant sur les *séquences*.

Motivation : Utiliser les fonctions génératrices pour résoudre des récurrences et des problèmes de dénombrement d'ensembles.

Notation : Dans ce chapitre, on dénotera les *séquences* en utilisant les symboles $\langle \ldots \rangle$.

Définition

Définition : La *fonction génératrice ordinaire* correspondant à la séquence infinie $\langle g_0, g_1, g_2, g_3, \ldots \rangle$ est la *série formelle*

$$G(x) = g_0 + g_1 x + g_2 x^2 + g_3 x^3 + \cdots = \sum_{n=0}^{\infty} g_n x^n.$$

Notations :

- $\langle g_0, g_1, g_2, g_3, \ldots \rangle \longleftrightarrow g_0 + g_1 x + g_2 x^2 + g_3 x^3 + \cdots$
- $[x^n]G(x)$ est le coefficient de x^n dans la séquence générée par $G(x)$.

Remarque : Les fonctions génératrices ne seront que très rarement évaluées. Dans ce chapitre, les questions de convergence n'ont donc en général pas d'importance.

Exemples :

- $\langle 0, 0, 0, 0, \ldots \rangle \longleftrightarrow 0 + 0x + 0x^2 + 0x^3 + \cdots = 0$
- $\langle 1, 0, 0, 0, \ldots \rangle \longleftrightarrow 1 + 0x + 0x^2 + 0x^3 + \cdots = 1$
- $\langle 3, 2, 1, 0, \ldots \rangle \longleftrightarrow 3 + 2x + 1x^2 + 0x^3 + \cdots = 3 + 2x + x^2$

Rappel : $1 + z + z^2 + z^3 + \cdots = \sum_{n=0}^{\infty} z^n = \frac{1}{1-z}$.

On a donc :

- $\langle 1, 1, 1, 1, \ldots \rangle \longleftrightarrow \sum_{n=0}^{\infty} x^n = \frac{1}{1-x}$
- $\langle 1, -1, 1, -1, \ldots \rangle \longleftrightarrow \left(\sum_{n=0}^{\infty} x^{2n}\right) - \left(\sum_{n=0}^{\infty} x^{2n+1}\right) = \left(\sum_{n=0}^{\infty} x^{2n}\right) - x\left(\sum_{n=0}^{\infty} x^{2n}\right) = \frac{1-x}{1-x^2} = \frac{1}{1+x}$
- $\langle 1, a, a^2, a^3, \ldots \rangle \longleftrightarrow \sum_{n=0}^{\infty} (ax)^n = \frac{1}{1-ax}$
- $\langle 1, 0, 1, 0, 1, 0, \ldots \rangle \longleftrightarrow \sum_{n=0}^{\infty} x^{2n} = \frac{1}{1-x^2}$

Multiplication par une constante

Propriété : Si $\langle f_0, f_1, f_2, \ldots \rangle \longleftrightarrow F(x)$, alors

$$\langle cf_0, cf_1, cf_2, \ldots \rangle \longleftrightarrow c \cdot F(x).$$

Démonstration :

$$\begin{aligned} \langle cf_0, cf_1, cf_2, \ldots \rangle &\longleftrightarrow \sum_{n=0}^{\infty} cf_n x^n \\ &= c \sum_{n=0}^{\infty} f_n x^n = c \cdot F(x) \end{aligned}$$

□

Addition

Propriété : Si

$$\langle f_0, f_1, f_2, \ldots \rangle \longleftrightarrow F(x) \quad \text{et} \quad \langle g_0, g_1, g_2, \ldots \rangle \longleftrightarrow G(x),$$

alors $\langle f_0 + g_0, f_1 + g_1, f_2 + g_2, \ldots \rangle \longleftrightarrow F(x) + G(x)$.

Démonstration :

$$\begin{aligned} \langle f_0 + g_0, f_1 + g_1, f_2 + g_2, \ldots \rangle &\longleftrightarrow \sum_{n=0}^{\infty} (f_n + g_n) x^n \\ &= \left(\sum_{n=0}^{\infty} f_n x^n \right) + \left(\sum_{n=0}^{\infty} g_n x^n \right) \\ &= F(x) + G(x) \end{aligned}$$

□

Exemples

- Multiplication par une constante :

$$\langle 1, 0, 1, 0, 1, 0, \ldots \rangle \longleftrightarrow 1 + x^2 + x^4 + x^6 + \ldots = \frac{1}{1 - x^2}$$

  En multipliant la fonction génératrice par 2 :

$$\langle 2, 0, 2, 0, 2, 0, \ldots \rangle \longleftrightarrow 2 + 2x^2 + 2x^4 + 2x^6 + \ldots = \frac{2}{1 - x^2}$$

- Addition :

$$\begin{array}{rccccccccc} & \langle & 1, & 1, & 1, & 1, & 1, & 1, & \ldots & \rangle & \longleftrightarrow & \frac{1}{1-x} \\ + & \langle & 1, & -1, & 1, & -1, & 1, & -1, & \ldots & \rangle & \longleftrightarrow & \frac{1}{1+x} \\ \hline & \langle & 2, & 0, & 2, & 0, & 2, & 0, & \ldots & \rangle & \longleftrightarrow & \frac{1}{1-x} + \frac{1}{1+x} \\ & & & & & & & & & & & = \frac{2}{1-x^2} \end{array}$$

Décalage vers la droite

Propriété : Si $\langle f_0, f_1, f_2, \ldots \rangle \longleftrightarrow F(x)$, alors

$$\langle \overbrace{0, 0, \ldots, 0}^{k \text{ zéros}}, f_0, f_1, f_2, \ldots \rangle \longleftrightarrow x^k \cdot F(x).$$

Démonstration :

$$\begin{aligned} \langle \overbrace{0, 0, \ldots, 0}^{k \text{ zéros}}, f_0, f_1, f_2, \ldots \rangle &\longleftrightarrow \sum_{n=0}^{\infty} f_n x^{n+k} \\ &= x^k \sum_{n=0}^{\infty} f_n x^n = x^k F(x) \end{aligned}$$

□

Dérivation et intégration

Propriété : Si $\langle f_0, f_1, f_2, \ldots \rangle \longleftrightarrow F(x)$, alors $\langle f_1, 2f_2, 3f_3, \ldots \rangle \longleftrightarrow F'(x)$.

Démonstration :

$$\langle f_1, 2f_2, 3f_3, \ldots \rangle \longleftrightarrow \sum_{n=1}^{\infty} n f_n x^{n-1} = \frac{d}{dx} \sum_{n=0}^{\infty} f_n x^n = \frac{d}{dx} F(x)$$

□

Propriété : Si $\langle f_0, f_1, f_2, \ldots \rangle \longleftrightarrow F(x)$, alors
$\langle 0, f_0, \frac{f_1}{2}, \frac{f_2}{3}, \ldots, \frac{f_n}{n}, \ldots \rangle \longleftrightarrow \int_0^x F(t)dt$.

(Dérivation = multiplication par l'index et décalage vers la gauche
Intégration = division par l'index et décalage vers la droite)

Application

Exercice : Trouver une fonction génératrice pour la séquence $\langle 0, 1, 4, 9, 16, \ldots \rangle$.

Réponse : Soit $F(x) = \dfrac{1}{1-x}$. On a successivement

- $\langle 1, 1, 1, 1, \ldots \rangle \longleftrightarrow F(x)$
- $\langle 1, 2, 3, 4, \ldots \rangle \longleftrightarrow F'(x)$
- $\langle 0, 1, 2, 3, \ldots \rangle \longleftrightarrow x \cdot F'(x)$
- $\langle 1, 4, 9, 16, \ldots \rangle \longleftrightarrow (x \cdot F'(x))'$
- $\langle 0, 1, 4, 9, 16, \ldots \rangle \longleftrightarrow x \cdot (x \cdot F'(x))'$.

En développant, on obtient $\langle 0, 1, 4, 9, 16, \ldots \rangle \longleftrightarrow \dfrac{x \cdot (1+x)}{(1-x)^3}$.

Produit

Propriété :
Si $\langle a_0, a_1, a_2, \ldots \rangle \longleftrightarrow A(x)$ et $\langle b_0, b_1, b_2, \ldots \rangle \longleftrightarrow B(x)$, alors

$$\langle c_0, c_1, c_2, \ldots \rangle \longleftrightarrow A(x) \cdot B(x),$$

où

$$c_n = a_0 b_n + a_1 b_{n-1} + a_2 b_{n-2} + \ldots + a_n b_0.$$

Démonstration : Soient $A(x) = \sum_{n=0}^{\infty} a_n x^n$ et $B(x) = \sum_{n=0}^{\infty} b_n x^n$, on a

$$C(x) = A(x) \cdot B(x) = \sum_{n=0}^{\infty} c_n x^n.$$

Coefficients c_n :

	b_0x^0	b_1x^1	b_2x^2	b_3x^3	$\cdots$
a_0x^0	$a_0b_0x^0$	$a_0b_1x^1$	$a_0b_2x^2$	$a_0b_3x^3$	$\cdots$
a_1x^1	$a_1b_0x^1$	$a_1b_1x^2$	$a_1b_2x^3$	$\cdots$	
a_2x^2	$a_2b_0x^2$	$a_2b_1x^3$	$\cdots$		
a_3x^3	$a_3b_0x^3$	$\cdots$			
$\vdots$	$\cdots$				

($\langle c_0, c_1, c_2, \ldots \rangle$ est appelée la *convolution* des séquences $\langle a_0, a_1, a_2, \ldots \rangle$ et $\langle b_0, b_1, b_2, \ldots \rangle$)

Sommes partielles

Propriété : Si $\langle a_0, a_1, a_2, \ldots \rangle \longleftrightarrow A(x)$, alors

$$\langle s_0, s_1, s_2, \ldots \rangle \longleftrightarrow \frac{A(x)}{1-x} \text{ où } s_n = \sum_{i=0}^{n} a_i \text{ pour } n \geq 0.$$

Démonstration : On a :

$$\langle 1, 1, 1, \ldots \rangle \longleftrightarrow \frac{1}{1-x}.$$

Par la règle du produit, le $n$ième terme de $A(x)/(1-x)$ est donné par :

$$a_0 \cdot 1 + a_1 \cdot 1 + a_2 \cdot 1 + \ldots + a_n \cdot 1 = \sum_{i=0}^{n} a_i.$$

□

Exemple : somme des carrés

Supposons qu'on veuille calculer $s_n = \sum_{i=0}^{n} i^2$ (voir chapitre 5).

On sait que (cf. transp. 410) :

$$\langle 0, 1, 4, 9, 16, \ldots \rangle \longleftrightarrow \frac{x \cdot (1+x)}{(1-x)^3}$$

Par la propriété précédente :

$$\langle s_0, s_1, s_2, s_3, \ldots \rangle \longleftrightarrow \frac{x \cdot (1+x)}{(1-x)^4}$$

s_n est donc le coefficient de x^n dans $\frac{x \cdot (1+x)}{(1-x)^4}$.

Extraction des coefficients

Propriété (séries de Taylor) : Si $F(x)$ est la fonction génératrice pour la séquence

$$\langle f_0, f_1, f_2, \ldots \rangle,$$

alors

$$f_0 = F(0), \quad f_n = \frac{F^{(n)}(0)}{n!} \text{ pour } n \geq 1$$

Démonstration :
Directe en dérivant $F(x) = f_0 + f_1 x + f_2 x^2 + \ldots$. □

Exemple :

- $F(x) = \frac{1}{1-x} \Rightarrow \frac{F^{(n)}(x)}{n!} = \frac{n!}{n!(1-x)^{n+1}} \Rightarrow \frac{F^{(n)}(0)}{n!} = \frac{n!}{n!(1-0)^{n+1}} = 1$
- $F(x) = e^x \Rightarrow \frac{F^{(n)}(0)}{n!} = \frac{1}{n!}$

Exemple : somme des carrés

- Calculons le $n$ième terme de

$$F(x) = \frac{x(1+x)}{(1-x)^4} = \frac{x}{(1-x)^4} + \frac{x^2}{(1-x)^4}.$$

- Par les propriétés d'addition et de décalage vers la droite, le coefficient de x^n dans $F(x)$ est donc le coefficient de x^{n-1} dans $\frac{1}{(1-x)^4}$ et le coefficient de x^{n-2} dans $\frac{1}{(1-x)^4}$.
- Soit $G(x) = 1/(1-x)^4$,

$$G^{(n)}(x) = \frac{(n+3)!}{6(1-x)^{n+4}} \Rightarrow \frac{G^{(n)}(0)}{n!} = \frac{(n+3)(n+2)(n+1)}{6}$$

- Finalement :

$$\sum_{i=0}^{n} i^2 = \frac{(n+2)(n+1)n}{6} + \frac{(n+1)n(n-1)}{6} = \frac{(2n+1)(n+1)n}{6}$$

Formule de Newton généralisée

Soit la fonction génératrice $F(x) = (1+x)^\alpha$, avec $\alpha > 0$. On a :

$$F^{(n)}(x) = \alpha(\alpha-1)(\alpha-2)\dots(\alpha-k+1)(1+x)^{\alpha-k},$$

dont on déduit :

$$(1+x)^\alpha = \sum_{n=0}^{\infty} C_\alpha^n x^n$$

où

$$C_\alpha^n = \frac{\prod_{i=0}^{k-1}(\alpha-i)}{n!}.$$

Dans le cas où $\alpha = k$ est un entier, on retrouve la formule du binôme de Newton :

$$(1+x)^k = \sum_{n=0}^{k} C_k^n x^n,$$

avec $C_k^n = \frac{k!}{(k-n)!n!}$. En d'autres mots :

$$\langle C_k^0, C_k^1, C_k^2, \dots, C_k^k, 0, 0, 0, \dots\rangle \longleftrightarrow (1+x)^k$$

Synthèse : fonctions génératrices usuelles

$$(1+x)^\alpha = \sum_{n=0}^{\infty} C_\alpha^n x^n \qquad \exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n \qquad \frac{1}{1-\lambda x} = \sum_{n=0}^{\infty} \lambda^n x^n$$

$$\frac{x}{(1-x)^2} = \sum_{n=0}^{\infty} n x^n \qquad \frac{x^2}{(1-x)^3} = \sum_{n=2}^{\infty} C_n^2 x^n$$

$$\frac{1}{(1-x)^{\alpha+1}} = \sum_{n=0}^{\infty} C_{n+\alpha}^n x^n \qquad \frac{x^m}{(1-x)^{m+1}} = \sum_{n=m}^{\infty} C_n^m x^n$$

$$\ln(1+x) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n!} x^n \qquad \ln\left(\frac{1}{1-x}\right) = \sum_{n=1}^{\infty} \frac{x^n}{n}$$

Exercice : prouvez les

Synthèse : opérations entre fonctions génératrices

Soit $U(x)$, $V(x)$, et $W(x)$ des fonctions génératrices avec $[x^n]U(x) = u_n$, $[x^n]V(x) = v_n$ et $[x^n]W(x) = w_n$.

$$\begin{aligned}
w_n = \alpha u_n + \beta v_n &\quad\Leftrightarrow\quad W(x) = \alpha U(x) + \beta V(x) \\
w_n = \sum_{k=0}^{n} u_{n-k} v_k &\quad\Leftrightarrow\quad W(x) = U(x)V(x) \\
v_n = u_{n-k} &\quad\Leftrightarrow\quad V(x) = x^k U(x) \\
v_n = u_n - u_{n-1} \text{ et } v_o = 0 &\quad\Leftrightarrow\quad V(x) = (1-x)U(x) \\
v_n = \sum_{k=0}^{n} u_k &\quad\Leftrightarrow\quad V(x) = \frac{U(x)}{1-x} \\
v_n = (n+1)u_{n+1} &\quad\Leftrightarrow\quad V(x) = \frac{d}{dx}U(x) \\
v_n = \frac{u_{n-1}}{n} \text{ et } v_0 = 0 &\quad\Leftrightarrow\quad V(x) = \int_0^x U(t)dt
\end{aligned}$$

Exercice : prouvez les

Plan

Résolution de récurrences

Principe général (pour toute récurrence)

- Trouver une fonction génératrice $F(x)$ pour la séquence définie par le récurrence (en multipliant la récurrence par x^n et en sommant sur $\sum_{n=0}^{\infty}$)
- Extraire une formulation analytique pour $[x^n]F(x)$.

Illustration sur la séquence de Fibonacci :

$$\begin{aligned} f_0 &= 0 \\ f_1 &= 1 \\ f_n &= f_{n-1} + f_{n-2} \quad (\text{pour } n \geq 2) \end{aligned}$$

Première étape : trouver $F(x)$ tel que

$$\langle 0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots \rangle \longleftrightarrow F(x) = \sum_{n=0}^{\infty} f_n x^n$$

Soit la récurrence

$$f_n = f_{n-1} + f_{n-2} \text{ (pour } n \geq 2)$$

En multipliant gauche et droite par x^n et en sommant sur n :

$$\begin{aligned} & \sum_{n=2}^{\infty} f_n x^n = \sum_{n=2}^{\infty} f_{n-1} x^n + \sum_{n=2}^{\infty} f_{n-2} x^n \\ \Leftrightarrow \quad & \sum_{n=0}^{\infty} f_n x^n - f_0 - x f_1 = x \sum_{n=2}^{\infty} f_{n-1} x^{n-1} + x^2 \sum_{n=2}^{\infty} f_{n-2} x^{n-2} \\ \Leftrightarrow \quad & \sum_{n=0}^{\infty} f_n x^n - f_0 - x f_1 = x \sum_{n=0}^{\infty} f_n x^n - f_0 + x^2 \sum_{n=0}^{\infty} f_n x^n \\ \Leftrightarrow \quad & F(x) = x + xF(x) + x^2 F(x) \\ \Leftrightarrow \quad & F(x) = \frac{x}{1 - x - x^2} \end{aligned}$$

Deuxième étape : trouver une formulation analytique pour le coefficient de x^n dans la série de puissance de $\frac{x}{1-x-x^2}$.

Extraction des coefficients

Calculons la décomposition en fractions partielles de $F(x)$:

- Factorisons le dénominateur :

$$(1 - x - x^2) = (1 - \alpha_1 x)(1 - \alpha_2 x),$$

où $\alpha_1 = (1 + \sqrt{5})/2$ et $\alpha_2 = (1 - \sqrt{5})/2$.

- Trouvons A_1 et A_2 tels que :

$$\frac{x}{1 - x - x^2} = \frac{A_1}{1 - \alpha_1 x} + \frac{A_2}{1 - \alpha_2 x}.$$

En prenant quelques valeurs de x, on obtient :

$$A_1 = \frac{1}{\alpha_1 - \alpha_2} = \frac{1}{\sqrt{5}}, \; A_2 = \frac{-1}{\alpha_1 - \alpha_2} = -\frac{1}{\sqrt{5}}.$$

En substituant :

$$\frac{x}{1 - x - x^2} = \frac{1}{\sqrt{5}} \left(\frac{1}{1 - \alpha_1 x} - \frac{1}{1 - \alpha_2 x} \right).$$

Puisque

$$\frac{1}{1 - \alpha x} = 1 + \alpha x + \alpha^2 x^2 + \ldots,$$

on obtient

$$F(x) = \frac{1}{\sqrt{5}} \left((1 + \alpha_1 x + \alpha_1^2 x^2 + \ldots) - (1 + \alpha_2 x + \alpha_2^2 x^2 + \ldots) \right)$$

Par identification :

$$[x^n]F(x) = f_n = \frac{\alpha_1^n - \alpha_2^n}{\sqrt{5}} = \frac{1}{\sqrt{5}} \left(\left(\frac{1 + \sqrt{5}}{2} \right)^n - \left(\frac{1 - \sqrt{5}}{2} \right)^n \right).$$

Récurrences linéaires (homogènes, à coefficients constants)

Propriété : Soit une récurrence linéaire homogène de la forme générale $u_n = a_1 u_{n-1} + \ldots + a_k u_{n-k}$ (pour $n \geq k$) et soit $G(x) = \sum_{n=0}^{\infty} u_n x^n$ la fonction génératrice associée à la séquence $\langle u_0, u_1, \ldots, u_n, \ldots \rangle$. On a

$$G(x) = \frac{U(x)}{V(x)},$$

où

$$\begin{aligned} U(x) &= P_{k-1}(x) - a_1 x P_{k-2}(x) - \ldots - a_{k-1} x^{k-1} P_0(x) \\ V(x) &= 1 - a_1 x - a_2 x - \ldots - a_k x^k \\ P_i(x) &= u_0 + u_1 x + \ldots + u_i x^i \ (i = 0, \ldots, k-1). \end{aligned}$$

NB :

- $V(\frac{1}{x}) = 0$ est l'équation caractéristique (voir transp. 379).
- $V(x)$ est indépendant des conditions initiales (pas $U(x)$).

Démonstration : En multipliant la récurrence par x^n, pour $n \geq k$:

$$\begin{aligned} u_n x^n &= a_1 u_{n-1} x^n + a_2 u_{n-2} x^n + \ldots + a_k u_{n-k} x^n \\ &= a_1 x u_{n-1} x^{n-1} + a_2 x^2 u_{n-2} x^{n-2} + \ldots + a_k x^k u_{n-k} x^{n-k} \end{aligned}$$

En sommant sur $n \geq k$:

$$\begin{aligned} u_n x^n &= a_1 x \sum_{n=k}^{\infty} u_{n-1} x^{n-1} \ldots + a_k x^k \sum_{n=k}^{\infty} u_{n-k} x^{n-k} \\ &= (a_1 x) \sum_{n=k-1}^{\infty} u_n x^n + \ldots + \sum_{n=0}^{\infty} u_n x^n \end{aligned}$$

En posant $P_i(x) = u_0 + u_1 x + \ldots + u_i x^i$ $(i = 1, \ldots, k-1)$, on obtient :

$$\begin{aligned} G(x) - P_{k-1}(x) &= a_1 x (G(x) - P_{k-2}(x)) + \ldots \\ &+ a_{k-1} x^{k-1} (G(x) - P_0(x)) + a_k x^k G(x). \end{aligned}$$

D'où on tire (*vérifier*) :

$$G(x) = \frac{U(x)}{V(x)}.$$

□

Nombres de Catalan

Théorème : Soit la récurrence (transp. 349) :

$$\begin{aligned} b_0 &= 1 \\ b_n &= \sum_{k=0}^{n-1} b_k b_{n-1-k} \text{ pour } n > 0. \end{aligned}$$

On a $b_n = \frac{1}{n+1} C_{2n}^n$.

Démonstration : Soit $B(x) = \sum_{n=0}^{\infty} b_n x^n$ la fonction génératrice correspondant à la séquence $\langle b_0, b_1, \ldots, b_n, \ldots \rangle$. Par définition de b_n, on a :

$$\begin{aligned} B(x) &= 1 + \sum_{n=1}^{\infty} \left(\sum_{k=0}^{n-1} b_k b_{n-k-1} \right) x^n = 1 + x \sum_{n=1}^{\infty} \left(\sum_{k=0}^{n-1} b_k b_{n-k-1} \right) x^{n-1} \\ &= 1 + x \sum_{n=0}^{\infty} \left(\sum_{k=0}^{n} b_k b_{n-k} \right) x^n \text{ (car } b_0 = 1) \\ &= 1 + x B(x)^2 \text{ (par définition du produit).} \end{aligned}$$

$B(x)$ est donc solution de $xB^2(x) - B(x) + 1 = 0$, ce qui donne :

$$B(x) = \frac{1 - \sqrt{1-4x}}{2x} \text{ ou } B(x) = \frac{1 + \sqrt{1-4x}}{2x}$$

Puisqu'on doit avoir $B(0) = b_0 = 1$, seule la première solution est acceptable. En utilisant la formule de Newton généralisée (transp. 417), on obtient

$$1 - (1-4x)^{1/2} = 1 - \sum_{n=0}^{\infty} C_{1/2}^n (-4x)^n = -\sum_{n=1}^{\infty} C_{1/2}^n (-4x)^n,$$

et donc en divisant par $2x$:

$$\begin{aligned} b_n &= -\frac{1}{2} C_{1/2}^n (-4)^{n+1} \\ &= -\frac{1}{2} \frac{\frac{1}{2}(\frac{1}{2}-1)(\frac{1}{2}-2)\ldots(\frac{1}{2}-n)(-4)^{n+1}}{(n+1)!} \\ &= \frac{1 \cdot 3 \cdots 5 \cdots (2n-1) \cdot 2^N}{(n+1)!} \\ &= \frac{1}{n+1} \frac{1 \cdot 3 \cdots 5 \cdots (2n-1)}{n!} \frac{2 \cdot 4 \cdot 6 \cdots 2n}{1 \cdot 2 \cdot 3 \cdot n} \\ &= \frac{1}{n+1} C_{2n}^n \end{aligned}$$

Plan

Dénombrements

Beaucoup de problèmes de dénombrements peuvent être modélisés par des récurrences.

Les fonctions génératrices permettent de résoudre des récurrences et donc indirectement des problèmes de dénombrement.

Dans la suite du cours, on va voir comment utiliser des fonctions génératrices pour modéliser *plus directement* des problèmes de dénombrements.

Classes combinatoires

Problème général de dénombrement : Soit un ensemble $\mathcal{A}$ d'objets (au sens très large) et une fonction $|a|$ qui mesure la "taille" d'un élément de $\mathcal{A}$. On cherche à calculer a_n, le nombre d'éléments de $\mathcal{A}$ de taille n.

Solution :

- On définit la séquence $\langle a_0, a_1, a_2, \ldots \rangle$ et la fonction génératrice correspondante $A(x)$ qui est telle que :

$$A(x) = \sum_{n=0}^{\infty} a_n x^n = \sum_{a \in \mathcal{A}} x^{|a|}.$$

- On dérive une solution analytique pour $A(x)$. Deux approches :
  - On exploite une définition récursive des éléments de $\mathcal{A}$
  - On construit $\mathcal{A}$ à partir d'ensembles dont les fonctions génératrices sont connues
- On déduit a_n de $A(x)$ (par ex. en utilisant Taylor)

Application 1 : chaîne de caractères

On cherche à déterminer le nombre de séquences de N bits qui contiennent exactement k 1, que nous noterons $b_{N,k}$. Soit $\mathcal{B}_N$ l'ensemble des séquences de bits de longueur N et définissons la "taille" $|b|$ d'un élément $b \in \mathcal{B}_N$ par le nombre de 1 dans b.
Soit la fonction génératrice $B_N(x) = \sum_{k=0}^{\infty} b_{N,k} x^k$. Etant donné qu'une séquence de N bits ($N > 1$) commence soit par 0, soit par 1, on a :

$$\begin{aligned} B_N(x) &= \sum_{b \in \mathcal{B}_N} x^{|b|} \\ &= \sum_{b \in \mathcal{B}_{N-1}} x^{|b|} + \sum_{b \in \mathcal{B}_{N-1}} x^{1+|b|} \\ &= (1+x) B_{N-1}(x) \end{aligned}$$

Puisque $B_1(x) = 1 + x$, on a :

$$B_N(x) = (1+x)^N,$$

dont on déduit (voir transp. 417) :

$$b_{N,k} = C_N^k$$

Application 2 : nombre d'arbres binaires

Soit $\mathcal{B}$ l'ensemble des arbres binaires et soit $|B|$ le nombre de nœuds d'un arbre $B \in \mathcal{B}$.

Etant donné la définition récursive d'un arbre binaire, on a :

$$\begin{aligned} B(x) &= \sum_{n=0}^{\infty} b_n x^b = \sum_{B \in \mathcal{B}} x^{|B|} && (1) \\ &= 1 + \sum_{B_L \in \mathcal{B}} \sum_{B_G \in \mathcal{B}} x^{|B_R|+|B_L|+1} && (2) \\ &= 1 + xB(x)^2 && (3) \end{aligned}$$

(où le 1 provient de l'existence d'un seul arbre vide).

On en déduit donc (voir transp. 427) :

$$b_n = \frac{1}{N+1} C_{2N}^N$$

Démonstration :

1. Soit a_n et b_n le nombre d'éléments de taille n dans $\mathcal{A}$ et $\mathcal{B}$ respectivement, $a_n + b_n$ est le nombre d'éléments de taille n dans $\mathcal{A} + \mathcal{B}$.
2. Le nombre d'objets de taille n dans $\mathcal{A} \times \mathcal{B}$ est donné par :

   $$\sum_{k=0}^{n} a_k b_{n-k}$$

   qui est le terme général de $A(x) \cdot B(x)$. On a aussi :

   $$\sum_{\gamma \in \mathcal{A} \times \mathcal{B}} z^{|\gamma|} = \sum_{\alpha \in \mathcal{A}} \sum_{\beta \in \mathcal{B}} x^{|\alpha|+|\beta|} = A(x) \cdot B(x).$$

3. De $\text{seq}(\mathcal{A}) = \epsilon + \mathcal{A} \times \mathcal{A} + \mathcal{A} \times \mathcal{A} \times \mathcal{A} + \ldots$, on en déduit que la fonction génératrice de $\text{seq}(\mathcal{A})$ est :

   $$1 + A(x) + A(x)^2 + A(x)^3 + \ldots = \frac{1}{1 - A(x)}$$

□

Opérations entre ensembles

Certaines opérations sur les ensembles peuvent se transposer à des opérations sur les fonctions génératrices.

Définition Soit deux ensembles $\mathcal{A}$ et $\mathcal{B}$:

- $\mathcal{A} + \mathcal{B}$ désigne la réunion disjointe de $\mathcal{A}$ et $\mathcal{B}$ (si un élément appartient aux deux ensembles, il sera copié deux fois dans $\mathcal{A} + \mathcal{B}$)
- $\mathcal{A} \times \mathcal{B}$ désigne le produit cartésien de $\mathcal{A}$ et $\mathcal{B}$ (c'est-à-dire l'ensemble des paires (a, b) où $a \in \mathcal{A}$ et $b \in \mathcal{B}$)
- $\text{seq}(A) = \epsilon + \mathcal{A} \times \mathcal{A} + \mathcal{A} \times \mathcal{A} \times \mathcal{A} + \ldots$ est l'ensemble des séquences d'éléments de A (où ϵ représente la séquence de longueur 0).

Théorème : Soit $A(x)$ et $B(x)$ les fonctions génératrices énumérant les éléments de taille n dans $\mathcal{A}$ et $\mathcal{B}$ respectivement. On a :

1. $A(x) + B(x)$ est la fonction génératrice pour $\mathcal{A} + \mathcal{B}$
2. $A(x) \cdot B(x)$ est la fonction génératrice pour $\mathcal{A} \times \mathcal{B}$
3. $\frac{1}{1-A(x)}$ est la fonction génératrice pour $\text{seq}(\mathcal{A})$.

Applications

- Etant donné la définition d'un arbre binaire, l'ensemble des arbres binaires $\mathcal{B}$ peut se définir comme suit :

  $$\mathcal{B} = \{\emptyset\} + \{\textbf{branch}(\emptyset, \emptyset)\} \times \mathcal{B} \times \mathcal{B}$$

  La fonction génératrice de $\{\emptyset\}$ étant 1 et celle de $\{\textbf{branch}(\emptyset, \emptyset)\}$ étant x, on obtient :

  $$B(x) = 1 + xB(x)^2$$

- Soit $\mathcal{B}$ l'ensemble des chaînes binaires. On a $\mathcal{B} = \text{seq}(\{0, 1\})$. Puisque la fonction génératrice de $\{0, 1\}$ est $2x$, on a :

  $$B(x) = \frac{1}{1 - 2x},$$

  dont on déduit :

  $$b_n = [x^n]B(x) = 2^n,$$

  où b_n est le nombre de chaînes de bits de longueur n.

  Note : on a aussi $\mathcal{B} = \epsilon + \{0, 1\} \times \mathcal{B}$, où ϵ est la chaîne de longueur 0.

Applications : choix avec répétition

Question : De combien de façons peut-on choisir n éléments (avec répétition) lorsque l'on a k sortes d'éléments disponibles ?

Solution :

- Soit $\mathcal{A}_i$ l'ensemble des ensembles d'éléments de la $i$ème sorte :

$$\mathcal{A}_i = \{\epsilon, \{s_i\}, \{s_i, s_i\}, \{s_i, s_i, s_i\}, \ldots\}.$$

  La fonction génératrice correspondante est

$$A_i(x) = 1 + x + x^2 + x^3 + \ldots = \frac{1}{1-x}.$$

- Une ensemble d'éléments choisis parmi k sortes avec répétitions est un tuple pris dans l'ensemble :

$$\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \ldots \times \mathcal{A}_k$$

- On en déduit que le nombre recherché est le coefficient de x^n de la fonction génératrice : $A(x) = \frac{1}{(1-x)^k}$

- Soit $A(x) = \dfrac{1}{(1-x)^k} = (1-x)^{-k}$.
- On obtient
  - $A'(x) = k(1-x)^{-k-1}$
  - $A''(x) = k(k+1)(1-x)^{-k-2}$
  - $A'''(x) = k(k+1)(k+2)(1-x)^{-k-3}$
  - $\ldots$
  - $A^{(n)}(x) = k(k+1)\cdots(k+n-1)(1-x)^{-k-n}$
- Le coefficient cherché est donc

$$\begin{aligned}\frac{A^{(n)}(0)}{n!} &= \frac{k(k+1)\cdots(k+n-1)}{n!} \\ &= \frac{(k+n-1)!}{(k-1)!n!} \\ &= C^n_{k+n-1}.\end{aligned}$$

Applications : comptage difficile

Problème : De combien de manières peut-on composer un panier avec n fruits (pommes, bananes, oranges et fraises) en respectant les contraintes suivantes ?

- Le nombre de pommes doit être pair ;
- Le nombre de bananes doit être un multiple de 5 ;
- Il y a au plus 4 oranges ;
- Il y a au plus 1 fraise.

Exemple : Il existe 7 façons de composer un panier de 6 fruits :

Pommes	6	4	4	2	2	0	0
Bananes	0	0	0	0	0	5	5
Oranges	0	2	1	4	3	1	0
Fraises	0	0	1	0	1	0	1

Ce type de problème est difficile à résoudre sans les fonctions génératrices.

Réponse :

- Fonction génératrice pour le choix des pommes :

$$P(x) = 1 + x^2 + x^4 + x^6 + \cdots = \sum_{n=0}^{\infty} x^{2n} = \frac{1}{1-x^2}.$$

- Fonction génératrice pour le choix des bananes :

$$B(x) = 1 + x^5 + x^{10} + x^{15} + \cdots = \sum_{n=0}^{\infty} x^{5n} = \frac{1}{1-x^5}.$$

- Fonction génératice pour le choix des oranges :

$$O(x) = 1 + x + x^2 + x^3 + x^4 = \frac{1-x^5}{1-x}.$$

- Fonction génératrice pour le choix des fraises :

$$F(x) = 1 + x.$$

- Par la propriété de convolution, la fonction génératrice pour la composition d'un panier de fruits est

$$\begin{aligned} & P(x)B(x)O(x)F(x) \\ = \; & \frac{1}{(1-x^2)}\frac{1}{(1-x^5)}\frac{(1-x^5)}{1-x}(1+x) \\ = \; & \frac{1}{(1-x)^2} \\ = \; & 1 + 2x + 3x^2 + 4x^3 + \cdots \end{aligned}$$

- Le coefficient de x^n est toujours $n+1$.
- Il y a donc $n+1$ façons de composer un panier de n fruits.

Résumé

Les fonctions génératrices constituent un outil très puissant pour résoudre des récurrences et des problèmes de dénombrement

Elles sont particulièrement utiles pour calculer la complexité en moyenne d'algorithmes, qui nécessite de calculer le coût moyen sur toutes les structures d'un certain type de taille n (arbres, graphes, séquences, etc.).

Il existe aussi des fonctions génératrices *exponentielles* qui permettent de prendre en compte des étiquetages d'objets combinatoires (par ex., des arbres avec des noeuds coloriés).

Pour en savoir plus :

- R. Sedgewick et P. Flagolet, *Analysis of Algorithms*, Addison-Wesley, 1995/2012. `http://aofa.cs.princeton.edu/`.