L2 - Logique

Olivier Gasquet, Jan Smaus, Martin Strecker

Université de Toulouse/IRIT

Année 2011/2012

Plan

Problématique

Logique: du grecque *logos* (mot, énoncé, propos)

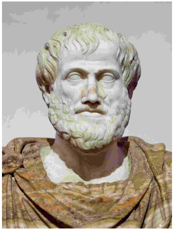
Histoire:

- Premiers développements au 4ème siècle av. J.-C.
- Scolastique médiévale: discipline essentielle (*trivium*: grammaire, rhétorique, logique)
- Refonte à la fin du 19ème siècle
- Crise existentielle au début du 20ème siècle

Objectifs de cette section:

- Connaître le développement historique de la discipline
- Apprécier ses limitations
- (sans comprendre tous les détails)

Histoire de la logique: Aristote



Aristote (-384 .. -322)

- Disciple de Platon
- Précepteur d'Alexandre le Grand
- Contributions essentielles aux sciences naturelles, l'éthique, la logique
- "Le Philosophe" par excellence de la philosophie médiévale

Histoire de la logique: Aristote

Le syllogisme d'Aristote (1)

Instance:

- *Prémisse majeure:*
  Tous les hommes sont mortels
- *Prémisse mineure:*
  Tous les Grecs sont des hommes
- *Conclusion:*
  Tous les Grecs sont mortels

En général:

- Tous les B sont des C
- Tous les A sont des B
- Tous les A sont des C

Histoire de la logique: Aristote

Le syllogisme d'Aristote (2)

Observation essentielle:

- On peut reconnaître un argument valide par sa *forme*
- ... sans regarder la *signification* des mots

En terminologie moderne:

- On peut raisonner de manière *syntaxique*
- ... sans connaissance de la *sémantique*

Histoire de la logique: Aristote

Le syllogisme d'Aristote (3)

Structure générale: deux prémisses, une conclusion

Quatre formes d'énoncé:

- Tout A est B (donc: $A \subseteq B$, pour $A \neq \{\}$)
- Aucun A n'est B (donc: $A \cap B = \{\}$, pour $A \neq \{\}$)
- Quelque A est B (donc: $A \cap B \neq \{\}$)
- Quelque A n'est pas B (donc: $A \not\subseteq B$)

Lesquels des syllogismes suivants sont valides?

- P_1: Tout B est C
- P_2: Quelque A est B
- C: Quelques A est C

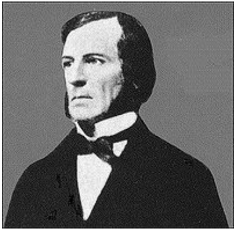
- P_1: Quelque B est C
- P_2: Quelque A est B
- C: Quelques A est C

Histoire de la logique: Leibniz

Leibniz (1646 - 1716)

- Codage binaire des nombres
- Construction de l'un des premiers calculateurs (mécanique, décimal)
- Développement du calcul infinitésimal (en concurrence avec Newton)
- *Ars combinatoria:* l'art de dériver des vérités de manière *calculatoire*, basé sur
  - une *characteristica universalis*, un langage mathématique non-ambigu
  - un *calculus ratiocinator*, un calcul / une machine manipulant la *characteristica*

Histoire de la logique: Boole

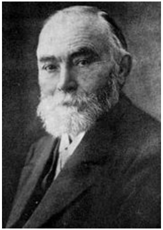


Boole (1815 - 1864)

Livre: An Investigation of the Laws of Thought

- Traitement *algébrique* de la logique propositionnelle
- (ce n'est pas "l'algèbre de Boole" actuelle!)
- Procédure de décision pour la logique propositionnelle

Histoire de la logique: Frege



Frege (1848 - 1925)

- Fondateur de la logique "moderne":
  - les connecteurs "essentiels"de la logique des propositions: $\longrightarrow$ et $\neg$
  - les quantificateurs de la logique des prédicats
  - un calcul formel

Histoire de la logique: Frege

“Begriffsschrift” de Frege (1879)

Les jugements

$$\vdash a \longrightarrow b \longrightarrow a$$

et

$$\vdash (c \longrightarrow b \longrightarrow a) \longrightarrow ((c \longrightarrow b) \longrightarrow (c \longrightarrow a))$$

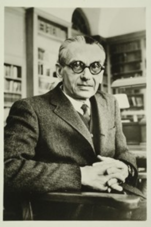
- Distinction entre
  - *formule:* représente une proposition (qui peut être vraie ou fausse)
  - *jugement:* formule dont on constate la vérité
    (dans un calcul donné)
- Notation deux-dimensionnelle qui a laissé des traces:
  - négation ($\neg$)
  - jugement ($\vdash$)

Histoire de la logique: Russel

Russel (1872 - 1970)

- Livre: *Principia Mathematica* (1910 - 1913, avec A. N. Whitehead)
  But: formalisation de la mathématique à partir de quelques notions élémentaires
- Découverte d'un paradoxe (1903) dans la théorie des ensembles de Cantor
  $\rightsquigarrow$ crise des fondements de la mathématique:
  Consistence des axiomes?
- (En plus: prix Nobel de littérature, emprisonné suite à des campagnes contre l'armement nucléaire, ...)

Histoire de la logique: Gödel



Gödel (1906 - 1978)

Le cataclysme: *"Sur des propositions indécidables de Principia Mathematica"* (1931)

- Toute théorie mathématique "suffisamment expressive" est
  - incomplète (ne peut pas prouver tout énoncé vrai)
  - *ou* contradictoire
- Surtout, elle ne permet pas de démontrer sa propre cohérence
- $\rightsquigarrow$ échec du programme rationaliste dans la tradition Leibniz - Hilbert

Histoire de la logique - Résumé

Quelques repères dans un paysage vaste:

- Syllogisme d'Aristote: Possibilité de raisonner en s'appuyant sur la *syntaxe*, sans connaissance de la *sémantique*
- Leibniz: Raisonnement exécutable par une machine (*calcul*)
- Forme moderne de la logique (Boole, Frege)
- Le programme rationnaliste mis à mal: Russel, Gödel

Conclusion: La logique

- a des limites
- ne fournit pas de certitude absolue
- mais est pourtant utile ...

Pour plus d'information sur le développement de la logique au tournant du 20ème siècle: Jean Van Heijenoort: From Frege to Gödel

Plan

Applications de la logique

La logique a d'importantes applications en informatique:

- Intelligence artificielle
- Sûreté de fonctionnement
- Sécurité

Objectifs de cette section:

- Comprendre le rôle de la logique dans l'informatique
- Gagner une intuition des méthodes mises en oeuvre

Application: Intelligence artificielle (1)

But:

- Comprendre mieux le fonctionnement de l’intelligence humaine
- Découvrir les facettes d’un comportement rationnel

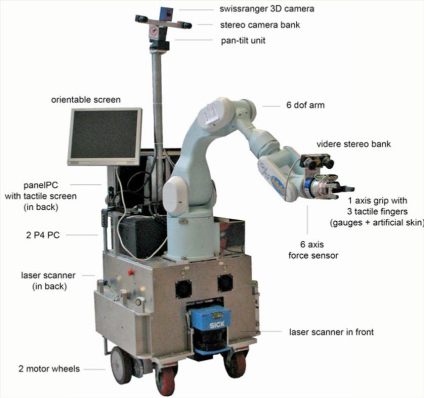
Applications:

- Simuler le comportement humain:
  comportement grégaire en économie (rationalité $\pm$ reduite)
- Guider l’action humaine:
  prévention de conflits
- Imiter le comportement humain: robotique

Application: Intelligence artificielle (2)

Robotique:

- *Contrôle des actions:* Comment déplacer un livre d'une chaise sur une table?
- *Contrôle des mouvements:* Comment se déplacer d'une salle à une autre?
- *Interaction homme-machine:* Comment interpréter une commande en langage naturel?



Robot Jido (LAAS)

Application: Sûreté de fonctionnement (1)

But: S'assurer du bon fonctionnement du matériel et logiciel de systèmes critiques:

- Transport (avions, trains)
- Nucléaire
- Systèmes médicaux

Contexte:

- Systèmes embarqués de plus en plus sophistiqués
- *Exemple:* Système *fly by wire* de l'A330

à lire: Wired: History's worst software bugs

Application: Sûreté de fonctionnement (2)

Explosion de la fusée Ariane 5 (juin 1996):

- 40 sec. après son lancement
- À bord: 4 satellites de recherche
- Dispersion de 745 tonnes de débris, y compris substances toxiques
- Coût des dégats: 290 millions d'Euros

Contexte:

- Premier lancement de la fusée
- Lors du développement: Réutilisation de logiciel de l'Ariane 4 (autres caractéristiques de vol, accéleration moins forte)
- Cause: dépassement arithmétique

Application: Sûreté de fonctionnement (3)

- Exemple du domaine médical: Therac-25 (entre 1985 et 1987)
  - Lors d'une radiothérapie, 6 personnes reçoivent une surdose massive (facteur 100)
  - Conséquence: 3 décès
  - Causes: Multiples, surtout:
    problème de synchronisation de deux tâches
- Mai 2007, à Toulouse:
  - 145 personnes irradiés au CHU de Rangueil
    "c'est encore une fois une déficience de l'informatique qui serait en cause [...] L'étalonnage était mal fait [...]"
  - lire l'article dans le Parisien

Application: Sécurité

Problème:

- Des virus et vers informatiques infectent des systèmes connectés par Internet
- *Exemple:* Ver “Slammer” (janvier 2003)
  - infecte 75.000 machines en 30 min
  - entre autres: blocage du système de surveillance de la centrale nucléaire Davis-Besse (Ohio) pendant 5h

Causes de la vulnérabilité:

- Langages de programmation de bas niveau (assembleur, C)
- Incompréhension du fonctionnement des protocoles de communication
- Cryptage insuffisant

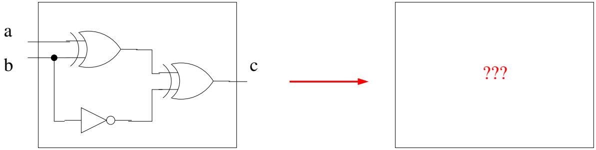
à lire: Wired: Slammer

Méthodes: Preuves d'équivalence

Circuits électroniques

Problème:

- Simplifiez le circuit suivant
- Vérifiez que le résultat est le même



Modélisation: $(a \oplus b) \oplus (\neg b) = ???$

où: $\oplus$ est "xor"

Remarque: Formule en logique propositionnelle

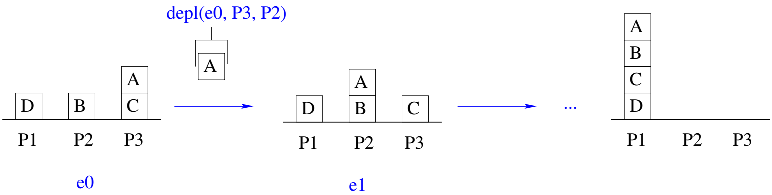
Question: Est-ce que la méthode est aussi praticable pour 1.000 variables?

Méthodes: Systèmes à état

Robotique / Planification:

Exemple:

- Un *état* est charactérisé par des objets A, B, C, D empilés arbitrairement sur trois positions P_1, P_2, P_3
- *Opérations:* Le robot peut déplacer un seul objet au sommet d'une pile vers le sommet d'une autre pile
- *But:* empiler A, B, C, D (dans l'ordre) sur P_1
- *Solution:* séquence d'opérations qui atteignent le but



Méthodes: Systèmes à état

Modélisation:

- Prédicat ternaire `sur`: un bloc est *sur* un autre bloc / sur une position dans un état
  Exemples: `sur(D, P1, e0)` et `sur(A, B, e1)`
- Fonction ternaire `depl`: convertir un état en déplaçant le sommet d'une position vers une autre
  Exemple: L'état `e1` est: `depl(e0, P3, P2)`

à faire:

- Décrivez entièrement l'état `e1`
- Trouvez la séquence d'opérations menant au but

Méthodes: Systèmes à état

Recherche d'une solution: On peut

- décrire le but par une formule logique:
  "*Pour tout* état initial, *il existe* un état final ayant la propriété . . . "
- faire une preuve constructive $\rightsquigarrow$ séquence d'opérations

Remarque: Formule et preuve en logique des prédicats

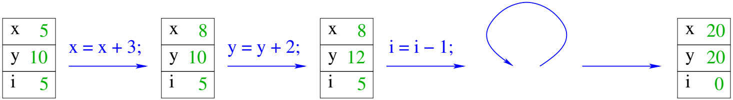
Questions plus générales:

- Comment trouver une solution pour n'importe quel état initial?
  $\rightsquigarrow$ stratégie de preuve
- Est-ce qu'on peut *assurer* qu'une solution existe toujours?
  $\rightsquigarrow$ complétude de la stratégie
- . . . aussi si on a seulement deux positions?

Méthodes: Systèmes à état

Programmes impératifs:

- Un *état* est charactérisé par un ensemble de variables et de valeurs associées.
- *Opérations:*
  - Affectations
  - Séquences, structures de contrôle, boucles



Méthodes: Systèmes à état

Pré- /Postconditions:

```
{ y = 2 * x ∧ x ≥ 0
∧ Y = y ∧ X = x }
i = x;
while (i > 0) {
  x = x + 3;
  y = y + 2;
  i = i - 1;
  }
{ y = x }
```

Correction du programme:

- Est-ce que le programme transforme tout état qui satisfait la *précondition* en un état qui satisfait la *postcondition*?
- Vérification à l'aide d'un invariant:
  $i \geq 0$
  $\wedge\ x = X + 3 * (X - i)$
  $\wedge\ y = Y + 2 * (X - i)$

$\leadsto$ plus de détails dans le cours “Programmation impérative”

Résumé

La logique est essentielle dans des domaines tels que

- l'intelligence artificielle
- la sûreté de fonctionnement et la sécurité

La logique permet de décrire

- l'équivalence fonctionnelle et comportementale (*exemple:* circuit)
- les propriétés des systèmes de transition:
  - *exemple planification:*
    Donné: État initial, état final.
    But: Recherche de séquence d'actions
  - *exemple programme impératif:*
    Donné: Séquence d'actions (le programme)
    But: Vérification de rapport entre un état initial et un état final

Plan

Langages logiques (1)

Les langages logiques sont:

- des abstractions du langage naturel ou mathématique:
  - *Lang. nat.:* “Il fait nuit, *et* la lumière est allumée”
  - *Logique:* $n \wedge l$
  - avec: $n \equiv$ il fait nuit; $l \equiv$ la lumière est allumée
- ...qui ne permettent pas d'exprimer certaines nuances:
  - *Lang. nat.:* “Il fait nuit *mais* la lumière est allumée”
  - *Logique:* $n \wedge l$
- ...mais plus précis:
  - *Lang. nat.:* “Je suis au bureau ou je suis dans le jardin et je lis un livre”
  - *Logique:* $(b \vee j) \wedge l$ ou $b \vee (j \wedge l)$
  - avec: $b \equiv$ je suis au bureau; $j \equiv$ je suis dans le jardin; $l \equiv$ je lis un livre

Langages logiques (2)

La logique des propositions LProp

- est une logique très simple
- fait une abstraction très grossière

Elle ne permet pas d'exprimer

- des rapports entre des *individus*:
  "entre deux nombres, il y a un troisième nombre"
  $\leadsto$ logique des prédicats, plus tard ...
- des rapports *temporels*:
  "S'il fait nuit, la lumière sera éventuellement allumée"
  $\leadsto$ logiques temporelles
- des souhaits, obligations, possibilités, ... :
  "Je pense que s'il fait nuit, la lumière devrait être allumée"
  $\leadsto$ logiques modales

Abstractions de la Lprop (1)

La logique des propositions

- remplace des *propositions* entières par des *variables propositionnelles*
  il fait nuit $\rightsquigarrow n$; la lumière est allumée $\rightsquigarrow l$
- ...et les relie par des *connecteurs logiques*:
  - "Il fait nuit, *et* la lumière est allumée": $n \wedge l$
  - "Il fait nuit, *ou* la lumière est allumée": $n \vee l$
  - "*S'*il fait nuit, *alors* la lumière est allumée": $n \longrightarrow l$

Abstractions de la Lprop (2)

Une proposition correspond à une *phrase élémentaire* en français:

- "Eve mange une grande pomme"
  ou simplément: "Eve mange"

mais pas à

- un substantif: "Eve", "une pomme"
- un verbe: "mange"
- un adjectif: "grand"

Pour avoir une correspondance juste: *réécrire* les phrases:

- *Lang. nat.:* "Eve mange une pomme et un abricot"
- *Logique:* $p \wedge a$
- avec: $p \equiv$ "Eve mange une pomme"; $a \equiv$ "Eve mange un abricot"
- mais pas(!): $p \equiv$ "Eve mange une pomme"; $a \equiv$ "un abricot"

Abstractions de la Lprop (3)

Inversion de l'ordre des connecteurs:

- "Je vais à la plage s'il fait beau"
  $\equiv$ "S'il fait beau, je vais à la plage"
  en logique: $b \longrightarrow p$
  avec: $b \equiv$ il fait beau; $p \equiv$ je vais à la plage

Attention à *l'ambiguïté* du langage naturel:

- "Je vais au cinéma ou à la plage s'il fait beau"
  1. $b \longrightarrow (c \vee p)$
  2. $c \vee (b \longrightarrow p)$

Définition formelle de la Lprop (1)

Définition inductive: Soit *PROP* un ensemble de *variables propositionnelles*.
On définit l'ensemble *FORM* des formules par induction (A et B représentent des chaînes de caractères) :
FORM est le plus petit ensemble qui satisfait les conditions:

1. *Variable propositionnelle:* Pour tout $p \in PROP$, $p \in FORM$
2. *Constante "faux":* $\perp \in FORM$
3. *Négation:* Si $A \in FORM$,
alors $(\neg A) \in FORM$
4. *Conjonction* ("et"): Si $A \in FORM$ et $B \in FORM$,
alors $(A \wedge B) \in FORM$
5. *Disjonction* ("ou"): Si $A \in FORM$ et $B \in FORM$,
alors $(A \vee B) \in FORM$
6. *Implication* ("si ... alors"): Si $A \in FORM$ et $B \in FORM$,
alors $(A \longrightarrow B) \in FORM$

Définition formelle de la Lprop (2)

Comprendre une définition inductive:

$((p \wedge (q \vee r)) \longrightarrow (\neg s)) \in FORM$, parce que:

- $(p \wedge (q \vee r)) \in FORM$, parce que:
  - $p \in FORM$, parce que $p \in PROP$
  - $(q \vee r) \in FORM$, parce que
    - $q \in FORM$, parce que $q \in PROP$
    - $r \in FORM$, parce que $r \in PROP$
- $(\neg s) \in FORM$, parce que
  - $s \in FORM$, parce que $s \in PROP$

Définition formelle de la Lprop (3)

Comprendre une définition inductive:

"*FORM* est le plus petit ensemble ...":

- $(p \bowtie q) \notin FORM$, parce qu'aucune règle ne génère $(p \bowtie q)$
- $((p \wedge (q \vee r))) \longrightarrow (\neg s)) \notin FORM$, parce que:
  - $(p \wedge (q \vee r))) \notin FORM$, parce que:
    - $(q \vee r)) \notin FORM$, parce que $r) \notin FORM$, parce que $r) \notin PROP$
- $(A \vee p)) \in FORM$ ou $\notin FORM$?
  - si A représente une formule oui, sinon non

Conventions syntaxiques (1)

On peut omettre des paranthèses selon les conventions suivantes:

- Associativité: Les opérateurs binaires associent à droite:
  - $A \wedge B \wedge C$ correspond à $(A \wedge (B \wedge C))$
  - $A \longrightarrow B \longrightarrow C$ correspond à $(A \longrightarrow (B \longrightarrow C))$

  Attention: $(A \longrightarrow (B \longrightarrow C))$ et $((A \longrightarrow B) \longrightarrow C)$ ont une sémantique différente!
- Priorité: Les opérateurs sont listés par priorité décroissante sur la page précédente. *Exemples:*
  - $(A \wedge B \vee C) \equiv ((A \wedge B) \vee C)$
  - $(\neg A \vee B) \equiv ((\neg A) \vee B)$
  - $(A \wedge B \longrightarrow A \vee B) \equiv ((A \wedge B) \longrightarrow (A \vee B))$

Conventions syntaxiques (2)

On peut introduire d'autres connecteurs comme abbréviations:

- *Constante "vrai":* $\top$ défini par $\neg\bot$
- *Équivalence:* $(A \leftrightarrow B)$ défini par $((A \longrightarrow B) \wedge (B \longrightarrow A))$
- *Ou exclusif:* $(A \oplus B)$ défini par $((A \vee B) \wedge (\neg(A \wedge B)))$

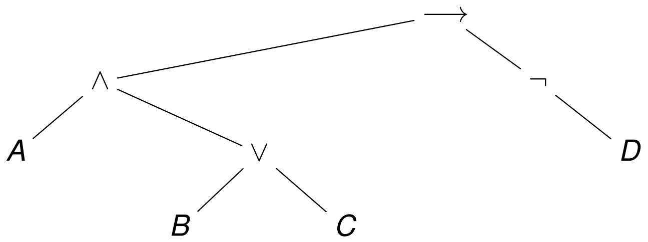
. . . ou les introduire comme nouveaux constructeurs de *FORM*
$\rightsquigarrow$ moins pratique pour des preuves inductives!

- Les lettres majuscules A, B, C,. . . représentent des formules !

Arbres syntaxiques (1)

Plusieurs représentations de la même formule:

- Arbre syntaxique
- Représentation textuelle: parenthésage maximal
- Représentation textuelle: parenthésage minimal



$$((A \wedge (B \vee C)) \longrightarrow (\neg D))$$

$$A \wedge (B \vee C) \longrightarrow \neg D$$

Arbres syntaxiques (2)

Étant donné l'arbre, comment obtenir sa représentation textuelle?

- *Parenthésage maximal:*
  Traverser récursivement l'arbre avec un *parcours infixe*:
  - *Cas de base:* (variable v ou constante c): écrire le symbole v ou c
  - *Cas unaire:* (négation $\neg A$):
    - écrire '(' et $\neg$; représenter A; écrire ')'
  - *Cas binaire:* (par exemple conjonction $A \wedge B$):
    - écrire '('; représenter A; écrire $\wedge$; représenter B; écrire ')'
- *Parenthésage minimal:*
  Similaire, sauf pour les
  - *Cas unaire et binaire:* écrire les parenthèses uniquement si le père du noeud actuel a une plus haute priorité.
    Exemple: Noeud: $\vee$, père: $\wedge$

Étant donnée la représentation textuelle, comment obtenir l'arbre?
$\rightsquigarrow$ plus difficile, voir cours "Analyse syntaxique", 4ème année

Induction (1)

La Triade:

- *Types inductifs:*
  - générés à partir de *cas de base*,
  - en applicant des *règles de construction*
- *Fonctions récursives* (récursion structurelle):
  - décomposent une structure inductive composée
  - s'arrêtent dans les cas de base

  ... et synthétisent un résultat en remontant
- *Preuves par induction structurelle:* Permettent de conclure qu'une propriété est satisfaite pour *tout* élément d'un type inductif
  - si elle est satisfaite pour les cas de base
  - si elle est héréditaire pour les éléments composés

Induction (2)

Histoire:

Maurolico (1494-1575)

Peano (1858 - 1932)

Première preuve par induction

Première axiomatisation des nombres naturels

Induction: Nombres Naturels (1)

Les nombres naturels comme type inductif

Nat est le plus petit ensemble qui satisfait les conditions:

1. *Zéro:* $0 \in Nat$
2. *Successeur:* Si $n \in Nat$, alors $suc(n) \in Nat$

Note: $suc(n)$ est une autre manière d'écrire $n + 1$

$$\bullet \xrightarrow{+1} \bullet \xrightarrow{+1} \bullet \xrightarrow{+1} \bullet$$

repr. décimale	0	1	2	3
repr. unaire	0	$suc(0)$	$suc(suc(0))$	$suc^3(0)$

Induction: Nombres Naturels (2)

Récursion sur les nombres naturels

Exemple: La fonction d'addition $n + m$ peut être définie par:

- *cas Zéro [eq. Z]:* $0 + m = m$
- *cas Successeur [eq. S]:* $suc(n) + m = suc(n + m)$

Schéma d'induction sur les nombres naturels:

- *base (Zéro):* Si $P(0)$,
- *hérédité (Successeur):* et si pour tout n, $P(n)$ implique $P(suc(n))$
- alors on peut conclure $\forall n.P(n)$

Induction: Nombres Naturels (3)

Preuve par induction

Exemple: Montrer que 0 est l'élément neutre à droite: $\forall n.(n + 0 = n)$

1. *Identifier le prédicat P:* Ici: $P(n) \equiv (n + 0 = n)$
2. *Preuve pour le cas de base:* Montrer: $P(0)$, donc: $0 + 0 = 0$ (par *[eq. Z]*)
3. *Preuve d'hérédité:* Montrer: si $P(n)$, alors $P(suc(n))$
    - Hypothèse d'induction *[eq. H]*: $n + 0 = n$
    - Montrer: $suc(n) + 0 = suc(n)$. Calculer:
        - $suc(n) + 0 = suc(n + 0)$ (par *[eq. S]*)
        - $= suc(n)$ (par *[eq. H]*)

Exercice: Montrer: l'addition est commutative: $\forall n.\forall m.(n + m = m + n)$

Retenir:

- $+$ est définie par *récursion* sur le 1er argument
- donc: *induction* sur le 1er argument de $+$

Induction: Listes (1)

Les listes comme type inductif

Soit A un type.

$A\ List$, le type des listes avec éléments de type A, est le plus petit ensemble qui satisfait les conditions:

1. *Vide:* $[] \in A\ List$
2. *Cons:* Si $a \in A$ et $\ell \in A\ List$ alors $a :: \ell \in A\ List$

Exemples:

- $3 :: 2 :: 1 :: [] \in Nat\ List$
  écriture lisible: $[3; 2; 1]$
- $true :: false :: true :: [] \in Bool\ List$
  écriture lisible: $[true; false; true]$

Induction: Listes (2)

Récursion sur les listes:

Exemple: La fonction de concaténation `append` peut être définie par:

- *cas Vide [eq. V]:* `append([],ys) = ys`
- *cas Cons [eq. C]:* `append(x::xs,ys) = x::append(xs,ys)`

Note: la fonction `append` est prédéfinie (infixe: `@`) en Caml.

La fonction `append` en Caml:

```
let rec append = function
    ([], ys) -> ys
  | (x :: xs, ys) -> x::append(xs, ys)
;;
# val append : 'a list * 'a list -> 'a list = <fun>

# append ([1;2], [3;4]) ;;
- : int list = [1; 2; 3; 4]
```

Induction: Listes (3)

Schéma d'induction sur les listes:

- *base (Vide):* Si $P([])$,
- *hérédité (Cons):* et si pour tout élément a et toute liste ℓ, $P(\ell)$ implique $P(a :: \ell)$
- alors on peut conclure $\forall \ell . P(\ell)$

Induction: Listes (4)

Preuve par induction

Exemple: Montrer que [] est l'élément neutre à droite:
$\forall \ell . append(\ell, []) = \ell$

1. *Identifier le prédicat P:* Ici: $P(\ell) \equiv (append(\ell, []) = \ell)$
2. *Preuve pour le cas de base:* Montrer: $P([])$, donc: $append([], []) = []$ (par *[eq. V]*)
3. *Preuve d'hérédité:* Montrer: si $P(\ell)$, alors $P(a :: \ell)$
   - Hypothèse d'induction *[eq. H]*: $append(\ell, []) = \ell$
   - Montrer: $append(a :: \ell, []) = a :: \ell$. Calculer:
     - $append(a :: \ell, []) = a :: append(\ell, [])$ (par *[eq. C]*)
     - $= a :: \ell$ (par *[eq. H]*)

Induction: Listes (5)

Exercices:

- La concaténation est-elle commutative?
- Montrer: la concaténation est associative: $\forall xs. \forall ys. \forall zs. append(append(xs, ys), zs) = append(xs, append(ys, zs))$
- Définir la fonction `longueur` sur des listes
- Montrer: La longeur de deux listes concaténées est la somme des longueurs des listes.

Induction: Formules (1)

Les formules comme type inductif: voir définition page 7

Récursion sur les formules:

Exemple: La fonction npb (nombre de parenthèses) est définie par:

- *Variable [eq. V]:* $\texttt{nbp}(\texttt{p}) = 0$
- *Constante [eq. C]:* $\texttt{nbp}(\bot) = 0$
- *Négation [eq. N]:* $\texttt{nbp}((\neg A)) = \texttt{nbp}(A) + 2$
- *Conjonction [eq. C]:* $\texttt{nbp}((A \wedge B)) = \texttt{nbp}(A) + \texttt{nbp}(B) + 2$
- *Disjonction [eq. D]:* $\texttt{nbp}((A \vee B)) = \texttt{nbp}(A) + \texttt{nbp}(B) + 2$
- *Implication [eq. I]:* $\texttt{nbp}((A \longrightarrow B)) = \texttt{nbp}(A) + \texttt{nbp}(B) + 2$

Quelques formules:

- $\texttt{nbp}(((p \vee q) \wedge r)) = 4$
- $\texttt{nbp}(((p \wedge q) \vee (\neg(r \vee \bot)))) = 8$

Induction: Formules (2)

Schéma d'induction sur les formules:

- *base (Variable):* Si $P(p)$
- *base (Constante):* et si $P(\perp)$
- *hérédité (Négation):* et si pour toute formule A, $P(A)$ implique $P((\neg A))$
- *hérédité (Conjonction):* et si pour tout A, B, $P(A)$ et $P(B)$ implique $P((A \wedge B))$
- *hérédité (Disjonction):* et si pour tout A, B, $P(A)$ et $P(B)$ implique $P((A \vee B))$
- *hérédité (Implication):* et si pour tout A, B, $P(A)$ et $P(B)$ implique $P((A \longrightarrow B))$
- alors on peut conclure: $\forall A \in FORM.P(A)$

Induction: Formules (3)

Preuve par induction *Exemple:* Montrer que le nombre de parenthèses est toujours pair: $\forall f.nbp(f) \mod 2 = 0$

- *Identifier le prédicat P:* Ici: $P(A) \equiv (nbp(A) \mod 2 = 0)$
- *Cas de base (Variable):* Montrer: $P(p)$, donc: $nbp(p) \mod 2 = 0 \mod 2 = 0$ (par *[eq. V]*)
- *Cas de base (Constante):* Montrer: $P(\bot)$, donc: $nbp(\bot) \mod 2 = 0 \mod 2 = 0$ (par *[eq. C]*)
- *Hérédité (Négation):* Montrer: si $P(A)$, alors $P((\neg A))$
  - Hypothèse d'induction *[eq. H]*: $nbp(A) \mod 2 = 0$
  - Montrer: $nbp((\neg A)) \mod 2 = 0$. Calculer:
    - $nbp((\neg A)) \mod 2 = (nbp(A) + 2) \mod 2$ (par *[eq. N]*)
    - $= nbp(A) \mod 2$ (arithmétique)
    - $= 0$ (par *[eq. H]*)

Induction: Formules (4)

- *Hérédité (Conjonction):* Montrer: si $P(A)$ et $P(B)$, alors $P((A \wedge B))$
  - Hypothèses d'induction
    - *[eq. H_1]*: $nbp(A) \mod 2 = 0$
    - *[eq. H_2]*: $nbp(B) \mod 2 = 0$
  - Montrer: $nbp((A \wedge B)) \mod 2 = 0$. Calculer:
    - $nbp((A \wedge B)) \mod 2 = (nbp(A) + nbp(B) + 2) \mod 2$ (par *[eq. C]*)
    - $= (nbp(A) \mod 2 + nbp(B) \mod 2) \mod 2$ (arithmétique)
    - $= (0 + nbp(B) \mod 2) \mod 2$ (par *[eq. H_1]*)
    - $= (0 + 0) \mod 2$ (par *[eq. H_2]*)
    - $= 0$ (arithmétique)
- *Hérédité (Disjonction et Implication):* Similaire

Substitutions (1)

Une substitution remplace *toutes* les occurrences d'une variable propositionnelle p dans une formule A par une formule B.

Notation: $A[B/p]$

Exemple:

- $(q \wedge (p \longrightarrow (\neg p)))[(r \vee s)/p]$
- $= (q[(r \vee s)/p] \wedge (p \longrightarrow (\neg p))[(r \vee s)/p])$
- $= (q \wedge (p[(r \vee s)/p] \longrightarrow (\neg p)[(r \vee s)/p]))$
- $= (q \wedge ((r \vee s) \longrightarrow (\neg p[(r \vee s)/p])))$
- $= (q \wedge ((r \vee s) \longrightarrow (\neg(r \vee s))))$

Substitutions (2)

Contre-exemples (dessinez les arbres syntaxiques!):

- Pourquoi $(\neg p)[(r \vee s)/p] \neq (\neg r \vee s)$??
- Pourquoi $(p \longrightarrow (\neg p))[(r \vee s)/p] \neq ((r \vee s) \longrightarrow (\neg p))$??

Exercices:

- Donner une définition (par induction sur A) de: $A[B/p]$
- Définir une fonction $nbocc(p, A)$ qui calcule le nombre d'occurrences d'une variablepropositionnelle p dans une formule A
- Si $nbocc(p, A) = 0$, alors $A[B/p] =$??? (*preuve!*)
- Définir une fonction $taille(A)$ qui calcule la taille d'une formule ($\equiv$ nombre de noeuds de l'arbre syntaxique)
- Quel est le rapport entre $taille(A)$, $taille(B)$, $nbocc(p, A)$ et $taille(A[B/p])$?

Résumé

Syntaxe de la logique des propositions

- Rapport avec le langage naturel
- Définition inductive de l'ensemble *FORM*
- Différentes représentations: textuelle, arborescente
- Définitions récursives (sur la structure des formules):
  - conversion arbre syntaxique $\rightsquigarrow$ repr. textuelle
  - nombre de parenthèses
  - substitution
  - ...

Induction

- Principe pour raisonner sur des structures inductives
- ...aussi sur des formules

Plan

Problématique (1)

Vu jusqu'à maintenant: la structure du langage logique: Syntaxe (grec: *syn-taxis*: composition)
Dans cette section: la signification du langage logique: Sémantique (grec: *sema*: signe)

Dichotomie syntaxe / sémantique dans tout langage :

- en logique et dans les langages de programmation
- en linguistique (signifiant/signifié $\sim$ le mot/la chose)

Postulats de la *logique classique*:

- *Bivalence:* deux valeurs de vérité: “faux” et “vrai”, resp. 0 et 1
- *Vérifonctionnalité:* La valeur de vérité d'une formule ne dépend *que* de celles des propositions qui la composent.

Problématique (2)

Questionnements:

- *Toujours vrai / faux* . . . ou aussi “je ne sais pas”?
  Quelle est la valeur de vérité de la conjecture de Goldbach :
  “Tout nombre pair $\geq$ 4 est la somme de deux nombres premiers”
  (sans preuve jusqu’à maintenant)
  $\rightsquigarrow$ logiques à trois valeurs de vérité
- *Statut de la disjonction:*
  Peut-on affirmer $a \vee b$ sans dire lequel de a ou b est vrai?
  En particulier: $a \vee \neg a$ toujours vrai?
  $\rightsquigarrow$ logique intuitionniste
- *Statut de l’implication:*
  Est-ce que $a \longrightarrow b$ est vrai si a est faux?
  “Si la lune est composée de frommage, alors 1 + 1 = 3”
  $\rightsquigarrow$ plusieurs notions d’implication:
  - matérielle (implication “classique”)
  - stricte (“nécessairement si . . . alors . . . ”)

Valuation et Interprétation

Valuation:

- Une valuation décrit quelle valeur de vérité est associée à une variable propositionnelle
- *Formellement:* une valuation est une fonction $v : PROP \Rightarrow \{0, 1\}$
- Donc: $v(p) = 0$: “p est faux”; $v(p) = 1$: “p est vrai”;

Une interprétation $\mathcal{I}_v$ est l'extension de v de *PROP* à *FORM*:
Définition récursive:

- *(Variable):* $\mathcal{I}_v\ (p) = v(p)$
- *(Constante):* $\mathcal{I}_v\ (\perp) = 0$
- *(Négation):* $\mathcal{I}_v\ (\neg A) = 1 - \mathcal{I}_v\ (A)$
- *(Conjonction):* $\mathcal{I}_v\ (A \wedge B) = min(\mathcal{I}_v\ (A), \mathcal{I}_v\ (B))$
- *(Disjonction):* $\mathcal{I}_v\ (A \vee B) = max(\mathcal{I}_v\ (A), \mathcal{I}_v\ (B))$
- *(Implication):* $\mathcal{I}_v\ (A \longrightarrow B) = max(1 - \mathcal{I}_v\ (A), \mathcal{I}_v\ (B))$

Notation: Souvent, on écrit $v(A)$ (pour formule A) au lieu de $\mathcal{I}_v\ (A)$

Interprétations et tables de vérité

Table de vérité:

p	q	r	$p \wedge q$	$\neg r$	$(p \wedge q) \vee (\neg r)$
0	0	0	0	1	1
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	0	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	1	0	1

Chaque *ligne* de la table de vérité correspond à une *valuation*.
Exemple: valuation v avec: $v(p) = 1, v(q) = 0, v(r) = 0$
$v((p \wedge q) \vee (\neg r)) = max(v(p \wedge q), v(\neg r)) =$
$max(min(v(p), v(q)), 1 - v(r)) = max(min(1, 0), 1 - 0) = max(0, 1) = 1$

Modèles

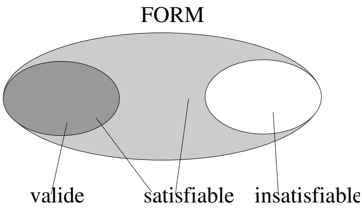
Modèle d'une formule: Une interprétation v est

- un *modèle* d'une formule A si $v(A) = 1$
  On dit: v satisfait A
  Ex.: $v(p) = 0, v(q) = 1$ est un modèle de $p \vee q$
- un *contre-modèle* d'une formule A si $v(A) = 0$
  On dit: v falsifie A
  Ex.: $v(p) = 0, v(q) = 1$ est un contre-modèle de $p \wedge q$

Modèle d'un ensemble de formules: Une interprétation v est

- un *modèle* d'un ensemble H si $v(A) = 1$ pour tout $A \in H$
  Ex.: $v(p) = 0, v(q) = 1$ est un modèle de $\{p \vee q, q\}$
- un *contre-modèle* d'un ensemble H si $v(A) = 0$ pour au moins un $A \in H$
  Ex.: $v(p) = 0, v(q) = 1$ est un contre-modèle de $\{p \vee q, p\}$

Validité (1)



- Une formule A est *valide* (une *tautologie*) si, pour toute valuation v, on a $v(A) = 1$
- Une formule A est *satisfiable* s'il existe une valuation v telle que $v(A) = 1$
- Une formule A est *insatisfiable* s'il n'existe pas de valuation v telle que $v(A) = 1$

Ces définitions s'étendent à un ensemble de formules.

Validité (2)

Déterminez si ces formules sont valides / satisfiables / insatisfiables:

1. $p \wedge \neg p$
2. $(p \vee q) \longrightarrow q$
3. $(p \wedge q) \longrightarrow p$

Complétez:

- Une formule A est *valide* si sa table de vérité …
- Une formule A est *satisfiable* si sa table de vérité …
- Une formule A est *insatisfiable* si sa table de vérité …

Rapports entre les notions:

- Si A est valide, qu'est-ce que vous savez de $\neg A$?
- Si A est satisfiable, est-ce que $\neg A$ est insatisfiable?

Conséquence (1)

Motivation: En mathématiques / informatique, on tire des *conclusions* à partir d'un ensemble *d'hypothèses*.
Exemple:

- H_1: La liste `xs` est triée
- H_2: La liste `ys` est triée
- H_3: Les éléments de `xs` sont plus petits que les éléments de `ys`
- C: La liste `xs @ ys` est triée

Notation: $\{H_1, H_2, H_3\} \models C$

Signification: Tout modèle de $\{H_1, H_2, H_3\}$ est aussi un modèle de C

Conséquence (2)

Cas particulier: Ensemble d'hypothèses vide:
On écrit $\models C$ au lieu de $\{\} \models C$

Différence subtile entre:

- C: une formule (qui peut être vraie ou fausse)
- $\models C$: une formule dont on affirme la validité

Théorème de la déduction:
$A \models B$ si et seulement si $\models A \longrightarrow B$

Exercices: Vrai ou faux?

- $\{A, B\} \models A \vee B$
- $\{A, B\} \models A \wedge B$
- $\{A, B\} \models A \longrightarrow B$
- $\{A \longrightarrow B, B\} \models A$
- $\{A \longrightarrow B, A\} \models B$
- $\{A \longrightarrow B, \neg B\} \models \neg A$

Équivalence (1)

Deux formules A et B sont *équivalentes* si elles ont les mêmes modèles. *Notation:* $A \equiv B$

Exercices:

- Montrer que $A \wedge B \equiv B \wedge A$
- Montrer que $A \longrightarrow B \not\equiv B \longrightarrow A$

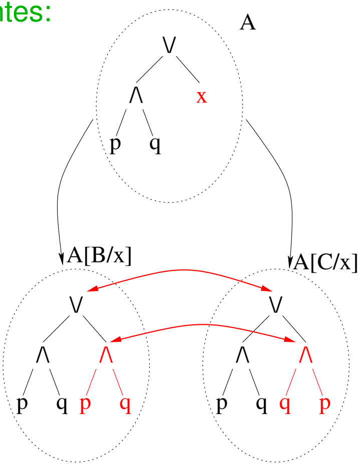
Méta-langage: $\models$ et $\equiv$ ne sont pas des constructeurs de formules!
Interdit: “formules” comme $(A \models B) \equiv (A \longrightarrow B)$

Équivalence (2)

Rappel: $A[B/p]$
Substitution d'une formule B pour une variable p dans une formule A
Remplacement de sous-formules équivalentes:

Exemple: Soient

- $B := (p \wedge q), C := (q \wedge p)$
- $A := (p \wedge q) \vee x$
- donc: $A[B/x] = (p \wedge q) \vee (p \wedge q)$
- donc: $A[C/x] = (p \wedge q) \vee (q \wedge p)$



Proposition: Si $B \equiv C$,
alors $A[B/x] \equiv A[C/x]$
Preuve: Par induction sur la structure de A
Complétéz!

Résumé

- Différence syntaxe / sémantique
- On définit la *sémantique* d'une formule à l'aide d'une fonction d'*interprétation*
- Correspondance entre interprétations et tables de vérité
- Modèles d'une formule
- Validité, satisfiabilité
- Équivalence logique, base pour le calcul booléen

Plan

Motivation (1)

Exemple (Tarski's world, restreint)
Sur une grille $n \times n$, placer des cercles et des carrés de façon que:

1. toute colonne contient un cercle
2. toute ligne contient un rectangle
3. il n'y a pas d'objets (= cercles, rectangles) sur les diagonales

Modélisation pour $n = 3$

Variables propositionnelles:

- $C_{c,\ell}$: il y a un cercle sur la colonne c et la ligne ℓ
- $R_{c,\ell}$: il y a un rectangle sur la colonne c et la ligne ℓ

Formules:

1. $(C_{1,1} \vee C_{1,2} \vee C_{1,3}) \wedge (C_{2,1} \vee C_{2,2} \vee C_{2,3}) \wedge (C_{3,1} \vee C_{3,2} \vee C_{3,3})$
2. Complétez!

Motivation (2)

Constat:

- Lourd à écrire et à lire
- Difficile à étendre et à modifier
- Impraticable pour un n élevé

Écriture avec quantificateurs sur ensembles finis
($\equiv$ "quantificateurs finis"):

1. $\bigwedge_{c\in\{1,2,3\}} \bigvee_{\ell\in\{1,2,3\}} C_{c,\ell}$
   "Pour tout $c \in \{1, 2, 3\}$, il y a un $\ell \in \{1, 2, 3\}$ tel que $C_{c,\ell}$"
2. $\bigwedge_{\ell\in\{1,2,3\}} \bigvee_{c\in\{1,2,3\}} R_{c,\ell}$
3. $(\bigwedge_{n\in\{1,2,3\}} \neg C_{n,n}) \wedge (\bigwedge_{n\in\{1,2,3\}} \neg R_{n,n}) \wedge (\bigwedge_{n\in\{1,2,3\}} \neg C_{n,4-n}) \wedge (\bigwedge_{n\in\{1,2,3\}} \neg R_{n,4-n})$

Quantificateurs finis comme abbréviation

Les quantificateurs sur ensembles finis n'augmentent pas l'expressivité de la logique, mais sont uniquement des abbréviations ("macros").

Réduction à la LP pure par récursion sur la structure des ensembles:

Conjonction:

- $\bigwedge_{i \in \{\}} P_i \equiv \top$
- $\bigwedge_{i \in \{e, \dots\}} P_i \equiv P_e \wedge \bigwedge_{i \in \{\dots\}} P_i$

Disjonction:

- $\bigvee_{i \in \{\}} P_i \equiv \perp$
- $\bigvee_{i \in \{e, \dots\}} P_i \equiv P_e \vee \bigvee_{i \in \{\dots\}} P_i$

Exemple:

$$\begin{array}{rcl}
\bigvee_{\ell \in \{1,2,3\}} C_{c,\ell} & \equiv & C_{c,1} \vee \bigvee_{\ell \in \{2,3\}} C_{c,\ell} \\
 & \equiv & C_{c,1} \vee C_{c,2} \vee \bigvee_{\ell \in \{3\}} C_{c,\ell} \\
 & \equiv & C_{c,1} \vee C_{c,2} \vee C_{c,3} \vee \bigvee_{\ell \in \{\}} C_{c,\ell} \\
 & \equiv & C_{c,1} \vee C_{c,2} \vee C_{c,3} \vee \perp
\end{array}$$

Quantificateurs finis avec contraintes

Exemple: Exprimer: “Toute case sauf la diagonale contient un cercle”

Solution:

$$\bigwedge_{c\in\{1,2,3\}} \bigwedge_{\ell\in\{1,2,3\}|c\neq\ell} C_{c,\ell}$$

Format général:

- $\bigwedge_{i\in E|C} P_i$, où
  - E est un ensemble fini (comme avant)
  - C est une *contrainte*
    (expression Booléenne qui peut contenir i et les variables liées précédemment)
- $\bigvee_{i\in E|C} P_i$: similaire

Exprimez:

- Toute case au-dessus de la diagonale contient un rectangle.
- Les cases dont la somme des indices est paire contient un cercle.

Plan

Un outil pédagogique

SATOULOUSE a été conçu par l'équipe pédagogique du module "Logique" pour

- avoir un format simple d'écriture de formules
- permettre la vérification de satisfiabilité d'une formule en un clic
- donner un modèle lisible du'une formule satisfiable

Autres outils (voir liens sur la page Moodle):

- Chaff: expressif, mais difficile à apprendre
- Minisat, Sat4j: format "assembleur" de bas niveau, difficile à comprendre

Le pourquoi et comment de SATOULOUSE

But: montrer la satisfiabilité d'une formule F

- Si F est satisfiable, SATOULOUSE fournit un modèle (variables interprétées comme “vraies”, les autres étant “fausses”)
- Sinon, SATOULOUSE affiche “insatisfiable”

Démarche:

1. écrire une ou plusieurs formules dans le panneau central (SATOULOUSE prend la *conjonction* des lignes)
2. Appuyer sur le “cerveau” pour vérifier la satisfiabilité
3. Eventuellement: Raffiner en rajoutant d'autres formules, et recommencer
   (pour obtenir un autre modèle)

Syntaxe de SATOULOUSE

- Une syntaxe textuelle, inspirée de Lisp
- Une syntaxe LaTeX, à composer avec l'éditeur syntaxique à gauche

Correspondances:

Textuel	LaTeX
p	p
(p i)	p_i
(p (i + 1))	p_{i+1}
(p i j)	$p_{i,j}$
(A and B)	$(A \wedge B)$
(A or B)	$(A \vee B)$
(bigand i (1 2 3) (p i))	$\bigwedge_{i \in \{1,2,3\}} p_i$
(bigor i (1 2 3) (p i))	$\bigvee_{i \in \{1,2,3\}} p_i$
(bigand i (1 2 3) (i diff j) (p i j))	$\bigwedge_{i \in \{1,2,3\} \mid i \neq j} p_{i,j}$

Limites (actuelles) de SATOULOUSE

- *Opérations sur indices:* limitées à des sommes d'indices et constantes:
  p_{i+j}, q_{i+3}, mais pas: q_{2*i-5}
- *Contraintes:* limitées à des inégalités:
  $i \neq j$, $i \neq j+2$, mais pas: $i \geq j$

Message publicitaire: Intéressé(e) par un stage pour étendre SATOULOUSE ?
Prenez contact avec nous pour en discuter !

Plan

Variables (1)

Chaque occurrence d'un quantificateur *lie* une variable.
Toute variable qui n'est pas liée est *libre*.

Exemple:

$$(\bigwedge_{i \in E} r_{i,j} \wedge (\bigvee_{k \in E} s_{i,k}))$$

Le concept est universel en

- mathématiques:

$$\Sigma_{i=a}^{b}(i + \Pi_{k=1}^{i} k)$$

- informatique:

```
# let inc = 2;;
# let rec sum = function (x) ->
      if x = 0 then 0 else inc + sum(x - 1) ;;
# sum 5 ;;
- : int = 10
```

Variables (2)

On peut renommer des variables liées sans modifier le sens d'une formule ou d'un programme.

- $$(\bigwedge_{i \in E} r_{i,j} \wedge (\bigvee_{k \in E} s_{i,k})) = (\bigwedge_{m \in E} r_{m,j} \wedge (\bigvee_{n \in E} s_{m,n}))$$
- $$\Sigma_{i=a}^{b}(i + \Pi_{k=1}^{i} k) = \Sigma_{i=a}^{b}(i + \Pi_{n=1}^{i} n)$$
- ```
  # let rec sum = function (n) ->
          if n = 0 then 0 else inc + sum(n - 1) ;;
  # sum 5 ;;
  - : int = 10
  ```

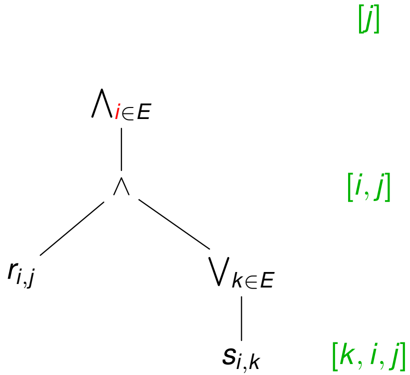
Des précautions s'imposent:

```
# let rec sum = function (inc) ->
        if inc = 0 then 0 else inc + sum(inc - 1) ;;
# sum 5 ;;
- : int = 15
```
```

Variables (3)

Petite complication: La notion de variable liée / libre est relative à un contexte / une sous-formule.

- Dans $(\bigwedge_{i \in E} r_{i,j} \wedge (\bigvee_{k \in E} s_{i,k}))$,
  - j est libre
  - i, k sont liées
- Dans $(\bigvee_{k \in E} s_{i,k})$,
  - i est libre
  - k est liée
- Dans $s_{i,k}$,
  - i, k sont libres
  - il n'y a pas de variables liées



Exercice: Définissez la fonction *vars* des variables libres d'une formule

Variables (4)

- Un nom de variable peut désigner en même temps une variable libre et une variable liée. *Exemple:* $(\bigwedge_{x \in E} r_{x,y} \wedge (\bigvee_{y \in E} s_{x,y}))$
  - y est libre dans le sous-terme $r_{x,y}$
  - y est liée dans le sous-terme $s_{x,y}$
- Bon style: *Renommer* les variables liées pour éviter des conflits. *Bon exemple:* $(\bigwedge_{x \in E} r_{x,y} \wedge (\bigvee_{z \in E} s_{x,z}))$
- En renommant, éviter des noms déjà utilisés dans le contexte actuel: *Mauvais exemple:* $(\bigwedge_{x \in E} r_{x,y} \wedge (\bigvee_{x \in E} s_{x,x}))$
  $\rightsquigarrow$ *capture* de la variable x

Variables et quantification avec contraintes

Dans $\bigwedge_{i \in E | C} p_i$,

- la variable i ne doit pas apparaître dans E,
- mais peut / devrait apparaître dans C.

(similaire pour $\bigvee_{i \in E | C} p_i$)
Exemple (utilise quelques simplifications arithmétiques):

$$\begin{array}{ll}
& \bigwedge_{i \in \{1,2,3,4\} | (i \mod 2 = 0)} p_i \\
= & ((1 \mod 2 = 0) \longrightarrow p_1) \wedge ((2 \mod 2 = 0) \longrightarrow p_2) \\
& \wedge ((3 \mod 2 = 0) \longrightarrow p_3) \wedge ((4 \mod 2 = 0) \longrightarrow p_4) \\
= & (\perp \longrightarrow p_1) \wedge (\top \longrightarrow p_2) \wedge (\perp \longrightarrow p_3) \wedge (\top \longrightarrow p_4) \\
= & p_2 \wedge p_4
\end{array}$$

Généralisez et donnez une définition récursive de la quantification avec contraintes.
Conclusion: Définition donne une formule valide de LProp uniquement pour des formules quantifiées *closes* (sans variables libres).

Plan

Motivation (1)

On s'intéresse désormais à des méthodes pour déterminer la *validité* d'une formule A.

Méthodes possibles:

Construction de la table de verité

- Question: Quelle est la taille de la table pour une formule avec n variables propositionnelles?
- $\rightsquigarrow$ Méthode non praticable si n est "grand"

Calcul booléen:

- Problèmes:
  - Quel sous-term réécrire?
  - Quand abandonner le calcul si une formule ne se réduit pas à $\top$?
- $\rightsquigarrow$ Méthode en général trop mal ciblée

Motivation (2)

Méthodes astucieuses: recherche plus ciblée:

- Méthode directe: Vérifie que la formule est *valide*
  Voir: Déduction naturelle
- Méthode indirecte: Vérifie que la négation de la formule est *insatisfiable*
  Rappel: A est valide ssi $\neg A$ est insatisfiable
  Voir: Méthode des tableaux, résolution

Vraiment "astucieuses"?

- *Oui:* On peut *souvent* éviter un coût exponentiel
- *Non:* On ne sait pas *toujours* l'éviter (à l'heure actuelle)

Digression: Classes de complexité (1)

(Mise en garde: Simplification très grossière!)

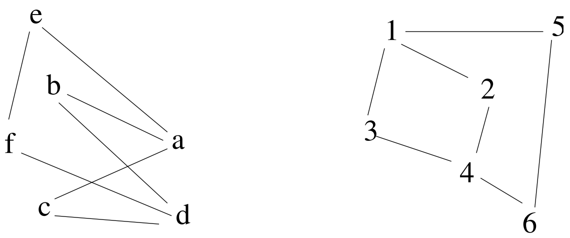
Classe *P*:

- Classe des problèmes dont le temps de calcul est borné par un *polynome* (en fonction de la taille de l'entrée)
- *Exemple:* Trier une liste de n éléments nécessite n^2 opérations (et on peut faire mieux)
- *Exemple:* Trouver le plus court chemin entre deux villes (pour un pays avec n villes) nécessite n^3 opérations
- *Exemple:* Étant donnée une interprétation v et une formule A, la *vérification* si $v(A) = 1$ nécessite $|A|$ opérations

Digression: Classes de complexité (2)

Classe *NP* (non-déterministe polynomial):

- Classe des problèmes résolubles en
  - devinant une solution (instantanément)
  - vérifiant cette solution (en temps polynomial)
- *Exemple:* Isomorphisme de graphes
  Est-ce que les deux graphes sont les "mêmes"
  (à renommage et déplacement de noeuds près)?



Chiffrez la complexité d'un algorithme "bête"

Digression: Classes de complexité (3)

Question $P = NP$

- Problème: Comment deviner une solution?
- Dans le pire cas: Faire une recherche exhaustive
  Complexité: exponentielle
- *Question ouverte:* Est-ce qu'on pourrait résoudre les problèmes de *NP* en temps polynomial?

Satisfiabilité *SAT* et *NP*:

- *SAT* est parmi les plus difficiles de *NP* (*NP*-complet)
- Si on pouvait résoudre *SAT* en temps polynomial, alors aussi tous les autres problèmes de *NP*

Pour résumer:

- Vérifier si une interprétation est un modèle: classe *P*
- Trouver une interprétation qui est un modèle: *NP*-complet
  surtout: le *pire temps connu* est exponentiel

Méthode des tableaux: Idée (1)

Exemple: Déterminer si $(\neg q) \wedge (p \longrightarrow q)$ est satisfiable.

p	q	$\neg q$	$p \longrightarrow q$	$(\neg q) \wedge (p \longrightarrow q)$
0	0	1	1	1
0	1	0	1	0
1	0	1	0	0
1	1	0	1	0

- $v((\neg q) \wedge (p \longrightarrow q)) = 1$, donc
  - $v(\neg q) = 1$, donc $v(q) = 0$
  - et $v(p \longrightarrow q) = 1$, donc
    1. $v(p) = 0$
    2. ou $v(q) = 1$ (impossible parce que $v(q) = 0$)

Résultat: $v((\neg q) \wedge (p \longrightarrow q)) = 1$ pour la valuation $v(q) = 0$, $v(p) = 0$

Méthode des tableaux: Idée (2)

Le même raisonnement, avec arbre de recherche:

$$(\neg q) \wedge (p \longrightarrow q)$$

$$\neg q$$

$$p \longrightarrow q$$

$$\neg p \qquad\qquad\qquad\qquad q$$

$$v(p) = 0, v(q) = 0 \qquad\qquad\qquad\qquad \perp$$

Structure de données (1)

Un arbre de recherche est:

1. une feuille
2. un noeud *unaire* dont le fils est un arbre de recherche
3. un noeud *binaire* dont les deux fils sont des arbres de recherche

Une branche est un chemin menant de la racine à une feuille.
Une branche peut être

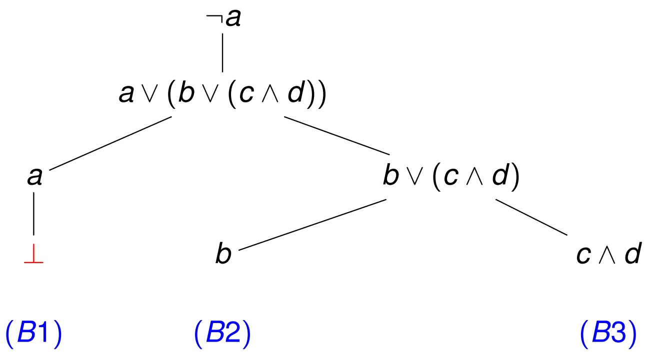
1. *fermée* si sa feuille est marquée $\perp$
2. *ouverte* si elle n'est pas fermée.

Un noeud sur une branche ouverte peut être

1. *passif* si une règle a déjà été appliquée à ce noeud sur cette branche
2. *actif* autrement

Structure de données (2)

Structure de base:

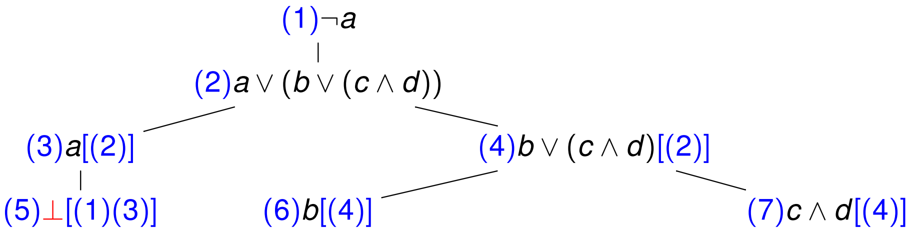


- Branche (B1) est fermée
- Branche (B2) est ouverte, avec noeuds actifs $\neg a$ et b
- Branche (B3) est ouverte, avec noeuds actifs $\neg a$ et $c \wedge d$

Structure de données (3)

Information administrative:

- Chaque noeud a un numéro, unique dans l'arbre
- Les noeuds dérivés sont annotés de leur origine: [noeud(s)]



Règles (1)

Condition d'applicabilité:

- *Fermeture* peut être appliquée à toute branche ouverte
- les autres règles peuvent être appliquées à tout noeud actif d'une branche ouverte

Effet de l'application d'une règle:

- *Fermeture:* Si la branche contient A et $\neg A$, créer un fils avec $\perp$
- Règles conjonctives: créent un seul fils:
  - *Règle* $\wedge$*:* Pour $A \wedge B$, créer un fils avec A et B
  - *Règle* $\neg\vee$*:* Pour $\neg(A \vee B)$, créer un fils avec $\neg A$ et $\neg B$
- Règles disjonctives: créent deux fils:
  - *Règle* $\vee$*:* Pour $A \vee B$, créer un fils avec A et un autre avec B
  - *Règle* $\neg\wedge$*:* Pour $\neg(A \wedge B)$, créer un fils avec $\neg A$ et un autre avec $\neg B$
- Règle pour $\neg\neg$: Pour $\neg\neg A$, créer un fils avec A

Autres règles: voir formulaire

Règles (2)

$(1) \neg(a \vee b)$

$(2) c \vee d$

$(3) c[(2)]$ $(4) d[(2)]$

$(5) \neg a[(1)]$

$(6) \neg b[(1)]$

$(B1)$ $(B2)$

- Dans (B1)
  - Règle $\vee$ n'est plus applicable
  - Règle $\neg\vee$ n'est plus applicable
- Dans (B2)
  - Règle $\vee$ n'est plus applicable
  - Règle $\neg\vee$ est encore applicable à noeud (1)

Noter: "Applicabilité" d'une règle est relative à une branche!

Algorithme

Algorithme: Tant que le tableau contient des branches ouvertes qui admettent l'application d'une règle:

1. sélectionner une telle branche B
2. sélectionner un noeud n dans B qui admet l'application de R
3. appliquer R

Résultat: Si l'algorithme termine,

- toutes les branches sont fermées
  $\rightsquigarrow$ le tableau est insatisfiable
- *ou bien* il existe des branches ouvertes, sans règles applicables
  $\rightsquigarrow$ le tableau est satisfiable
  $\rightsquigarrow$ chaque branche ouverte contient uniquement des littéraux (p ou $\neg p$, pour $p \in PROP$)
  $\rightsquigarrow$ permet la construction d'un modèle

Satisfiabilité et validité

Pour vérifier la satisfiabilité d'une formule F:

1. Appliquer l'algorithme des tableaux à F
2. Résultat:
   - toutes les branches sont fermées
     $\rightsquigarrow$ F est insatisfiable
   - il existe des branches ouvertes
     $\rightsquigarrow$ F est satisfiable
     $\rightsquigarrow$ on peut construire un modèle de F

Pour vérifier la validité d'une formule F:

1. Appliquer l'algorithme des tableaux à $\neg F$
2. Résultat:
   - toutes les branches sont fermées
     $\rightsquigarrow$ $\neg F$ est insatisfiable
     donc: F est valide
   - il existe des branches ouvertes
     $\rightsquigarrow$ $\neg F$ est satisfiable
     donc: F n'est pas valide
     le modèle de $\neg F$ est un *contre-modèle* de F

Extensions: Conséquence logique

Au lieu de vérifier $\models F$ (“F est valide”), on s’intéresse souvent à:
$\{H_1, \dots H_n\} \models F$ (“F est valide sous hypothèses $H_1 \dots H_n$”)
Ceci est le cas seulement si $\{H_1, \dots H_n, \neg F\}$ est insatisfiable
Algorithme:

1. Mettre les formules $H_1, \dots H_n, \neg F$ sur une branche du tableau
2. Développer le tableau
3. Résultat:
   - toutes les branches sont fermées: $\{H_1, \dots H_n\} \models F$
   - il existe des branches ouvertes: $\{H_1, \dots H_n\} \not\models F$

Algorithme: Propriétés (1)

Terminaison: L'algorithme de calcul des tableaux termine.
Argument informel:

- Définir le *poids* d'une branche comme la "liste" des tailles de ses noeuds actifs
- Le poids des branches diminue lors de l'application des règles
- ... et ceci ne peut pas continuer à l'infini

Exemple:

- Avant application de la règle:
  Branche: $[a \wedge b; c \vee (b \wedge \neg a)]$
  Poids de la branche: $[3; 6]$
- Après application de la règle $\vee$:
  Branches $[a \wedge b; c \vee (b \wedge \neg a); c]$ et $[a \wedge b; c \vee (b \wedge \neg a); (b \wedge \neg a)]$
  Poids des branches: $[3; 1]$ et $[3; 4]$
  Il y a plus de branches, mais à moindre poids

Algorithme: Propriétés (2)

Correction de l'algorithme:

Informellement:

- L'algorithme ne trouve pas de "faux modèles"
- Si l'algorithme dit que le tableau est *satisfiable*, il l'est effectivement

Quelques définitions:

- Une branche B est dite *satisfiable* par une interprétation v si pour toutes les formules actives F de B, on a $v(F) = 1$
- Un tableau est dit *satisfiable* par une interprétation v s'il existe une branche qui est satisfiable par v

Algorithme: Propriétés (3)

Correction de l'algorithme: suit de la
Correction de l'application d'une règle:
Si le tableau T' est le résultat de l'application de la règle R à T, si T' est satisfiable par v, alors aussi T est satisfiable par v

Preuve: Soit T' satisfiable par v, avec B' satisfiable par v

- B' est déjà dans T.
  Alors: T est satisfiable par v
- autrement: examiner les règles:
  - Règle $\wedge$: alors $B' = [\ldots; p \wedge q; p; q]$, avec $v(p) = v(q) = 1$ alors $v(p \wedge q) = 1$, donc $B = [\ldots; p \wedge q]$ dans T est satisfiable.
  - Prouvez les autres règles

Algorithme: Propriétés (4)

Complétude de l'algorithme:
Informellement: L'inverse de la correction:

- L'algorithme ne perd pas de modèles
- Si l'algorithme dit que le tableau est *insatisfiable*, il l'est effectivement

Complétude de l'algorithme: suit de la
Complétude de l'application d'une règle:
Si le tableau T' est le résultat de l'application de la règle R à T, si T est satisfiable par v, alors aussi T' est satisfiable par v
Preuve: (un peu moins en détail):

- Règle $\vee$: Soit $B = [\ldots, p \vee q]$ satisfiable par v, donc
  1. $v(p) = 1$, donc $B' = [\ldots, p \vee q, p]$ satisfiable par v
  2. *ou:* $v(q) = 1$, donc $B' = [\ldots, p \vee q, q]$ satisfiable par v
- Prouvez les autres règles

Correction et complétude

Exercice 1: Vous avez besoin d'une règle pour le *"ou exclusif"* $(A \oplus B)$ défini par $((A \vee B) \wedge (\neg(A \wedge B)))$
On propose la règle:

- Si le noeud contient $A \oplus B$, créer un fils contenant A et $\neg B$

Prouvez avec cette règle:

- $(A \oplus B) \longrightarrow (A \longrightarrow \neg B)$
- $(A \oplus B) \longrightarrow \neg B$

Cette règle est-elle correcte? complète?
Sinon, proposez une règle correcte et complète.

Exercice 2:

- Notre règle de *Fermeture* permet de fermer une branche contenant A et $\neg A$, pour n'importe quelle formule A.
- Introduison une règle *Fermeture'* qui permet de fermer une branche uniquement pour p et $\neg p$, où $p \in PROP$?
- *Question:* Conséquences pour la correction / complétude?

Stratégies

Observation: L'algorithme de page 14 est non-déterministe:

- sélection d'une branche
- sélection d'un noeud sur la branche

Des stratégies rendent l'algorithme plus déterministe, en imposant des préférences:

- Si vous avez le choix entre la règle $(\wedge)$ et la règle $(\vee)$, laquelle préférez-vous?
- Généralisez!

Plan

Motivation (1)

Calcul des tableaux:

- *Avantage:* Facile à mécaniser
  - ...parce qu'il suffit de décomposer des formules données
  - ...pas nécessaire d'"inventer" quelque chose
- *Désavantage:* Ne correspond pas à un raisonnement "naturel"

Déduction naturelle:

- correspond à un style de raisonnement "naturel"
- difficile à automatiser: pas de propriété de sous-formule

Motivation (2)

Exemple de raisonnement “naturel”

Quelques propositions:

1. Quand il neige, il fait froid: $(N \longrightarrow F)$
2. Quand il y a du verglas, il fait froid: $(V \longrightarrow F)$
3. Il neige ou il y a du verglas $(N \vee V)$
4. En été, il ne fait pas froid $(E \longrightarrow \neg F)$
5. Donc, il n’est pas été: $\neg E$

Dérivation de (5) à partir de (1) . . . (4):

6. Pour montrer (5), supposons E ((6)), et dérivons une contraditiction $(\perp)$
7. Distinction de cas (avec (3)):
    1. Soit N. Alors, avec (1), on a F
    2. Soit V. Alors, avec (2), on a F

    Donc, de (1),(2),(3), on peut conclure F
8. Avec (6) et (4), inférer $\neg F$.
9. (7) et (8) permettent de conclure $\perp$, comme demandé dans (6)

Motivation (3)

Arbre de preuve:

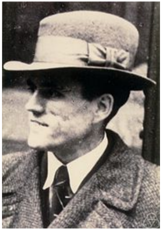
$$\dfrac{\dfrac{\overset{(3)}{N \vee V} \qquad \dfrac{\overset{(7.1)}{N} \quad \overset{(1)}{N \longrightarrow F}}{F} \qquad \dfrac{\overset{(7.2)}{V} \quad \overset{(2)}{V \longrightarrow F}}{F}}{F} \qquad \dfrac{\overset{(6)}{E} \quad \overset{(4)}{E \longrightarrow \neg F}}{\neg F}}{\dfrac{\perp}{\neg E}}$$

Histoire: Hilbert

Hilbert (1862 - 1943)

- Contributions essentielles aux fondements de la mathématique:
  - Axiomatisation de la géométrie Euclidienne
  - . . . et dans la suite: axiomatisation de la mathématique
- "Programme de Hilbert":
  - Trouver un système d'axiomes et règles de déduction qui assurent la consistance de la mathématique
  - (mis à mal par les résultats de Gödel)

Histoire: Gentzen



Gentzen (1909 - 1945)

- Développement de deux calculs (1934):
  - *Déduction naturelle*, pour rendre l'axiomatisation de Hilbert plus utilisable
  - *Calcul des séquents:* une procédure *effective* de preuve
  - Les deux calculs sont équivalents ("élimination des coupures")
- Méthode des tableaux:
  - Variante du calcul des séquents
  - . . . introduite vers 1955 par E.W.Beth

Voir article The Development of Proof Theory

Validité et Prouvabilité

Validité: $\{H_1, \ldots H_n\} \models C$

- une notion *sémantique*
- définie à l'aide d'*interprétations:* toute interprétation qui satisfait $\{H_1, \ldots H_n\}$ satisfait aussi C
- voir définitions de validité et conséquence

Prouvabilité: $\{H_1, \ldots H_n\} \vdash C$

- une notion *syntaxique*
- relative à un *calcul:* avec les règles du calcul, on peut déduire C à partir des hypothèses $\{H_1, \ldots H_n\}$
- (ici, le calcul est la *déduction naturelle*)

Critères pour un “bon” calcul:

- *Correction:* si $\{H_1, \ldots H_n\} \vdash C$, alors $\{H_1, \ldots H_n\} \models C$
- *Complétude:* si $\{H_1, \ldots H_n\} \models C$, alors $\{H_1, \ldots H_n\} \vdash C$

Règles d'inférence (1)

Un jugement a la forme $\{H_1, \dots H_n\} \vdash C$, où

- $H_1, \dots H_n$ sont les *hypothèses*
- C est la *conclusion*

Une règle

$$\frac{\{H_1^1, \dots H_n^1\} \vdash C^1 \quad \dots \quad \{H_1^m, \dots H_n^m\} \vdash C^m}{\{H_1^c, \dots H_n^c\} \vdash C^c}$$

est composée de

- 0, 1 ou plusieurs *antécédents*
- un seul *conséquent*

Lecture informelle:
"Si tous les antécédents sont prouvables, alors aussi le conséquent"

Règles d'inférence (2)

Exemples

- sans antécédents:

$$\frac{}{\{\ldots, A, \ldots\} \vdash A}\ (Ax)$$

- sans modification des hypothèses:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \land B}\ (I\land)$$

- avec décharge d'hypothèses:

$$\frac{\Gamma \cup \{A\} \vdash B}{\Gamma \vdash A \longrightarrow B}\ (I \longrightarrow)$$

Notation allégée

Notation explicite

- $$\frac{}{\{\ldots, A, \ldots\} \vdash A}\ (Ax)$$
- $$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ (I\wedge)$$
- $$\frac{\Gamma \cup \{A\} \vdash B}{\Gamma \vdash A \longrightarrow B}\ (I \longrightarrow)$$

Désavantage:
Lourdeur notationelle

Notation allégée

- Si A est une hypothèse:
  $$\frac{}{A}\ (Ax)$$
- $$\frac{A \qquad B}{A \wedge B}\ (I\wedge)$$
- $$\frac{\begin{array}{c} A(i) \\ \vdots \\ B \end{array}}{A \longrightarrow B}\ (I \longrightarrow)(i)$$

Désavantage: Nécessite une gestion scrupuleuse des hypothèses!!

Typologie des règles: Règles pour $\wedge$ (1)

Pour chaque connecteur, il y a une (éventuellement deux)

- *règle(s) d'introduction* (connecteur dans le conséquent), par exemple ($I\wedge$)
- *règle(s) d'élimination* (connecteur dans l'antécédent *principal*), par exemple ($E\wedge_1$) et ($E\wedge_2$)

Notation explicite:

- Règle d'introduction:

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ (I\wedge)$$

- Règles d'élimination:

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ (E\wedge_1) \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ (E\wedge_2)$$

Typologie des règles: Règles pour ∧ (2)

Notation allégée:

- Règle d'introduction:

$$\frac{A \qquad B}{A \wedge B} \; (I\wedge)$$

- Règles d'élimination:

$$\frac{A \wedge B}{A} \; (E\wedge_1) \qquad \frac{A \wedge B}{B} \; (E\wedge_2)$$

Typologie des règles: Règles pour $\longrightarrow$

Notation explicite

- Intro:

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \longrightarrow B} \; (I \longrightarrow)$$

- Elim:

$$\frac{\Gamma \vdash A \longrightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B} \; (E \longrightarrow)$$

Notation allégée

- Intro:

$$\frac{\begin{array}{c} A(i) \\ \cdots \\ B \end{array}}{A \longrightarrow B} \; (I \longrightarrow)(i)$$

- Elim:

$$\frac{A \longrightarrow B \qquad A}{B} \; (E \longrightarrow)$$

Notes:

- On écrit $\Gamma, A \vdash B$ au lieu de $\Gamma \cup \{A\} \vdash B$
- $(E \longrightarrow)$ aussi appelée *modus ponens*

Typologie des règles: Règles pour $\perp$

Notation explicite

- Intro:

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash \neg A}{\Gamma \vdash \perp}\ (I\perp)$$

- Elim:

$$\frac{\Gamma \vdash \perp}{\Gamma \vdash A}\ (E\perp)$$

Notation allégée

- Intro:

$$\frac{A \qquad \neg A}{\perp}\ (I\perp)$$

- Elim:

$$\frac{\perp}{A}\ (E\perp)$$

Note:

- $(E\perp)$ aussi appelée *ex falso quodlibet*

Typologie des règles: Règles pour $\vee$ (1)

Notation explicite

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ (I\vee_1) \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ (I\vee_2)$$

$$\frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C}\ (E\vee)$$

Note: $(E\vee)$ correspond à une distinction de cas

Typologie des règles: Règles pour $\vee$ (2)

Notation allégée

$$\frac{A}{A \vee B}\ (I\vee_1) \qquad \frac{B}{A \vee B}\ (I\vee_2)$$

$$\frac{A \vee B \qquad \begin{matrix} A(j) \\ \cdots \\ C \end{matrix} \qquad \begin{matrix} B(k) \\ \cdots \\ C \end{matrix}}{C}\ (E\vee)(j, k)$$

Typologie des règles: Diverses

Règle “Axiome”:

Notation explicite

$$\frac{A \in \Gamma}{\Gamma \vdash A}\ (Ax)$$

Notation allégée

Si A est une hypothèse:

$$\frac{}{A}\ (Ax)$$

Double négation (une spécificité de la logique “classique”):

Notation explicite

$$\frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A}\ (\neg\neg)$$

Notation allégée

$$\frac{\neg\neg A}{A}\ (\neg\neg)$$

Règles dérivées:

Règles pour $\neg$: On peut définir $\neg A$ comme: $A \longrightarrow \bot$

- Dérivez la règle $(I\neg)$ à l'aide de $(I \longrightarrow)$
- Montrez que l'élimination de $\neg$ correspond à $(I\bot)$

Règles pour $\leftrightarrow$: On peut définir $A \leftrightarrow B$ comme: $(A \longrightarrow B) \wedge (B \longrightarrow A)$

- Dérivez les règles $(I \leftrightarrow)$ et $(E \leftrightarrow)$

Portée (1)

Rappel: Portée dans des langages de programmation

Bien typé en Ocaml:

```
# let x = 5 in
      (let y = x + 7 in
       y + 2)
      ,
      (let z = x - 3 in
       z + x) ;;

  - : int * int = (14, 7)
```

Mal typé en Ocaml:

```
# let x = 5 in
      (let y = x + 7 in
       y + 2)
      ,
      (let z = x - 3 in
       y + x) ;;

Error: Unbound value y
```

Portée (2)

…en Déduction Naturelle:

1. Exercice: Faire la preuve de:
   $A \longrightarrow ((B \longrightarrow A \wedge B) \wedge (C \longrightarrow C \wedge A))$
2. Exercice: Donner un contre-modèle de:
   $A \longrightarrow ((B \longrightarrow A \wedge B) \wedge (C \longrightarrow B \wedge A))$

Où est le problème dans la "preuve" (fausse!):

$$
\dfrac{\dfrac{\dfrac{\dfrac{\overset{(1)}{A} \qquad \overset{(2)}{B}}{A \wedge B}\,(I\wedge)}{B \longrightarrow A \wedge B}\,(I \longrightarrow)(2) \qquad \dfrac{\dfrac{\overset{(2)}{B} \qquad \overset{(1)}{A}}{B \wedge A}\,(I\wedge)}{C \longrightarrow B \wedge A}\,(I \longrightarrow)(3)}{(B \longrightarrow A \wedge B) \wedge (C \longrightarrow B \wedge A)}\,(I\wedge)}{A \longrightarrow ((B \longrightarrow A \wedge B) \wedge (C \longrightarrow B \wedge A))}\,(I \longrightarrow)(1)
$$

Portée (3)

Important: On peut utiliser une hypothèse uniquement dans le sous-arbre qui l'a générée.
La notation explicite est plus claire. *Comparez:*

Correct:

$$\dfrac{\dfrac{\{A, B\} \vdash A \quad \{A, B\} \vdash B}{\{A, B\} \vdash A \wedge B}\,(I\wedge)}{\{A\} \vdash B \longrightarrow A \wedge B}\,(I \longrightarrow)$$

Incorrect:

$$\dfrac{\dfrac{\{A, C\} \vdash B \quad \{A, C\} \vdash A}{\{A, C\} \vdash B \wedge A}\,(I\wedge)}{\{A\} \vdash C \longrightarrow B \wedge A}\,(I \longrightarrow)$$

Déduction Naturelle: Résumé

Principes:

- Motivation: Modéliser le raisonnement humain
- Structure de base: Arbre de déduction
  - *Lecture:* De haut en bas (antécédent $\rightsquigarrow$ conséquent)
  - *Construction:* De bas en haut

Règles:

- ...d'introduction: connecteur dans le conséquent
- ...d'élimination: connecteur dans l'antécédent

Subtil:

- Certaines règles modifient les hypothèses en vigueur dans un sous-arbre
- Difficile à automatiser (pas de *propriété de sous-formule*)

Plan

Motivation et Historique (1)

Résolution: Une méthode de déduction assez récente:
Alan Robinson:
A Machine-Oriented Logic Based on the Resolution Principle (1965)

But:

- Contexte: Premières applications en Intelligence Artificielle
- Implantation simple et efficace sur ordinateur
- . . . sans ambition d'être compréhensible pour des humains

Principe:

- Une seule règle d'inférence appliquée à un ensemble de *clauses*
- . . . nécessitant un pré-traitement de formules structurées

Motivation et Historique (2)

Idée: Propagation de faits

Donné:

- des faits: $A, \; C$
- des implications: $A \longrightarrow B, \; C \longrightarrow B \longrightarrow D, \; B \longrightarrow D \longrightarrow F$
- un but: prouver F

Démarche: Propagation des faits dans les implications ("modus ponens"):

1.
   - A et $A \longrightarrow B$ donnent B
   - C et $C \longrightarrow B \longrightarrow D$ donnent $B \longrightarrow D$
2.
   - B et $B \longrightarrow D \longrightarrow F$ donnent $D \longrightarrow F$
   - B et $B \longrightarrow D$ donnent D
3.
   - D et $D \longrightarrow F$ donnent F

Ceci est la base du langage de programmation Prolog

Motivation et Historique (3)

Les bases de la résolution:

- Au lieu d'implications ($A \longrightarrow B$): des disjonctions: $\neg A \lor B$
- La règle de résolution: A et $\neg A \lor B$ donnent B

...dans un cadre plus général:

- Des *clauses* avec plusieurs variables positives et négatives: $\neg A \lor B \lor C$
- Résolution entre clauses:
  $A \lor B \lor \neg D$ et $\neg A \lor B \lor C$ donnent $B \lor C \lor \neg D$

Étapes principales: Pour une formule F

1. Construction d'une forme clausale de F
2. Application de la résolution

Formes Normales

Un littéral est une variable propositionnelle ou sa négation, *par exemple:* p, $\neg q$, *mais pas:* $p \vee \neg q$

Une formule est en

- Forme Normale Disjonctive (FND) si elle est une disjonction de conjonctions de littéraux
- Forme Normale Conjonctive (FNC) si elle est une conjonction de disjonctions de littéraux

Exercice: FND, FNC, ou ...???

- $(\neg A \vee B) \wedge (C \vee D)$
- $(\neg A \wedge B) \vee (C \wedge D)$
- $\neg(A \wedge B) \vee (C \wedge D)$
- $(A \vee \bot) \wedge (C \wedge \neg D)$
- $A \vee (C \vee \neg D)$
- $A \wedge \neg B$

Conversion en Forme Normale Conjonctive

1. Éliminer les connecteurs autres que $\neg, \wedge, \vee$
   1. Réécrire $A \leftrightarrow B$ en $(A \longrightarrow B) \wedge (B \longrightarrow A)$
   2. Réécrire $A \longrightarrow B$ en $\neg A \vee B$
2. Tirer les négations à l'intérieur, éliminer les doubles négations:
   - $\neg(A \wedge B)$ devient $\neg A \vee \neg B$
   - $\neg(A \vee B)$ devient $\neg A \wedge \neg B$
   - $\neg\neg A$ devient A
3. Distribuer $\vee$ sur $\wedge$:
   - $A \vee (B \wedge C)$ devient $(A \vee B) \wedge (A \vee C)$
   - et pareil pour $(B \wedge C) \vee A$
4. Faire des simplifications triviales,
   par exemple de $A \wedge \neg A$, $A \vee \neg A$, $A \vee \neg\bot$ ...

Exercice: Convertir $\neg((A \wedge \neg B) \longrightarrow (A \leftrightarrow \neg B))$ en FNC

Clauses

Une clause est un ensemble de littéraux.
Une clause représente la disjonction de ses littéraux.
Exemple: $\{A, \neg B\}$ représente $A \vee \neg B$
Notes:

- Deux formules syntaxiquement différentes peuvent être représentées par la même clause:
  $\{A, \neg B\}$ représente $A \vee \neg B$ et $A \vee \neg B \vee A$
- ... mais ces formules sont équivalentes: $(A \vee \neg B) \equiv (A \vee \neg B \vee A)$
- La clause vide $\{\}$ représente $\perp$

Un ensemble de clauses représente la conjonction des disjonctions des littéraux des clauses.
Exemple: $\{\{A, \neg B\}, \{B, \neg A\}\}$ représente $(A \vee \neg B) \wedge (B \vee \neg A)$

Règle de Résolution

Étant donné:

- une clause $\mathcal{C} = \{\ell_1 \dots \ell_m, p\}$
- une clause $\mathcal{C}' = \{\ell'_1 \dots \ell'_n, \neg p\}$

La résolvante de $\mathcal{C}$ et $\mathcal{C}'$ (écrit $res(\mathcal{C}, \mathcal{C}')$)
est la clause $\{\ell_1 \dots \ell_m, \ell'_1 \dots \ell'_n\}$
(*Petite imprécision:* on ne détaille pas quel littéral p est éliminé.)

Exemples:

- $res(\{A, B\}, \{\neg C, \neg B\}) = \{A, \neg C\}$
- $res(\{\neg A, B, C\}, \{\neg B, A\}) = \{\neg A, A, C\}$
- $res(\{\neg A, B, C\}, \{\neg B, A\}) = \{\neg B, B, C\}$

Cas spécial: La résolvante peut être vide: $res(\{A\}, \{\neg A\}) = \{\}$

Preuve par Résolution

Donné: Un ensemble de clauses $\mathcal{E}$

Algorithme:

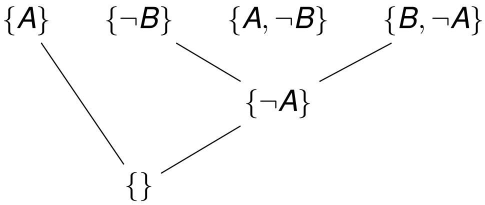
Tant que $\{\} \notin \mathcal{E}$ et tant qu'il existe $\mathcal{C}, \mathcal{C}' \in \mathcal{E}$ avec $res(\mathcal{C}, \mathcal{C}') \notin \mathcal{E}$:

- $\mathcal{E} := \mathcal{E} \cup \{res(\mathcal{C}, \mathcal{C}')\}$

Résultats possibles:

1. $\{\} \in \mathcal{E}$. Alors $\mathcal{E}$ est insatisfiable
2. $\{\} \notin \mathcal{E}$ et impossible de former de nouvelles résolvantes. Alors $\mathcal{E}$ est satisfiable

Exemple d'un ensemble de clauses insatisfiable:



Résolution: Propriétés (1)

Préservation des interprétations:

$\mathcal{I}(\mathcal{E} \cup \{res(\mathcal{C}, \mathcal{C}')\}) = \mathcal{I}(\mathcal{E})$ pour $\mathcal{C}, \mathcal{C}' \in \mathcal{E}$

Preuve: Soit $\mathcal{C} = \{\ell_1 \dots \ell_m, p\}$, $\mathcal{C}' = \{\ell'_1 \dots \ell'_n, \neg p\}$

- Soit $\mathcal{I}$ un modèle de $\mathcal{E} \cup \{res(\mathcal{C}, \mathcal{C}')\}$. Alors, $\mathcal{I}$ est un modèle de $\mathcal{E}$.
- Soit $\mathcal{I}$ un modèle de $\mathcal{E}$,
  donc surtout: $\mathcal{I}(\mathcal{C}) = 1$ et $\mathcal{I}(\mathcal{C}') = 1$
  donc $\mathcal{I}(\ell_1 \vee \dots \vee \ell_m \vee p) = 1$ et $\mathcal{I}(\ell'_1 \vee \dots \vee \ell'_n \vee \neg p) = 1$.
  Alors, $\mathcal{I}(\ell_1 \vee \dots \vee \ell_m \vee \ell'_1 \vee \dots \vee \ell'_n) = 1$
  Pourquoi? Complétez les détails!
  Donc: $\mathcal{I}$ est un modèle de $\mathcal{E} \cup \{res(\mathcal{C}, \mathcal{C}')\}$.

Résolution: Propriétés (2)

Correction: Étant donné $\mathcal{E}$. Si l'algorithme de résolution fourni un $\mathcal{E}'$ avec $\{\} \in \mathcal{E}'$, alors $\mathcal{E}$ n'est pas satisfiable.

Preuve: par induction sur le nombre *n* d'itérations de l'algorithme.

- $n = 0$: Alors, $\mathcal{E} = \{\mathcal{C}_1, \ldots, \mathcal{C}_k, \{\}\}$ représente une formule $C_1 \wedge \ldots \wedge C_k \wedge \perp \equiv \perp$
- $n \to n + 1$:
  D'après la préservation des interprétations:
  Si $\mathcal{E} \cup \{res(\mathcal{C}, \mathcal{C}')\}$ n'a pas de modèle, alors $\mathcal{E}$ non plus.

Résolution: Propriétés (3)

Complétude: Étant donné $\mathcal{E}$. Si l'algorithme de résolution fourni un $\mathcal{E}'$ avec $\{\} \notin \mathcal{E}'$, alors $\mathcal{E}$ est satisfiable.

Preuve: par induction sur le nombre *n* d'itérations de l'algorithme.

- $n = 0$: Si
  - $\{\} \notin \mathcal{E}$
  - et $\mathcal{E}$ est *saturé* (n'admet pas la dérivation d'une nouvelle résolvante)

  alors $\mathcal{E}$ a un modèle.
  voir: Extraction d'un modèle
- $n \to n + 1$:
  D'après la préservation des interprétations:
  Tout modèle de $\mathcal{E} \cup \{res(\mathcal{C}, \mathcal{C}')\}$ est un modèle de $\mathcal{E}$.

Extraction d'un modèle (1)

Substitution d'une constante dans un ensemble de clauses:

- $\mathcal{E}[\perp/p]$ est l'ensemble de clauses où:
  - le littéral p est éliminé de toutes les clauses
  - toute clause contenant $\neg p$ est éliminée

  Exemple: $\{\{A, \neg B\}, \{\neg A, C\}, \{A, C\}\}[\perp/A] = \{\{\neg B\}, \{C\}\}$
- $\mathcal{E}[\top/p]$ est l'ensemble de clauses où:
  - toute clause contenant p est éliminée
  - le littéral $\neg p$ est éliminé de toutes les clauses

  Exemple: $\{\{A, \neg B\}, \{\neg A, C\}, \{A, C\}\}[\top/A] = \{\{C\}\}$

Justifiez cette définition!
Montrez: Si $\mathcal{E}$ est saturé, alors aussi $\mathcal{E}[\perp/p]$ et $\mathcal{E}[\top/p]$.

Extraction d'un modèle (2)

Construction d'un modèle par preuve de:
Si $\{\} \notin \mathcal{E}$ et $\mathcal{E}$ est saturé, alors $\mathcal{E}$ a un modèle.

Induction sur le nombre n de variables $p_1, \ldots p_n$ dans $\mathcal{E}$.
La preuve construit une valuation v pour $p_1, \ldots p_n$ tq. $\mathcal{I}_v(\mathcal{E}) = 1$

- $n = 0$: $\mathcal{E}$ est de la forme:
  - $\{\}$: Alors, $\mathcal{E}$ est valide
  - $\{\{\}\}$: impossible, parce que $\{\} \notin \mathcal{E}$
- $n \to n + 1$:
  1. Construire $\mathcal{E}' := \mathcal{E}[\perp/p_{n+1}]$, qui est saturé. Deux cas:
     - $\{\} \in \mathcal{E}'$: Alors, $\mathcal{E}'$ n'a pas de modèle.
     - $\{\} \notin \mathcal{E}'$: Par hyp. d'induction, il existe valuation v' tq. $\mathcal{I}_{v'}(\mathcal{E}') = 1$. Alors, pour v défini comme $v'(p_{n+1} := 0)$, on a $\mathcal{I}_v(\mathcal{E}) = 1$
  2. Construire $\mathcal{E}'' := \mathcal{E}[\top/p_{n+1}]$, et procéder en analogie à (1)

  Pourquoi est-ce qu'on ne peut pas avoir $\{\} \in \mathcal{E}'$ et $\{\} \in \mathcal{E}''$ en même temps?

Extraction d'un modèle (3)

Exemple: Ensemble de clauses initial: $\{\{A, \neg B\}, \{B, \neg C\}, \{A, C\}\}$

Ensemble saturé: $\mathcal{E} = \{\{A, \neg B\}, \{B, \neg C\}, \{A, C\}, \{A, \neg C\}, \{A\}\}$

Construction des modèles:

- $\mathcal{E}[\bot/A] = \{\{\neg B\}, \{B, \neg C\}, \{C\}, \{\neg C\}, \{\}\}$ n'a pas de modèle
- $\mathcal{E}[\top/A] = \{\{B, \neg C\}\}$
  - $\{\{B, \neg C\}\}[\bot/B] = \{\neg C\}$
    - $\{\neg C\}[\bot/C] = \{\}$ est valide
      $\rightsquigarrow$ valuation $v_0(A) = 1$, $v_0(B) = 0$, $v_0(C) = 0$
    - $\{\neg C\}[\top/C] = \{\{\}\}$ n'a pas de modèle
  - $\{\{B, \neg C\}\}[\top/B] = \{\}$ est valide
    $\rightsquigarrow$ valuation $v_1(A) = 1$, $v_1(B) = 1$

Optimisations

Subsomption: La clause $\mathcal{C}$ *subsume* la clause $\mathcal{C}'$ si $\mathcal{C} \subseteq \mathcal{C}'$.
Signification:
Si la clause $\mathcal{C}$ représente la formule C et $\mathcal{C}'$ représente C', alors $C \longrightarrow C'$.
Exemple: $\{A\}$ subsume $\{A, \neg B\}$, et $A \longrightarrow (A \vee \neg B)$
Constat: Si $\mathcal{E} \cup \{\mathcal{C}, \mathcal{C}'\}$ est insatisfiable et $\mathcal{C} \subseteq \mathcal{C}'$, alors déjà $\mathcal{E} \cup \{\mathcal{C}\}$ est insatisfiable.
Idée de preuve: Si $\neg(E \wedge C \wedge C')$ et $C \longrightarrow C'$, alors $\neg(E \wedge C)$
Optimisation: “On peut supprimer les clauses subsumées par d’autres”
Dérivez la clause vide à partir de
$\{\{A, \neg B\}, \{\neg A, C\}, \{A\}, \{B, C\}, \{\neg C\}\}$

Résolution: Résumé (1)

Idée générale: La résolution permet de vérifier qu'un ensemble de clauses est insatisfiable.

Pour vérifier la validité d'une formule F:

1. Convertir $\neg F$ en Forme Normale Conjonctive
2. Construire l'ensemble de clauses correspondant
3. Appliquer l'algorithme de résolution:
   - Si $\{\}$ est une résolvante, alors F est valide
   - Si on ne peut pas obtenir $\{\}$, alors F n'est pas valide (contre-modèle par extraction d'un modèle de l'ensemble de clauses saturé)

Résolution: Résumé (2)

Astuces Pour vérifier que $\{H_1, \ldots H_N\} \models F$:

- Convertir $\neg((H_1 \wedge \ldots \wedge H_n) \longrightarrow F)$ en FNC
- équivalent à convertir $(H_1 \wedge \ldots \wedge H_n \wedge \neg F)$ en FNC
- équivalent à convertir H_1 en FNC, $\ldots$ H_n en FNC, $\neg F$ en FNC et prendre l'union des clauses
- vérifier que cet ensemble est insatisfiable

Vérifier que $\{A \longrightarrow B, B \wedge C \longrightarrow \neg D, A, D\} \models \neg C$

Plan

Motivation (1)

La logique des *propositions* est limitée en expressivité.

- elle ne permet pas de parler d'individus:
  "Si l'objet $o1$ est bleu, alors $o1$ est aussi rond"
  - Tentative de modélisation: $B \longrightarrow R$
    Problème: Traduire "Si $o1$ est bleu, alors $o1$ est rond et $o2$ carré"
  - Tentative de modélisation: $B1 \longrightarrow R1 \wedge C2$
    Problème: Différence structurelle avec $X \longrightarrow R1 \wedge C2$ ou $B2 \longrightarrow R1 \wedge C2$?
- elle n'est pas concise (voir quantif. sur ensembles finis):
  "Tout objet bleu est rond" (on connaît les objets $o1, o2, o3$):
  $(B1 \longrightarrow R1) \wedge (B2 \longrightarrow R2) \wedge (B3 \longrightarrow R3)$
- elle ne permet pas de parler d'une infinité d'individus:
  "Tout objet bleu est rond"
- elle ne permet pas de parler d'actions / opérations:
  "Tout objet repaint deux fois apparaît neuf"

Motivation (2)

La logique des *prédicats* LPred raffine la logique des *propositions*.

Individus et prédicats

- Langage naturel: “Si l’objet $o1$ est bleu, alors $o1$ est aussi rond”
- LProp: $B1 \longrightarrow R1$, où
  - $B1, R1$ sont des phrases élémentaires
- LPred: $B(o1) \longrightarrow R(o1)$, où
  - $o1$ est le *sujet* de la phrase: “l’objet $o1$”
  - B est le *prédicat*: “est bleu”
  - On garde les connecteurs de LProp ($\longrightarrow$)

Motivation (3)

Quantificateurs

- Langage naturel: “Tout objet bleu est rond”
- LProp: Pas exprimable en général
- LPred: $\forall x.(B(x) \longrightarrow R(x))$

Opérations / Fonctions

- Langage naturel: “Tout objet repaint deux fois apparaît neuf”
- LProp: Pas exprimable en général
- LPred: $\forall x.N(r(r(x)))$, où
  - x est un individu
  - r est une fonction qui transforme des individus (“repaindre”)
  - N est un prédicat

Syntaxe de LPred: Termes et formules (1)

Termes: Soit

- VAR un ensemble de variables d'individus
- FON_n un ensemble des fonctions n-aires

L'ensemble $TERM$ des termes est défini par induction comme le plus petit ensemble qui satisfait:

1. *Variable:* si $x \in VAR$, alors $x \in TERM$
2. *Application de fonction:* si $t_1 \in TERM, \ldots t_n \in TERM$ et $f \in FON_n$, alors $f(t_1, \ldots, t_n) \in TERM$

On appelle des éléments de FON_0 des *constantes*.
Souvent, on écrit c au lieu de $c()$.
Exemples:

- $f(x, c) \in TERM$, pour $f \in FON_2, x \in VAR, c \in FON_0$
- $f(x, g(y, \pi)) \in TERM$, pour $f \in FON_2, x, y \in VAR, g \in FON_1, \pi \in FON_0$

Syntaxe de LPred: Termes et formules (2)

Formules: Soit $PRED_n$ un ensemble des prédicats n-aires

L'ensemble $FORM$ des formules est défini par induction comme le plus petit ensemble qui satisfait:

1. *Constante "faux":* $\perp \in FORM$
2. *Application de prédicat:* si $t_1 \in TERM, \ldots t_n \in TERM$ et $P \in PRED_n$, alors $P(t_1, \ldots, t_n) \in FORM$
3. *Négation:* Si $A \in FORM$, alors $(\neg A) \in FORM$
4. *Connecteurs binaires:* Si $A \in FORM$ et $B \in FORM$, alors $(A \wedge B) \in FORM, (A \vee B) \in FORM, (A \longrightarrow B) \in FORM$
5. *Quantificateur universel* ("pour tout"): Si $A \in FORM$ et $x \in VAR$, alors $(\forall x.A) \in FORM$
6. *Quantificateur existentiel* ("il existe"): Si $A \in FORM$ et $x \in VAR$, alors $(\exists x.A) \in FORM$

à noter: $FORM$ dépend de $TERM$, mais pas inversement

Fonctions et Prédicats

Utilisation: Fonctions et prédicats traduisent des *verbes*.
Les fonctions décrivent la transformation d'un ou plusieurs objet(s):

- "une tarte faite de farine et de pommes": $fairetarte(farine, pomme)$

Les prédicats décrivent un état / une situation:

- "Eve achète une pomme": $Acheter(eve, pomme)$
- "Eve mange une tarte de pomme": $Manger(eve, fairetarte(farine, pomme))$

Un prédicat ne peut pas prendre des formules comme arguments:
"Eve mange les pommes qu'elle achète":

- *Faux:* $Manger(eve, Acheter(eve, pomme))$
- *Correct:* $\forall p.Acheter(eve, p) \longrightarrow Manger(eve, p)$

Convention: nom de fonctions: minuscules; de prédicats: majuscules

Limites de l'expressivité

Contournables: par exemple: propriétés temporelles:

- "Si le composant tombe en panne, il émet éventuellement un signal"
- Peut être codé par:
  $\forall t. Panne(c, t) \longrightarrow (\exists t'. t' > t \wedge Signal(c, t'))$

Essentielles:

- Utilisation des fonctions comme objets:
  - Quantification sur des fonctions: "Toute fonction a une inverse":
    Interdit: $\forall f. \exists g. \forall x. f(g(x)) = x$
  - Fonctions qui prennent des fonctions comme argument:
    Interdit: $\forall f. \forall x. f(inv(f, x)) = x$
- De même: Quantification sur des prédicats ou ensembles

$\leadsto$ inapproprié pour raisonner sur certains programmes fonctionnels

Variables libres et variables liées

La quantification de LPred est sujette aux mêmes conditions que la quantification sur des ensembles finis.
Comparez:

$$(\bigwedge_{i \in E} r_{i,j} \wedge (\bigvee_{k \in E} s_{i,k}))$$

et

$$(\forall i . r(i, j) \wedge (\exists k . s(i, k)))$$

Différence:

- Une formule quantifiée sur des ensembles finis n'est pas bien définie dans LProp si elle contient des indices libres.
- Une formule de LPred peut contenir des variables libres.

Substitutions (1)

- Substitution dans un terme:
  $t[s/x]$, où x est une variable et t et s sont des termes:
  1. *Variable:* $x[s/x] = s$ et $y[s/x] = y$ pour $y \neq x$
  2. *Application de fonction:* $f(t_1, \ldots, t_n)[s/x] = f(t_1[s/x], \ldots, t_n[s/x])$
- Substitution dans une formule:
  $F[s/x]$, où x est une variable, s est un terme et F une formule:
  1. *Application de prédicat:* $P(t_1, \ldots, t_n)[s/x] = P(t_1[s/x], \ldots, t_n[s/x])$
  2. *Const. "faux", connecteurs propositionnels:* comme d'habitude
  3. *Quantificateur universel:* $(\forall y.A)[s/x] = (\forall y.(A[s/x]))$
  4. *Quantificateur existentiel:* $(\exists y.A)[s/x] = (\exists y.(A[s/x]))$

Substitutions (2)

Dans le cas des quantificateurs: Une substitution est dite saine si la variable quantifiée y diffère de x et des variables de s

- *Subst. saine:* $(\forall y.R(y,x))[f(z)/x] = (\forall y.R(y,f(z)))$
- *Subst. pas saine:* $(\forall y.R(y,x))[f(y)/x] = (\forall y.R(y,f(y)))$

Pour obtenir une substitution saine:
Renommer des variables liées avant d'effectuer la substitution
$(\forall y.R(y,x))[f(y)/x] = \quad (\forall y_0.R(y_0,x))[f(y)/x] = \quad (\forall y_0.R(y_0,f(y)))$

Exercices: Calculer les substitutions (saines!) de:

- $(\forall x.\exists y.R(x,y) \wedge P(z))[f(v)/z]$
- $(\forall x.\exists y.R(x,y) \wedge P(z))[f(x)/z]$
- $(\forall x.\exists y.R(x,y) \wedge P(z))[f(v)/x]$

Plan

Valuation et Interprétation

Rappel: Logique propositionnelle

- Une *valuation* v détermine la valeur de vérité d'une variable propositionnelle. *Exemple:* $v(A) = 0, v(B) = 1$
- Une *interprétation* $\mathcal{I}_v$ étend une valuation à une formule. *Exemple:* $\mathcal{I}_v(\neg A \wedge B) = 1$

Logique des prédicats: Plus complexe:

- Distinction entre "termes" et "formules":
  $\rightsquigarrow$ deux fonctions d'interprétation différentes
- Prédicats n-aires au lieu de propositions 0-aires:
  $\rightsquigarrow$ interprétation de prédicats comme "ensembles"

Interprétation des termes (1)

Quelle est la signification du terme $f(x, y)$?

1.
   - Si x représente 2 et y représente 3 et f est la fonction d'addition, alors $f(x, y)$ représente $2 + 3$, donc 5.
   - Si x représente 2 et y représente 3 et f est la fonction de multiplication, alors $f(x, y)$ représente $2 * 3$, donc 6.
2.
   - Si x représente "vrai" et y représente "faux", et f est la fonction de conjonction, alors $f(x, y)$ représente "faux".

Une valuation pour les termes est donnée par
$(D, v_{VAR}, \{v_{FON_n} | n \in Nat\})$:

- Un domaine d'interprétation D: un ensemble non vide
- v_{VAR} : $VAR \Rightarrow D$ est une valuation pour les variables
- v_{FON_n} : $FON_n \Rightarrow D^n \Rightarrow D$ est une valuation pour les fonctions

Interprétation des termes (2)

Exemples:

1. - Domaine $D = Nat$
   - Interprétation des variables: $v_{VAR} : VAR \Rightarrow Nat$ avec $v_{VAR}(x) = 2, v_{VAR}(y) = 3$
   - Interprétation des fonctions binaires:
     $v_{FON_2}(f)$ la fonction d'addition
     $v_{FON_2}(g)$ la fonction de multiplication

   Le terme $f(x, g(x, y))$ est interprété comme $2 + (2 * 3) = 8$
2. - Domaine $D = Bool$
   - Interprétation des variables: $v_{VAR} : VAR \Rightarrow Bool$ avec $v_{VAR}(x) = true, v_{VAR}(y) = false$
   - Interprétation des fonctions binaires:
     $v_{FON_2}(f)$ la fonction “et”
     $v_{FON_2}(g)$ la fonction “ou”

   Le terme $f(x, g(x, y))$ est interprété comme $true \wedge (true \vee false) = true$

Interprétation des termes (3)

Une interprétation des termes $\mathcal{I}_v$ est l'extension de $v = (D, v_{VAR}, \{v_{FON_n} | n \in Nat\})$ à $TERM$:

Définition récursive:

- *(Variable):* $\mathcal{I}_v\ (x) = v_{VAR}(x)$
- *(Application):* $\mathcal{I}_v\ (f(t_1, \ldots, t_n)) = v_{FON_n}(f)(\mathcal{I}_v(t_1),\ \ldots,\ \mathcal{I}_v(t_n))$

Exemple:

$v = (Nat, v_{VAR} = [x \mapsto 2, y \mapsto 3], \{v_{FON_2} = [f \mapsto (+),\ g \mapsto (*)]\})$

- $\mathcal{I}_v\ (f(x, g(x, y))) = 2 + 6 = 8$
  - $v_{FON_2}(f) = (+)$
  - $\mathcal{I}_v\ (x) = v_{VAR}(x) = 2$
  - $\mathcal{I}_v\ (g(x, y)) = 2 * 3 = 6$
    - $v_{FON_2}(g) = (*)$
    - $\mathcal{I}_v\ (x) = v_{VAR}(x) = 2$
    - $\mathcal{I}_v\ (y) = v_{VAR}(y) = 3$

Interprétation des formules (1)

Quelle est la signification de la formule (élémentaire) $P(x)$?

1. Si x représente 2 et P représente la propriété “être un nombre pair”, alors $P(x)$ est une proposition vraie.
2. Si x représente π et P représente la propriété “être un nombre rationnel”, alors $P(x)$ est une proposition fausse.

Une valuation pour les formules est donnée par
$(D, v_{VAR}, \{v_{FON_n} | n \in Nat\}, \{v_{PRED_n} | n \in Nat\})$:

- $(D, v_{VAR}, \{v_{FON_n} | n \in Nat\})$ est une valuation pour les termes
- $v_{PRED_n} : PRED_n \Rightarrow D^n \Rightarrow \{\mathbf{0}, \mathbf{1}\}$ est une valuation pour les prédicats

Interprétation des formules (2)

Une interprétation des formules $\mathcal{I}_v$ est l'extension de $v = (D, v_{VAR}, \{v_{FON_n} | n \in Nat\}, \{v_{PRED_n} | n \in Nat\})$ à $FORM$:

- *Application de prédicat:* $\mathcal{I}_v\ (P(t_1, \ldots, t_n)) = v_{PRED_n}(P)(\mathcal{I}_v(t_1),\ \ldots,\ \mathcal{I}_v(t_n))$
- *Constante "faux", négation, connecteurs binaires:* comme pour la logique propositionnelle
- *Quantificateur universel:* $\mathcal{I}_v\ (\forall x.P(x)) = min_{d \in D}\{\mathcal{I}_{v(x:=d)}(P(x))\}$
- *Quantificateur existentiel:* $\mathcal{I}_v\ (\exists x.P(x)) = max_{d \in D}\{\mathcal{I}_{v(x:=d)}(P(x))\}$

Actualisation d'une fonction pour un argument:

$$v(x := d)(y) = \begin{cases} d & \text{si } x = y \\ v(y) & \text{si } x \neq y \end{cases}$$

Interprétation des formules (3)

Cas spécial: Domaine fini. Soit

- $N_3 = \{0, 1, 2\}$
- $\oplus$ l'addition modulo 3: $x \oplus y \equiv (x + y) \mod 3$

Soit la valuation v avec

- Domaine N_3
- $v_{VAR} = [x_1 \mapsto 1, x_2 \mapsto 2]$
- $\{v_{FON_0} = [n_0 \mapsto 0, n_1 \mapsto 1, n_2 \mapsto 2], v_{FON_2} = [a \mapsto \oplus]\})$
- $\{v_{PRED_1} = [P \mapsto$ "est un nombre pair"$]\}$

Interprétation de:

- $\mathcal{I}_v(P(x_1)) = v_{PRED_1}(P)(v_{FON_0}(1)) = pair(1) = \mathbf{0}$
- $\mathcal{I}_v(\exists x.P(a(x_2, x))) = max_{d \in N_3}\{\mathcal{I}_{v(x:=d)}(P(a(x_2, x)))\}$
  $= max_{d \in N_3}\{pair(2 \oplus d)\} = max\{pair(2 \oplus 0), pair(2 \oplus 1), pair(2 \oplus 2)\}$
  $= \mathbf{1}$

Interprétation des formules (4)

Exercices (dans le domaine N_3):

- Développez $\mathcal{I}_v(\forall x.P(a(x_2, x)))$
- Comparez $\mathcal{I}_v(\exists x.P(x))$ et $\mathcal{I}_v(P(n_0) \vee P(n_1) \vee P(n_2))$
- Comparez $\mathcal{I}_v(\forall x.P(x))$ et $\mathcal{I}_v(P(n_0) \wedge P(n_1) \wedge P(n_2))$

Les notions de modèle, satisfiabilité, validité, conséquence et équivalence logique sont définies comme en logique propositionnelle.

Exercices (en général):

- Montrez: $\exists x.(P(x) \vee Q(x)) \equiv (\exists x.P(x)) \vee (\exists x.Q(x))$
- Montrez: $\forall x.(P(x) \wedge Q(x)) \equiv (\forall x.P(x)) \wedge (\forall x.Q(x))$
- Vous savez que $\neg(F_0 \vee F_1) \equiv (\neg F_0 \wedge \neg F_1)$.
  Alors: $\neg(\exists x.P(x)) \equiv ???$
- Vous savez que $\neg(F_0 \wedge F_1) \equiv (\neg F_0 \vee \neg F_1)$.
  Alors: $\neg(\forall x.P(x)) \equiv ???$

Interprétation des formules (4)

Vrai ou faux?

- Si vrai, fournissez un modèle fini (avec $\wedge$ et $\vee$)
  (c'est un argument de plausibilité, pas une preuve)!
- Si faux, fournissez un contre-modèle

$\forall x.\forall y.R(x,y)$ $\equiv \forall y.\forall x.R(x,y)$	vrai	$(A_{1,1} \wedge A_{1,2}) \wedge (A_{2,1} \wedge A_{2,2})$ $\equiv (A_{1,1} \wedge A_{2,1}) \wedge (A_{1,2} \wedge A_{2,2})$
$\forall x.\exists y.R(x,y)$ $\equiv \exists y.\forall x.R(x,y)$	faux	*contre-mod.:* domaine: *Nat*, $v_{PRED}(R) = (<)$
$\exists x.\exists y.R(x,y)$ $\equiv \exists y.\exists x.R(x,y)$		
$\exists x.(P(x) \wedge Q(x))$ $\equiv (\exists x.P(x)) \wedge (\exists x.Q(x))$		
$\forall x.(P(x) \vee Q(x))$ $\equiv (\forall x.P(x)) \vee (\forall x.Q(x))$		

Interprétation des formules (5)

Exercice: Montrez par induction le *lemme d'indifférence*:
Si $x \notin vars(F)$, alors pour tout d: $\mathcal{I}_v(F) = \mathcal{I}_{v(x:=d)}(F)$
"les interprétations ne s'intéressent qu'aux variables libres"
Exercice: Utilisez ce lemme pour montrer:

- $\exists x.(P \wedge Q(x)) \equiv P \wedge (\exists x.Q(x))$, si $x \notin vars(P)$
- $\forall x.(P \vee Q(x)) \equiv P \vee (\forall x.Q(x))$, si $x \notin vars(P)$

Forme Normale Prénexe (1)

Une formule est en Forme Normale Prénexe (FNP) si elle est de la forme

$$Q_1 x_1 . Q_2 x_2 . \dots . Q_n x_n . F$$

où $Q_1, \dots Q_n$ sont des quantificateurs $\forall, \exists$ et F ne contient pas de quantificateurs.

Exemples:

- $\forall x . \exists y . R(x, y)$
- $\forall x . \exists y . (P(x) \wedge Q(y))$
- $\exists x . \exists y . \forall z . (R(x, z) \longrightarrow Q(y))$

Contre-Exemples:

- $\forall x . (P(x) \wedge (\exists y . Q(y)))$
- $\exists x . (P(x) \vee (\forall z . (R(x, z) \wedge Q(x))))$

Forme Normale Prénexe (2)

Montrez par induction: Si *fnp_neg* est défini par:

- $fnp\_neg(\forall x.F) = \exists x.(fnp\_neg(F))$
- $fnp\_neg(\exists x.F) = \forall x.(fnp\_neg(F))$
- $fnp\_neg(F) = \neg F$ si F est sans quantificateur

et F' est en FNP, alors $fnp\_neg(F')$ est en FNP et $\neg F \equiv fnp\_neg(F')$

Exercices:

- Définissez *fnp_bin* avec:
  Si F' et G' sont en FNP et $*$ est un connecteur $\wedge, \vee$ ou $\longrightarrow$,
  alors $fnp\_bin(F', *, G')$ est en FNP et $F' * G' \equiv fnp\_bin(F', *, G')$
- Définissez la fonction *fnp* telle que $fnp(F)$ est en FNP et $F \equiv fnp(F)$, pour toute formule F.
- Convertissez les “contre-exemples” d’en haut en FNP.

Plan

Déduction Naturelle pour LPred

Extension de la déduction naturelle pour LProp:

- Les règles pour LProp restent en vigueur
- Nouvelles règles pour les quantificateurs: $(I\exists), (E\exists), (I\forall), (E\forall)$
- Construction d'un arbre de dérivation ... comme pour LProp

Exemple d'une preuve "naturelle" de $\forall x.\exists y.x < y$

1. Soit x une valeur arbitraire (mais fixe), pour laquelle il faut prouver $\exists y.x < y$.
2. Pour montrer $\exists y.x < y$, il faut trouver un y qui satisfait $x < y$ (et qui peut dépendre de x). On choisit $x + 1$ pour y.
3. On vérifie que $x < x + 1$ est satisfait.

Règles: $(I\exists)$

Notation explicite

$$\frac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x.A}\ (I\exists)$$

Notation allégée

$$\frac{A[t/x]}{\exists x.A}\ (I\exists)$$

Condition: $[t/x]$ est une substitution saine.

Lecture informelle:
A est vrai pour un t. Donc, il existe un x pour lequel A est vrai.

Règles: $(E\forall)$

Notation explicite

$$\frac{\Gamma \vdash \forall x.A}{\Gamma \vdash A[t/x]}\ (E\forall)$$

Notation allégée

$$\frac{\forall x.A}{A[t/x]}\ (E\forall)$$

Condition: $[t/x]$ est une substitution saine.

Lecture informelle:
A est vrai pour tout x. Donc, il est vrai en particulier pour un t (que je peux choisir).

Règles: ($I\forall$) (1)

Notation explicite

$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x.A}\ (I\forall)$$

Condition: $x \notin vars(\Gamma)$

Notation allégée

$$\frac{A}{\forall x.A}\ (I\forall)$$

Condition: x n'est pas libre dans les hypothèses de la prémisse.

Lecture informelle:

- Si A est vrai pour n'importe quel x, alors aussi $\forall x.A$ est vrai.
- *Condition:* ne pas découpler les variables qui se réfèrent au même objet

Règles: ($I\forall$) (2)

Exemple d'une preuve incorrecte (!) qui ne respecte pas la condition:

$$
\dfrac{\dfrac{\dfrac{\dfrac{x = 0 \vdash x = 0}{x = 0 \vdash \forall x.x = 0}\;(I\forall)}{\vdash x = 0 \longrightarrow (\forall x.x = 0)}\;(I \longrightarrow)}{\vdash \forall x.(x = 0 \longrightarrow (\forall x.x = 0))}\;(I\forall)}{\vdash 0 = 0 \longrightarrow (\forall x.x = 0)}\;(E\forall)
$$

Règles: $(E\exists)$ (1)

Notation explicite

$$\frac{\Gamma \vdash \exists x.A \qquad \Gamma, A \vdash C}{\Gamma \vdash C}\ (E\exists)$$

Condition: $x \notin vars(\Gamma) \cup vars(C)$

Notation allégée

$$\frac{\exists x.A \qquad \begin{matrix} (i)A \\ \vdots \\ C \end{matrix}}{C}\ (E\exists)(i)$$

Condition: x n'est libre ni dans les hypothèses de C, ni dans C

Lecture informelle: Pour montrer C, sachant que $\exists x.A$, il suffit de postuler A (pour un x dont on ne connaît pas l'identité exacte) et de montrer C.

Règles: $(E\exists)$ (2)

Exemple d'une preuve:

$$\dfrac{\mathcal{E} \vdash \mathcal{E} \qquad \dfrac{\dfrac{\mathcal{E}, (P(x) \wedge Q(x)) \vdash P(x) \wedge Q(x)}{\mathcal{E}, (P(x) \wedge Q(x)) \vdash P(x)}\,(E\wedge_1)}{\mathcal{E}, (P(x) \wedge Q(x)) \vdash \exists x.P(x)}\,(I\exists)}{\dfrac{\exists x.(P(x) \wedge Q(x)) \vdash \exists x.P(x)}{\vdash (\exists x.(P(x) \wedge Q(x))) \longrightarrow (\exists x.P(x))}\,(I \longrightarrow)}\,(E\exists)$$

Ici, $\mathcal{E}$ abbrévie $\exists x.(P(x) \wedge Q(x))$

Règles: $(E\exists)$ (3)

Exemple d'une preuve *incorrecte*(!):

$$\cfrac{\cfrac{\cfrac{\exists x.P(x) \vdash \exists x.P(x) \quad \exists x.P(x), P(x) \vdash P(x)}{\exists x.P(x) \vdash P(x)}\,(E\exists)}{\exists x.P(x) \vdash \forall x.P(x)}\,(I\forall)}{\vdash (\exists x.P(x)) \longrightarrow (\forall x.P(x))}\,(I \longrightarrow)$$

Discussion:

- L'application de $(I\forall)$ est correcte: x n'apparaît pas *libre* dans l'hypothèse
- L'application de $(E\exists)$ est incorrecte: x apparaît libre dans la conclusion

Égalité (1)

Problématique: *à priori*, rien n'est connu des symboles relationnels:

- $2 \leq 3$ $(2,3) \leq (3,4)$??? $(2,3) \leq (3,2)$???
- `"majuscule" < "minuscule"` `"MOT" < "mot"` ???

L'égalité de deux termes dépend souvent de l'interprétation des opérateurs:

- 2 + 3 = 3 + 2 $a(2,3) = a(3,2)$???
- $2 \oplus 5 = 3 \oplus 1$??? $\rightsquigarrow$ OK pour addition modulo 3

Ici: Notion la plus stricte possible:
"égal" signifie: "syntaxiquement identique"

- $a(3,2) = a(3,2)$
- mais: $\neg(a(2,3) = a(3,2))$
- donc aussi: $\neg(2 + 3 = 3 + 2)$

Égalité (2)

Règle d'introduction:

$$\frac{}{t = t} \ (I =)$$

Règle d'élimination:

$$\frac{t_1 = t_2 \quad P[t_1/x]}{P[t_2/x]} \ (E =)$$

Définition de l'égalité de Leibniz:

- Deux objets sont égaux s'ils sont indistinguibles (par rapport à toute propriété P)

Exercices: Montrez: l'égalité est

- symétrique: $t_1 = t_2 \longrightarrow t_2 = t_1$
- transitive: $t_1 = t_2 \longrightarrow t_2 = t_3 \longrightarrow t_1 = t_3$

Égalité (3)

Deux preuves ...

Correct:

$$\cfrac{\cfrac{\overline{x = x}\ (I =)}{\exists y. x = y}\ (I\exists)}{\forall x. \exists y. x = y}\ (I\forall)$$

Incorrect:

$$\cfrac{\cfrac{\overline{x = x}\ (I =)}{\forall x. x = x}\ (I\forall)}{\exists y. \forall x. x = y}\ (I\exists)$$

Trouvez l'erreur!

Décidabilité (1)

Quels *problèmes* peuvent être résolus avec un *algorithme*?

Problème:

- Une question qui admet une réponse “vrai” ou “faux”
- *Alternativement:* un ensemble S, avec la question: “Est-ce que $x \in S$, pour n'importe quel x donné?”

Algorithme:

- un programme en Caml
  - qui termine toujours
  - et fournit une réponse “vrai” ou “faux”
- ou en C, Java, Assembleur, ...
  Tous les langages de programmation sont essentiellement équivalents (*thèse de Church*)

Un problème est décidable s'il existe un algorithme qui le résout.

Décidabilité (2)

Exemples de problèmes décidables:

- Nombres pairs:
  - *Problème:* est-ce que x appartient à l'ensemble des nombres pairs?
  - *Algorithme* qui le résout:

    ```
    let pair (x) = (x mod 2 = 0);;
    ```
- Nombres premiers:
  - *Problème:* est-ce que x appartient à l'ensemble des nombres premiers?
  - *Algorithme* qui le résout:

    ```
    let rec test_div (d, x) =
      (d = 1) or (x mod d <> 0) && test_div (d - 1, x)
    let premier (x) = (x > 1) && test_div (x - 1, x)
    ```

Décidabilité (3)

Correspondances

- *Donné:* Une liste (finie) de couples de mots binaires $C = [(x_1, y_1); \ldots; (x_k, y_k)]$, avec x_i, y_i des mots binaires
- Une *correspondance* pour C est une séquence finie d'indices $i_1, \ldots i_n \in \{1 \ldots k\}$, avec $x_{i_1} \cdot x_{i_2} \ldots x_{i_n} = y_{i_1} \cdot y_{i_2} \ldots y_{i_n}$

Exemple: $[(1, 101); (10, 00); (011, 11)]$ a la correspondance $1, 3, 2, 3$:
$1 \cdot 011 \cdot 10 \cdot 011 = 101 \cdot 11 \cdot 00 \cdot 11$
Exercice: Corresp. pour $[(001, 0); (01, 011); (01, 101); (10, 001)]$
(Solution la plus courte: séquence de 66 indices)

Problème de correspondance de Post:

- Pour n'importe quel C, est-ce que C a une correspondance?

Théorème (Post): Le problème de correspondance est indécidable.

Décidabilité (4)

Problème de validité de LPred:

- Pour une formule F de la logique des prédicats, est-ce que F est valide?

Théorème (Church): Le problème de validité de LPred est indécidable.

Principe de preuve:
Réduction au problème de correspondance de Post:

- Codage d'une instance C du problème de Post par une formule F_C, telle que C a une solution si et seulement si F_C est valide.
- Si on pouvait déterminer la validité de toute formule:
  - alors aussi la validité de tout F_C
  - donc aussi résoudre toute instance C du problème de Post. *Contradiction.*

Décidabilité (5)

Construction de F_C pour un $C = [(x_1, y_1); \ldots ; (x_k, y_k)]$

- F_C a la forme $(F_1 \wedge F_2) \longrightarrow F_3$
- F_1 code des séquences initiales non-vides:
  $F_1 = P(\overline{x_1}(a), \overline{y_1}(a)) \wedge \ldots \wedge P(\overline{x_k}(a), \overline{y_k}(a))$
  Ici:
    - a représente une séquence vide
    - $\overline{x_1}(u)$ rajoute des symboles de fonction f_0, f_1 à la séquence u
      Exemple: $\overline{001}(u) = f_1(f_0(f_0(u)))$
- F_2 code la concaténation:
  $F_2 = \forall uv.(P(u, v) \longrightarrow P(\overline{x_1}(u), \overline{y_1}(v)) \wedge \ldots \wedge P(\overline{x_k}(u), \overline{y_k}(v)))$
- F_3 code l'existence de la correspondance: $\exists z.P(z, z)$