Outils Logiques

Antonio Bucciarelli et Delia Kesner
buccia@irif.fr,kesner@irif.fr
(cours original de Ralf Treinen)

Université Paris Diderot
UFR Informatique
Institut de Recherche en Informatique Fondamentale

September 8, 2017

Organisation

- La page du cours:

  `http://www.irif.fr/~kesner/enseignement/ol3/`

  contient les transparents du cours, les feuilles de TD, les dates d'examen et les consignes.... à consulter régulièrement.
- 13 semaines de cours-td. Aujourd'hui : Introduction
- Un poly vous sera fourni (mais attention : tout n'est pas dans le poly. La participation active au cours-TD est essentiel).

Organisation

- La page du cours:

  `http://www.irif.fr/~kesner/enseignement/ol3/`

  contient les transparents du cours, les feuilles de TD, les dates d'examen et les consignes.... à consulter régulièrement.
- 13 semaines de cours-td. Aujourd'hui : Introduction
- Un poly vous sera fourni (mais attention : tout n'est pas dans le poly. La participation active au cours-TD est essentiel).

Organisation

- La page du cours:

  `http://www.irif.fr/~kesner/enseignement/ol3/`

  contient les transparents du cours, les feuilles de TD, les dates d'examen et les consignes.... à consulter régulièrement.
- 13 semaines de cours-td. Aujourd'hui : Introduction
- Un poly vous sera fourni (mais attention : tout n'est pas dans le poly. La participation active au cours-TD est essentiel).

Modalités de contrôle de connaissances

- Partiel obligatoire, le samedi 4 novembre.
  A noter dès maintenant dans vos agendas!
- Note de première session =
  $\frac{1}{2}$ Note Examen de première session + $\frac{1}{2}$ Note Partiel
- Note de deuxième session = Note Examen de deuxième session

Modalités de contrôle de connaissances

- Partiel obligatoire, le samedi 4 novembre.
  A noter dès maintenant dans vos agendas!
- Note de première session =
  $\frac{1}{2}$ Note Examen de première session + $\frac{1}{2}$ Note Partiel
- Note de deuxième session = Note Examen de deuxième session

Modalités de contrôle de connaissances

- Partiel obligatoire, le samedi 4 novembre.
  A noter dès maintenant dans vos agendas!
- Note de première session =
  $\frac{1}{2}$ Note Examen de première session + $\frac{1}{2}$ Note Partiel
- Note de deuxième session = Note Examen de deuxième session

Définition ?

> *L'étude des règles formelles que doit respecter toute déduction correcte.*

(trouvé sur Wikipédia)

- Qu'est-ce que cela veut dire ?
- À quoi est-ce que cela sert en Informatique ?

Définition ?

> *L'étude des règles formelles que doit respecter toute déduction correcte.*

(trouvé sur Wikipédia)

- Qu'est-ce que cela veut dire ?
- À quoi est-ce que cela sert en Informatique ?

Définition ?

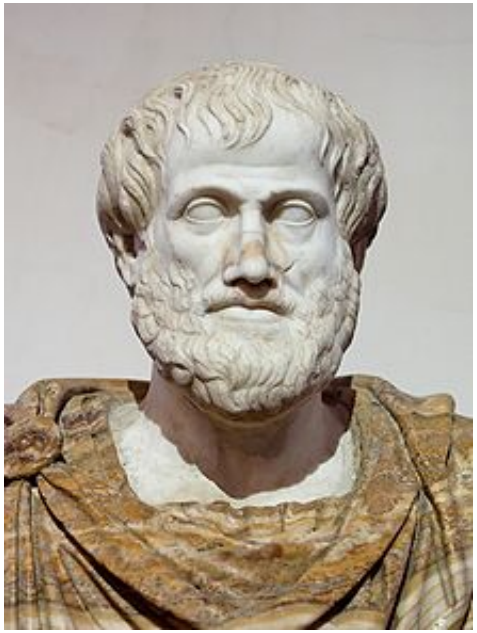
> *L'étude des règles formelles que doit respecter toute déduction correcte.*

(trouvé sur Wikipédia)

- Qu'est-ce que cela veut dire ?
- À quoi est-ce que cela sert en Informatique ?

Une très brève histoire de la Logique 1

Depuis l'antiquité : La logique philosophique a pour but de comprendre les règles du raisonnement humain.



Aristote

Une très brève histoire de la Logique 2

Depuis le 19-ème siècle : Développement de la logique mathématique qui cherche à établir des nouveaux fondements des Mathématiques.

Frege

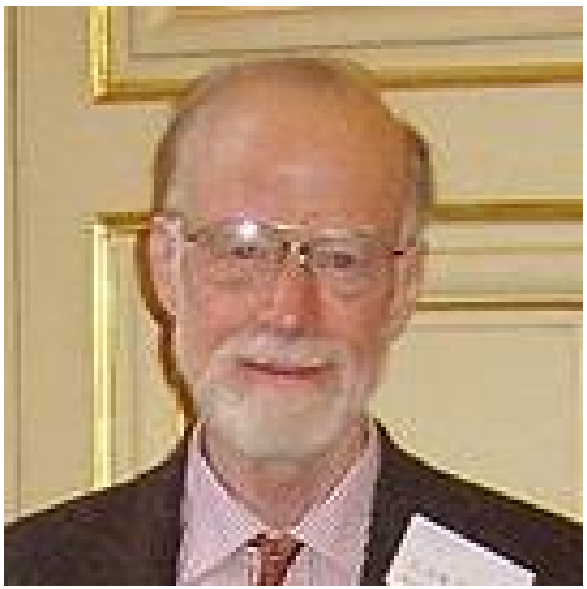
Hilbert

Russel

Une très brève histoire de la Logique 3

Depuis le 20-ème siècle : la logique contemporaine s'intéresse à la logique comme fondement de l'informatique, à l'intelligence Artificielle, et aux Preuves Automatiques.

Gödel



Hoare

Sifakis

Systèmes Logiques

Un système logique consiste en trois parties :

1. Syntaxe: la forme des énoncés.
2. Sémantique: le sens des énoncés.
3. Inférence: les règles qui permettent de prouver, ou de refuter, un énoncé.

Dans ce cours nous étudions le Calcul Propositionnel, mais vous verrez d'autres systèmes logiques pendant vos études.

Systèmes Logiques

Un système logique consiste en trois parties :

1. Syntaxe: la forme des énoncés.
2. Sémantique: le sens des énoncés.
3. Inférence: les règles qui permettent de prouver, ou de refuter, un énoncé.

Dans ce cours nous étudions le Calcul Propositionnel, mais vous verrez d'autres systèmes logiques pendant vos études.

Systèmes Logiques

Un système logique consiste en trois parties :

1. Syntaxe: la forme des énoncés.
2. Sémantique: le sens des énoncés.
3. Inférence: les règles qui permettent de prouver, ou de refuter, un énoncé.

Dans ce cours nous étudions le Calcul Propositionnel, mais vous verrez d'autres systèmes logiques pendant vos études.

Systèmes Logiques

Un système logique consiste en trois parties :

1. Syntaxe: la forme des énoncés.
2. Sémantique: le sens des énoncés.
3. Inférence: les règles qui permettent de prouver, ou de refuter, un énoncé.

Dans ce cours nous étudions le Calcul Propositionnel, mais vous verrez d'autres systèmes logiques pendant vos études.

La Syntaxe

- Définition formelle (c'est-à-dire, mathématique) de l'ensemble des énoncés qui sont considérés dans un système logique donné.
- Les énoncés sont normalement exprimés dans un langage symbolique.
- On appelle souvent de tels énoncés des formules.

La Syntaxe

- Définition formelle (c'est-à-dire, mathématique) de l'ensemble des énoncés qui sont considérés dans un système logique donné.
- Les énoncés sont normalement exprimés dans un langage symbolique.
- On appelle souvent de tels énoncés des formules.

La Syntaxe

- Définition formelle (c'est-à-dire, mathématique) de l'ensemble des énoncés qui sont considérés dans un système logique donné.
- Les énoncés sont normalement exprimés dans un langage symbolique.
- On appelle souvent de tels énoncés des formules.

Exemple Syntaxe 1

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

- formule de la logique propositionnelle.
- Les formules de cette logique sont composées de variables propositionnelles, ici x, y et z, et d'opérateurs logiques, ici $\wedge$ et $\neg$.
- Cette logique est pertinente pour décrire le comportement d'un circuit, par exemple.

Exemple Syntaxe 1

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

- formule de la logique propositionnelle.
- Les formules de cette logique sont composées de variables propositionnelles, ici x, y et z, et d'opérateurs logiques, ici $\wedge$ et $\neg$.
- Cette logique est pertinente pour décrire le comportement d'un circuit, par exemple.

Exemple Syntaxe 1

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

- formule de la logique propositionnelle.
- Les formules de cette logique sont composées de variables propositionnelles, ici x, y et z, et d'opérateurs logiques, ici $\wedge$ et $\neg$.
- Cette logique est pertinente pour décrire le comportement d'un circuit, par exemple.

Exemple Syntaxe 2

$$\neg(x = 0) \wedge x * y = 0$$

- formule de la logique du premier ordre.
- Cette logique sera étudiée dans le cours Logique du L3.
- Cette logique est utile pour la description du comportement des programmes.

Exemple Syntaxe 2

$$\neg(x = 0) \wedge x * y = 0$$

- formule de la logique du premier ordre.
- Cette logique sera étudiée dans le cours Logique du L3.
- Cette logique est utile pour la description du comportement des programmes.

Exemple Syntaxe 2

$$\neg(x = 0) \wedge x * y = 0$$

- formule de la logique du premier ordre.
- Cette logique sera étudiée dans le cours Logique du L3.
- Cette logique est utile pour la description du comportement des programmes.

Définir une syntaxe

- La syntaxe doit être définie rigoureusement.
- La définition ne doit laisser aucun doute sur ce qui est une formule, et ce qui ne l'est pas.
- Normalement la syntaxe est définie par une définition inductive (voir la suite du cours) ou par une grammaire.
- Une définition rigoureuse permet d'obtenir un programme qui peut analyser la structure d'un formule.

Définir une syntaxe

- La syntaxe doit être définie rigoureusement.
- La définition ne doit laisser aucun doute sur ce qui est une formule, et ce qui ne l'est pas.
- Normalement la syntaxe est définie par une définition inductive (voir la suite du cours) ou par une grammaire.
- Une définition rigoureuse permet d'obtenir un programme qui peut analyser la structure d'un formule.

Définir une syntaxe

- La syntaxe doit être définie rigoureusement.
- La définition ne doit laisser aucun doute sur ce qui est une formule, et ce qui ne l'est pas.
- Normalement la syntaxe est définie par une définition inductive (voir la suite du cours) ou par une grammaire.
- Une définition rigoureuse permet d'obtenir un programme qui peut analyser la structure d'un formule.

Définir une syntaxe

- La syntaxe doit être définie rigoureusement.
- La définition ne doit laisser aucun doute sur ce qui est une formule, et ce qui ne l'est pas.
- Normalement la syntaxe est définie par une définition inductive (voir la suite du cours) ou par une grammaire.
- Une définition rigoureuse permet d'obtenir un programme qui peut analyser la structure d'un formule.

La sémantique

- La sémantique associe une valeur de vérité (vrai ou faux, 1 ou 0) à chaque formule.
- Elle est donnée, au moins pour les systèmes logiques qui nous intéressent ici, par une définition rigoureuse qui fait bien sûr référence à la définition formelle de la syntaxe.
- Définition de la sémantique par récurrence sur la syntaxe (voir la suite du cours).

La sémantique

- La sémantique associe une valeur de vérité (vrai ou faux, 1 ou 0) à chaque formule.
- Elle est donnée, au moins pour les systèmes logiques qui nous intéressent ici, par une définition rigoureuse qui fait bien sûr référence à la définition formelle de la syntaxe.
- Définition de la sémantique par récurrence sur la syntaxe (voir la suite du cours).

La sémantique

- La sémantique associe une valeur de vérité (vrai ou faux, 1 ou 0) à chaque formule.
- Elle est donnée, au moins pour les systèmes logiques qui nous intéressent ici, par une définition rigoureuse qui fait bien sûr référence à la définition formelle de la syntaxe.
- Définition de la sémantique par récurrence sur la syntaxe (voir la suite du cours).

Interprétation dans un contexte

- Pour la plupart des systèmes logiques, la valeur de vérité d'une formule dépend d'un contexte.
- Dans le cas de la logique propositionnelle, un contexte associe à chaque variable propositionnelle une des deux valeurs de vérité vrai (1) ou faux (0).

Interprétation dans un contexte

- Pour la plupart des systèmes logiques, la valeur de vérité d'une formule dépend d'un contexte.
- Dans le cas de la logique propositionnelle, un contexte associe à chaque variable propositionnelle une des deux valeurs de vérité vrai (1) ou faux (0).

Exemple sémantique

Une fois syntaxe et sémantique de la logique définis, on aura par exemple que

- La formule $\neg p \wedge r$ est vraie dans tous les contextes qui associent à p la valeur faux, et à r la valeur vrai.
- La formule $p \vee \neg p$ est vraie dans tous les contextes. On dira que cette formule est valide.

Exemple sémantique

Une fois syntaxe et sémantique de la logique définis, on aura par exemple que

- La formule $\neg p \wedge r$ est vraie dans tous les contextes qui associent à p la valeur faux, et à r la valeur vrai.
- La formule $p \vee \neg p$ est vraie dans tous les contextes. On dira que cette formule est valide.

Inférence

- Règles formelles de raisonnement.
- Les règles d'inférence permettent de déduire la véracité d'une formule (la conclusion de la règle) du fait que certaines autres fromules (les hypothèses de la règle) sont vraies.
- Plus généralement, il s'agit de déterminer (ou faire calculer par un programme) si une formule est vraie ou pas, ou encore si elle valide ou pas.

Inférence

- Règles formelles de raisonnement.
- Les règles d'inférence permettent de déduire la véracité d'une formule (la conclusion de la règle) du fait que certaines autres fromules (les hypothèses de la règle) sont vraies.
- Plus généralement, il s'agit de déterminer (ou faire calculer par un programme) si une formule est vraie ou pas, ou encore si elle valide ou pas.

Inférence

- Règles formelles de raisonnement.
- Les règles d'inférence permettent de déduire la véracité d'une formule (la conclusion de la règle) du fait que certaines autres fromules (les hypothèses de la règle) sont vraies.
- Plus généralement, il s'agit de déterminer (ou faire calculer par un programme) si une formule est vraie ou pas, ou encore si elle valide ou pas.

Inférence dans ce cours

- Logique propositionnelle : nous présenterons deux algorithmes qui permettent de dire si une formule est valide ou pas (algorithme DPLL, la base des SAT-solveurs et Calcul des Séquents).

Attention : simplification !

- Cette présentation de la logique (dite logique classique) n'est pas unique.
- En particulier, la façon de définir la sémantique peut être contestée. Une école importante de la logique moderne (l'école intuitionniste ou constructiviste) n'admet pas le principe du tiers exclu (qui affirme que pour toute formule p, la formule $(p \vee \neg p)$ est valide).

Attention : simplification !

- Cette présentation de la logique (dite logique classique) n'est pas unique.
- En particulier, la façon de définir la sémantique peut être contestée. Une école importante de la logique moderne (l'école intuitionniste ou constructiviste) n'admet pas le principe du tiers exclu (qui affirme que pour toute formule p, la formule $(p \vee \neg p)$ est valide).

Expressions Booléennes

Existent dans tous les langages de programmation :

Formule logique	Expression en Java
$(x \wedge (y \wedge \neg z))$	`x && (y && (!z))`

Les expressions booléennes sont par exemple utilisées comme gardes dans des instructions conditionnelles ou des boucles.

Questions intéressantes

- Est-ce qu'on peut remplacer une formule par une formule équivalente plus simple ? Cela permettra à un compilateur de simplifier la formule avant d'engendrer le code, et ainsi d'optimiser le temps de calcul.
- Est-ce qu'une formule donnée est toujours vraie ou toujours fausse ? Cela permettra à un compilateur d'omettre un test inutile, et ainsi d'optimiser le temps de calcul et aussi la taille du programme.

Questions intéressantes

- Est-ce qu'on peut remplacer une formule par une formule équivalente plus simple ? Cela permettra à un compilateur de simplifier la formule avant d'engendrer le code, et ainsi d'optimiser le temps de calcul.
- Est-ce qu'une formule donnée est toujours vraie ou toujours fausse ? Cela permettra à un compilateur d'omettre un test inutile, et ainsi d'optimiser le temps de calcul et aussi la taille du programme.

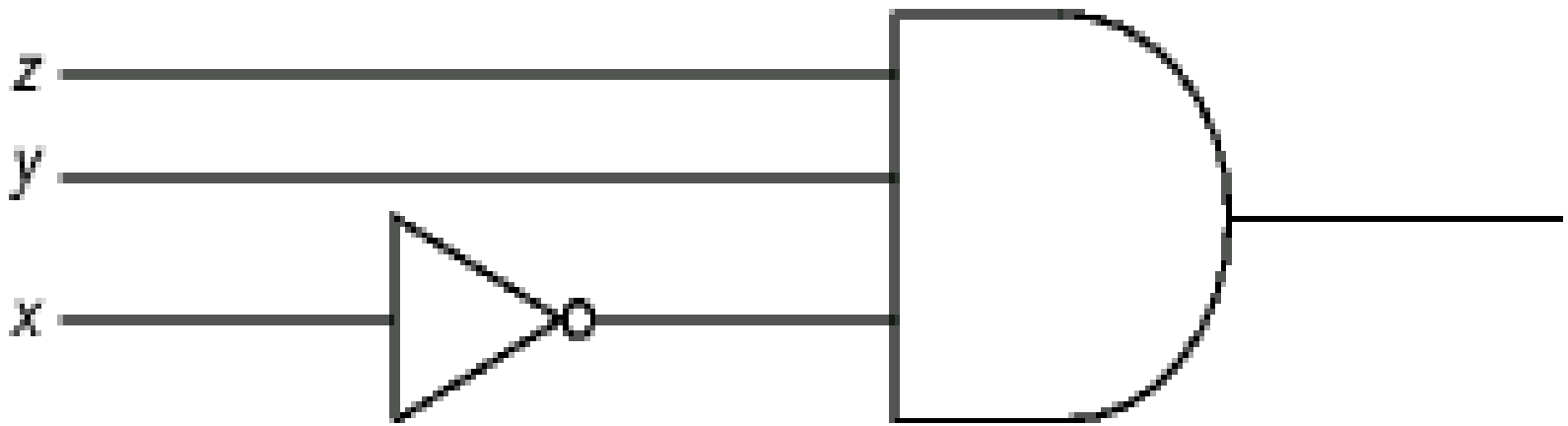
Circuits

Les circuits sont une réalisation matérielle des formules de la logique propositionnelle, dans le sens que

- les variables propositionnelles correspondent aux entrées d'un circuit,
- les opérateurs logiques correspondent aux portes logiques,
- et la valeur de vérité de la formule correspond à la sortie du circuit.

Exemple Circuit

La formule $(z \wedge (y \wedge \neg x))$ correspond au circuit suivant :



Questions intéressantes

- Construire un circuit (c'est-à-dire, une formule logique) qui réalise une fonction donnée (par exemple un additionneur).
- Déterminer la fonctionnalité d'un circuit donné.
- Est-ce que deux circuits donnés sont équivalents ?
- Minimiser un circuit : étant donné un circuit, construire le plus petit circuit avec la même fonctionnalité.
- Cours Circuits et Architecture en M1

Questions intéressantes

- Construire un circuit (c'est-à-dire, une formule logique) qui réalise une fonction donnée (par exemple un additionneur).
- Déterminer la fonctionnalité d'un circuit donné.
- Est-ce que deux circuits donnés sont équivalents ?
- Minimiser un circuit : étant donné un circuit, construire le plus petit circuit avec la même fonctionnalité.
- Cours Circuits et Architecture en M1

Questions intéressantes

- Construire un circuit (c'est-à-dire, une formule logique) qui réalise une fonction donnée (par exemple un additionneur).
- Déterminer la fonctionnalité d'un circuit donné.
- Est-ce que deux circuits donnés sont équivalents ?
- Minimiser un circuit : étant donné un circuit, construire le plus petit circuit avec la même fonctionnalité.
- Cours Circuits et Architecture en M1

Questions intéressantes

- Construire un circuit (c'est-à-dire, une formule logique) qui réalise une fonction donnée (par exemple un additionneur).
- Déterminer la fonctionnalité d'un circuit donné.
- Est-ce que deux circuits donnés sont équivalents ?
- Minimiser un circuit : étant donné un circuit, construire le plus petit circuit avec la même fonctionnalité.
- Cours Circuits et Architecture en M1

Questions intéressantes

- Construire un circuit (c'est-à-dire, une formule logique) qui réalise une fonction donnée (par exemple un additionneur).
- Déterminer la fonctionnalité d'un circuit donné.
- Est-ce que deux circuits donnés sont équivalents ?
- Minimiser un circuit : étant donné un circuit, construire le plus petit circuit avec la même fonctionnalité.
- Cours Circuits et Architecture en M1

Vérification et développement de programmes

```
z := 0;
i := 0;
  while i ≠ y do
  z := z + x;
  i := i + 1;
done
```

Nous prétendons qu'à la fin de l'exécution de ce morceau de programme, la valeur de la variable z est $y * x$.

Vérification et développement de programmes

```
z := 0;
i := 0;
while i ≠ y do
  { z = i * x }
  z := z + x;
  i := i + 1;
done
{ z = y * x }
```

La formule $z = i * x$ est un invariant du programme.

Question intéressantes

- Comment trouver les bonnes assertions pour un programme donné ?
- Étant donné un programme annoté avec des assertions, comment vérifier que les assertions sont vraies ?

Preuves automatiques de programmes

- Extrêmement intéressantes pour des applications critiques (par exemple, système de pilotage d'un avion, ou contrôle d'une centrale nucléaire).
- Développement commencé à la fin des années 60, devient aujourd'hui utilisable pour des grands systèmes, même des compilateurs ou des systèmes d'exploitation.
- Plusieurs cours de vérification dans nos parcours M1/M2.

Preuves automatiques de programmes

- Extrêmement intéressantes pour des applications critiques (par exemple, système de pilotage d'un avion, ou contrôle d'une centrale nucléaire).
- Développement commencé à la fin des années 60, devient aujourd'hui utilisable pour des grands systèmes, même des compilateurs ou des systèmes d'exploitation.
- Plusieurs cours de vérification dans nos parcours M1/M2.

Preuves automatiques de programmes

- Extrêmement intéressantes pour des applications critiques (par exemple, système de pilotage d'un avion, ou contrôle d'une centrale nucléaire).
- Développement commencé à la fin des années 60, devient aujourd'hui utilisable pour des grands systèmes, même des compilateurs ou des systèmes d'exploitation.
- Plusieurs cours de vérification dans nos parcours M1/M2.

Bases de données

- Indispensable pour des systèmes d'informations actuels.
- Bases de données relationnels utilisent le langage SQL :

```
SELECT nom FROM persons WHERE age > 18 AND
    ( children > 0 OR married = yes)
```

- revient à évaluer la formule logique

$$age > 18 \wedge (children > 0 \vee married = yes)$$

sur tous les éléments de la table avec le nom *persons*.

Bases de données

- Indispensable pour des systèmes d'informations actuels.
- Bases de données relationnels utilisent le langage SQL :

```
SELECT nom FROM persons WHERE age > 18 AND
    ( children > 0 OR married = yes)
```

- revient à évaluer la formule logique

$$age > 18 \wedge (children > 0 \vee married = yes)$$

sur tous les éléments de la table avec le nom *persons*.

Bases de données

- Indispensable pour des systèmes d'informations actuels.
- Bases de données relationnels utilisent le langage SQL :

```
SELECT nom FROM persons WHERE age > 18 AND
    ( children > 0 OR married = yes)
```

- revient à évaluer la formule logique

$$age > 18 \wedge (children > 0 \vee married = yes)$$

sur tous les éléments de la table avec le nom *persons*.

Questions intéressantes

- Comment évaluer efficacement une requête (formule logique) ?
- Comment transformer une requête pour rendre son évaluation plus efficace ? Cela est une question importante pour des bases de données de très grande taille.
- Cours Bases de Données du L3, M1, M2.

Questions intéressantes

- Comment évaluer efficacement une requête (formule logique) ?
- Comment transformer une requête pour rendre son évaluation plus efficace ? Cela est une question importante pour des bases de données de très grande taille.
- Cours Bases de Données du L3, M1, M2.

Questions intéressantes

- Comment évaluer efficacement une requête (formule logique) ?
- Comment transformer une requête pour rendre son évaluation plus efficace ? Cela est une question importante pour des bases de données de très grande taille.
- Cours Bases de Données du L3, M1, M2.

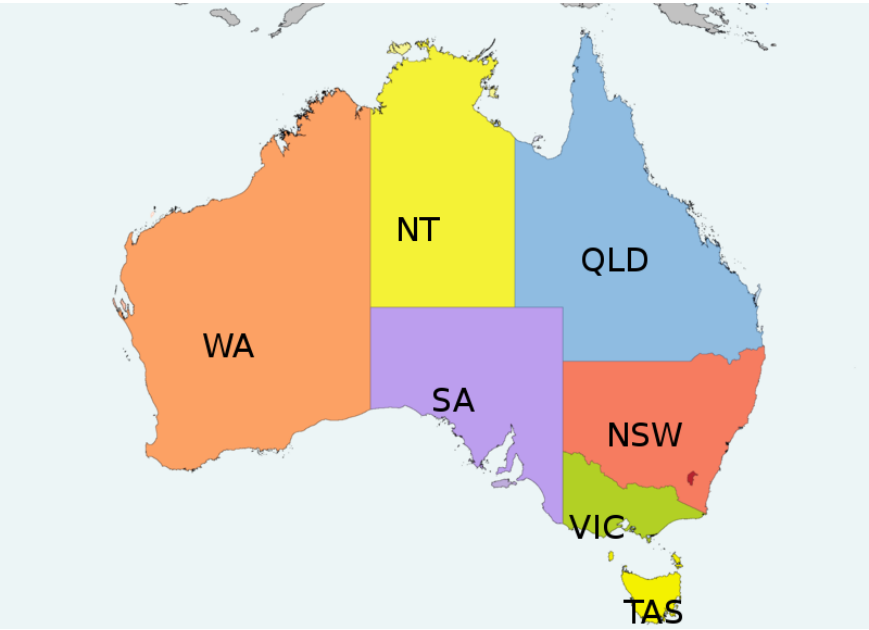
Problèmes de satisfaction de contraintes

- Problème donné par un domaine fini par variable (un ensemble fini de valeurs possibles), et des contraintes (conditions) que toute solution doit satisfaire.
- Trouver une solutions qui satisfait toutes les contraintes.

Problèmes de satisfaction de contraintes

- Problème donné par un domaine fini par variable (un ensemble fini de valeurs possibles), et des contraintes (conditions) que toute solution doit satisfaire.
- Trouver une solutions qui satisfait toutes les contraintes.

Exemple : colorer une carte



Colorier avec 3 couleurs, de telle manière que deux états adjacents aient des couleurs différentes.

Exemple : résoudre un Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Autre exemples

- Trouver un emploi du temps
- Découper une surface
- Planifier des tâches
- Trouver une installation optimale de paquets Linux
- ...

Autre exemples

- Trouver un emploi du temps
- Découper une surface
- Planifier des tâches
- Trouver une installation optimale de paquets Linux
- ...

Autre exemples

- Trouver un emploi du temps
- Découper une surface
- Planifier des tâches
- Trouver une installation optimale de paquets Linux
- ...

Autre exemples

- Trouver un emploi du temps
- Découper une surface
- Planifier des tâches
- Trouver une installation optimale de paquets Linux
- . . .

Autre exemples

- Trouver un emploi du temps
- Découper une surface
- Planifier des tâches
- Trouver une installation optimale de paquets Linux
- ...

Encore plus d'applications

- Programmation Logique ($\rightarrow$ cours M1, M2)
- Assistants de preuves ($\rightarrow$ cours M1, M2)
- Démonstration Automatique ($\rightarrow$ cours M1, MPRI)
- Intelligence Artificielle ($\rightarrow$ cours M1)
- Théorie de types, complexité descriptive (cours MPRI)

Encore plus d'applications

- Programmation Logique ($\rightarrow$ cours M1, M2)
- Assistants de preuves ($\rightarrow$ cours M1, M2)
- Démonstration Automatique ($\rightarrow$ cours M1, MPRI)
- Intelligence Artificielle ($\rightarrow$ cours M1)
- Théorie de types, complexité descriptive (cours MPRI)

Encore plus d'applications

- Programmation Logique ($\rightarrow$ cours M1, M2)
- Assistants de preuves ($\rightarrow$ cours M1, M2)
- Démonstration Automatique ($\rightarrow$ cours M1, MPRI)
- Intelligence Artificielle ($\rightarrow$ cours M1)
- Théorie de types, complexité descriptive (cours MPRI)

Encore plus d'applications

- Programmation Logique ($\rightarrow$ cours M1, M2)
- Assistants de preuves ($\rightarrow$ cours M1, M2)
- Démonstration Automatique ($\rightarrow$ cours M1, MPRI)
- Intelligence Artificielle ($\rightarrow$ cours M1)
- Théorie de types, complexité descriptive (cours MPRI)