UNIVERSITÉ TOULOUSE CAPITOLE
FACULTÉ D'INFORMATIQUE

IRIT

Intelligence Artificielle

Logique, SAT et Programmation par Contraintes

Jean-Guy Mailly (FI), Leila Moudjari (FA)

Master 1 MIAGE – 2025-2026

TD: Audren Boulic-Bouadjio, Leila Moudjari, Paul Saves

UNIVERSITÉ TOULOUSE CAPITOLE
FACULTÉ D'INFORMATIQUE

IRIT

人工智能

逻辑、SAT与约束规划

Jean-Guy Mailly（法国），Leila Moudjari（阿尔及利亚）

MIAGE硕士1年级 - 2025-2026学年

TD：奥德伦·布利克-布阿吉奥、莱拉·穆贾里、保罗·塞夫斯

Introduction

导言

Espaces d'états vs Espaces de variables

Modélisation par espaces d'états

- Avantages : modélisation plus naturelle et possibilité de calculer des séquences d'actions
- Inconvénients : il faut considérer un nombre d'états possible très grand (exponentiel par rapport aux nombre de $\ll$ composants $\gg$ dans les états)

Modélisation des états avec un ensemble de variables

- Inconvénient : modélisation potentiellement plus complexe
- Avantages : pas besoin d'énumérer les états, possibilité d'utiliser des algorithmes très efficaces (solveurs) pour résoudre des gros problèmes rapidement

状态空间与变量空间

状态空间建模

- 优势：更自然的建模方式及动作序列计算能力
- 缺点：需要考虑可能的状态数量非常大（相对于状态中的 组件数 呈指数级增长）

基于变量集合的状态建模

- 缺点：建模可能更为复杂
- 优势：无需逐项枚举状态，可借助高效求解器快速处理复杂问题

Espaces d'états vs Espaces de variables

Modélisation par espaces d'états

- Avantages : modélisation plus naturelle et possibilité de calculer des séquences d'actions
- Inconvénients : il faut considérer un nombre d'états possible très grand (exponentiel par rapport aux nombre de $\ll$ composants $\gg$ dans les états)

Modélisation des états avec un ensemble de variables

- Inconvénient : modélisation potentiellement plus complexe
- Avantages : pas besoin d'énumérer les états, possibilité d'utiliser des algorithmes très efficaces (solveurs) pour résoudre des gros problèmes rapidement

On ne considère plus les chemins entre l'état initial et l'état final, mais uniquement les valeurs des variables dans l'état but

状态空间与变量空间

状态空间建模

- 优势：更自然的建模方式及动作序列计算能力
- 缺点：需要考虑可能的状态数量非常大（相对于状态中的 组件数 呈指数级增长）

基于变量集合的状态建模

- 缺点：建模可能更为复杂
- 优势：无需逐项枚举状态，可借助高效求解器快速处理复杂问题

不再考虑从初始状态到最终状态的路径，而仅关注目标状态中变量的取值。

Exemple : Le Sudoku

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

Résolution avec la recherche arborescente

- Action : placer un nombre dans une case libre
- États buts : la grille est pleine, et il n'y a pas deux fois la même valeur dans une ligne, une colonne, ou un secteur 3×3
- Facteur de branchement : très grand ! (81 si la grille de départ est vide)

Résolution avec la programmation par contraintes

- 81 variables (une par case) et des contraintes pour décrire les règles du Sudoku
- un solveur de contraintes fournit une solution (ou la preuve qu'il n'en existe pas)

示例：数独

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

树形搜索的分辨率

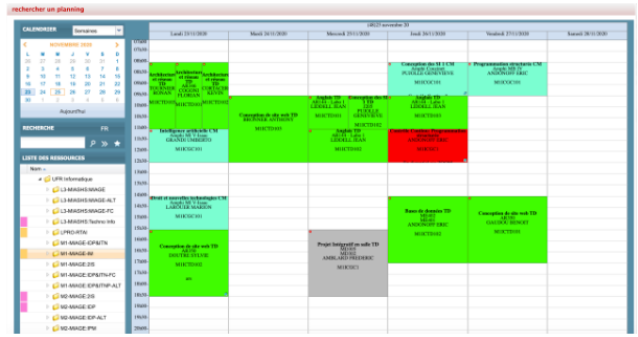
- 操作：在空白单元格中输入数字
- “目标状态：网格已填满，且同一行、同一列或同一3 × 3区域中不会出现重复值”
- 接线系数：超大！（若起始网格为空则为81）

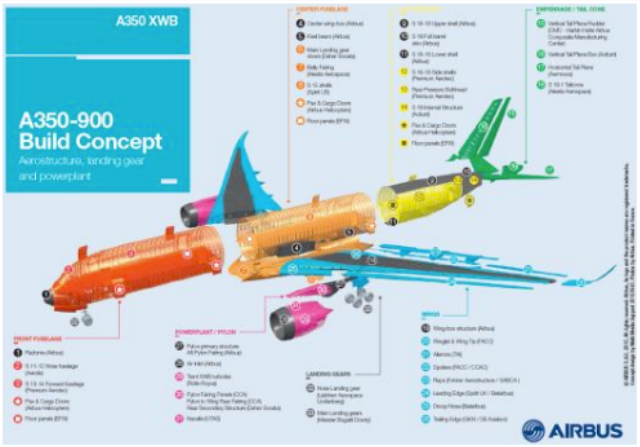
约束规划求解

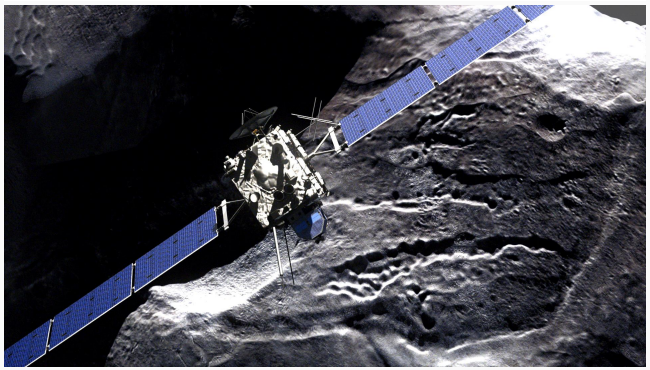
- 81个变量（每个单元格对应一个变量）
  以及用于编写数独规则的约束条件
- 约束求解器提供解决方案（或证明不存在该问题）

Applications

- Emplois du temps
- Allocation de tâches ou de ressources (entre des travailleurs, des citoyens, des processeurs,. . .)
- Ordonnancement/planifications de tâches (production d'avion chez Airbus, observations de la sonde spatiale Rosetta)



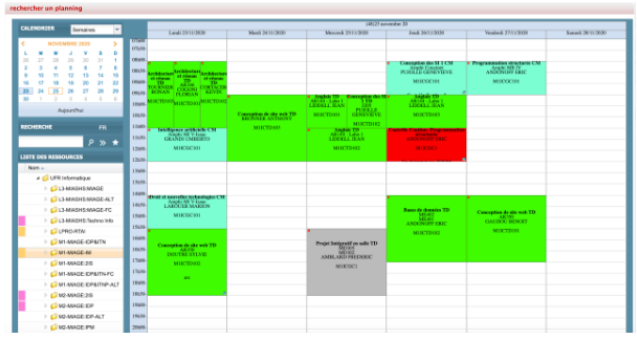


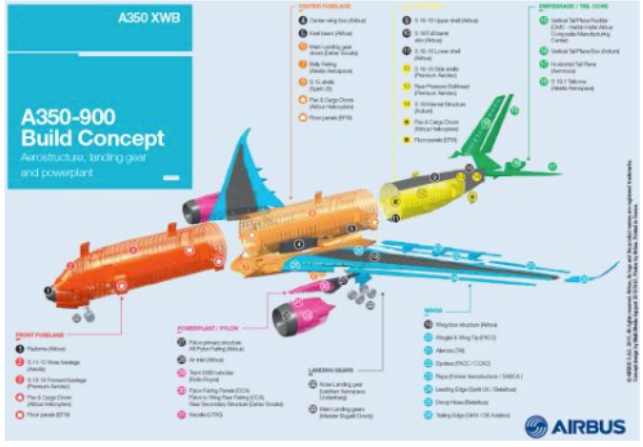


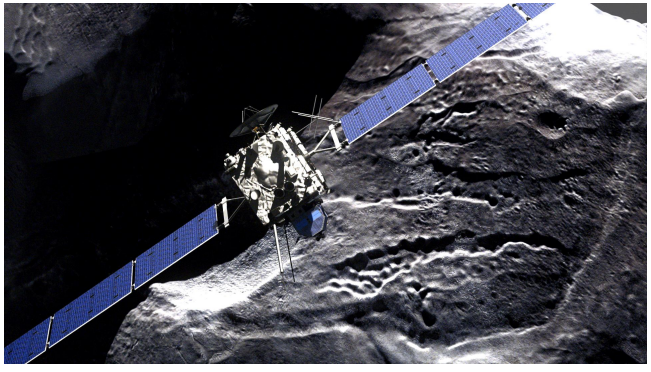
https://www.a4cp.org/node/1058

应用

- 日程安排
- 任务或资源分配（在工作者、公民、处理器等之间）
- 任务调度与规划（空客飞机制造、罗塞塔号探测器观测）







https://www.a4cp.org/node/1058

Application : Rosetta

- Mission : recueillir des données sur la composition du noyau de la comète Tchouri (https://fr.wikipedia.org/wiki/Rosetta_(sonde_spatiale))
- La sonde dispose de peu d'énergie et d'un temps limité pour décider quelles expériences et mesures effectuer en fonction des conditions observées sur place
- Les scientifiques souhaitent réaliser un maximum d'expériences en un minimum de temps, étant donné le coût très élevé de la mission
- Plusieurs pays ont financé la mission, mais pas tous avec le même montant : comment répartir le temps d'expérimentation entre eux de manière équitable ?

应用:罗塞塔

- 任务：收集关于楚里彗星核心成分的数据（https://fr.wikipedia.org/wiki/Rosetta_(sonde_spatiale)）
- 该探测器能量有限且时间紧迫，需根据现场观测条件来决定开展哪些实验和测量。
- 科学家们希望在最短时间里完成尽可能多的实验，因为该任务的成本非常高。
- 多个国家为此次任务提供了资金支持，但资助金额不一：如何公平分配各国在试验阶段的投入时间？

Gestion d'une cargaison

Nombreuses contraintes pour la sécurité et l'efficacité du transport de marchandises

- Répartition équilibrée du poids
- L'ordre dans lequel le cargo visite les ports est important : on ne veut pas être obligé de décharger tout le cargo pour accéder aux conteneurs du premier port !
- De manière générale : minimisation du temps de chargement/déchargement

货物管理

货物运输安全与效率面临的诸多限制

- 平衡重分配
- 货轮的港口停靠顺序至关重要：我们可不想被迫在第一个港口卸完整船货物才能进入集装箱区！
- 通用方法：最小化装卸时间

Logique et SAT

逻辑与SAT

Logique Propositionnelle : les bases

- Cadre formel simple pour exprimer des contraintes
- Manipulation de *propositions* : énoncés qui peuvent être *V*rai ou *F*aux
  - *Il pleut*, *Il fait du soleil*, *Nous sommes à Toulouse*, *Nous sommes à Paris*
  - *Il y a un* 1 *dans la première case de la première ligne*
- Combinaison de propositions (*ET*, *OU*, *NON*, *SI X ALORS Y*, ...)
  - *Nous sommes à Toulouse OU nous sommes à Paris*, *SI nous sommes à Toulouse ALORS il y a du soleil*
  - *SI il y a un* 1 *dans la première case de la première ligne, ALORS il n'y a pas de* 1 *ailleurs sur la première ligne*
- Ce qu'on met généralement dans un `if` ou un `while`, c'est de la logique propositionnelle !

命题逻辑：基础

- 用于表达约束条件的简单公式框架
- *命题*的操纵：可以是*真*或*假*的陈述
  - *下雨了，阳光灿烂，我们在图卢兹，我们在巴黎*
  - *第一行第一列有一个*1
- 命题组合（*与*、*或*、*非*、*如果X那么Y*、...）
  - *我们在图卢兹还是在巴黎？如果我们在图卢兹，那就有阳光。*
  - *如果第一行的第一个单元格里有一个*1，*那么第一行其他地方就没有*1
- 在if或while语句中通常使用的，都是命题逻辑！

Syntaxe de la logique propositionnelle

- On définit un vocabulaire $\mathcal{V} = \{x_1, \ldots, x_n\}$: ensemble de variables qui peuvent recevoir une valeur *V*rai (parfois noté 1, ou $\top$) ou *F*aux (0, $\bot$)
- Une formule propositionnelle ϕ est
  - un atome (juste une variable) : $\phi = x$
  - une négation (*NON*) : $\phi = \neg\phi_1$
  - une conjonction (*ET*) : $\phi = \phi_1 \wedge \phi_2$
  - une disjonction (*OU*) : $\phi = \phi_1 \vee \phi_2$
  - une implication (*SI ... ALORS*) : $\phi = \phi_1 \rightarrow \phi_2$
  - une équivalence (*SI ET SEULEMENT SI*) : $\phi = \phi_1 \leftrightarrow \phi_2$
  - (et on peut en définir d'autres...)
- *SI nous sommes à Toulouse ALORS il y a du soleil* devient *toulouse* $\rightarrow$ *soleil*
- Pour le Sudoku : $x_i^{j,k}$ représente « il y a la valeur i dans la case j de la ligne k »
  - *SI il y a un* 1 *dans la première case de la première ligne, ALORS il n'y a pas de* 1 *ailleurs sur la première ligne* devient $x_1^{1,1} \rightarrow (\neg x_1^{2,1} \wedge \cdots \wedge \neg x_1^{9,1})$

命题逻辑语法

- 我们定义一个词汇表$V=\{x_1,\ldots,x_n\}$：一组变量，它们可以取值为*真*（有时记作1，或 ）或*假*（0，$\perp$）。
- 一个命题公式ϕ是
  - 一个原子（仅一个变量）：$\phi=x$
  - 一个否定（*NON*）：$\phi=\neg\phi_1$
  - 一个合取（*ET*）：$\phi=\phi_1\wedge\phi_2$
  - 一个析取（*或*）：$\phi=\phi_1\vee\phi_2$
  - 一个蕴含（*SI...ALORS*）：$\phi=\phi_1\rightarrow\phi_2$
  - 一个“等价（*仅当且仅当*）：$\phi=\phi_1\leftrightarrow\phi_2$
  - （当然还可以列举更多例子.....）
- *如果我们在图卢兹，那么那里就有阳光* 变成 *图卢兹*$\rightarrow$*阳光*
- 对于数独：xji,k表示 在第k行第j单元格中的值i
  - *如果第一行的第一个单元格中有一个*1，*那么第一行其他位置就没有*1*了*变为$x11,1\rightarrow(\neg x21,1\wedge\cdots\wedge\neg x91,1)$

Sémantique de la logique propositionnelle (1/2)

- Sémantique : association d'une valeur 0 ou 1 à chaque variable (interprétation)
- Des règles permettent de connaître la valeur d'une formule à partir de l'interprétation des variables
- Table de vérité : représentation de la valeur d'une formule pour chaque interprétation possible
- Par exemple :
  - $\neg\phi$ est vrai quand ϕ est faux
  - $\phi_1 \wedge \phi_2$ est vrai quand ϕ_1 et ϕ_2 sont vrais tous les deux

x	$\neg x$
0	1
1	0

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

- S'il y a n variables, il y a 2^n lignes dans la table !

命题逻辑的语义（上）

- 语义：为每个变量（解释）分配0或1的值
- 通过规则可从变量解释中获取公式值
- 真值表：对每个可能解释的公式值进行表示
- 例如：
  - $\neg\phi$ 当 ϕ 为假时为真
  - $\phi_1 \wedge \phi_2$ 当且仅当 ϕ_1 和 ϕ_2 同时为真时成立

X	非x
0	1
1	0

X	Y	x与y的乘积
0	0	0
0	1	0
1	0	0
1	1	1

- 如果有 n 个变量，表格里就有 2^n 行！

Sémantique de la logique propositionnelle (2/2)

- $\phi_1 \vee \phi_2$ est vrai quand au moins un parmi ϕ_1 et ϕ_2 est vrai
- $\phi_1 \rightarrow \phi_2$ est vrai quand ϕ_1 et ϕ_2 sont vrais, ou ϕ_1 est faux
- $\phi_1 \leftrightarrow \phi_2$ est vrai quand ϕ_1 et ϕ_2 ont la même valeur

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

x	y	$x \rightarrow y$
0	0	1
0	1	1
1	0	0
1	1	1

x	y	$x \leftrightarrow y$
0	0	1
0	1	0
1	0	0
1	1	1

- On peut combiner l'utilisation de ce principe pour savoir quand une formule plus complexe est vraie
- Les interprétations où la formule vaut 1 sont appelées les *modèles* de la formule

命题逻辑的语义（二）

- $\phi_1 \vee \phi_2$当且仅当ϕ_1和ϕ_2中至少有一个为真时成立
- $\phi_1 \rightarrow \phi_2$当$\phi_1$和$\phi_2$为真时成立，或$\phi_1$为假时成立
- $\phi_1 \leftrightarrow \phi_2$当且仅当$\phi_1$与$\phi_2$具有相同值时成立

x	y	x 与 y 的并集
0	0	0
0	1	1
1	0	1
1	1	1

x	y	$x \rightarrow y$
0	0	1
0	1	1
1	0	0
1	1	1

x	y	x与y的交换
0	0	1
0	1	0
1	0	0
1	1	1

- 可以结合这一原理来判断更复杂的公式何时成立
- 公式值为1的解释称为该公式的*模型*

Exemple simple : l'implication revisitée

- $\phi = \neg x \vee y$

x	y	$\neg x \vee y$
0	0	1
0	1	1
1	0	0
1	1	1

简单示例：被重新审视的参与

- $\varphi = \neg x \vee y \; x \; y$ 非

与	或	
0	0	1
0	1	1
1	0	0
1	1	1

Exemple simple : l'implication revisitée

- $\phi = \neg x \vee y$

x	y	$\neg x \vee y$
0	0	1
0	1	1
1	0	0
1	1	1

x	y	$x \to y$
0	0	1
0	1	1
1	0	0
1	1	1

- $\phi_1 \to \phi_2$ est équivalent à $\neg\phi_1 \vee \phi_2$!

简单示例：被重新审视的参与

- $\varphi = \neg x \vee y\, x$

y		非与或
0	0	1
0	1	1
1	0	0
1	1	1

x	$y0$	$x \to y$
0	1	1
0		1
1	0	0
1	1	1

- $\phi_1 \to \phi_2$等价于$\neg\phi_1 \vee \phi_2$!

Exemple simple : le “ou exclusif”

- $\phi = x \oplus y$ représente le $\ll$ ou exclusif $\gg$
- Rappel : avec une disjonction (le $\ll$ ou classique $\gg$ $\vee$), la formule est vraie si *au moins* une sous-formule est vraie
- Avec $\oplus$, la formule est vraie si *exactement* une sous-formule est vraie

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

简单示例：“或”或“非”

- $\phi = x \oplus y$表示 或排他性
- 提示：使用析取（ 或经典 $\vee$），当*至少*一个子公式为真时，该公式为真
- 对于$\oplus$，当*恰好*一个子公式为真时，该公式为真x y x *与* y *的并集*

0	0	0
0	1	1
1	0	1
1	1	0

Exemple simple : le “ou exclusif”

- $\phi = x \oplus y$ représente le « ou exclusif »
- Rappel : avec une disjonction (le « ou classique » $\vee$), la formule est vraie si *au moins* une sous-formule est vraie
- Avec $\oplus$, la formule est vraie si *exactement* une sous-formule est vraie

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

x	y	$x \leftrightarrow y$
0	0	1
0	1	0
1	0	0
1	1	1

- $\phi_1 \oplus \phi_2$ est équivalent à $\neg(\phi_1 \leftrightarrow \phi_2)$!

简单示例：“或”或“非”

- $\phi = x \oplus y$表示 或排他性
- 提示：使用析取（ 或经典 $\vee$），当*至少*一个子公式为真时，该公式为真
- 对于$\oplus$，当*恰好*一个子公式为真时，该公式为真$x\ y$ x *与* y *的并集*

x *与* y x*与*y*的交换*

0	0	0
0	1	1
1	0	1
1	1	0

0	0	1
0	1	0
1	0	0
1	1	1

- $\phi_1 \oplus \phi_2$等价于$\neg(\phi_1 \leftrightarrow \phi_2)$!

Un exemple plus complexe

- $\phi = (x \vee \neg y) \rightarrow (x \wedge z)$
- On peut décomposer la formule en sous-formules plus simples

x	y	z	$\neg y$	$x \vee \neg y$	$x \wedge z$	ϕ
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

一个更复杂的例子

- $\varphi = (x \vee \neg y) \rightarrow (x \wedge z)$
- 可以将公式分解为更简单的子公式$x\ y\ z \mid y\ x \vee \neg y\ x \wedge z\ \phi$

0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Un exemple plus complexe

- $\phi = (x \vee \neg y) \rightarrow (x \wedge z)$
- On peut décomposer la formule en sous-formules plus simples

x	y	z	$\neg y$	$x \vee \neg y$	$x \wedge z$	ϕ
0	0	0	1			
0	0	1	1			
0	1	0	0			
0	1	1	0			
1	0	0	1			
1	0	1	1			
1	1	0	0			
1	1	1	0			

- $\neg y$ est vrai quand y est faux

一个更复杂的例子

- $\varphi = (x \vee \neg y) \rightarrow (x \wedge z)$
- 可以将公式分解为更简单的子公式$x\ y\ z \mid y\ x \vee \neg y\ x \wedge z\ \phi$

0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

- $\neg y$当y为假时为真

Un exemple plus complexe

- $\phi = (x \vee \neg y) \rightarrow (x \wedge z)$
- On peut décomposer la formule en sous-formules plus simples

x	y	z	$\neg y$	$x \vee \neg y$	$x \wedge z$	ϕ
0	0	0	1	1		
0	0	1	1	1		
0	1	0	0	0		
0	1	1	0	0		
1	0	0	1	1		
1	0	1	1	1		
1	1	0	0	1		
1	1	1	0	1		

- $\neg y$ est vrai quand y est faux
- $x \vee \neg y$ est vrai quand x est vrai ou $\neg y$ est vrai

一个更复杂的例子

- $\varphi = (x \vee \neg y) \rightarrow (x \wedge z)$
- 可以将公式分解为更简单的子公式$x\ y\ z \mid y\ x \vee \neg y\ x \wedge z\ \phi$

0	0	0	1	1
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	0	1
1	1	1	0	1

- $\neg y$当y为假时为真
- $x \vee \neg y$当且仅当x为真或$\neg y$为真时为真

Un exemple plus complexe

- $\phi = (x \vee \neg y) \rightarrow (x \wedge z)$
- On peut décomposer la formule en sous-formules plus simples

x	y	z	$\neg y$	$x \vee \neg y$	$x \wedge z$	ϕ
0	0	0	1	1	0	
0	0	1	1	1	0	
0	1	0	0	0	0	
0	1	1	0	0	0	
1	0	0	1	1	0	
1	0	1	1	1	1	
1	1	0	0	1	0	
1	1	1	0	1	1	

- $\neg y$ est vrai quand y est faux
- $x \vee \neg y$ est vrai quand x est vrai ou $\neg y$ est vrai
- $x \wedge z$ est vrai quand x est z sont vrais tous les deux

一个更复杂的例子

- $\varphi = (x \vee \neg y) \rightarrow (x \wedge z)$
- 可以将公式分解为更简单的子公式$x\ y\ z \mid y\ x \vee \neg y\ x \wedge z\ \phi$

0	0	0	1	1	0
0	0	1	1	1	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	0	1	0
1	1	1	0	1	1

- $\neg y$当y为假时为真
- $x \vee \neg y$当且仅当x为真或$\neg y$为真时为真
- $x \wedge z$当且仅当x为真且z为真时为真

Un exemple plus complexe

- $\phi = (x \vee \neg y) \rightarrow (x \wedge z)$
- On peut décomposer la formule en sous-formules plus simples

x	y	z	$\neg y$	$x \vee \neg y$	$x \wedge z$	ϕ
0	0	0	1	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	1	0	0	0	1
1	0	0	1	1	0	0
1	0	1	1	1	1	1
1	1	0	0	1	0	0
1	1	1	0	1	1	1

- $\neg y$ est vrai quand y est faux
- $x \vee \neg y$ est vrai quand x est vrai ou $\neg y$ est vrai
- $x \wedge z$ est vrai quand x est z sont vrais tous les deux
- l'implication (ϕ) est vraie quand ses deux parties sont vraies, ou la partie gauche est fausse

一个更复杂的例子

- $\varphi = (x \vee \neg y) \rightarrow (x \wedge z)$
- 可以将公式分解为更简单的子公式$x\ y\ z \mid y\ x \vee \neg y\ x \wedge z\ \phi$

0	0	0	1	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	1	0	0	0	1
1	0	0	1	1	0	0
1	0	1	1	1	1	1
1	1	0	0	1	0	0
1	1	1	0	1	1	1

- $\neg y$当y为假时为真
- $x \vee \neg y$当且仅当x为真或$\neg y$为真时为真
- $x \wedge z$当且仅当x为真且z为真时为真
- 当蕴含关系（ϕ）的两个部分均为真时，或左部分为假时，该蕴含关系成立。

Exercice

- Trouvez la table de vérité de $\phi = (\neg x \vee \neg y) \wedge (x \vee \neg z)$

x	y	z	$\neg x \vee \neg y$	$x \vee \neg z$	ϕ
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

- 求出真值表 $\phi = (\neg x \vee \neg y) \wedge (x \vee \neg z)$ $x\ y\ z \mid \neg x \vee \neg y \quad x \vee \neg z$

ϕ		
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Exercice

- Trouvez la table de vérité de $\phi = (\neg x \vee \neg y) \wedge (x \vee \neg z)$

x	y	z	$\neg x \vee \neg y$	$x \vee \neg z$	ϕ
0	0	0	1		
0	0	1	1		
0	1	0	1		
0	1	1	1		
1	0	0	1		
1	0	1	1		
1	1	0	0		
1	1	1	0		

- $\neg x \vee \neg y$ est vrai quand x ou y est faux (ou les deux)

- 求出真值表$\phi = (\neg x \vee \neg y) \wedge (x \vee \neg z)$

x	y	z	$\neg x \vee \neg y$	$x \vee \neg z$	ϕ
0	0	0	1	1	1
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	1	1	1
1	0	0	1	1	1
1	0	1	1	0	1
1	1	0	0	1	0
1	1	1	0	1	0

- $\neg x \vee \neg y$当且仅当x或y为假（或两者皆为假）时为真

Exercice

- Trouvez la table de vérité de $\phi = (\neg x \vee \neg y) \wedge (x \vee \neg z)$

x	y	z	$\neg x \vee \neg y$	$x \vee \neg z$	ϕ
0	0	0	1	1	
0	0	1	1	0	
0	1	0	1	1	
0	1	1	1	0	
1	0	0	1	1	
1	0	1	1	1	
1	1	0	0	1	
1	1	1	0	1	

- $\neg x \vee \neg y$ est vrai quand x ou y est faux (ou les deux)
- $x \vee \neg z$ est vrai quand x est vrai ou $\neg z$ est vrai (donc z est faux)

- 求出真值表$\phi = (\neg x \vee \neg y) \wedge (x \vee \neg z)$

x	y	z	$\neg x \vee \neg y$	$x \vee \neg z$
0	0	0	1	1
0	0	1	1	0
0	1	0	1	1
0	1	1	1	0
1	0	0	1	1
1	0	1	1	1
1	1	0	0	1
1	1	1	0	1

- $\neg x \vee \neg y$当且仅当x或y为假（或两者皆为假）时为真
- $x \vee \neg z$当且仅当x为真或$\neg z$为真（即z为假）

Exercice

- Trouvez la table de vérité de $\phi = (\neg x \vee \neg y) \wedge (x \vee \neg z)$

x	y	z	$\neg x \vee \neg y$	$x \vee \neg z$	ϕ
0	0	0	1	1	1
0	0	1	1	0	0
0	1	0	1	1	1
0	1	1	1	0	0
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	0	1	0
1	1	1	0	1	1

- $\neg x \vee \neg y$ est vrai quand x ou y est faux (ou les deux)
- $x \vee \neg z$ est vrai quand x est vrai ou $\neg z$ est vrai (donc z est faux)
- ϕ est vrai quand les deux sous-formules sont vraies

- 求出真值表$\phi = (\neg x \vee \neg y) \wedge (x \vee \neg z)$

x	y	z	$\neg x \vee \neg y$	$x \vee \neg z$	ϕ
0	0	0	1	1	1
0	0	1	1	0	0
0	1	0	1	1	1
0	1	1	1	0	0
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	0	1	0
1	1	1	0	1	1

- $\neg x \vee \neg y$当且仅当x或y为假（或两者皆为假）时为真
- $x \vee \neg z$当且仅当x为真或$\neg z$为真（即z为假）
- ϕ当两个子公式都为真时成立

Un petit casse-tête : tigres et princesses

Inspiré de *Le livre qui rend fou* (Raymond Smullyan)

- Dans un royaume ancien, le roi met un prisonnier face à deux cellules, qui peuvent contenir soit un tigre, soit un princesse
- Le prisonnier doit entrer dans une cellule. Si elle contient un tigre, il se fait dévorer, mais si elle contient une princesse il est libéré et épouse la princesse
- Des affiches sur les portes donnent des indices pour choisir, et le roi lui précise qu'une affiche dit la vérité, et l'autre ment
  - Porte 1 : ≪ Il y a une princesse dans cette cellule et un tigre dans l'autre ≫
  - Porte 2 : ≪ Il y a une princesse dans une cellule et il y a un tigre dans une cellule ≫
- Quelle porte choisiriez vous ?

一个小小的谜题：老虎与公主

灵感来自《*疯狂之书*》（雷蒙德·斯穆里安）

- 在一座古老的王国里，国王将一名囚犯置于两个牢房之间，每个牢房里可能关着一只老虎或一位公主。
- 囚犯必须进入牢房。若牢房里有老虎，他就会被老虎吃掉；但若牢房里有公主，他就会获释并娶公主为妻。
- 门上的告示为他提供了选择的线索，国王明确指出：一张告示说真话，另一张则说谎话。
  - 门1：　这个牢房里有位公主，另一个牢房里有只老虎
  - 门2：　一个牢房里有位公主，另一个牢房里有只老虎
- 你会选择哪扇门？

Tigres et princesses, modélisation

On peut modéliser de différentes façons

- t_1, t_2 : il y a un tigre dans la cellule 1/cellule 2
- p_1, p_2 : il y a une princesse dans la cellule 1/cellule 2

老虎与公主的建模

可通过多种方式建立模型

- t_1，t_2：在单元格1/单元格2中有一只老虎
- p_1，p_2：在细胞1/细胞2中存在一位公主

Tigres et princesses, modélisation

On peut modéliser de différentes façons

- t_1, t_2 : il y a un tigre dans la cellule 1/cellule 2
- p_1, p_2 : il y a une princesse dans la cellule 1/cellule 2
- Problème : ça complexifie la résolution du problème ($2^4 = 16$ lignes pour la table de vérité, et il faut représenter explicitement le fait qu'il n'y a pas un tigre et une princesse dans la même cellule)

老虎与公主的建模

可通过多种方式建立模型

- t_1，t_2：在单元格1/单元格2中有一只老虎
- p_1，p_2：在细胞1/细胞2中存在一位公主
- 问题：这使得问题的求解变得复杂（验证表包含$2^4 = 16$行，且需要明确表示同一单元格中不存在老虎和公主的情况）

Tigres et princesses, modélisation

On peut modéliser de différentes façons

- t_1, t_2 : il y a un tigre dans la cellule 1/cellule 2
- p_1, p_2 : il y a une princesse dans la cellule 1/cellule 2
- Problème : ça complexifie la résolution du problème ($2^4 = 16$ lignes pour la table de vérité, et il faut représenter explicitement le fait qu'il n'y a pas un tigre et une princesse dans la même cellule)
- On n'a besoin que des variables t_1 et t_2 : pour dire qu'il y a une princesse dans une cellule, on peut utiliser $\neg t_1$ et $\neg t_2$

老虎与公主的建模

可通过多种方式建立模型

- t_1，t_2：在单元格1/单元格2中有一只老虎
- p_1，p_2：在细胞1/细胞2中存在一位公主
- 问题：这使得问题的求解变得复杂（验证表包含$2^4 = 16$行，且需要明确表示同一单元格中不存在老虎和公主的情况）
- 只需要变量t_1和t_2：要表示某个单元格中存在公主，可以使用$\neg t_1$和$\neg t_2$

Tigres et princesses, modélisation

On peut modéliser de différentes façons

- t_1, t_2 : il y a un tigre dans la cellule 1/cellule 2
- p_1, p_2 : il y a une princesse dans la cellule 1/cellule 2
- Problème : ça complexifie la résolution du problème ($2^4 = 16$ lignes pour la table de vérité, et il faut représenter explicitement le fait qu'il n'y a pas un tigre et une princesse dans la même cellule)
- On n'a besoin que des variables t_1 et t_2 : pour dire qu'il y a une princesse dans une cellule, on peut utiliser $\neg t_1$ et $\neg t_2$
- Porte 1 : $\ll$ Il y a une princesse dans cette cellule et un tigre dans l'autre $\gg$, $\phi_1 = \neg t_1 \wedge t_2$

老虎与公主的建模

可通过多种方式建立模型

- t_1，t_2：在单元格1/单元格2中有一只老虎
- p_1，p_2：在细胞1/细胞2中存在一位公主
- 问题：这使得问题的求解变得复杂（验证表包含$2^4 = 16$行，且需要明确表示同一单元格中不存在老虎和公主的情况）
- 只需要变量t_1和t_2：要表示某个单元格中存在公主，可以使用$\neg t_1$和$\neg t_2$
- 门1：　这个牢房里有位公主，另一个牢房里有只老虎　，$\phi_1 = \neg t_1 \wedge t_2$

Tigres et princesses, modélisation

On peut modéliser de différentes façons

- t_1, t_2 : il y a un tigre dans la cellule 1/cellule 2
- p_1, p_2 : il y a une princesse dans la cellule 1/cellule 2
- Problème : ça complexifie la résolution du problème ($2^4 = 16$ lignes pour la table de vérité, et il faut représenter explicitement le fait qu'il n'y a pas un tigre et une princesse dans la même cellule)
- On n'a besoin que des variables t_1 et t_2 : pour dire qu'il y a une princesse dans une cellule, on peut utiliser $\neg t_1$ et $\neg t_2$
- Porte 1 : « Il y a une princesse dans cette cellule et un tigre dans l'autre », $\phi_1 = \neg t_1 \wedge t_2$
- Porte 2 : « Il y a une princesse dans une cellule et il y a un tigre dans une cellule », $\phi_2 = (\neg t_1 \vee \neg t_2) \wedge (t_1 \vee t_2)$

老虎与公主的建模

可通过多种方式建立模型

- t_1，t_2：在单元格1/单元格2中有一只老虎
- p_1，p_2：在细胞1/细胞2中存在一位公主
- 问题：这使得问题的求解变得复杂（验证表包含$2^4 = 16$行，且需要明确表示同一单元格中不存在老虎和公主的情况）
- 只需要变量t_1和t_2：要表示某个单元格中存在公主，可以使用$\neg t_1$和$\neg t_2$
- 门1：　这个牢房里有位公主，另一个牢房里有只老虎　，$\phi_1 = \neg t_1 \wedge t_2$
- 门2：　一个牢房里有位公主，另一个牢房里有只老虎　，$\phi_2 = (\neg t_1 \vee \neg t2) \wedge (t_1 \vee t2)$

Tigres et princesses, modélisation

On peut modéliser de différentes façons

- t_1, t_2 : il y a un tigre dans la cellule 1/cellule 2
- p_1, p_2 : il y a une princesse dans la cellule 1/cellule 2
- Problème : ça complexifie la résolution du problème ($2^4 = 16$ lignes pour la table de vérité, et il faut représenter explicitement le fait qu'il n'y a pas un tigre et une princesse dans la même cellule)
- On n'a besoin que des variables t_1 et t_2 : pour dire qu'il y a une princesse dans une cellule, on peut utiliser $\neg t_1$ et $\neg t_2$
- Porte 1 : « Il y a une princesse dans cette cellule et un tigre dans l'autre », $\phi_1 = \neg t_1 \wedge t_2$
- Porte 2 : « Il y a une princesse dans une cellule et il y a un tigre dans une cellule », $\phi_2 = (\neg t_1 \vee \neg t_2) \wedge (t_1 \vee t_2)$
- Une affiche dit la vérité, et l'autre ment, $\phi = \phi_1 \leftrightarrow \neg\phi_2$

老虎与公主的建模

可通过多种方式建立模型

- t_1，t_2：在单元格1/单元格2中有一只老虎
- p_1，p_2：在细胞1/细胞2中存在一位公主
- 问题：这使得问题的求解变得复杂（验证表包含$2^4 = 16$行，且需要明确表示同一单元格中不存在老虎和公主的情况）
- 只需要变量t_1和t_2：要表示某个单元格中存在公主，可以使用$\neg t_1$和$\neg t_2$
- 门1：　这个牢房里有位公主，另一个牢房里有只老虎　，$\phi_1 = \neg t_1 \wedge t_2$
- 门2：　一个牢房里有位公主，另一个牢房里有只老虎　，$\phi_2 = (\neg t_1 \vee \neg t2) \wedge (t_1 \vee t2)$
- 一张海报说真话，另一张说谎，$\phi = \phi_1 \leftrightarrow \neg \phi_2$

Tigres et princesses, résolution

- $\phi_1 = \neg t_1 \wedge t_2$, $\phi_2 = (\neg t_1 \vee \neg t_2) \wedge (t_1 \vee t_2)$, $\phi = \phi_1 \leftrightarrow \neg\phi_2$

t_1	t_2	$\phi_1 = \neg t_1 \wedge t_2$	$\neg t_1 \vee \neg t_2$	$t_1 \vee t_2$	ϕ_2	ϕ
0	0	0	1	0	0	0
0	1	1	1	1	1	0
1	0	0	1	1	1	1
1	1	0	0	1	0	0

老虎与公主，分辨率

- $\phi_1 = \neg t_1 \wedge t_2$，$\phi_2 = (\neg t_1 \vee \neg t2) \wedge (t_1 \vee t_2)$，$\phi = \phi_1 \leftrightarrow \neg\phi_2$

t_1	t_2	$\phi_1 = \neg t_1 \wedge t_2$	$\neg t_1 \vee \neg t_2$	$t_1 \vee t_2$	ϕ_2	ϕ
0	0	0	1	0	0	0
0	1	1	1	1	1	0
1	0	0	1	1	1	1
1	1	0	0	1	0	0

Tigres et princesses, résolution

- $\phi_1 = \neg t_1 \wedge t_2$, $\phi_2 = (\neg t_1 \vee \neg t_2) \wedge (t_1 \vee t_2)$, $\phi = \phi_1 \leftrightarrow \neg\phi_2$

t_1	t_2	$\phi_1 = \neg t_1 \wedge t_2$	$\neg t_1 \vee \neg t_2$	$t_1 \vee t_2$	ϕ_2	ϕ
0	0	0	1	0	0	0
0	1	1	1	1	1	0
1	0	0	1	1	1	1
1	1	0	0	1	0	0

- Le seul modèle de ϕ : $t_1 = 1$ et $t_2 = 0$, donc il y a un tigre dans la cellule 1 et une princesse dans la cellule 2

老虎与公主，分辨率

- $\phi_1 = \neg t_1 \wedge t_2$，$\phi_2 = (\neg t_1 \vee \neg t2) \wedge (t_1 \vee t_2)$，$\phi = \phi_1 \leftrightarrow \neg\phi_2$

t_1	t_2	$\phi_1 = \neg t_1 \wedge t_2$	$\neg t_1 \vee \neg t_2$	$t_1 \vee t_2$	ϕ_2	ϕ
0	0	0	1	0	0	0
0	1	1	1	1	1	0
1	0	0	1	1	1	1
1	1	0	0	1	0	0

- 唯一的ϕ模型：$t_1 = 1$且$t_2 = 0$，因此单元格1中有一只老虎，单元格2中有一只公主

Logique et Sudoku

Essayez (après le cours) de trouver une représentation logique des règles du Sudoku

- De quelles variables propositionnelles avez vous besoin ?
- Quelles règles logiques permettent de résoudre une grille de Sudoku ?

逻辑与数独

课后请尝试用逻辑方式表达数独规则

- 您需要哪些命题变量？
- 有哪些逻辑规则可以用来解决数独谜题？

Forme Normale Conjonctive

- Les algorithmes pour raisonner avec des formules logiques utilisent la forme normale conjonctive (CNF, pour *Conjunctive Normal Form*)
- Toute formule propositionnelle peut-être transformée en une CNF équivalente (avec les mêmes modèles)

正常结膜形态

- 用于逻辑公式推理的算法采用合取范式（CNF，即*合取范式*）
- 任何命题公式都可以转换为等价的CNF（采用相同模型）

Forme Normale Conjonctive

- Les algorithmes pour raisonner avec des formules logiques utilisent la forme normale conjonctive (CNF, pour *Conjunctive Normal Form*)
- Toute formule propositionnelle peut-être transformée en une CNF équivalente (avec les mêmes modèles)
- Littéral : soit une variable seule, soit la négation d'une variable (x est un littéral, $\neg y$ est un littéral, mais $x \vee \neg y$ n'en est pas un !)

正常结膜形态

- 用于逻辑公式推理的算法采用合取范式（CNF，即 *合取范式* ）
- 任何命题公式都可以转换为等价的CNF（采用相同模型）
- 字面量：要么是一个变量，要么是一个变量的否定（x是一个字面量，$\neg y$是一个字面量，但$x \vee \neg y$不是！）

Forme Normale Conjonctive

- Les algorithmes pour raisonner avec des formules logiques utilisent la forme normale conjonctive (CNF, pour *Conjunctive Normal Form*)
- Toute formule propositionnelle peut-être transformée en une CNF équivalente (avec les mêmes modèles)
- Littéral : soit une variable seule, soit la négation d'une variable (x est un littéral, $\neg y$ est un littéral, mais $x \vee \neg y$ n'en est pas un !)
- Clause : disjonction de littéraux (au moins 1, ou plusieurs littéraux, par exemple $\neg y$ est une clause, $x \vee \neg y$ est une clause, mais $x \wedge \neg y$ n'en est pas une)

正常结膜形态

- 用于逻辑公式推理的算法采用合取范式（CNF，即*合取范式*）
- 任何命题公式都可以转换为等价的CNF（采用相同模型）
- 字面量：要么是一个变量，要么是一个变量的否定（x是一个字面量，$\neg y$是一个字面量，但$x \vee \neg y$不是！）
- 子句：析取子句（至少1个或多个子句，例如$\neg y$是一个子句，$x \vee \neg y$是一个子句，但$x \wedge \neg y$不是）

Forme Normale Conjonctive

- Les algorithmes pour raisonner avec des formules logiques utilisent la forme normale conjonctive (CNF, pour *Conjunctive Normal Form*)
- Toute formule propositionnelle peut-être transformée en une CNF équivalente (avec les mêmes modèles)
- Littéral : soit une variable seule, soit la négation d'une variable (x est un littéral, $\neg y$ est un littéral, mais $x \vee \neg y$ n'en est pas un !)
- Clause : disjonction de littéraux (au moins 1, ou plusieurs littéraux, par exemple $\neg y$ est une clause, $x \vee \neg y$ est une clause, mais $x \wedge \neg y$ n'en est pas une)
- Formule CNF : conjonction de clauses (au moins 1, ou plusieurs clauses)

正常结膜形态

- 用于逻辑公式推理的算法采用合取范式（CNF，即*合取范式*）
- 任何命题公式都可以转换为等价的CNF（采用相同模型）
- 字面量：要么是一个变量，要么是一个变量的否定（x是一个字面量，$\neg y$是一个字面量，但$x \vee \neg y$不是！）
- 子句：析取子句（至少1个或多个子句，例如$\neg y$是一个子句，$x \vee \neg y$是一个子句，但$x \wedge \neg y$不是）
- CNF公式：由至少一个子句（或多个子句）组成的合取式

Forme Normale Conjonctive

- Les algorithmes pour raisonner avec des formules logiques utilisent la forme normale conjonctive (CNF, pour *Conjunctive Normal Form*)
- Toute formule propositionnelle peut-être transformée en une CNF équivalente (avec les mêmes modèles)
- Littéral : soit une variable seule, soit la négation d'une variable (x est un littéral, $\neg y$ est un littéral, mais $x \vee \neg y$ n'en est pas un !)
- Clause : disjonction de littéraux (au moins 1, ou plusieurs littéraux, par exemple $\neg y$ est une clause, $x \vee \neg y$ est une clause, mais $x \wedge \neg y$ n'en est pas une)
- Formule CNF : conjonction de clauses (au moins 1, ou plusieurs clauses)
- Une interprétation satisfait une clause si elle satisfait au moins un littéral dans la clause (car c'est une disjonction)

正常结膜形态

- 用于逻辑公式推理的算法采用合取范式（CNF，即*合取范式*）
- 任何命题公式都可以转换为等价的CNF（采用相同模型）
- 字面量：要么是一个变量，要么是一个变量的否定（x是一个字面量，$\neg y$是一个字面量，但$x \vee \neg y$不是！）
- 子句：析取子句（至少1个或多个子句，例如$\neg y$是一个子句，$x \vee \neg y$是一个子句，但$x \wedge \neg y$不是）
- CNF公式：由至少一个子句（或多个子句）组成的合取式
- 当解释满足该条款时，只要满足其中至少一个字面条件即可（因为这是一个析取关系）

Forme Normale Conjonctive

- Les algorithmes pour raisonner avec des formules logiques utilisent la forme normale conjonctive (CNF, pour *Conjunctive Normal Form*)
- Toute formule propositionnelle peut-être transformée en une CNF équivalente (avec les mêmes modèles)
- Littéral : soit une variable seule, soit la négation d'une variable (x est un littéral, $\neg y$ est un littéral, mais $x \vee \neg y$ n'en est pas un !)
- Clause : disjonction de littéraux (au moins 1, ou plusieurs littéraux, par exemple $\neg y$ est une clause, $x \vee \neg y$ est une clause, mais $x \wedge \neg y$ n'en est pas une)
- Formule CNF : conjonction de clauses (au moins 1, ou plusieurs clauses)
- Une interprétation satisfait une clause si elle satisfait au moins un littéral dans la clause (car c'est une disjonction)
- Une interprétation satisfait une CNF si elle satisfait toutes les clauses (car c'est une conjonction)

正常结膜形态

- 用于逻辑公式推理的算法采用合取范式（CNF，即*合取范式*）
- 任何命题公式都可以转换为等价的CNF（采用相同模型）
- 字面量：要么是一个变量，要么是一个变量的否定（x是一个字面量，$\neg y$是一个字面量，但$x \vee \neg y$不是！）
- 子句：析取子句（至少1个或多个子句，例如$\neg y$是一个子句，$x \vee \neg y$是一个子句，但$x \wedge \neg y$不是）
- CNF公式：由至少一个子句（或多个子句）组成的合取式
- 当解释满足该条款时，只要满足其中至少一个字面条件即可（因为这是一个析取关系）
- 当一个解释满足所有子句时，它就满足CNF（因为这是个合取式）。

Forme Normale Conjonctive

- Les algorithmes pour raisonner avec des formules logiques utilisent la forme normale conjonctive (CNF, pour *Conjunctive Normal Form*)
- Toute formule propositionnelle peut-être transformée en une CNF équivalente (avec les mêmes modèles)
- Littéral : soit une variable seule, soit la négation d'une variable (x est un littéral, $\neg y$ est un littéral, mais $x \vee \neg y$ n'en est pas un !)
- Clause : disjonction de littéraux (au moins 1, ou plusieurs littéraux, par exemple $\neg y$ est une clause, $x \vee \neg y$ est une clause, mais $x \wedge \neg y$ n'en est pas une)
- Formule CNF : conjonction de clauses (au moins 1, ou plusieurs clauses)
- Une interprétation satisfait une clause si elle satisfait au moins un littéral dans la clause (car c'est une disjonction)
- Une interprétation satisfait une CNF si elle satisfait toutes les clauses (car c'est une conjonction)
- Un solveur SAT est un algorithme qui prend en entrée une CNF et qui détermine si elle est SATisfiable, c'est-à-dire si elle a au moins un modèle
  - Le solveur SAT permet aussi de retourner un modèle, si il en existe

正常结膜形态

- 用于逻辑公式推理的算法采用合取范式（CNF，即*合取范式*）
- 任何命题公式都可以转换为等价的CNF（采用相同模型）
- 字面量：要么是一个变量，要么是一个变量的否定（x是一个字面量，$\neg y$是一个字面量，但$x \vee \neg y$不是！）
- 子句：析取子句（至少1个或多个子句，例如$\neg y$是一个子句，$x \vee \neg y$是一个子句，但$x \wedge \neg y$不是）
- CNF公式：由至少一个子句（或多个子句）组成的合取式
- 当解释满足该条款时，只要满足其中至少一个字面条件即可（因为这是一个析取关系）
- 当一个解释满足所有子句时，它就满足CNF（因为这是个合取式）。
- SAT求解器是一种算法，它以CNF（标准范式）作为输入，判断该公式是否可满足，即是否至少存在一个模型。
    - SAT求解器也可返回模型，若存在此类模型。

Transformation en CNF

- Élimination des doubles négations : $\neg\neg\phi \equiv \phi$
- Lois de De Morgan :
  - $\neg(\phi_1 \vee \phi_2) = \neg\phi_1 \wedge \neg\phi_2$
  - $\neg(\phi_1 \wedge \phi_2) = \neg\phi_1 \vee \neg\phi_2$
- Distributivité
  - $\phi_1 \vee (\phi_2 \wedge \phi_3) \equiv (\phi_1 \vee \phi_2) \wedge (\phi_1 \vee \phi_3)$
  - $\phi_1 \wedge (\phi_2 \vee \phi_3) \equiv (\phi_1 \wedge \phi_2) \vee (\phi_1 \wedge \phi_3)$

- 消去双重否定：$\neg\neg\boldsymbol{\phi \equiv \phi}$
- 德摩根定律
  - $\neg(\phi_1 \vee \phi_2) = \neg\phi_1 \wedge \neg\phi_2$
  - $\neg(\phi_1 \wedge \phi_2) = \neg\phi_1 \vee \neg\phi_2$
- 分配律
  - $\phi_1 \vee (\phi_2 \wedge \phi_3) \equiv (\phi_1 \vee \phi_2) \wedge (\phi_1 \vee \phi_3)$
  - $\phi_1 \wedge (\phi_2 \vee \phi_3) \equiv (\phi_1 \wedge \phi_2) \vee (\phi_1 \wedge \phi_3)$

Transformation en CNF

- Élimination des doubles négations : $\neg\neg\phi \equiv \phi$
- Lois de De Morgan :
  - $\neg(\phi_1 \vee \phi_2) = \neg\phi_1 \wedge \neg\phi_2$
  - $\neg(\phi_1 \wedge \phi_2) = \neg\phi_1 \vee \neg\phi_2$
- Distributivité
  - $\phi_1 \vee (\phi_2 \wedge \phi_3) \equiv (\phi_1 \vee \phi_2) \wedge (\phi_1 \vee \phi_3)$
  - $\phi_1 \wedge (\phi_2 \vee \phi_3) \equiv (\phi_1 \wedge \phi_2) \vee (\phi_1 \wedge \phi_3)$
- L'application successive de ces règles permet de transformer n'importe quelle formule en une CNF équivalente
  - Attention, cette transformation peut mener à une formule exponentiellement plus grande que la formule d'origine
  - Il existe une méthode un peu plus complexe qui permet de garantir une taille linéaire (transformation de Tseytin)

- 消去双重否定：$\neg\neg\phi \equiv \phi$
- 德摩根定律
  - $\neg(\phi_1 \vee \phi_2) = \neg\phi_1 \wedge \neg\phi_2$
  - $\neg(\phi_1 \wedge \phi_2) = \neg\phi_1 \vee \neg\phi_2$
- 分配律
  - $\phi_1 \vee (\phi_2 \wedge \phi_3) \equiv (\phi_1 \vee \phi_2) \wedge (\phi_1 \vee \phi_3)$
  - $\phi_1 \wedge (\phi_2 \vee \phi_3) \equiv (\phi_1 \wedge \phi_2) \vee (\phi_1 \wedge \phi_3)$
- 通过连续应用这些规则，可将任意公式转换为等价的CNF形式。
  - 注意，这种转换可能导致公式规模呈指数级增长，远超原始公式。
  - 存在一种较为复杂的方法，可确保线性尺寸（Tseytin变换）

Transformation en CNF

- Élimination des doubles négations : $\neg\neg\phi \equiv \phi$
- Lois de De Morgan :
  - $\neg(\phi_1 \vee \phi_2) = \neg\phi_1 \wedge \neg\phi_2$
  - $\neg(\phi_1 \wedge \phi_2) = \neg\phi_1 \vee \neg\phi_2$
- Distributivité
  - $\phi_1 \vee (\phi_2 \wedge \phi_3) \equiv (\phi_1 \vee \phi_2) \wedge (\phi_1 \vee \phi_3)$
  - $\phi_1 \wedge (\phi_2 \vee \phi_3) \equiv (\phi_1 \wedge \phi_2) \vee (\phi_1 \wedge \phi_3)$
- L'application successive de ces règles permet de transformer n'importe quelle formule en une CNF équivalente
  - Attention, cette transformation peut mener à une formule exponentiellement plus grande que la formule d'origine
  - Il existe une méthode un peu plus complexe qui permet de garantir une taille linéaire (transformation de Tseytin)

Vous pouvez prouver que ces règles fonctionnent en utilisant la méthode des tables de vérité

- 消去双重否定：$\neg\neg\phi \equiv \phi$
- 德摩根定律
  - $\neg(\phi_1 \vee \phi_2) = \neg\phi_1 \wedge \neg\phi_2$
  - $\neg(\phi_1 \wedge \phi_2) = \neg\phi_1 \vee \neg\phi_2$
- 分配律
  - $\phi_1 \vee (\phi_2 \wedge \phi_3) \equiv (\phi_1 \vee \phi_2) \wedge (\phi_1 \vee \phi_3)$
  - $\phi_1 \wedge (\phi_2 \vee \phi_3) \equiv (\phi_1 \wedge \phi_2) \vee (\phi_1 \wedge \phi_3)$
- 通过连续应用这些规则，可将任意公式转换为等价的CNF形式。
  - 注意，这种转换可能导致公式规模呈指数级增长，远超原始公式。
  - 存在一种较为复杂的方法，可确保线性尺寸（Tseytin变换）

你可以通过使用真值表法来证明这些规则的有效性。

Fonctionnement des solveurs SAT

- La plupart des solveurs SAT utilisant l'algorithme DPLL (Davis–Putnam–Logemann–Loveland)
- Principe de base :

SAT求解器的工作原理

- 大多数采用DPLL（戴维斯-普特南-洛格曼-洛夫兰）算法的SAT求解器
- 基本原则：

Fonctionnement des solveurs SAT

- La plupart des solveurs SAT utilisant l'algorithme DPLL (Davis–Putnam–Logemann–Loveland)
- Principe de base :
  - Choisir une variable x_i et lui assigner une valeur de vérité (par exemple 1)

SAT求解器的工作原理

- 大多数采用DPLL（戴维斯-普特南-洛格曼-洛夫兰）算法的SAT求解器
- 基本原则：
  - 选择一个变量*xi*并为其赋值为真值（例如1）

Fonctionnement des solveurs SAT

- La plupart des solveurs SAT utilisant l'algorithme DPLL (Davis–Putnam–Logemann–Loveland)
- Principe de base :
  - Choisir une variable x_i et lui assigner une valeur de vérité (par exemple 1)
  - Toutes les clauses qui contiennent x_i sont satisfaites, elles peuvent être supprimées

SAT求解器的工作原理

- 大多数采用DPLL（戴维斯-普特南-洛格曼-洛夫兰）算法的SAT求解器
- 基本原则：
  - 选择一个变量*xi*并为其赋值为真值（例如1）
  - 只要满足包含*xi*的所有条款，即可予以删除

Fonctionnement des solveurs SAT

- La plupart des solveurs SAT utilisant l'algorithme DPLL (Davis–Putnam–Logemann–Loveland)
- Principe de base :
  - Choisir une variable x_i et lui assigner une valeur de vérité (par exemple 1)
  - Toutes les clauses qui contiennent x_i sont satisfaites, elles peuvent être supprimées
  - Toutes les clauses qui contiennent $\neg x_i$ peuvent être raccourcies : $x_1 \vee x_2 \vee \neg x_i$ devient $x_1 \vee x_2$

- 大多数采用DPLL（戴维斯-普特南-洛格曼-洛夫兰）算法的SAT求解器
- 基本原则：
  - 选择一个变量*xi*并为其赋值为真值（例如1）
  - 只要满足包含*xi*的所有条款，即可予以删除
  - 所有包含¬*xi*的子句都可以被简化：*x1* ∨ *x2* ∨ ¬*xi*变为*x1* ∨ *x2*

Fonctionnement des solveurs SAT

- La plupart des solveurs SAT utilisant l'algorithme DPLL (Davis–Putnam–Logemann–Loveland)
- Principe de base :
  - Choisir une variable x_i et lui assigner une valeur de vérité (par exemple 1)
  - Toutes les clauses qui contiennent x_i sont satisfaites, elles peuvent être supprimées
  - Toutes les clauses qui contiennent $\neg x_i$ peuvent être raccourcies : $x_1 \vee x_2 \vee \neg x_i$ devient $x_1 \vee x_2$
  - Si on obtient une clause $\ll$ vide $\gg$ (i.e. tous les littéraux sont falsifiés), c'est qu'on a pris une décision incohérente : on revient en arrière et on essaye d'assigner la valeur 0 à x_1

- 大多数采用DPLL（戴维斯-普特南-洛格曼-洛夫兰）算法的SAT求解器
- 基本原则：
  - 选择一个变量*xi*并为其赋值为真值（例如1）
  - 只要满足包含*xi*的所有条款，即可予以删除
  - 所有包含¬*xi*的子句都可以被简化：*x1* ∨ *x2* ∨ ¬*xi*变为*x1* ∨ *x2*
  - 如果得到一个 空 条款（即所有字面表达式都被证伪），那就说明我们做出了一个不一致的决定：需要回溯并尝试将*x1*赋值为0

Fonctionnement des solveurs SAT

- La plupart des solveurs SAT utilisant l'algorithme DPLL (Davis–Putnam–Logemann–Loveland)
- Principe de base :
  - Choisir une variable x_i et lui assigner une valeur de vérité (par exemple 1)
  - Toutes les clauses qui contiennent x_i sont satisfaites, elles peuvent être supprimées
  - Toutes les clauses qui contiennent $\neg x_i$ peuvent être raccourcies : $x_1 \vee x_2 \vee \neg x_i$ devient $x_1 \vee x_2$
  - Si on obtient une clause $\ll$ vide $\gg$ (i.e. tous les littéraux sont falsifiés), c'est qu'on a pris une décision incohérente : on revient en arrière et on essaye d'assigner la valeur 0 à x_1
  - Si 0 mène aussi à une incohérence, alors la formule n'est pas satisfiable (aucun modèle)

- 大多数采用DPLL（戴维斯-普特南-洛格曼-洛夫兰）算法的SAT求解器
- 基本原则：
  - 选择一个变量*xi*并为其赋值为真值（例如1）
  - 只要满足包含*xi*的所有条款，即可予以删除
  - 所有包含¬*xi*的子句都可以被简化：*x1* ∨ *x2* ∨ ¬*xi*变为*x1* ∨ *x2*
  - 如果得到一个 空 条款（即所有字面表达式都被证伪），那就说明我们做出了一个不一致的决定：需要回溯并尝试将*x1*赋值为0
  - 若0也存在不一致性，则该公式无法满足（即不存在对应模型）

Fonctionnement des solveurs SAT

- La plupart des solveurs SAT utilisant l'algorithme DPLL (Davis–Putnam–Logemann–Loveland)
- Principe de base :
  - Choisir une variable x_i et lui assigner une valeur de vérité (par exemple 1)
  - Toutes les clauses qui contiennent x_i sont satisfaites, elles peuvent être supprimées
  - Toutes les clauses qui contiennent $\neg x_i$ peuvent être raccourcies : $x_1 \vee x_2 \vee \neg x_i$ devient $x_1 \vee x_2$
  - Si on obtient une clause $\ll$ vide $\gg$ (i.e. tous les littéraux sont falsifiés), c'est qu'on a pris une décision incohérente : on revient en arrière et on essaye d'assigner la valeur 0 à x_1
  - Si 0 mène aussi à une incohérence, alors la formule n'est pas satisfiable (aucun modèle)
  - Sinon, on continue jusqu'à avoir trouvé une valeur pour chaque variable, ce qui donne un modèle

- 大多数采用DPLL（戴维斯-普特南-洛格曼-洛夫兰）算法的SAT求解器
- 基本原则：
  - 选择一个变量*xi*并为其赋值为真值（例如1）
  - 只要满足包含*xi*的所有条款，即可予以删除
  - 所有包含¬*xi*的子句都可以被简化：*x1* ∨ *x2* ∨ ¬*xi*变为*x1* ∨ *x2*
  - 如果得到一个 空 条款（即所有字面表达式都被证伪），那就说明我们做出了一个不一致的决定：需要回溯并尝试将*x1*赋值为0
  - 若0也存在不一致性，则该公式无法满足（即不存在对应模型）
  - 否则，继续迭代直至为每个变量找到对应值，从而构建出一个模型

Fonctionnement des solveurs SAT

- La plupart des solveurs SAT utilisant l'algorithme DPLL (Davis–Putnam–Logemann–Loveland)
- Principe de base :
  - Choisir une variable x_i et lui assigner une valeur de vérité (par exemple 1)
  - Toutes les clauses qui contiennent x_i sont satisfaites, elles peuvent être supprimées
  - Toutes les clauses qui contiennent $\neg x_i$ peuvent être raccourcies : $x_1 \vee x_2 \vee \neg x_i$ devient $x_1 \vee x_2$
  - Si on obtient une clause $\ll$ vide $\gg$ (i.e. tous les littéraux sont falsifiés), c'est qu'on a pris une décision incohérente : on revient en arrière et on essaye d'assigner la valeur 0 à x_1
  - Si 0 mène aussi à une incohérence, alors la formule n'est pas satisfiable (aucun modèle)
  - Sinon, on continue jusqu'à avoir trouvé une valeur pour chaque variable, ce qui donne un modèle
- Il existe des techniques pour simplifier la formule
  - Propagation unitaire : s'il y a une clause unitaire (e.g. x ou $\neg y$), on peut propager dans toutes les clauses (x vaut 1 et y vaut 0 partout)

- 大多数采用DPLL（戴维斯-普特南-洛格曼-洛夫兰）算法的SAT求解器
- 基本原则：
  - 选择一个变量*xi*并为其赋值为真值（例如1）
  - 只要满足包含*xi*的所有条款，即可予以删除
  - 所有包含¬*xi*的子句都可以被简化：*x1* ∨ *x2* ∨ ¬*xi*变为*x1* ∨ *x2*
  - 如果得到一个 空 条款（即所有字面表达式都被证伪），那就说明我们做出了一个不一致的决定：需要回溯并尝试将*x1*赋值为0
  - 若0也存在不一致性，则该公式无法满足（即不存在对应模型）
  - 否则，继续迭代直至为每个变量找到对应值，从而构建出一个模型
- 存在一些技术可以简化公式
  - 单位传播：如果存在一个单位子句（例如*x*或¬*y*），则可以在所有子句中进行传播（*x*在所有地方为1且*y*在所有地方为0）

Fonctionnement des solveurs SAT

- La plupart des solveurs SAT utilisant l'algorithme DPLL (Davis–Putnam–Logemann–Loveland)
- Principe de base :
  - Choisir une variable x_i et lui assigner une valeur de vérité (par exemple 1)
  - Toutes les clauses qui contiennent x_i sont satisfaites, elles peuvent être supprimées
  - Toutes les clauses qui contiennent $\neg x_i$ peuvent être raccourcies : $x_1 \vee x_2 \vee \neg x_i$ devient $x_1 \vee x_2$
  - Si on obtient une clause $\ll$ vide $\gg$ (i.e. tous les littéraux sont falsifiés), c'est qu'on a pris une décision incohérente : on revient en arrière et on essaye d'assigner la valeur 0 à x_1
  - Si 0 mène aussi à une incohérence, alors la formule n'est pas satisfiable (aucun modèle)
  - Sinon, on continue jusqu'à avoir trouvé une valeur pour chaque variable, ce qui donne un modèle
- Il existe des techniques pour simplifier la formule
  - Propagation unitaire : s'il y a une clause unitaire (e.g. x ou $\neg y$), on peut propager dans toutes les clauses (x vaut 1 et y vaut 0 partout)
  - Littéraux purs : si une variable apparaît avec la même polarité dans toutes les clauses (par exemple, x apparaît dans certaines clauses mais $\neg x$ n'apparaît nulle part) on peut aussi propager de l'information (on assigne à x la valeur 1 et les clauses correspondantes sont satisfaites)

- 大多数采用DPLL（戴维斯-普特南-洛格曼-洛夫兰）算法的SAT求解器
- 基本原则：
  - 选择一个变量*xi*并为其赋值为真值（例如1）
  - 只要满足包含*xi*的所有条款，即可予以删除
  - 所有包含¬*xi*的子句都可以被简化：*x1* ∨ *x2* ∨ ¬*xi*变为*x1* ∨ *x2*
  - 如果得到一个 空 条款（即所有字面表达式都被证伪），那就说明我们做出了一个不一致的决定：需要回溯并尝试将*x1*赋值为0
  - 若0也存在不一致性，则该公式无法满足（即不存在对应模型）
  - 否则，继续迭代直至为每个变量找到对应值，从而构建出一个模型
- 存在一些技术可以简化公式
  - 单位传播：如果存在一个单位子句（例如*x*或¬*y*），则可以在所有子句中进行传播（*x*在所有地方为1且*y*在所有地方为0）
  - 纯逻辑变量：若某个变量在所有子句中保持相同极性（例如，*x*出现在某些子句中，而¬*x*则完全不存在），我们仍可传递信息（通过赋值`a *x*为 1 且相关条款已满足）

DPLL et recherche arborescente

DPLL partage des points communs avec la recherche arborescente

- États : ensembles de variables assignées ou non à une valeur de vérité
- Actions : Assigner une valeur 0 ou 1 à une variable
- États buts : toutes les variables ont une valeur et toutes les clauses sont satisfaites
- Il existe aussi des heuristiques (e.g. pour choisir en priorité des variables qui permettront de trouver rapidement une solution)

DPLL与树形搜索的共同点

- 状态：一组被赋值或未被赋值为真值的变量集合
- 操作：为变量a赋值0或1
- 目标状态：所有变量均具有值，且所有条件均满足
- 此外还存在启发式方法（例如优先选择某些变量，以便快速找到解决方案）

DPLL et recherche arborescente

DPLL partage des points communs avec la recherche arborescente

- États : ensembles de variables assignées ou non à une valeur de vérité
- Actions : Assigner une valeur 0 ou 1 à une variable
- États buts : toutes les variables ont une valeur et toutes les clauses sont satisfaites
- Il existe aussi des heuristiques (e.g. pour choisir en priorité des variables qui permettront de trouver rapidement une solution)
- Le fait d'utiliser des CNF et des techniques comme la propagation unitaire, l'élimination des littéraux purs (et d'autres techniques plus complexes) permet de résoudre le problème beaucoup plus efficacement qu'avec une recherche arborescente ≪ simple ≫

DPLL与树形搜索的共同点

- 状态：一组被赋值或未被赋值为真值的变量集合
- 操作：为变量a赋值0或1
- 目标状态：所有变量均具有值，且所有条件均满足
- 此外还存在启发式方法（例如优先选择某些变量，以便快速找到解决方案）
- 使用CNF以及单元传播、纯分支消除（以及其他更复杂的技术）能够比树状搜索 简单 更有效地解决问题。

Conclusion sur SAT

- La logique propositionnelle et les solveurs SAT sont des outils très puissants pour modéliser des problèmes complexes et les résoudre ≪ simplement ≫
- Aujourd'hui, les solveurs SAT peuvent traiter (assez rapidement) des modèles avec des milliers (voire millions) de variables et de clauses
- Pour la partie pratique associée à ce cours, vous utiliserez une API Python qui permet de modéliser un problème sous forme de clauses et de le résoudre directement (sans avoir besoin d'écrire soi-même l'algorithme DPLL !)

SAT总结

- 命题逻辑和SAT求解器是建模复杂问题并求解问题的非常强大的工具 简单地
- 如今，SAT求解器能够（相当快速地）处理包含数千甚至数百万个变量和子句的模型。
- 在本课程的实践环节中，您将使用Python API来将问题建模为语句形式，并直接求解（无需自己编写DPLL算法！）

Programmation par Contraintes

约束规划

Principe de base

- Intuition :
  - ensemble de variables qui décrivent un problème
  - ensemble de contraintes (logiques, arithmétiques, etc) sur ces variables
- SAT est une forme de programmation par contraintes avec des restrictions (les variables sont booléennes, et les contraintes des clauses)
- Problème de satisfaction de contraintes (CSP) : représentation de domaines et contraintes plus riches qu'avec SAT
  - domaines : nombres, lettres, couleurs, objets quelconques,...
  - contraintes : de manière générale, n'importe quelle combinaison de valeurs possibles pour les variables. Souvent représentées e.g. par des contraintes arithmétiques ($x \geq y$, $x \neq y$, $x + y \geq z + w$, ...)

基本原理

- 直觉：
  - 描述问题的变量集合
  - 对这些变量施加的约束集合（逻辑、算术等）
- SAT是一种约束编程形式，其特点是变量为布尔值，且约束条件由子句定义。
- 约束满足问题（CSP）：相较于SAT，具有更丰富的域表示和约束条件
  - 领域：数字、字母、颜色、任意对象.....
  - 约束条件：通常指变量可能取值的任意组合。常见形式包括算术约束（$x \geq y$、$x) \neq y$，$x+y \geq z+w$，。）

Principe de base

- Intuition :
  - ensemble de variables qui décrivent un problème
  - ensemble de contraintes (logiques, arithmétiques, etc) sur ces variables
- SAT est une forme de programmation par contraintes avec des restrictions (les variables sont booléennes, et les contraintes des clauses)
- Problème de satisfaction de contraintes (CSP) : représentation de domaines et contraintes plus riches qu'avec SAT
  - domaines : nombres, lettres, couleurs, objets quelconques,...
  - contraintes : de manière générale, n'importe quelle combinaison de valeurs possibles pour les variables. Souvent représentées e.g. par des contraintes arithmétiques ($x \geq y$, $x \neq y$, $x + y \geq z + w$, ...)
- L'utilisation de domaines de variables et de contraintes plus riches qu'avec SAT peut permettre une modélisation plus facile du problème

基本原理

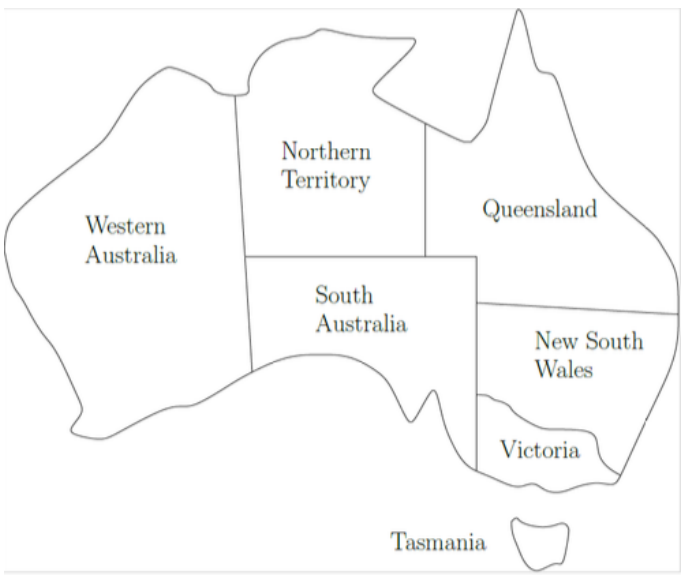
- 直觉：
  - 描述问题的变量集合
  - 对这些变量施加的约束集合（逻辑、算术等）
- SAT是一种约束编程形式，其特点是变量为布尔值，且约束条件由子句定义。
- 约束满足问题（CSP）：相较于SAT，具有更丰富的域表示和约束条件
  - 领域：数字、字母、颜色、任意对象.....
  - 约束条件：通常指变量可能取值的任意组合。常见形式包括算术约束（$x \geq y$、x）$\neq y$，$x+y \geq z+w$，。)
- 相较于SAT，采用更丰富的变量域和约束条件，可使问题建模更为简便。

Exemple classique : coloriage de cartes

- Règles : on doit colorier une carte avec k couleurs, sans que deux pays voisins soient de la même couleur
- Selon la carte et le nombre de couleurs k, il n'y a pas forcément de solution
- Variables : $\{WA, NT, Q, SA, NSW, V, T\}$, avec le même domaine e.g. $\{R, V, B\}$
- Contraintes : $WA \neq NT$, $WA \neq SA$, $NT \neq SA$, $NT \neq Q$, $SA \neq Q$, $SA \neq NSW$, $SA \neq V$, $Q \neq NSW$, $NSW \neq V$
- Solution : affectation de couleurs qui respecte toutes les contraintes

经典示例：地图涂色

- 规则：需要用k种颜色给地图上色，且相邻的两个国家不能用同一种颜色。
- 根据地图和颜色数量k，不一定存在解
- 变量：$\{WA，NT，Q，SA，NSW，V，T\}$，具有相同域，例如$\{R，V，B\}$
- 限制：$WA \neq NT$，华盛顿州$\neq SA$、$NT \neq SA$、$NT \neq Q$，$SA \neq Q$、$SA \neq$ 新南威尔士州、南澳大利亚州$\neq V$、$Q \neq$ 新南威尔士州，新南威尔士州$\neq V$
- 解决方案：符合所有约束条件的配色方案



Pourquoi pas la recherche arborescente ?

- Tout comme pour l'algorithme DPLL des solveurs SAT, les solveurs CSP utilisent une méthode similaire à la recherche arborescente
  - États : combinaisons de couples (variable, valeur)
  - Actions : assigner à une variable une valeur de son domaine
  - États buts : toutes les variables ont une valeur et toutes les contraintes sont satisfaites

为什么不试试树形搜索呢？

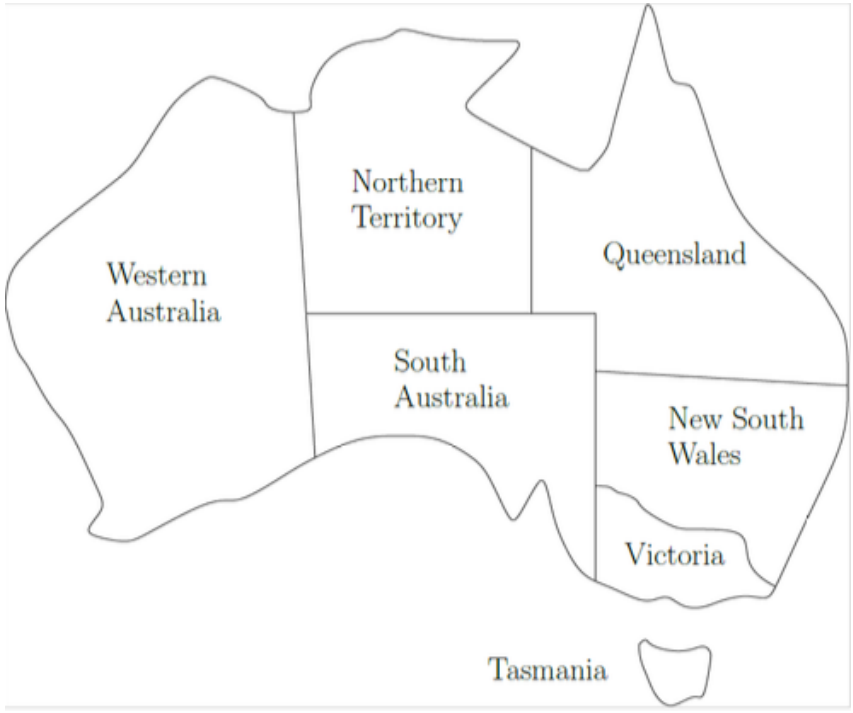
- 与SAT求解器的DPLL算法类似，CSP求解器也采用了类似的树形搜索方法。
  - 状态：配对组合（变量，值）
  - 操作：将变量a赋值为其定义域中的值
  - 目标状态：所有变量均具有数值且所有约束条件均得到满足

Pourquoi pas la recherche arborescente ?

- Tout comme pour l'algorithme DPLL des solveurs SAT, les solveurs CSP utilisent une méthode similaire à la recherche arborescente
  - États : combinaisons de couples (variable, valeur)
  - Actions : assigner à une variable une valeur de son domaine
  - États buts : toutes les variables ont une valeur et toutes les contraintes sont satisfaites
- Et comme pour SAT, des techniques permettent d'accélérer la recherche
  - E.g. quand on assigne une valeur à une variable, ça a un impact sur d'autres variables qui sont reliées par des contraintes (e.g. si $WA = R$, on retire cette valeur du domaine de NT et SA) : possibilité de réduire l'espace de recherche et de détecter des conflits

为什么不试试树形搜索呢？

- 与SAT求解器的DPLL算法类似，CSP求解器也采用了类似的树形搜索方法。
  - 状态：配对组合（变量，值）
  - 操作：将变量a赋值为其定义域中的值
  - 目标状态：所有变量均具有数值且所有约束条件均得到满足
- 与SAT考试类似，也有专门的技巧能帮助加快搜索速度。
  - 例如，当将一个值a赋给变量时，这会影响其他通过约束条件关联的变量（例如，如果WA=R，则从NT和SA的取值域中移除该值）：可以缩小搜索空间并检测冲突



Pourquoi pas SAT ?

- On peut traduire un CSP en problème SAT, par exemple :
  - *wa_bleu*, *wa_rouge*, *wa_vert*, etc : variables booléennes pour indiquer ≪ Western Australia est en bleu/rouge/vert ≫
  - Contraintes logiques : $wa\_bleu \vee wa\_rouge \vee wa\_vert$, $\neg(wa\_bleu \wedge wa\_rouge)$, $\neg(wa\_bleu \wedge wa\_vert)$, $\neg(wa\_rouge \wedge wa\_vert)$, etc
- En général, la modélisation CSP est plus compacte et plus intuitive (on verra des contraintes très riches, difficiles à exprimer en CNF)
- Remarque : quand un encodage SAT est possible, un solveur SAT sera (en général) plus rapide que le solveur CSP
  $\Rightarrow$ compromis entre expressivité pour la modélisation et rapidité de la résolution

为什么不选择SAT呢？

- 例如，可以将CSP转化为SAT问题：
  - *wa__bleu*、*wa__rouge*、*wa__vert*等：布尔变量，用于表示 西澳大利亚州为蓝色/红色/绿色
  - 逻辑约束：$wa\_\_bleu \vee wa\_\_rouge \vee wa\_\_vert$，$\neg(wa\_\_bleu \wedge wa\_\_rouge)$，$\neg(wa\_\_bleu \wedge wa\_\_vert)$，$\neg(wa\_\_rouge \wedge wa\_\_vert)$，等等
- 总体而言，CSP建模方式更为简洁直观（我们将看到一些极其复杂的约束条件，这些约束在CNF中难以表达）
- 注意：当支持SAT编码时，SAT求解器通常比CSP求解器运行更快。

  ☞ 表达力与建模速度与求解速度之间的折衷

Contraintes Globales : All Different

- Contraintes avec un sens précis qui reviennent souvent
- Exemple classique : AllDiff (pour *all different*)
  - Cette contrainte globale porter une un ensemble de variables quelconque, et exprime l'obligation de donner une valeur différente à chaque variable
  - Utile pour des problèmes comme le Sudoku : $x_{i,j}$ représente la valeur de la case sur la ligne i et colonne j (dans $\{1, \ldots, 9\}$)
  - $\texttt{allDiff}(x_{1,1}, x_{1,2}, \ldots, x_{1,9})$: toutes les cases de la première ligne doivent contenir une valeur différente
  - On reproduit cette contrainte pour les autres lignes, les colonnes, et les secteurs 3×3
- Beaucoup plus dur à représenter en SAT !

$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$	$x_{1,5}$	$x_{1,6}$	$x_{1,7}$	$x_{1,8}$	$x_{1,9}$	$\texttt{allDiff}(x_{1,1}, x_{1,2}, \ldots, x_{1,9})$
8			2		3			6	$\texttt{allDiff}(x_{2,1}, x_{2,2}, \ldots, x_{2,9})$
	3			6			7		
		1				6			⋮
5	4						1	9	
		2				7			
	9			3			8		
2			8		4			7	
	1		9		7		6		$\texttt{allDiff}(x_{9,1}, x_{9,2}, \ldots, x_{9,9})$

全球性限制：各不相同

- 具有明确意义的约束条件，这类约束条件经常出现
- 经典示例：AllDiff（全不同）
  - 这个全局约束会作用于一组任意变量，并规定必须为每个变量赋予不同的值。
  - 适用于数独等题目：xi，j表示第i行第j列单元格的值（取值范围为{1，...，9}）
  - allDiff($x1，_1，x1，_2，...，x1，_9$)：第一行的所有单元格都必须包含不同的值
  - 将此约束条件应用于其他行、列及扇区3 × 3
- 在SAT考试中表现得更加吃力！

$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$	$x1.5$	$x1.6$	$x1.7$	$x1.8$	$x1.9$	allDiff($x_{1,1}, x_{1,2}, \ldots, x_{1，9}$)
8			2		3			6	allDiff($x_{2,1}, x_{2,2}, \ldots, x_{2，9}$)
	3			6			7		
		1				6			⋮
5	4						1	9	
		2				7			
	9			3			8		
2			8		4			7	
	1		9		7		6		allDiff($x_{9,1}, x_{9,2}, \ldots, x_{9，9}$)

Contraintes globales : quelques exemples

- La plupart des contraintes globales sont associées à des mécanismes de propagation dédiés : l'arbre de recherche peut être ≪ coupé ≫ pour accélérer la recherche de solutions
- `atLeastNValues`$(n, x_1, \ldots, x_k)$: les variables x_1, ..., x_k doivent avoir au moins n valeurs différentes
- `allEqual`$(x_1, \ldots, x_k)$: l'inverse du `allDiff`
- `disjoint`$(x_1, \ldots, x_k, y_1, \ldots, y_l)$: chaque variable x_i doit avoir une valeur distincte de toutes les valeurs des variables y_i
- `increasingSum`$(x_1, \ldots, x_k, y)$: les variables x_1, ..., x_k ont des valeurs rangées dans l'ordre croissant, et leur somme est égale à y
- Etc
- Un catalogue des (423) contraintes étudiées dans la littérature est disponible : `https://sofdem.github.io/gccat/`

全球性约束：若干实例

- 大多数全局约束条件都与传播机制相关：搜索树可以被 截断 以加速解的搜索
- atLeastNValues(n，$x_1, \ldots, x_k$)：变量$x_1, \ldots, x_k$至少需要n个不同的值
- 全相等（$x_1, \ldots, x_k$）：全不等的逆运算
- disjoint($x_1, \ldots, x_k, y_1, \ldots, y_l$)：每个变量$x_i$必须具有一个与所有变量$y_i$的值不同的值
- increasingSum($x_1, \ldots, x_k$，y)：变量$x_1, \ldots, x_k$的值按升序排列，且它们的总和等于y
- 等等
- 文献中研究的（423）种约束条件目录可获取：`https://sofdem.github.io/gccat/`

Optimisation

- Dans certains cas, toutes les solutions n'ont pas la même « valeur »
  - Coût à minimiser
  - Utilité à maximiser
- Exemple :
  - Un parent partage 30 euros d'argent de poche entre ses enfants, A (14 ans), B (12 ans) et C (10 ans)
  - Il n'y a que des billets de 5 euros, le parent donne un peu plus d'argent en fonction de l'âge, et il ne dépasse pas 20 euros par enfant
  - On peut exprimer cela comme un CSP avec trois variables A, B, C dont le domaine est $\{0, 5, \ldots, 20\}$ et les contraintes $A \geq B + 5$, $B \geq C + 5$, $A + B + C = 30$
  - Deux solutions : $A = 20, B = 10, C = 0$ et $A = 15, B = 10, C = 5$
  - Pour être plus équitable, le parent peut souhaiter maximiser la valeur de C (pour limiter la jalousie entre les enfants), dans ce cas la deuxième solution est meilleure

优化

- 在某些情况下，并非所有解决方案都具有相同的 值
  - 最小化成本
  - 效用最大化
- 示例：
  - 一位家长将30欧元零用钱分给了他的孩子，A（14岁）、B（12岁）和C（10岁）
  - 只有5欧元的票，家长会根据孩子的年龄多给一些钱，但每个孩子的花费不会超过20欧元。
  - 可以将其表示为一个具有三个变量A、B、C的CSP，其取值域为{0,5，...，20}，约束条件为$A \geq B+5$、$B \geq C+5$、$A+B+C=30$
  - 两种解决方案：A=20，B=10，C=0 和A=15，B=10，C=5
  - 为了更“公平”，家长可能希望最大化C的数值（以减少孩子间的嫉妒），这种情况下第二种方案会更合适

Optimisation

- Dans certains cas, toutes les solutions n'ont pas la même « valeur »
  - Coût à minimiser
  - Utilité à maximiser
- Exemple :
  - Un parent partage 30 euros d'argent de poche entre ses enfants, A (14 ans), B (12 ans) et C (10 ans)
  - Il n'y a que des billets de 5 euros, le parent donne un peu plus d'argent en fonction de l'âge, et il ne dépasse pas 20 euros par enfant
  - On peut exprimer cela comme un CSP avec trois variables A, B, C dont le domaine est $\{0, 5, \ldots, 20\}$ et les contraintes $A \geq B + 5$, $B \geq C + 5$, $A + B + C = 30$
  - Deux solutions : $A = 20, B = 10, C = 0$ et $A = 15, B = 10, C = 5$
  - Pour être plus équitable, le parent peut souhaiter maximiser la valeur de C (pour limiter la jalousie entre les enfants), dans ce cas la deuxième solution est meilleure
- Formellement, un problème d'optimisation de contraintes (COP) est un CSP auquel on ajoute une fonction objectif, c'est-à-dire une fonction qui dépend des variables du CSP et dont on doit maximiser (ou minimiser) la valeur

优化

- 在某些情况下，并非所有解决方案都具有相同的 值
  - 最小化成本
  - 效用最大化
- 示例：
  - 一位家长将30欧元零用钱分给了他的孩子，A（14岁）、B（12岁）和C（10岁）
  - 只有5欧元的票，家长会根据孩子的年龄多给一些钱，但每个孩子的花费不会超过20欧元。
  - 可以将其表示为一个具有三个变量A、B、C的CSP，其取值域为{0,5，...，20}，约束条件为$A \geq B+5$、$B \geq C+5$、$A+B+C=30$
  - 两种解决方案：A=20，B=10，C=0 和A=15，B=10，C=5
  - 为了更“公平”，家长可能希望最大化C的数值（以减少孩子间的嫉妒），这种情况下第二种方案会更合适
- 从数学定义上讲，约束优化问题（COP）是在约束满足问题（CSP）基础上增加目标函数的变体。该目标函数是基于CSP变量定义的函数，需要通过优化其数值来实现最大化或最小化。

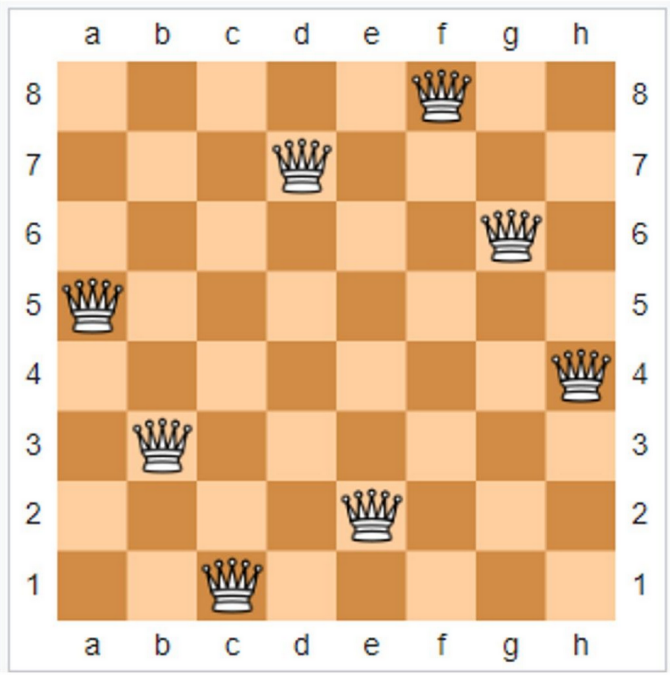
Conclusion

- SAT et CSP sont dans le domaine de la programmation déclarative : on décrit le problème à résoudre (sous forme de clauses/contraintes) et un outil existant le résoud pour nous !
- Étape clé : la modélisation
  - choix des variables (booléennes ou dans des domaines discrets)
  - définition des contraintes
- Les algorithmes pour SAT et CSP peuvent être vus comme des formes améliorées de recherche arborescente, qui permettent dans certains cas de gagner du temps de façon exponentielle
- Remarque : il existe beaucoup de formes plus complexes (plusieurs formes d'optimisation, utilisation de domaines continus pour les variables, etc)

结论

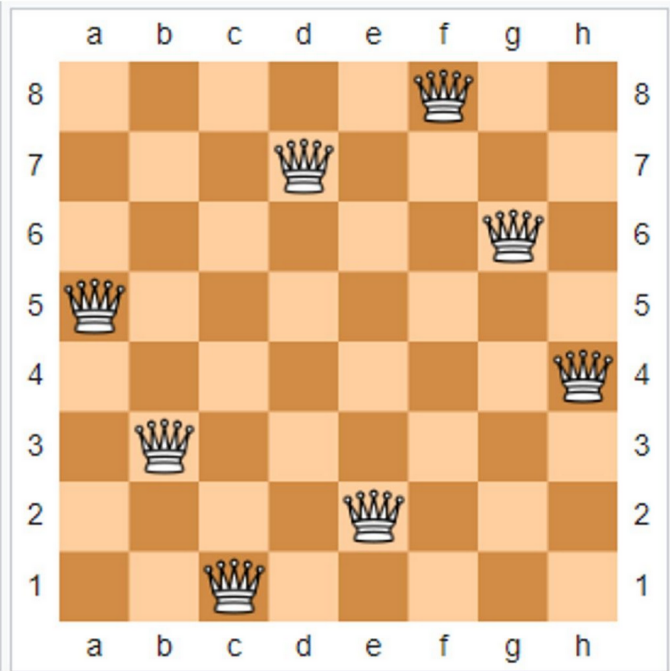
- SAT和CSP都属于声明式编程领域：我们只需用约束条件的形式描述待解决问题，剩下的就交给现成的工具来搞定啦！
- 关键步骤：建模
  - 变量选择（布尔型或离散域）
  - 约束条件的定义
- SAT和CSP算法可视为树形搜索的改进形式，这类算法在某些情况下能以指数级速度显著提升搜索效率。
- 注意：存在许多更复杂的形态（包括多种优化形式、变量连续域的应用等）

Exercice : le problème des n reines



- On doit disposer n reines sur un échiquier de taille $n \times n$, en s'assurant qu'elles ne sont pas en mesure de se menacer les unes les autres
- Rappel : aux échecs, une reine peut se déplacer (et donc menacer d'autres pièces) sur n'importe quelle distance, en ligne, en colonne ou en diagonale
- On considère le cas $n = 8$. Essayez de modéliser ce problème en SAT et en CSP

练习：n皇后问题



- 需要在n个皇后上放置于一个$n \times n$大小的棋盘上，同时确保它们之间不会互相威胁
- 国际象棋中，王后可自由移动（从而威胁其他棋子），其移动距离不受限制，可沿直线、纵线或斜线进行。
- 我们以n= 8的情况为例，尝试将该问题建模为SAT和CSP问题。