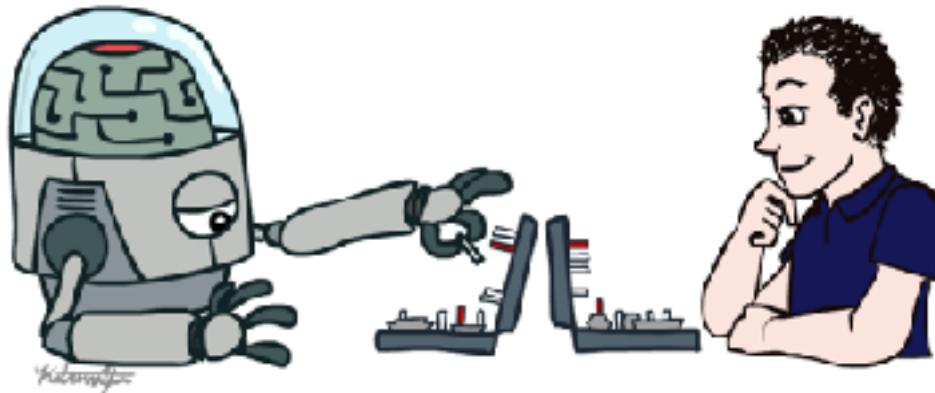


Intelligence Artificielle

Khaled Ben Lamine



Plan

- Introduction
- Recherche
 - ◆ Recherche heuristique
 - ◆ Recherche pour les jeux compétitifs
 - ◆ Constraint Satisfaction Problems (backtracking search)
- Representation de connaissance
 - ◆ Logique du premier ordre
 - ◆ Plannification
- Raisonnement probabiliste (Bayes nets)

AI a un impact sur la réalité

- ◆ imagination publique
 - » assistants textuels



AI a un impact sur la réalité

- ◆ imagination
 - » assistants textuels
 - » Génération d'images

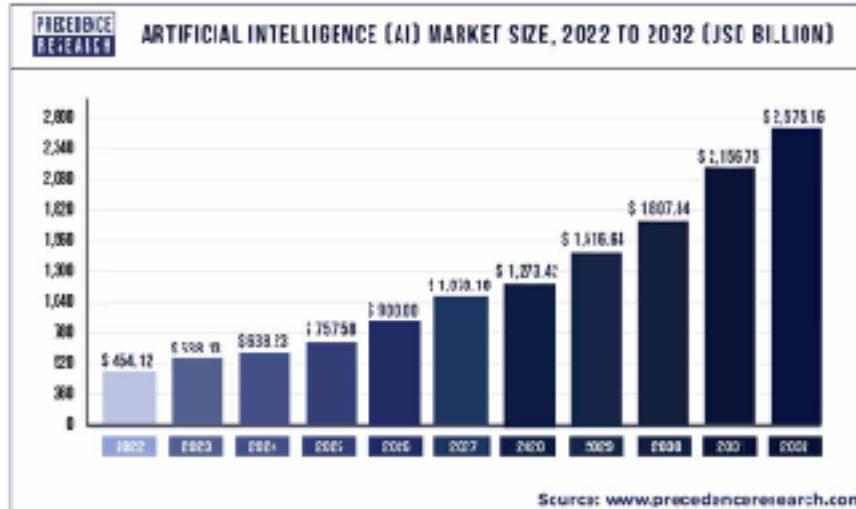


AI a un impact sur la réalité

◆ Economie

» 454 Milliard de \$

The global artificial intelligence (AI) market size was valued at USD 454.12 billion in 2022 and is expected to hit around USD 2,575.16 billion by 2032, progressing with a CAGR of 19% from 2023 to 2032. The North America artificial intelligence market was valued at USD 167.30 billion in 2022.



AI a un impact sur la réalité

- ◆ imagination
- ◆ Economie
- ◆ Politique



AI a un impact sur la réalité

- ◆ imagination
- ◆ Economie
- ◆ Politique
- ◆ Travail

Finance & economics | Free exchange

New research shows the robots are coming for jobs—but stealthily

Look beneath the aggregate economic numbers, and change is afoot

The Optimist's Guide to Artificial Intelligence and Work

The focus of much discussion is on how it will replace jobs, but nothing is inevitable.

AI a un impact sur la réalité

- ◆ imagination
- ◆ Economie
- ◆ Politique
- ◆ Travail
- ◆ Sciences

nature
BIOTECH

AlphaFold Developers Win \$3-Million Breakthrough Prize in Life Sciences

DeepMind's system for predicting the 3D structure of proteins is among five recipients of awards

By 22.2022

DeepMind Has Trained an AI to Control Nuclear Fusion

The company has trained a reinforcement learning algorithm to control the plasma inside a tokamak reactor.



AI a un impact sur la réalité

- ◆ imagination
- ◆ Economie
- ◆ Politique
- ◆ Travail
- ◆ Sciences
- ◆ Education

BREAKING

ChatGPT In Schools: Here's Where It's Banned—And How It Could Potentially Help Students

Arianna Johnson Forbes Staff
I cover the latest trends in science, tech and healthcare.

Follow

2

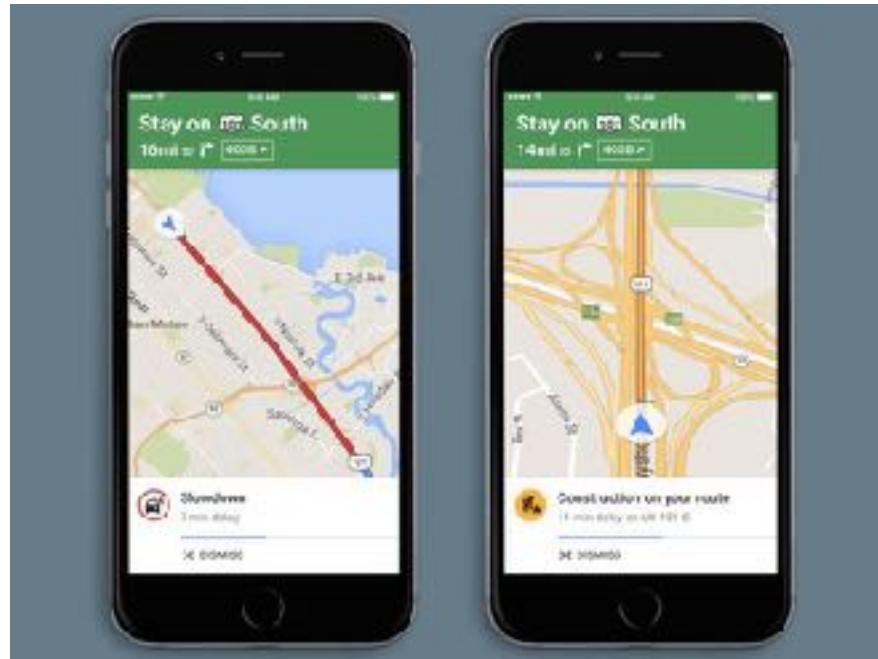
Jan 18, 2023, 02:31pm EST

AI & science fiction



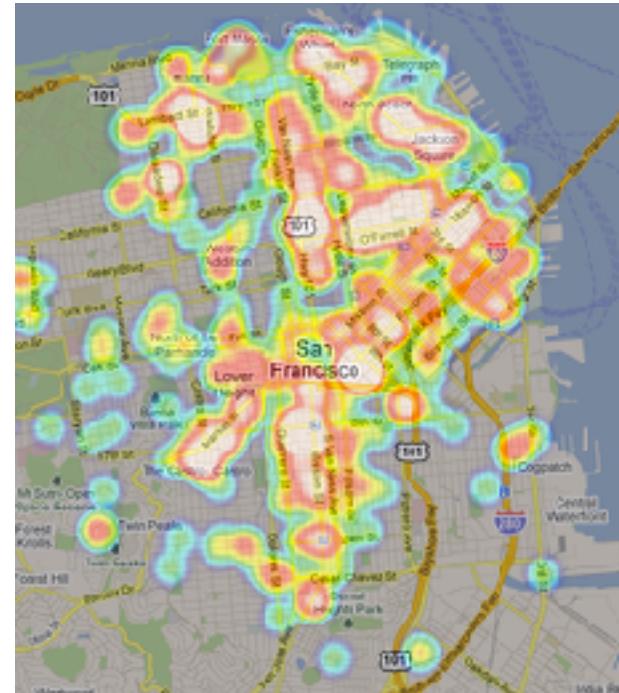
Google maps

- Utilisation de données anonymes de localisation pour analyser le traffic et ainsi suggérer les routes les plus rapides



Uber

- Utilise ML pour déterminer prix de la course, minimiser temps d'attente, etc.



Voiture Autonome

- Voiture électrique Autonome



Email



- Filtre SPAM
 - google filtre 99% des spams
- Catégorisation des emails
- Google Allo , Réponse automatique, Chatbots



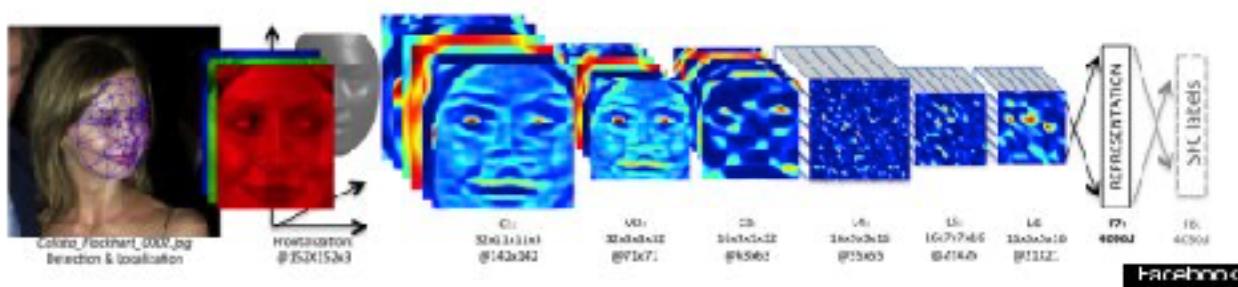
Banque

- Détection de fraude
- Décision de crédit
- Score



Facebook/Réseaux sociaux

- Suggestion de tags d'amis



- Pinterest : détection des objets d'intérêt dans une image et suggérer des pins similaires

Applications robotiques



asimo.honda.com

- Plusieurs prototypes de robots, de plus en plus agiles, visant plusieurs domaines d'application.

Applications robotiques



- Plusieurs prototypes de robots, de plus en plus agiles, visant plusieurs domaines d'application.

Applications robotiques

- Planification de trajectoires pour un corps articulé, avec évitement d'obstacles



Bras-robot canadien



Station de contrôle

Applications dans les jeux

- Intelligence artificielle dans les jeux



Deep Blue
IBM

Applications dans les jeux

- Google AlphaGo



AlphaGo

Applications dans les jeux



IA Générative

- Les services de génération d'images et d'œuvres d'art (**Midjourney, DALL-E et Stable Diffusion**)



Une image créée avec le système Midjourney selon les instructions suivantes : "A black old lady playing chess with a child in a NYC park, at sunset, digital art, 4k" (MIDJOURNEY / FRANCEINFO)



Une image créée avec le système Dall-E 2 selon les instructions suivantes : "A crowd of individuals staring at their phones, not noticing huge explosions in the background, in a renaissance-style painting, high details" (DALL-E 2 / FRANCEINFO)

IA Générative

- Les générateurs d'avatars stylisés (**Lens, AI Time Machine**)



IA Générative

- Le logiciel qui donne l'impression de discuter avec un humain (**ChatGPT**)
- Generative Pretrained Transformer (GPT)

K Explain quantum computing in simple terms

K How do I make an HTTP request in Javascript?

Q Quantum computing is a type of computation that uses quantum-mechanical phenomena such as superposition and entanglement to perform calculations. Unlike a classical computer, data is represented using quantum bits (qubits) which can be in multiple states simultaneously, representing a state of 0, 1, or a combination of both. This allows quantum computers to perform certain types of calculations much faster than classical computers, especially for problems like integer factorization of large integers, searching unsorted databases, and simulating quantum systems.

Q There are several ways to make an HTTP request in JavaScript, including using the built-in `XMLHttpRequest` object, or the more modern `fetch()` API. Here is an example of using `fetch()` to make a GET request to a JSON endpoint:

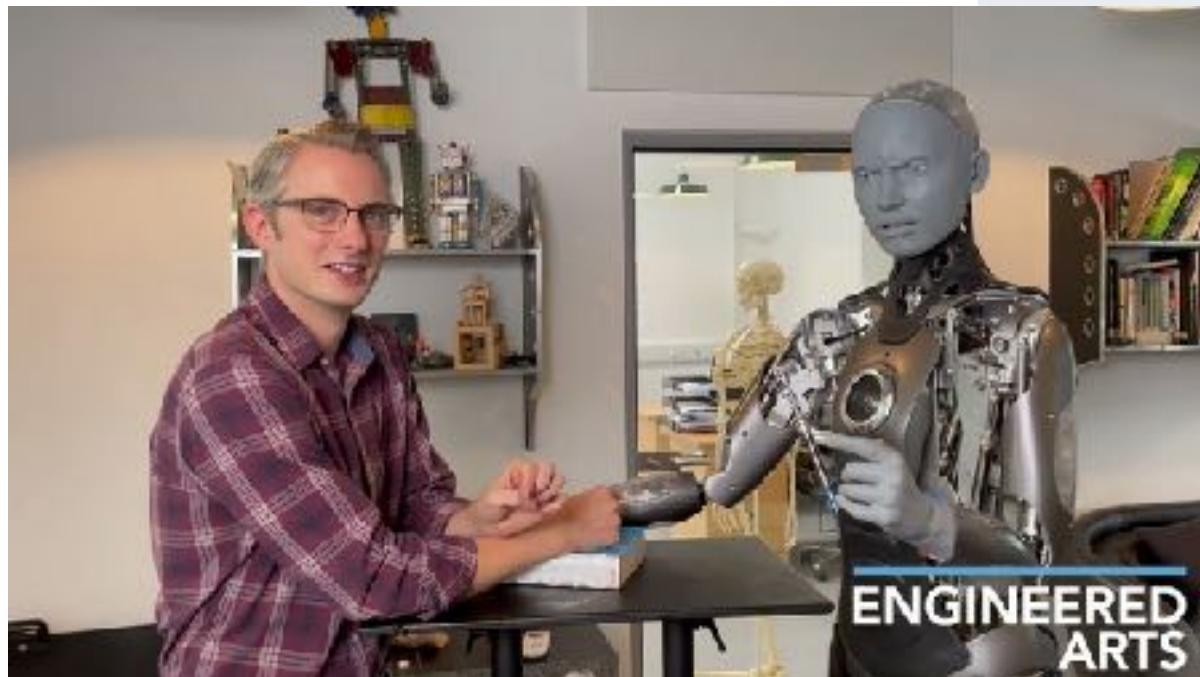
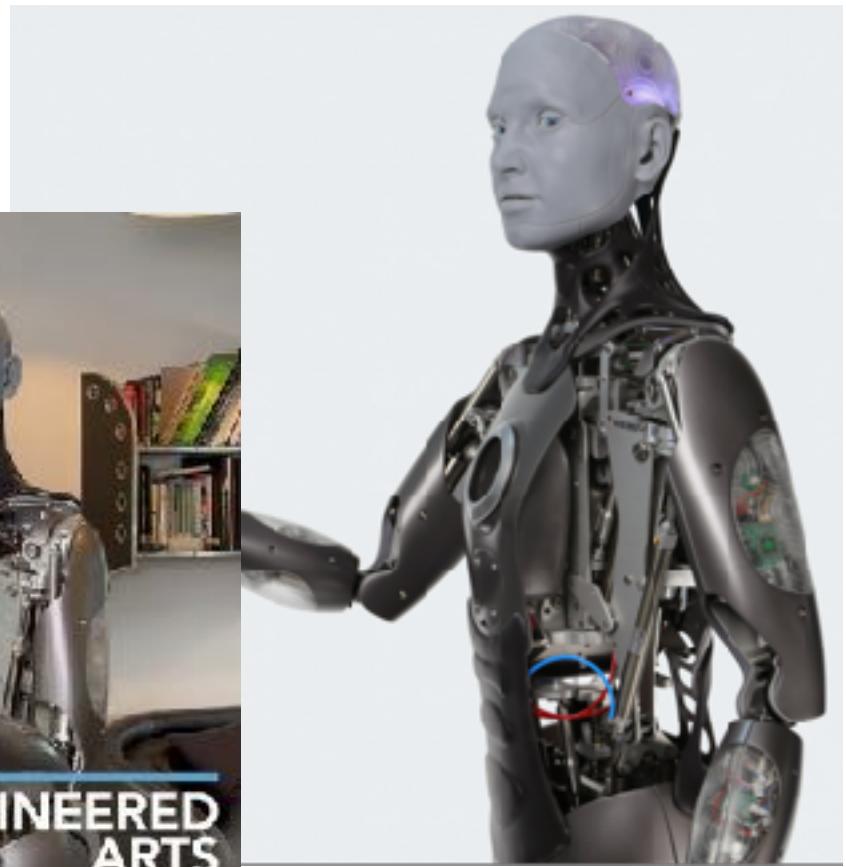
```
fetch('https://jsonplaceholder.typicode.com/posts/1')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error(error));
```

In this example, `fetch()` is called with the URL of the JSON endpoint as the first argument. The returned promise is resolved with a `Response` object, which has a `json()` method that returns a promise that resolves with the parsed JSON data. The `then()` method is used to handle the successful resolution of the promise, and the `catch()` method is used to handle any errors that occur.

You can also use other methods like POST, PUT, DELETE etc using `fetch()`

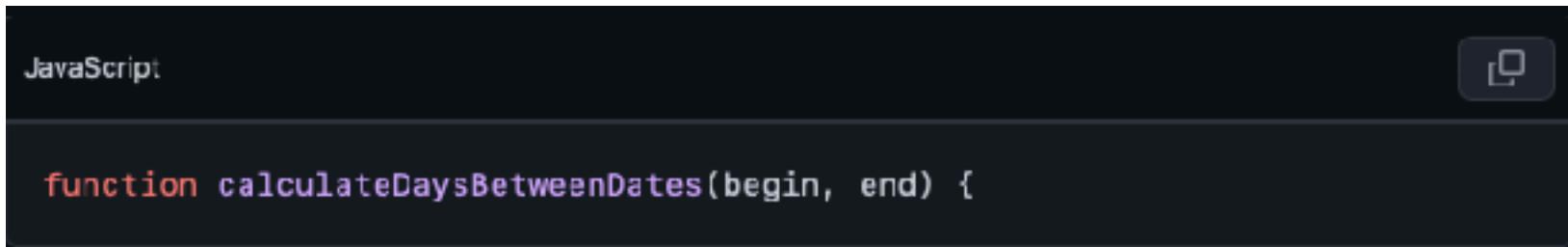
IA Générative

- Le logiciel qui donne l'impression de discuter avec un humain + robot (ChatGPT + Ameca)



IA Générative

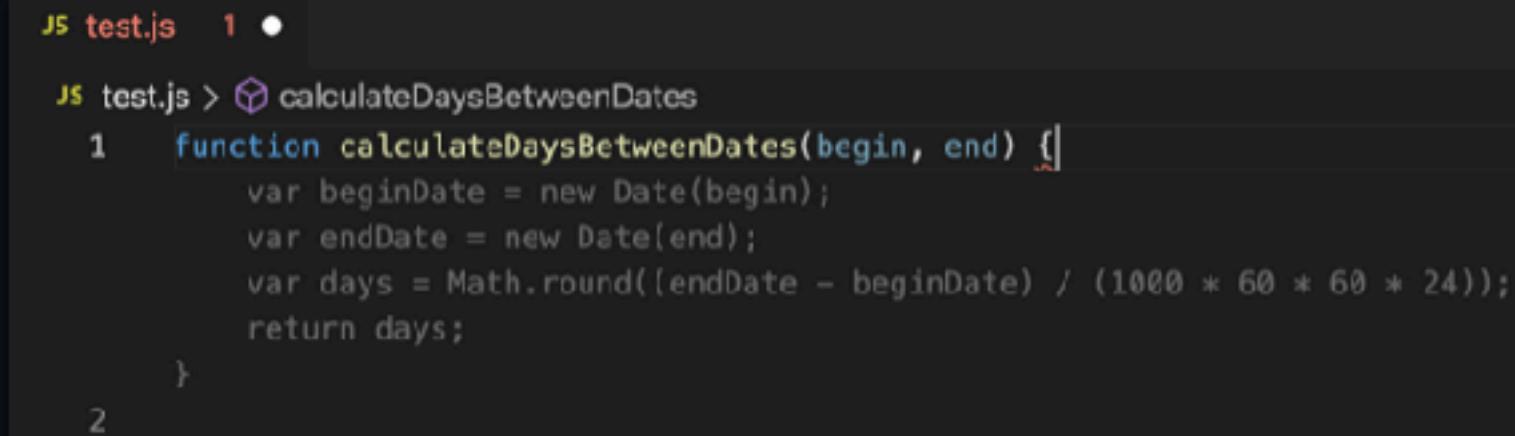
- Logiciel d'aide à la programmation ([GitHub Copilot](#))



The screenshot shows a dark-themed code editor window for JavaScript. At the top left, it says "JavaScript". At the top right, there's a small icon. The main area contains the following code:

```
function calculateDaysBetweenDates(begin, end) {
```

GitHub Copilot will automatically suggest an entire function body in grayed text, as shown below. The exact suggestion may vary.



The screenshot shows a terminal window with the command "JS test.js" followed by a cursor. The terminal output shows:

```
JS test.js 1 ●
```

Then, the user types "JS test.js > calculateDaysBetweenDates" and the terminal displays the function definition with some parts in grayed text, indicating a suggestion:

```
JS test.js > calculateDaysBetweenDates
1   function calculateDaysBetweenDates(begin, end) {
    var beginDate = new Date(begin);
    var endDate = new Date(end);
    var days = Math.round((endDate - beginDate) / (1000 * 60 * 60 * 24));
    return days;
}
```

At the bottom left, the number "2" is visible.

IA

Quiz: de quoi est capable l'IA Aujourd'hui?

Gagné les meilleurs joueurs au jeu d'échec ?

Gagné les meilleurs joueurs de Go?

Joué au tennis de table ?

Vider un lave vaisselle ?

Conduire sûrement dans les autoroutes ?

Conduire sûrement dans les villes ?

Commander des courses en ligne ?

Acheter des courses d'un magasin ?

Découvrir et démontrer des nouveaux théorème?

Faire une opération chirurgicale ?

Traduire en temps réel du chinois ?

Gagner une compétition d'art ?

Ecrire une histoire drôle?

Construire un immeuble?

IA

Quiz: de quoi est capable l'IA Aujourd'hui?

- Gagné les meilleurs joueurs au jeu d'échec ?
- Gagné les meilleurs joueurs de Go ?

Joué au tennis de table ?

Vider un lave vaisselle ?

Conduire sûrement dans les autoroutes ?

Conduire sûrement dans les villes ?

Commander des courses en ligne ?

Acheter des courses d'un magasin ?

Découvrir et démontrer des nouveaux théorème?

Faire une opération chirurgicale ?

Traduire en temps réel du chinois ?

Gagner une compétition d'art ?

Ecrire une histoire drôle?

Construire un immeuble?



IA

Quiz: de quoi est capable l'IA Aujourd'hui?

- Gagné les meilleurs joueurs au jeu d'échec ?
- Gagné les meilleurs joueurs de Go ?
- Joué au tennis de table ?

Vider un lave vaisselle ?

Conduire sûrement dans les autoroutes ?

Conduire sûrement dans les villes ?

Commander des courses en ligne ?

Acheter des courses d'un magasin ?

Découvrir et démontrer des nouveaux théor

Faire une opération chirurgicale ?

Traduire en temps réel du chinois ?

Gagner une compétition d'art ?

Ecrire une histoire drôle?

Construire un immeuble?



IA

Quiz: de quoi est capable l'IA Aujourd'hui?

- Gagné les meilleurs joueurs au jeu d'échec ?
- Gagné les meilleurs joueurs de Go?
- Joué au tennis de table ?
- Vider un lave vaisselle ?
- Conduire sûrement dans les autoroutes ?
- Conduire sûrement dans les villes ?
- Commander des courses en ligne ?
- Acheter des courses d'un magasin ?
- Découvrir et démontrer des nouveaux théorème?
- Faire une opération chirurgicale ?
- Traduire en temps réel du chinois ?
Gagner une compétition d'art ?
- Ecrire une histoire drôle?
- Construire un immeuble?

IA

Quiz: de quoi est capable l'IA Aujourd'hui?

- Gagné les meilleurs joueurs au jeu d'échec ?
 - Gagné les meilleurs joueurs de Go?
 - Joué au tennis de table ?
 - Vider un lave vaisselle ?
 - Conduire sûrement dans les autoroutes ?
 - Conduire sûrement dans les villes ?
 - Commander des courses en ligne ?
 - Acheter des courses d'un magasin ?
 - Découvrir et démontrer des nouveaux t
 - Faire une opération chirurgicale ?
 - Traduire en temps réel du chinois ?
 - Gagner une compétition d'art ?
- Ecrire une histoire drôle?
Construire un immeuble?



IA

Quiz: de quoi est capable l'IA Aujourd'hui?

- Gagné les meilleurs joueurs au jeu d'échec ?
- Gagné les meilleurs joueurs de Go?
- Joué au tennis de table ?
- Vider un lave vaisselle ?
- Conduire sûrement dans les autoroutes ?
- Conduire sûrement dans les villes ?
- Commander des courses en ligne ?
- Acheter des courses d'un magasin ?
- Découvrir et démontrer des nouveaux théorème?
- Faire une opération chirurgicale ?
- Traduire en temps réel du chinois ?
- Gagner une compétition d'art ?
- Ecrire une histoire drôle?
- Construire un immeuble?



AI dans les infos



Elon Musk

@elonmusk

...

Tesla Full Self-Driving Beta is now available to anyone in North America who requests it from the car screen, assuming you have bought this option.

Congrats to Tesla Autopilot/AI team on achieving a major milestone!

11:34 PM · Nov 23, 2022

12.9K Retweets

2,651 Quote Tweets

174.1K Likes



AI dans les infos



Elon Musk

@elonmusk

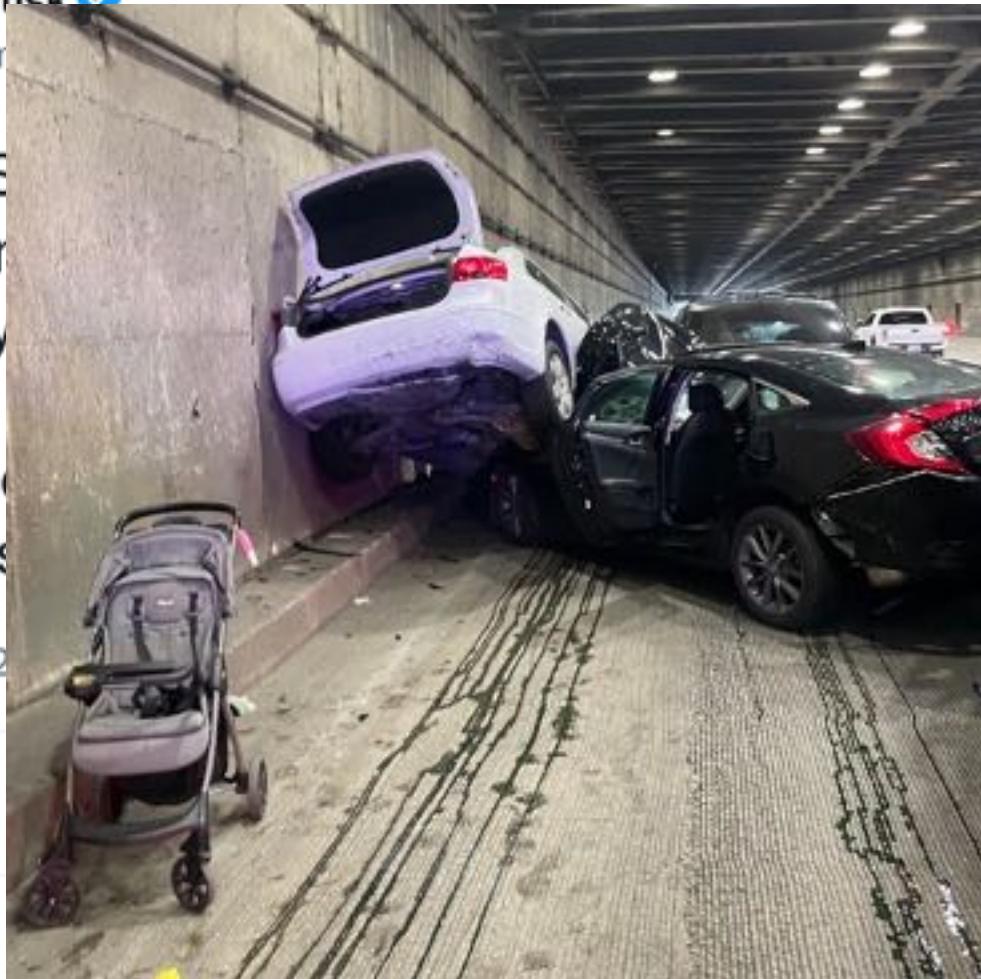
...

Tesla Full S
in North Ar
assuming y

Congrats to
major miles

11:34 PM · Nov 2

12.9K Retweets



anyone
screen,

ng a



Intelligence artificielle

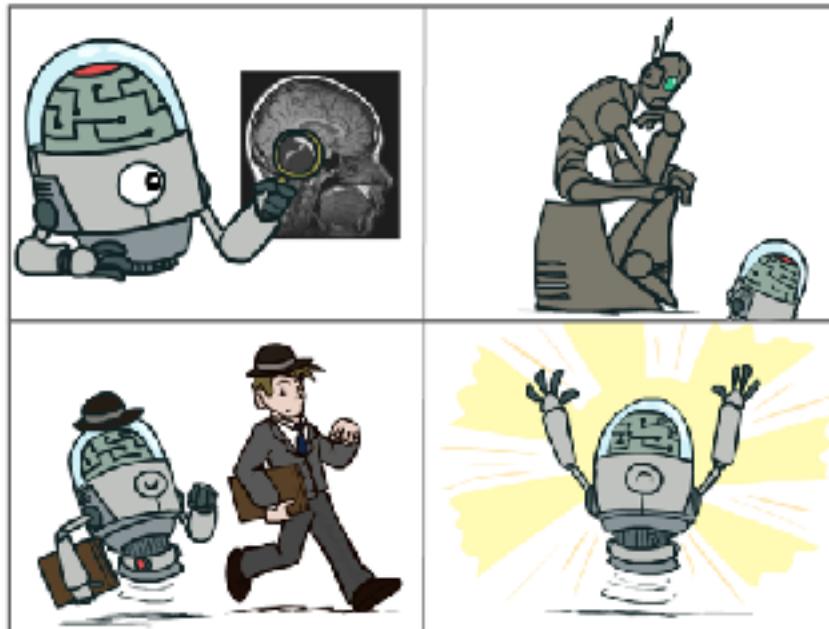
- Selon « *One Hundred Year Study on Artificial Intelligence (AI100)* » <https://ai100.stanford.edu/>
 - ◆ L'intelligence artificielle est à la fois une **science** et un **ensemble de technologies** inspirées—mais typiquement opérant différemment— de la façon dont les humains utilisent leur cerveau et leur corps pour **percevoir/sentir, apprendre, raisonner et agir**.

Intelligence artificielle

Science permettant aux machines de :

**penser comme
des humains**

Sciences cognitives



**agir comme des
humain**

Test de Turing

**penser
rationnellement**

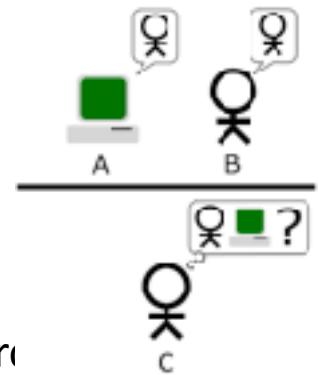
“Right thinking”
syllogism,
logique, proba.

**agir
rationnellement**

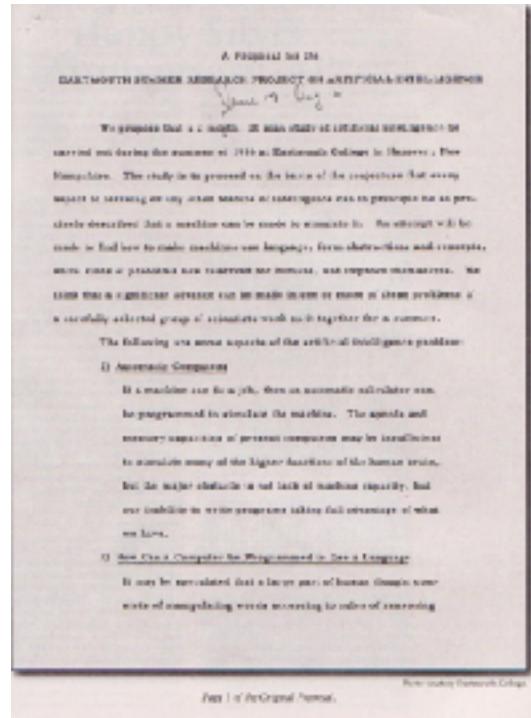
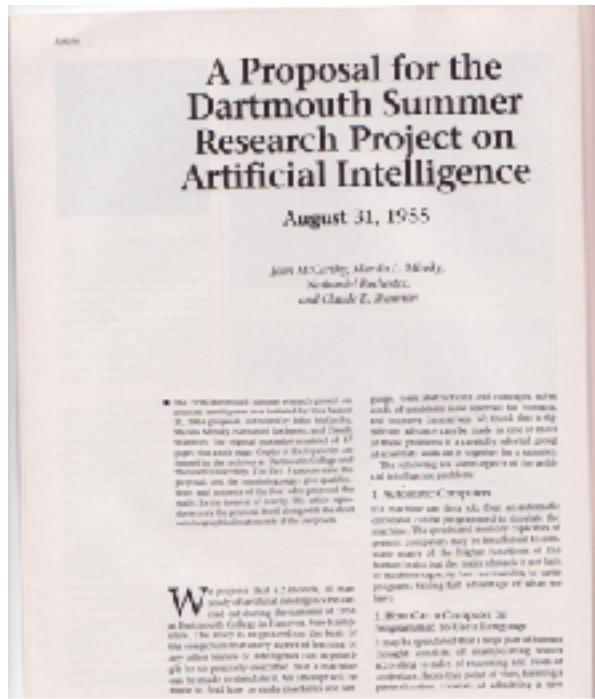
agent rationnel,
“do the right thing”

Comment savoir si une machine est intelligente?

- Test de Turing :
 - ◆ un interrogateur humain pose des questions écrites à une machine et à une personne, les deux cachées par un rideau
 - ◆ si l'interrogateur ne peut distinguer les réponses données par la machine de celles données par la personne, alors la machine est intelligente
- Pour réussir le test, le système a besoin des capacités suivantes :
 - ◆ traitement du langage naturel
 - ◆ représentation des connaissances
 - ◆ raisonnement
 - ◆ apprentissage
- Le test de Turing complet permet les interactions physiques entre l'interrogateur et la machine, ce qui ajoute les capacités de :
 - ◆ perception (pour le test complet)
 - ◆ robotique
- Chacune de ces capacités correspond à une sous-discipline de l'IA



Perspective historique de l'IA



“The proposal [for the meeting] is to proceed on the basis of the conjecture that every aspect of . . . intelligence can in principle be so precisely described that a machine can be made to simulate it”

Perspective historique de l'IA

- De 1956 jusqu'au milieu des années 1980, les recherches en IA sont dominées par des approches à base de connaissances (***knowledge based***).
 - ◆ Critique : L'IA conventionnelle [*knowledge based*] n'est qu'une « application de règles », mais l'intelligence [humaine] ne l'est pas (Haugeland)
- Dès les années 1980, les approches comportementales deviennent populaire (***behaviour based*** ou ***situated AI***).
 - ◆ Leitmotiv : La représentation des connaissances n'est pas nécessaire, elle est même nuisible (Brooks)
- Dès les années 1990, les approches probabilistes deviennent populaire (***Markov decision processes, Hidden Markov Models, Bayesian networks***)
 - ◆ Leitmotiv : L'inférence nécessaire [pour l'IA] est probabiliste, mais pas logique.
- Aujourd'hui, les approches connexionnistes
 - ◆ Leitmotiv : Orienté-données, mieux représentatifs de la réalité du domaine d'application.

Beaucoup d'autres applications

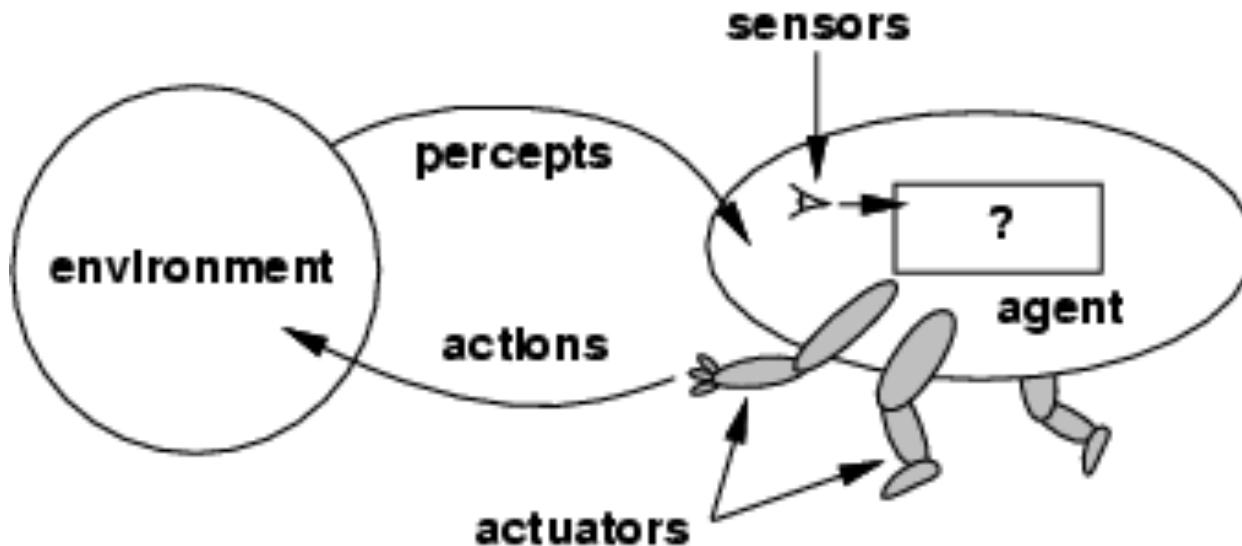
- Vision
- Traitement du langage naturelle
- Domotique
- Véhicules autonomes
- Défense
- Aérospatiale
- Le web
- Etc.

II- Agents Intelligents

- Agents intelligents
- Rationalité
- Modèle générique PEAS de conception des agents
 - ◆ mesure de ***Performance***, modélisation de ***l'Environnement***, et l'implémentation des ***Actionneurs*** ainsi que des ***Senseurs***
- Types d'environnements
 - ◆ Déterministe, stochastique, etc.
- Types d'agents
 - ◆ Reflex, orienté-but, orienté-utilité, etc.

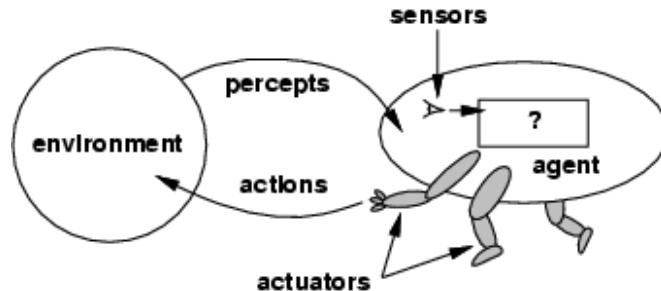
C'est quoi un agent?

- Un agent est n'importe quelle entité qui perçoit son environnement par des **capteurs** (*sensors*) et agit sur cet environnement par des **actionneurs** (*actuators*)

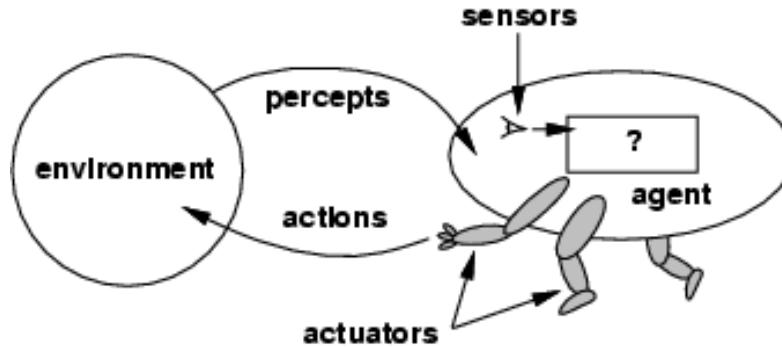


Exemple d'agents

- Agent humain :
 - ◆ Senseurs: yeux, oreilles, odorat, toucher, etc.
 - ◆ Actionneurs: mains, jambes, bouche, etc.
- Agent robot, drone ou voiture autonome :
 - ◆ Senseurs: Odomètre, GPS, caméras, capteurs infra rouges, microphone, etc.
 - ◆ Actionneurs: Roues, jambes, bras-articulés, speaker, etc.
- Agent virtuel comme Siri ou Google Now :
 - ◆ Senseurs : caméra, microphone, GPS, etc.
 - ◆ Actionneurs : affichage écran, speaker, vibration, etc.



Fonction mathématique « agent »

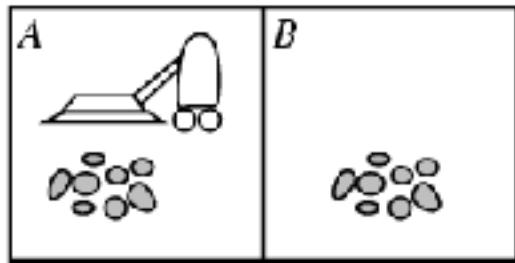


- La ***fonction agent*** f prend en entrée une séquence d'**observations** (percepts) et retourne une **action** :

$$f : P^* \rightarrow A$$

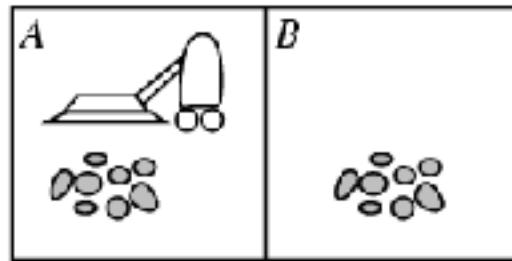
- En pratique la fonction est implémentée par un programme sur une architecture matérielle particulière

Exemple : Aspirateur robotisé



- Observations (données sensorielles) : position et état des lieux
Par exemple : $[A, Clean]$,
 $[A, Dirty]$,
 $[B, Clean]$,
- Actions : *Left, Right, Suck, NoOp*

Exemple : Aspirateur robotisé



- $f :$
 - $[A, Clean] \rightarrow Right$
 - $[A, Dirty] \rightarrow Suck$
 - ...
 - $[A, Clean] [A, Clean] [A, Dirty] \rightarrow Suck$
 - $[A, Clean] [A, Clean] [A, Clean] \rightarrow Right$
 - ...

Agents rationnels

- Un agent rationnel doit agir « correctement » en fonction de ce qu'il perçoit et de ses capacités d'action :
 - ◆ l'**action correcte** est celle permettant à l'agent de réussir le mieux
- **Mesure de performance** :
 - ◆ une fonction objective mesurant la qualité d'un comportement de l'agent
- Par exemple, une mesure de performance pour un robot aspirateur pourrait être :
 - ◆ la quantité de déchets aspirés
 - ◆ la propreté des lieux
 - ◆ la durée de la tâche
 - ◆ le bruit généré
- **Agent rationnel** : étant donné une séquence d'observations (données sensorielles) et des connaissances propres, un agent rationnel devrait **choisir une action qui maximise la mesure de performance**

Agents rationnels

- **Rationalité ne veut pas dire « qui sait tout »**
(par exemple, connaît tous les effets de ses actions)!
- **Rationnel ne veut pas dire « parfait »**
 - ◆ la rationalité maximise la performance escomptée
 - ◆ la perfection maximise la performance réelle
 - ◆ mais souvent on ne peut pas connaître la performance réelle avant l'action
- **La rationalité est mesurée par le résultat** —la décision /action prise —et non le processus de calcul de la décision.
- Un agent peut effectuer des actions d'observation pour cueillir des informations nécessaires à sa tâche
- Un agent est **autonome** s'il est capable d'adapter son comportement aux changements dans l'environnement (capable d'apprendre, de planifier, de raisonner)

Modèle PEAS

- PEAS : Un modèle générique de conception des agents par la spécification des composantes suivantes :
 - ◆ mesure de **performance**
 - ◆ éléments de l'**environnement**
 - ◆ les **actions** que l'agent peut effectuer (**Actionneurs**)
 - ◆ la séquence des **observations** ou **percepts** de l'agent (**Senseurs**)
- **PEAS = *Performance, Environnement, Actuateurs, Senseurs***

Exemple : Modèle PEAS pour voiture autonome

- **Agent** : Voiture autonome
- **Mesure de performance** : sécurité, vitesse, respect du code routier, voyage confortable, maximisation des profits (pour un taxi)
- **Environnement** : route, trafic, piétons, clients
- **Actionneurs** : volant, changement de vitesse, accélérateur, frein, clignotants, klaxon
- **Senseurs** : caméras, sonar, GPS, odomètre, compteur de vitesse, témoins du moteur, etc.



Exemple : Modèle PEAS pour Pacman

- **Agent** : Pacman
- **Mesure de performance** : score
- **Environnement** : le labyrinthe, les biscuits, les fantômes
- **Actionneurs** : se déplacer, manger, crier
- **Senseurs** : senseur de fantômes, senseur de biscuits, senseur pour la position,



Caractéristiques d'environnement

- Différents problèmes auront des environnements avec des caractéristiques différentes
- Caractéristiques que l'on distingue:
 - ◆ **Complètement observable** (vs. partiellement observable)
 - ◆ **Déterministe** (vs. stochastique)
 - ◆ **Épisodique** (vs. séquentiel)
 - ◆ **Statique** (vs. dynamique)
 - ◆ **Discret** (vs. continu)
 - ◆ **Agent unique** (vs. multi-agent)

Caractéristiques d'environnement

- **Complètement observable** (vs. partiellement observable) : grâce à ses capteurs, l'agent a accès à l'état complet de l'environnement à chaque instant
- Le jeu des échecs est complètement observable
 - ◆ on voit la position de toutes les pièces
- Le jeu du poker est partiellement observable
 - ◆ on ne connaît pas les cartes dans les mains de l'adversaire

Caractéristiques d'environnement

- **Déterministe** (vs. stochastique) : l'état suivant de l'environnement est entièrement déterminé par l'état courant et l'action effectuée par le ou les agents
- Le jeu des échecs est déterministe
 - ◆ déplacer une pièce donne toujours le même résultat
- Le jeu du poker est stochastique
 - ◆ la distribution des cartes est aléatoire
- **Notes importantes :**
 - ◆ on considère comme stochastique les phénomènes qui ne peuvent pas être prédits parfaitement
 - ◆ on ne tient pas compte des actions des autres agents pour déterminer si déterministe ou pas

Caractéristiques d'environnement

- **Épisodique** (vs. séquentiel) : les opérations/comportements de l'agent sont divisés en épisodes :
 - ◆ chaque épisode consiste à observer l'environnement et effectuer une seule action
 - ◆ cette action n'a pas d'influence sur l'environnement dans l'épisode suivant
- La reconnaissance de caractères est épisodique
 - ◆ la prédiction du système n'influence pas le prochain caractère à reconnaître
- Le jeu du poker est séquentiel
 - ◆ décider si je mise ou pas a un impact sur l'état suivant de la partie

Caractéristiques d'environnement

- **Statique** (vs. dynamique) : l'environnement ne change pas lorsque le ou les agents n'agissent pas
- Le jeu des échecs est statique
 - ◆ l'état du jeu ne change pas si personne joue
- Le jeu de stratégie en temps réel, comme StarCraft, est dynamique
 - ◆ Les unités ont une certaine autonomie; elles peuvent évoluer même si aucun joueur ne fait une action.

Caractéristiques d'environnement

- **Discret** (vs. continu) : un nombre limité et clairement distincts de **données sensorielles et d'actions**
- Le jeu des échecs est dans un environnement discret
 - ◆ toutes les actions et état du jeu peuvent être énumérées
- La conduite automatique d'une voiture est dans un environnement continu
 - ◆ l'angle du volet est un nombre réel

Caractéristiques d'environnement

- **Agent unique** (vs. multi-agent) : un agent opérant seul dans un environnement
- Résoudre un Sudoku est à agent unique
 - ◆ aucun adversaire
- Le jeu des échecs est multi-agent
 - ◆ il y a toujours un adversaire

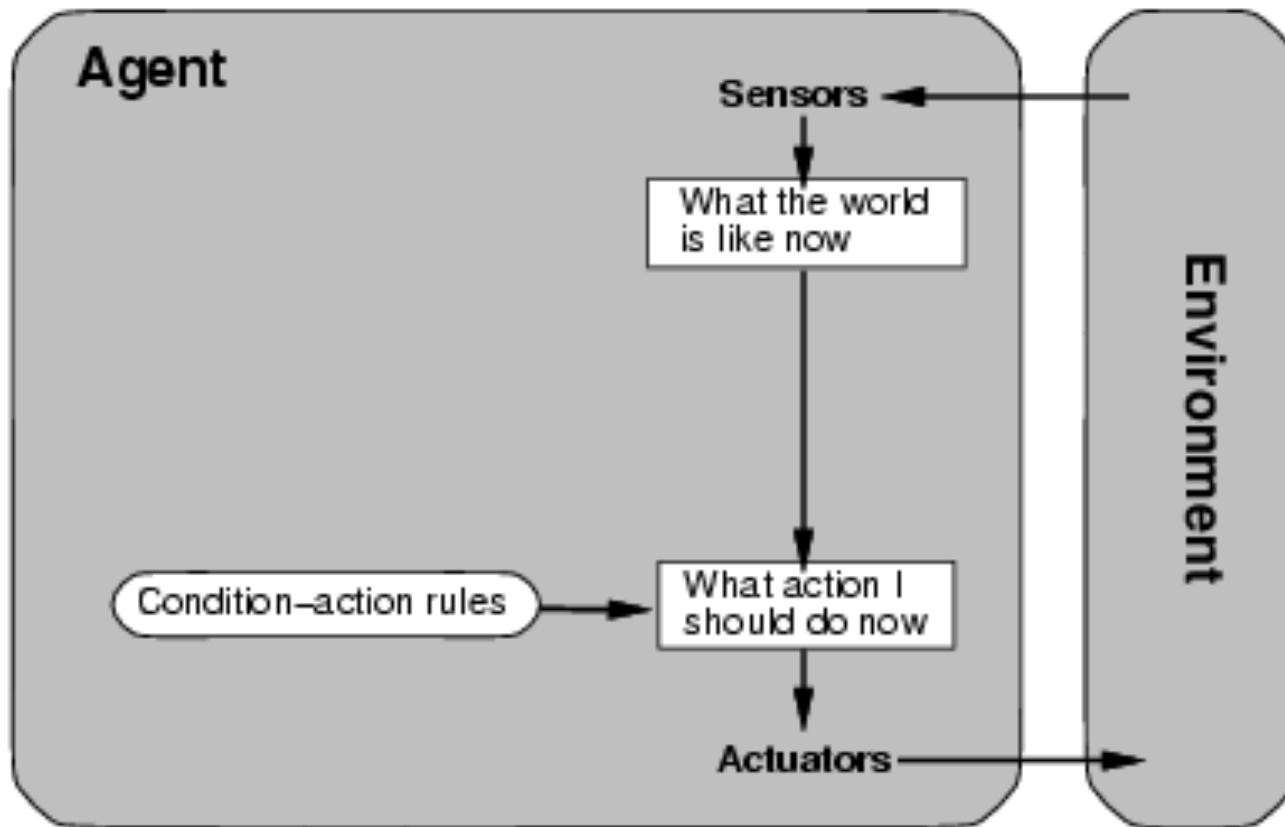
Caractéristiques d'environnement

- Parfois, plus d'une caractéristique est appropriée
- Déplacement d'un robot
 - ◆ si seul dans un environnement, ses déplacements sont théoriquement déterministes (la physique mécanique est déterministe)
 - ◆ par contre, puisqu'un robot ne contrôle pas parfaitement ses mouvements, on préfère normalement modéliser comme stochastique
- On identifie souvent les caractéristiques d'environnement en réfléchissant à **comment on programmerait/simulerait cet environnement**

Architectures des agents

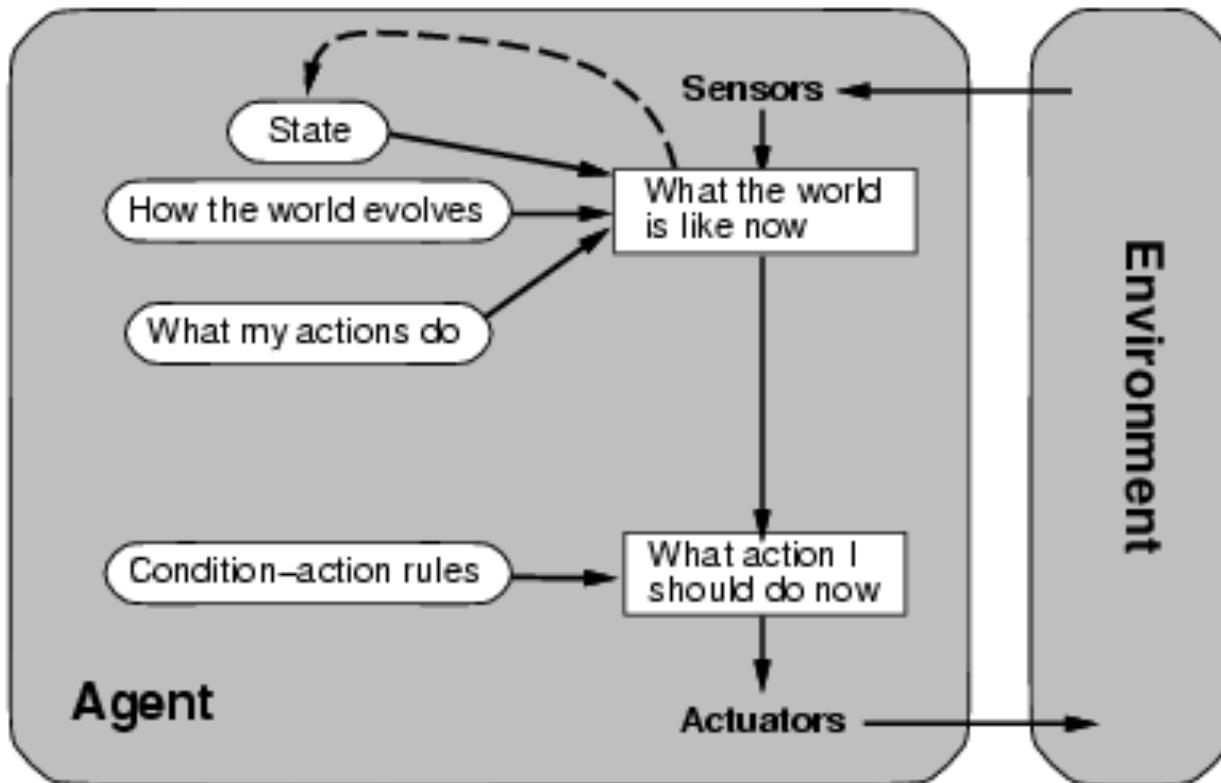
- Simple reflex agents
- Model-based reflex agents
- Goal-based agents
- Utility-based agents

Simple reflex agents



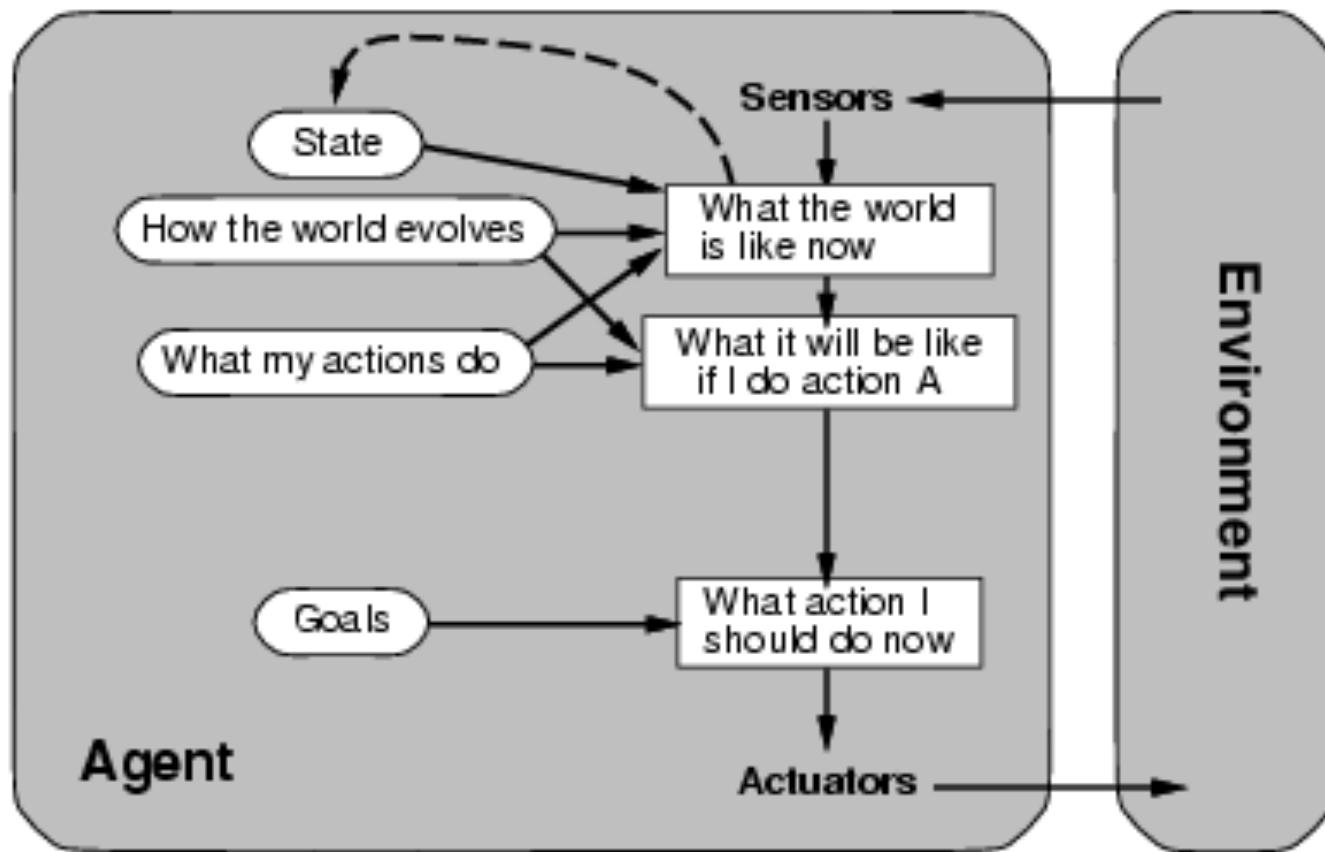
Il agit selon des règles qui dépendent de la situation actuelle définie par ses percepts

Model-based reflex agents



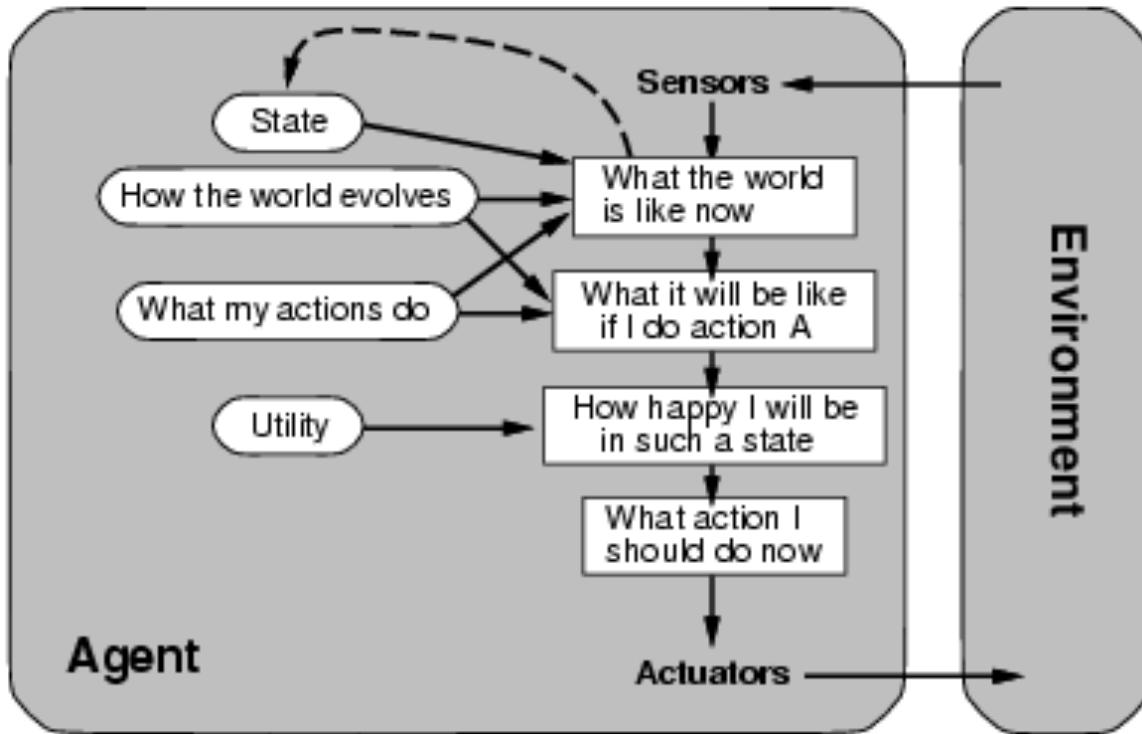
Il agit selon des règles qui dépendent de d'un modèle de la situation actuelle (observation partielle)

Goal-based agents



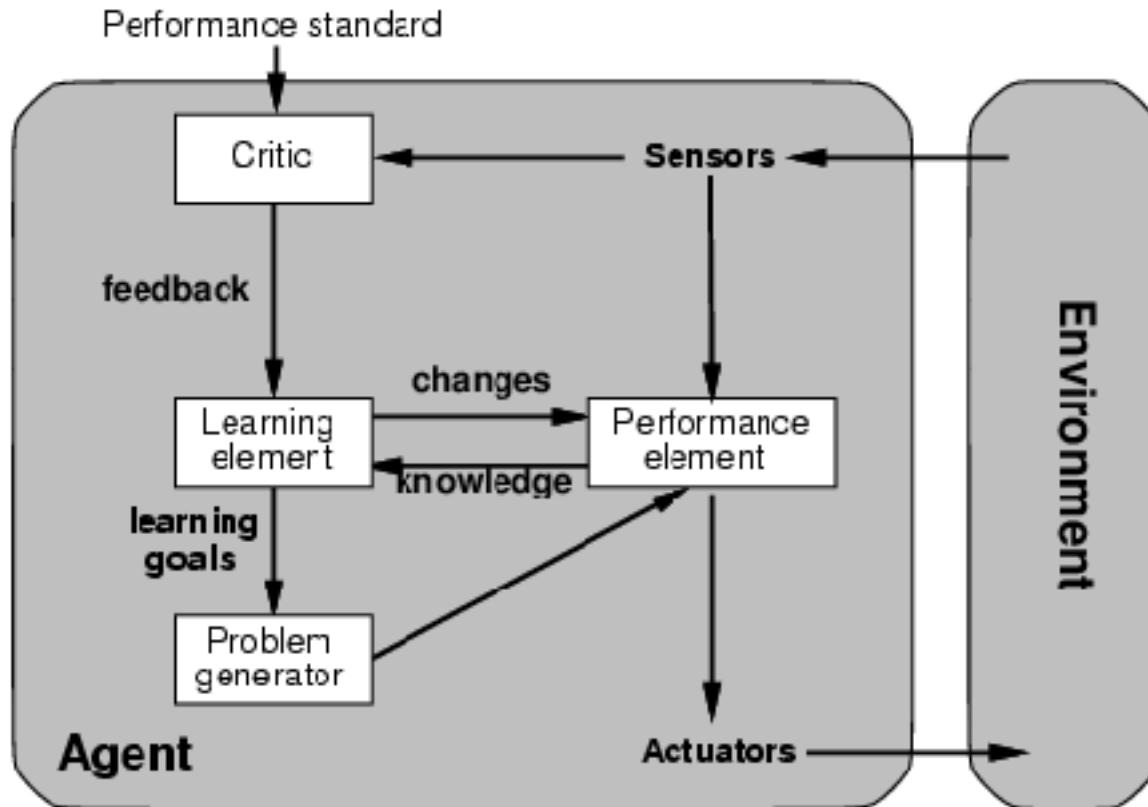
Il agit selon un but qui dépend d'un modèle de la situation actuelle et en considérant aussi l'évolution future du modèle

Utility-based agents



Il agit selon une utilité (mesure de performance) qui dépend d'un modèle de la situation actuelle et en considérant aussi l'évolution future du modèle

Learning agents



Conclusion

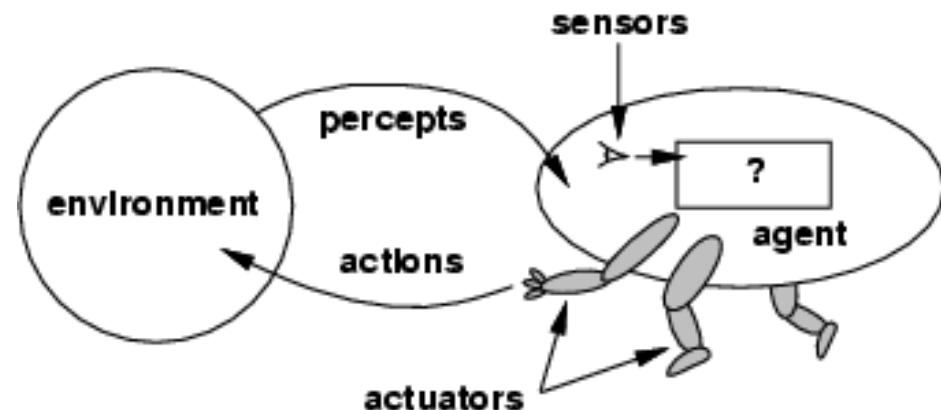
- En résumé, l'intelligence artificielle s'intéresse à tout sujet qui permettrait de reproduire toute capacité de l'intelligence humaine
- Un agent est quelque chose qui perçoit et agit sur son environnement
- Idéalement, on aimerait concevoir un agent rationnel
 - ◆ par rationnel, on veut dire qui maximise sa performance espérée (moyenne)
- L'espace des agents possibles est très large
 - ◆ dépend de la tâche à résoudre
 - ◆ chaque algorithme qu'on va voir est associé à un type d'agent spécifique
- Il existe plusieurs types d'environnement
 - ◆ leurs caractéristiques vont déterminer quel algorithme on devrait utiliser

III-Résolution de problèmes par une recherche dans un graphe

- Une des techniques fondamentales en IA
- Peut résoudre plusieurs problèmes mieux que les humains
- Peut atteindre des performances sur-humaine (échecs, go)
- méthode algorithmique utile pour résoudre des problèmes (en IA et autres domaines)

Résolution de problème par recherche dans un graphe

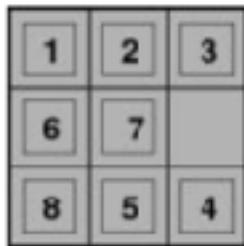
- Algorithme d'un agent basé sur la recherche dans l'espace d'états
- Comprendre l'algorithme A*
- Appliquer A* à un problème donné
- Comprendre la notion d'heuristique



Résolution de problèmes par une recherche dans un graphe

- Étapes intuitives par un humain
 1. modéliser la situation actuelle
 2. énumérer les options possibles
 3. évaluer la valeur des options
 4. retenir la meilleure option possible satisfaisant le but
- Mais comment parcourir efficacement la liste des options?
- La résolution de beaucoup de problèmes peut être faite par **une recherche dans un graphe**
 - ◆ chaque nœud correspond à un état de l'environnement
 - ◆ chaque chemin à travers un graphe représente alors une suite d'actions prises par l'agent
 - ◆ pour résoudre notre problème, il suffit de **chercher le chemin qui satisfait le mieux notre mesure de performance**

Résolution de problèmes par une recherche dans un graphe



Slide 7

Résolution de problèmes par une recherche dans un graphe



DeepBlue 1997

Kasparov
Champion
mondial d'échec

AlphaGo 2017
Ke Jie joueur #1 mondial

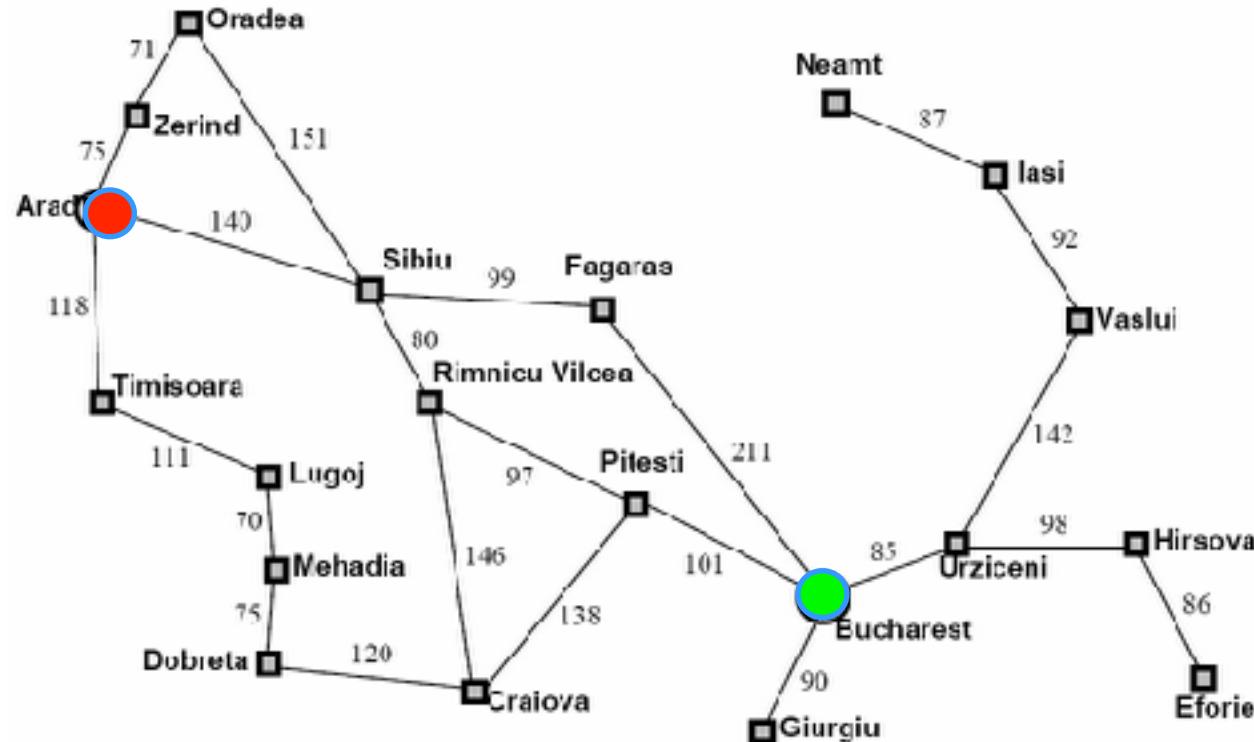


Representation d'un problème

- Pour formuler un problème comme un problème de recherche nous avons besoin :
 1. **Espace d'états** : l'espace d'états est un moyen de representer les états du problème réel
 2. **Espace d'actions (ou transitions)** : Les actions représente les actions qu'on peut faire dans le monde réel mais appliquées à l'espace d'état.
 3. **Un état initial et un état but** : identifie l'état initial et le but ou la condition que nous voulons satisfaire.
 4. **heuristiques** : permettant de guider le processus de recherche
- une fois le problème formulé comme un problème de recherche dans un espace d'états plusieurs algorithmes peuvent être utilisés pour résoudre le problème.

Exemple 1

- de Arad nous voulons aller à Bucharest. Quel est l'espace d'états ?



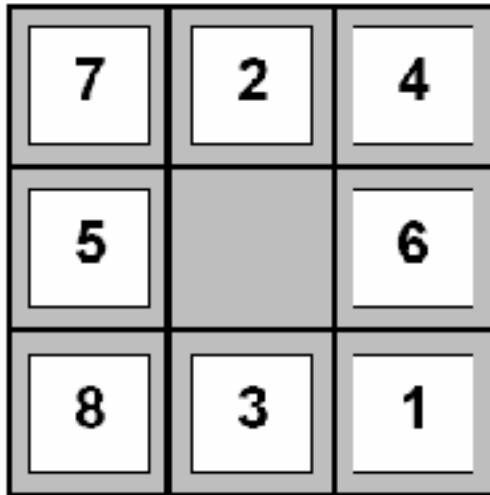
Exemple 1

- Espace d'états : les différentes villes
 - ◆ nous ignorons les détails de la conduite d'une ville à une autre
- Actions : se déplacer d'une ville à une autre ville voisine.
- état initial : dans Arad
- but (condition désirée) : un état où nous sommes dans Bucharest.
- Une solution serait la route (séquence de villes pour atteindre Bucharest)

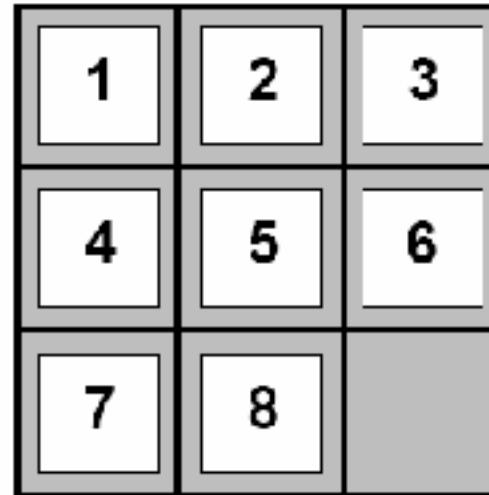
Exemple 2:

- problème de cruches d'eau:
 - ◆ nous avons une cruche d'eau pouvant contenir 3 litres et une autre pouvant contenir 4 litres. Nous pouvons soit remplir les cruches à leur contenance maximale soit les vider soit verser de l'eau d'une cruche dans l'autre (au plus jusqu'à la remplir).
 - ◆ états : paire de nombres (l_3, l_4)
 - » l_3 nombre de litres dans la cruche de 3 litres
 - » l_4 nombre de litres dans la cruche de 4 litres
 - ◆ actions : Vider3, Vider4, Remplir3, Remplir4, Verser3ds4, Verser4ds3.
 - ◆ état initial : exple= (0, 0).
 - ◆ but : (0,2) ou (*,3). ou * signifie n'importe quelle valeur.

Exemple 3: Puzzle



Start State



Goal State

- ◆ Règle: peut glisser une tuile dans la place vide.
- ◆ ou déplacer la case vide.

Exemple 3: Puzzle

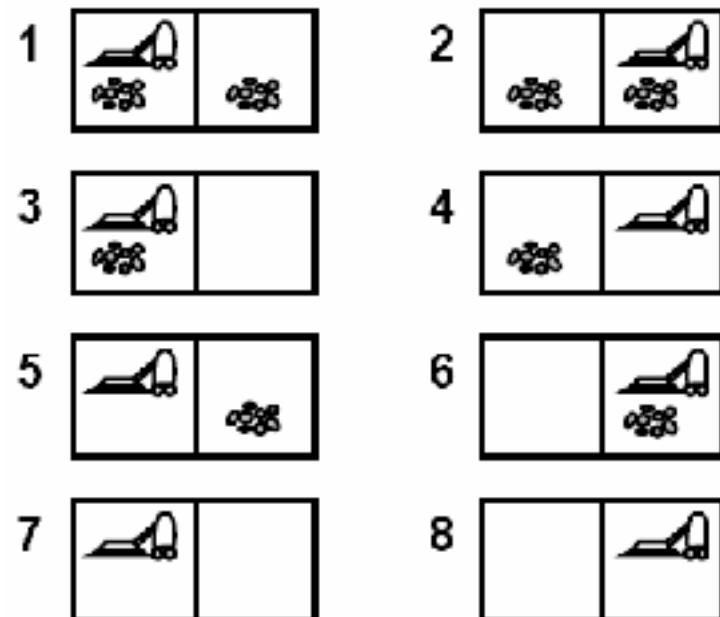
- Espace d'états : les différentes configurations des tuiles.
 - ◆ combien d'états différents ?
- Actions : déplacer la case vide en Haut, Bas, droite, Gauche.
 - ◆ certaines actions ne peuvent pas s'exécuter dans certains états.
- état initial : état représenté dans le slide précédent.
- but (condition désirée) : état représenté dans le slide précédent.
- Solution: une séquence de déplacements de la case vide de l'état initial à l'état final.

Exemple 4: Aspirateur

- Dans les exemples précédents un état dans l'espace de recherche correspond à un état du monde.
- cependant, les états ne doivent pas nécessairement correspondre directement à un état du monde. Mais, un état peut correspondre à un état de connaissance (knowledge state).
- un état de connaissance est un **ensemble** d'états- les états que vous croyais possible.
- Si vous avez une connaissance exacte du monde cet ensemble contiendrait un seul état.

Exemple 4: Aspirateur

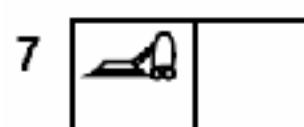
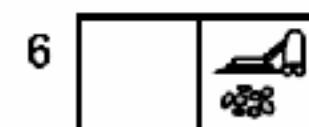
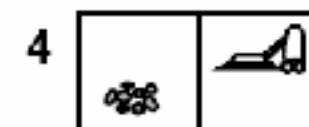
- un aspirateur et deux chambres
- chaque chambre peut être nettoyé ou sale.
- l'aspirateur peut aller **à gauche** ou **à droite**
- l'aspirateur peut **aspirer** et nettoyé une chambre.
- le but est de nettoyer toutes les chambres



états physique

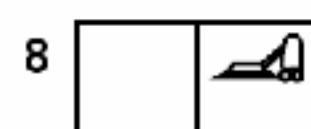
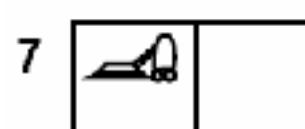
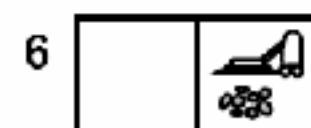
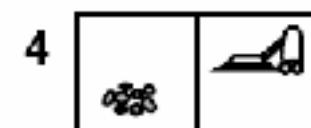
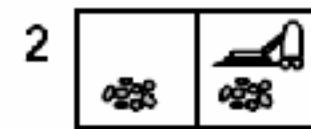
Exemple 4: Aspirateur

- une connaissance complète:
l'agent connaît exactement un état physique.
- start : [5].
- solution : <droite, aspirer>

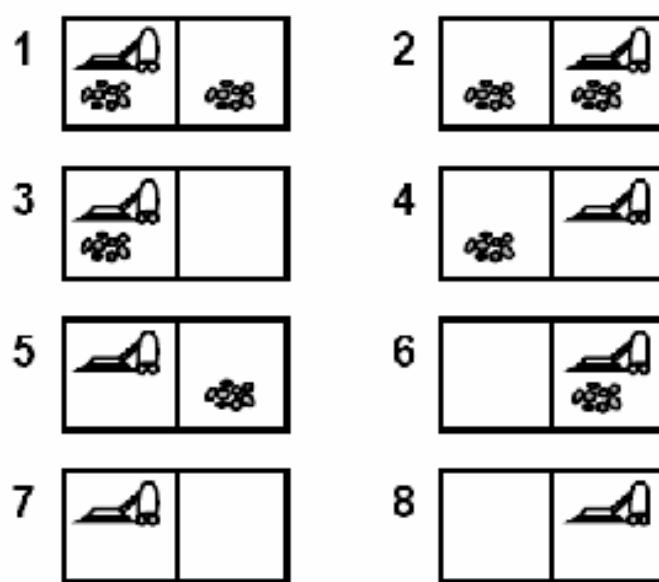


Exemple 4: Aspirateur

- pas de connaissance: état est un ensemble d'états.
- start : {1,2,3,4,5,6,7,8}.
- solution : <droite, aspirer, gauche, aspirer>



Exemple 4: Aspirateur

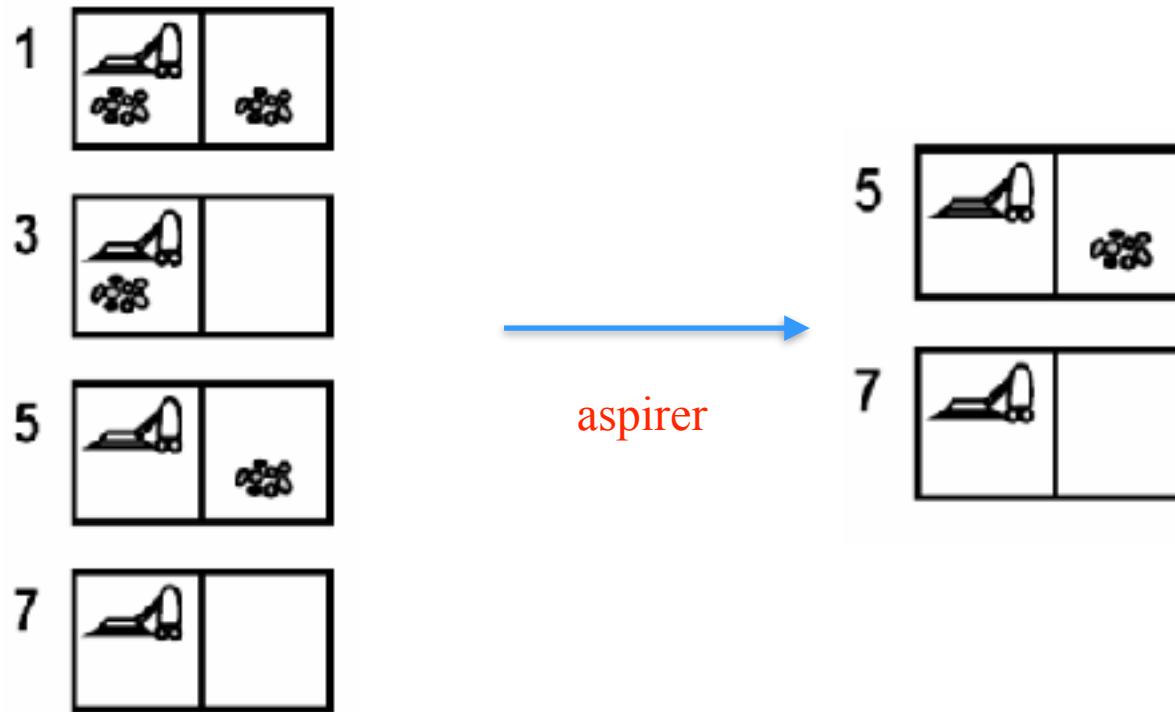


→ gauche



- état initial : {1,2,3,4,5,6,7,8}

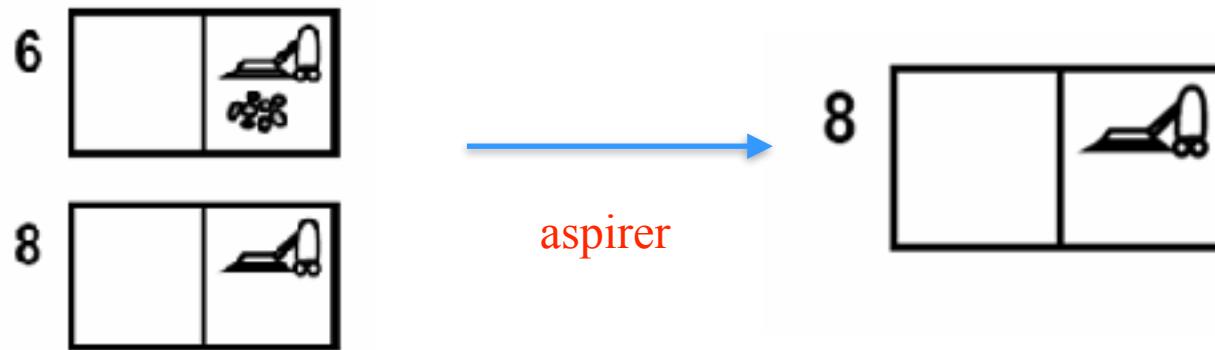
Exemple 4: Aspirateur



Exemple 4: Aspirateur



Exemple 4: Aspirateur



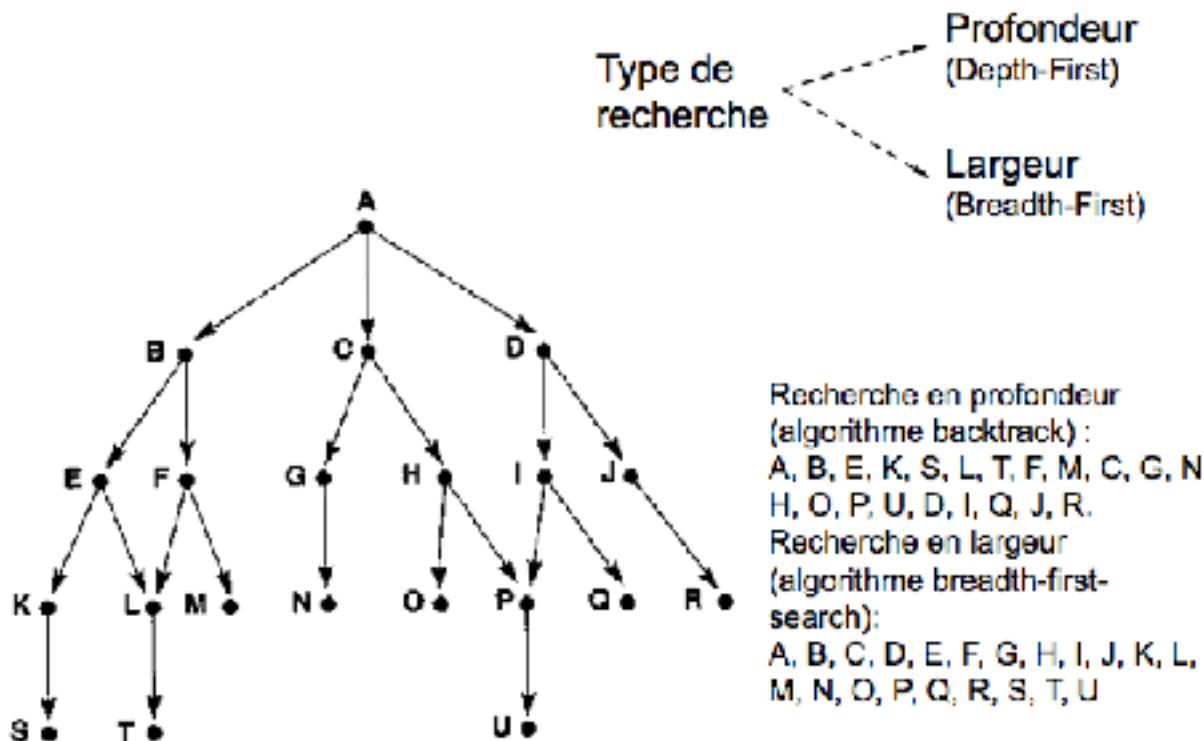
Algorithmes de recherche

- utilisent un espace d'état **implicite**.
- l'espace d'état est généralement un nombre d'état exponentiel : impossible de les représenter
- Les configurations possibles d'un jeu de GO et de 3^{361} (19×19 échiquier)
- nous construisons les états que nous avons besoin. Dans le pire des cas nous aurons à chercher un nombre exponentiel d'états
- les actions sont des fonctions qui pour un état S donne ses successeurs.

Variables importantes : *open* et *closed*

- *Open* contient les nœuds non encore traités, c'est à dire à la frontière de la partie du graphe explorée jusque là
- *Closed* contient les nœuds déjà traités, c'est à dire à l'intérieur de la frontière délimitée par *open*

Algorithmes de recherche



Recherche en profondeur d'abord

```
function depth-first-search
begin
  Open := [Start]; Closed := []% initialize
  while Open != [] do begin % while there are states to be tried
    remove leftmost state from open, called it X;
    if X is a goal then return SUCCESS % goal found
    else begin
      generate children of X;
      put X on closed;
      discard children of X if already on open or closed;
    end
    put remaining children on left end of open %heap
  end
  return FAIL % no states left
End.
```

Trace d'exécution

1. open = [A]; closed = []
2. open = [B, C, D]; closed = [A]
3. open = [E, F, C, D]; closed = [B, A]
4. open = [K, L, F, C, D]; closed = [E, B, A]
5. open = [S, L, F, C, D]; closed = [K, E, B, A]
6. open = [L, F, C, D]; closed = [S, K, E, B, A]
7. open = [T, F, C, D]; closed = [L, S, K, E, B, A]
8. open = [F, C, D]; closed = [T, L, S, K, E, B, A]
9. open = [M,C,D]; as L is already on closed; closed = [F,T,L,S,K,E,B,A]
10. open = [C,D]; closed = [M,F,T,L,S,K,E,B,A]
11. open = [G,H,D]; closed = [M,F,T,L,S,K,E,B,A]

Recherche en largeur d'abord

function breadth-first-search

begin

 Open := [Start]; Closed := [];
 % initialize

 while Open != [] do begin % while there are states to be tried

 remove **leftmost** state from open, called it X;

 if X is a goal then return SUCCESS % goal found

 else begin

 generate children of X;

 put X on closed;

 discard children of X if already on open or closed; %loops

 check

 put remaining children on **right end** of open %queue

 end

 end

 return FAIL % no states left

End.

Trace d'exécution

open = [A]; closed = []

open = [B, C, D]; closed = [A]

open = [C, D, E, F]; closed = [B, A]

open = [D, E, F, G, H]; closed = [B, A]

open = [E, F, G, H, I, J]; closed = [D, C, B, A]

open = [F, G, H, I, K, L]; closed = [E, D, C, B, A]

open = [G, H, I, K, L, M]; (as L is already on
open) closed = [F, E, D, C, B, A]

open = [H, I, K, L, M, N]; closed = [G, F, E, D, C, B,
A]

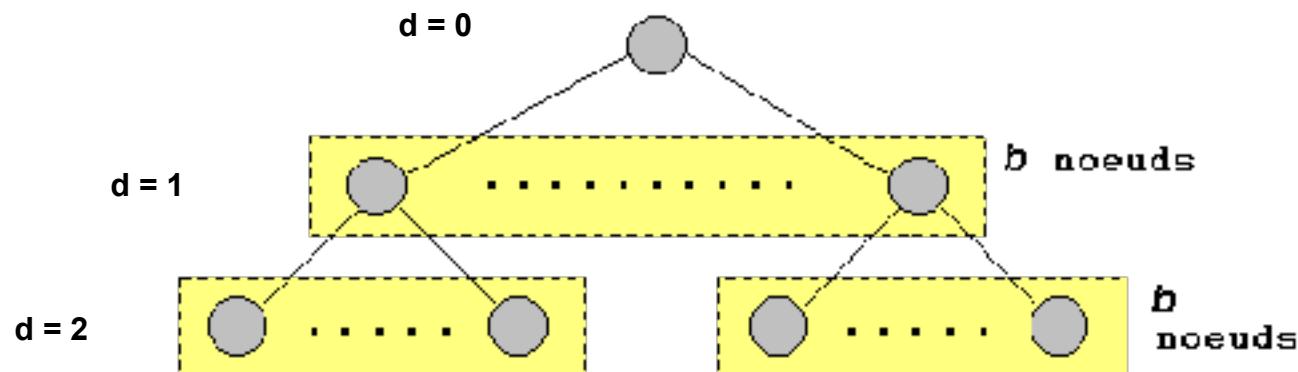
and so on until either U is found or open = []

Propriétés de Alg de recherche

- complétude : l'algorithme trouve une solution si elle existe.
- optimalité: l'algorithme trouve la solution ayant le moindre coût.
- complexité en temps : Quel est le nombre maximal de noeuds générés
- complexity en espace : quel est le nombre maximal de noeuds qui doivent être gardés en mémoire.

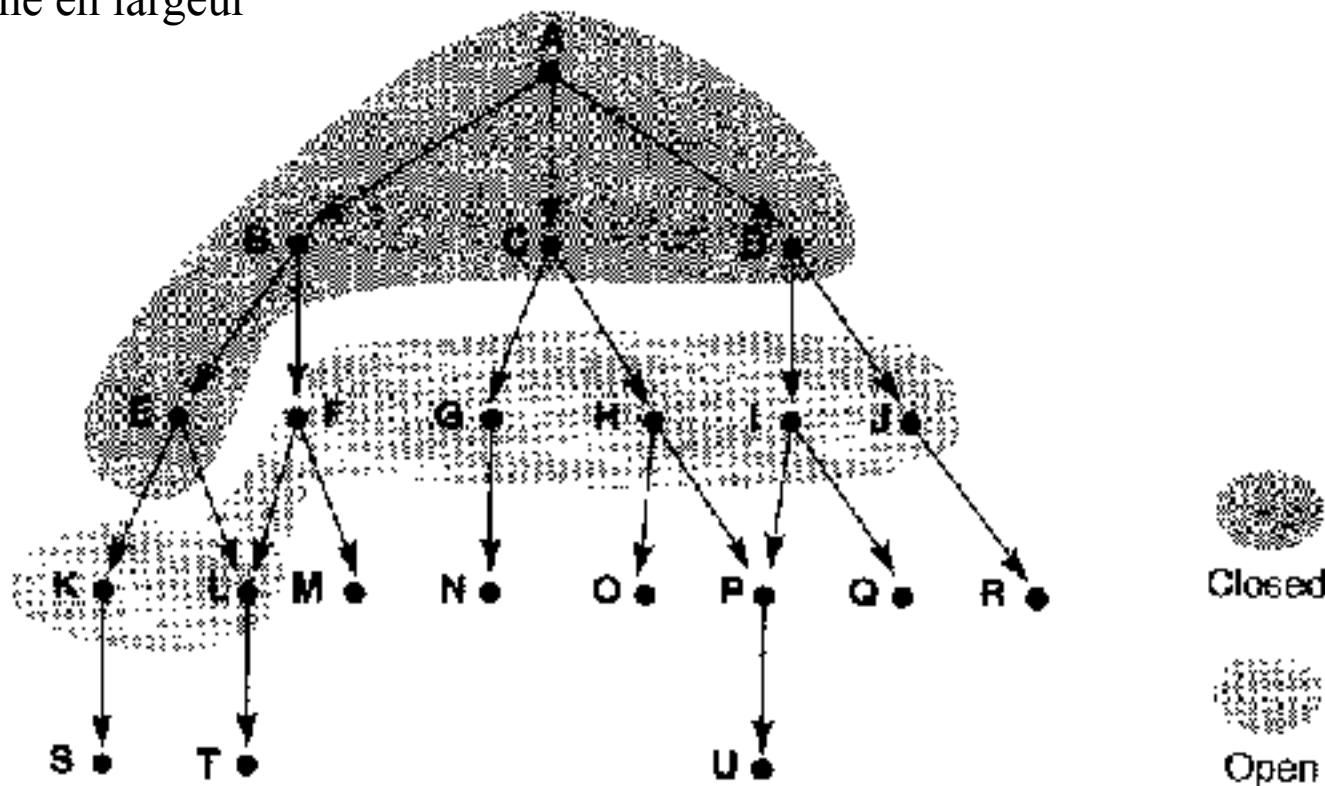
Évaluation des méthodes de recherche

- facteur de branchement



Évaluation des méthodes de recherche

Recherche en largeur



Évaluation des méthodes de recherche

Recherche en largeur

- Complétude: oui, une solution est retrouvée
 - Optimalité: oui, la meilleure solution (nombre d'étapes)
 - Complexité en temps: $O(b^d)$,
 - explorer tout l'arbre de niveau d
 - Complexité en espace: $O(b^d)$,
 - garde en mémoire tout les nœuds du niveau d
- d = la profondeur de la solution la moins profonde

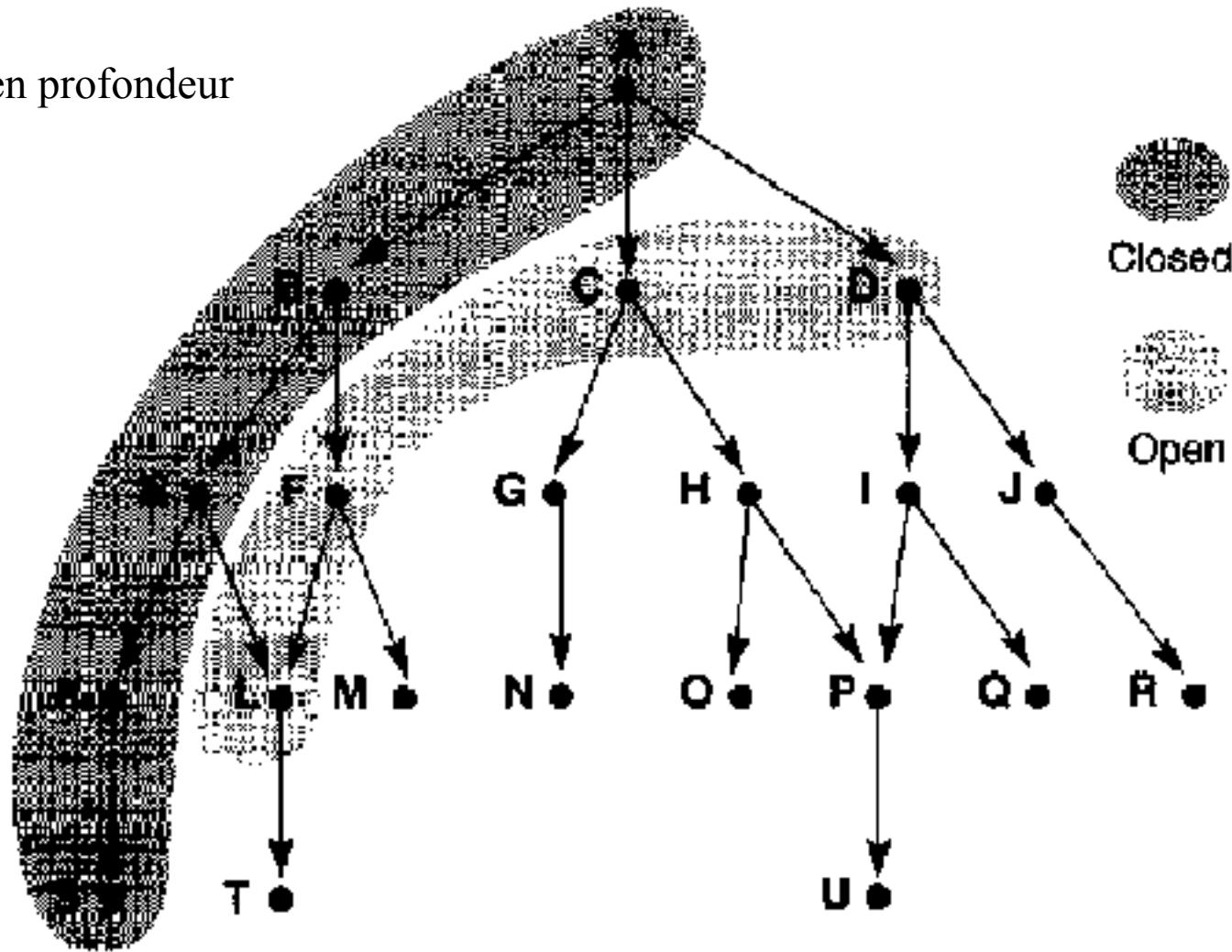
Évaluation des méthodes de recherche

- Recherche en largeur : $b = 10$, temps d'expansion d'un nœud 1 micro-seconde. Un nœud occupe 100 octets

Solution at depth	Nodes expanded (worst case)	Time consumed	Memory required
0	1	1 m-sec	100 b
2	111	0.1 secs	11 kb
4	11,111	11 secs	1 meg
6	10^6	18 mins	111 megs
8	10^8	31 hrs	11 gigs
10	10^{10}	128 days	1 tera
12	10^{12}	35 years	111 teras

Évaluation des méthodes de recherche

Recherche en profondeur



Évaluation des méthodes de recherche

- Complétude: oui, une solution est retrouvée
- Optimalité: non
- Complexité en temps: $O(b^d)$,
- explorer tout l'arbre
- Complexité en espace: $O(b^d)$,
- seulement un nœud par niveau est étendu

d = niveau le plus profond de l'arbre

Évaluation des méthodes de recherche

- Recherche en profondeur
 - Efficace en en mémoire.
 - Ne garantie pas de trouver une solution optimale si elle existe
- Recherche en largeur
 - Moins efficace en en mémoire.
 - Garantie de trouver une solution si elle existe.
 - Trouve toujours la solution optimale.

Recherche heuristique

- heuristique : emprunté au grec euristikê, de découvrir.
- But d'une heuristique : diriger la recherche dans l'espace d'états, de façon à réduire le temps de résolution du problème.
- Une heuristique $h(n)$ est une fonction d'**estimation** du coût entre un nœud n d'un graphe et le *but* (le nœud à atteindre)
- Heuristique est spécifique au domaine du problème.
- Alpha Go utilise des techniques de ML pour calculer l'estimation heuristique d'un état

Recherche heuristique

- si $h(n_1) < h(n_2)$ c-à-d qu'il est plus "économique" d'arriver au but à partir de n_1 que n_2 .
- $h(n) = 0$ pour tout n qui satisfait le but.
- coût zero pour satisfaire un but à partir d'un noeud qui déjà satisfait le but.

Résolution de problème par une recherche heuristique dans un graphe

- La recherche heuristique est à la base de beaucoup d'approches en IA
- Le graphe est défini récursivement (plutôt qu'explicitement)
- Une heuristique est utilisée pour guider la recherche :
 - ◆ les heuristiques exploitent les connaissances du domaine d'application

Exemple : trouver chemin dans une ville

Domaine :

Routes entre les villes :

$$transitions(n_0) = (n_3, n_2, n_1)$$

Distance entre les villes :

$$c(n_0, n_2) = 4$$

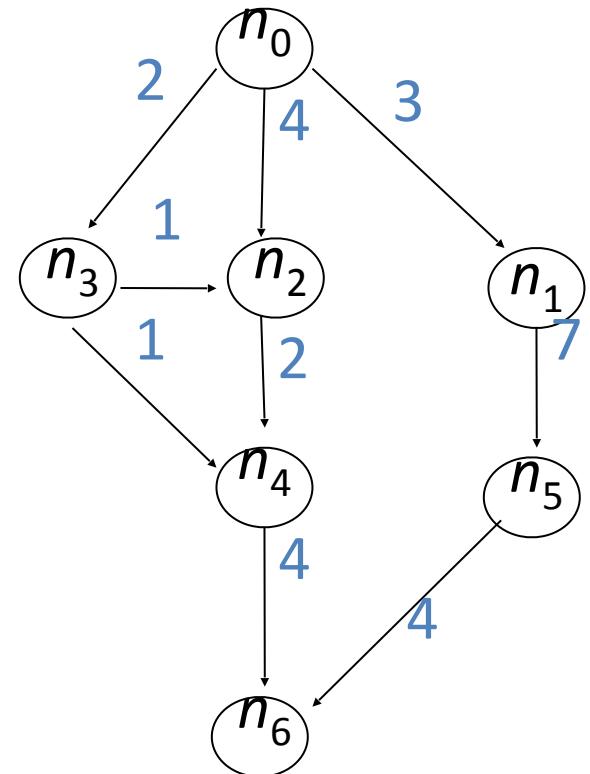
Problème posé (initNode, goal) :

n_0 : ville de départ (état initial)

n_6 : destination (but)

En d'autres termes :

$$goal(n) : \text{vrai si } n=n_6$$



Rappel sur les algorithmes de recherche dans des graphes

- Recherche sans heuristique et coût uniforme
 - ◆ Recherche en profondeur (*depth-first Search*)
 - » pour un noeud donné, explore le premier enfant avant d'explorer un noeud frère
 - ◆ Recherche en largeur (*breadth-first search*)
 - » pour un noeud donné, explore les noeuds frères avant leurs enfants
- Recherche **sans heuristique** et coût variable
 - ◆ Algorithme de Dijkstra
 - » trouve le chemin le plus court entre un noeud source et tous les autres noeuds
- Recherche **avec heuristique** et coût variable :
 - ◆ *best-first search*
 - ◆ *greedy best-first search*
 - ◆ A*

Algorithme A*

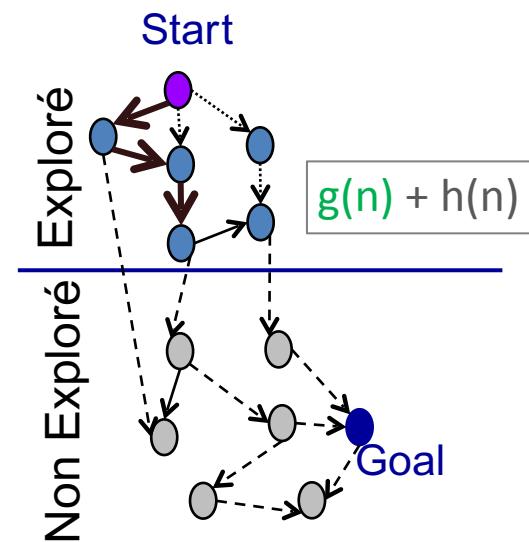
- A* est une extension de l'algorithme de Dijkstra
 - ◆ utilisé pour trouver un chemin optimal dans un graphe via **l'ajout d'une heuristique**
- Une **heuristique $h(n)$** est une fonction d'**estimation du coût entre un nœud n d'un graphe et le but** (le nœud à atteindre)
- Les heuristiques sont à la base de beaucoup de travaux en IA :
 - ◆ recherche de meilleures heuristiques
 - ◆ apprentissage automatique d'heuristiques

Insertion des nœuds dans *open*

- Les nœuds n dans *open* sont triés selon l'estimé $f(n)$ de leur « valeur »
 - ◆ on appelle $f(n)$ une **fonction d'évaluation**
- Pour chaque nœud n , $f(n)$ est un nombre réel positif ou nul, **estimant le coût du meilleur chemin partant de la racine, passant par n , et arrivant au but**
- Dans *open*, les nœuds se suivent en ordre croissant selon les valeurs $f(n)$.
 - ◆ le tri se fait par insertion : on s'assure que le nouveau nœud va au bon endroit
 - ◆ on explore donc les noeuds les plus « prometteurs » en premier

Définition de f

- La fonction d'évaluation $f(n)$ tente d'estimer le coût du chemin optimal entre le nœud initial et le but, et qui passe par n
- En pratique on ne connaît pas ce coût : c'est ce qu'on cherche !
- À tout moment, on connaît seulement le coût optimal pour la partie explorée entre la racine et un nœud déjà exploré
- Dans A*, on sépare le calcul de $f(n)$ en deux parties :
 - ◆ $g(n)$: coût du meilleur chemin ayant mené au noeud n depuis la racine
 - » c'est le coût du meilleur chemin trouvé jusqu'à maintenant qui se rend à n
 - ◆ $h(n)$: coût estimé du reste du chemin optimal partant de n jusqu'au but. $h(n)$ est la fonction heuristique
 - » on suppose que $h(n)$ est non négative et $h(n) = 0$ si n est le noeud but



Algorithme générique de recherche dans un graphe

Algorithme RECHERCHE-DANS-GRAPHE(*noeudInitial*)

1. déclarer deux nœuds : n, n'
2. déclarer deux listes : *open*, *closed* // toutes les deux sont vides au départ
3. insérer *noeudInitial* dans *open*
4. tant que (1) // la condition de sortie (exit) est déterminée dans la boucle
 5. si *open* est vide, sortir de la boucle avec échec
 6. $n = \text{noeud au début de } open;$
 7. enlever n de *open* et l'ajouter dans *closed*
 8. si n est le but, sortir de la boucle avec succès en retournant le chemin;
 9. pour chaque successeur n' de n
 10. initialiser la valeur $g(n')$ à : $g(n) + \text{le coût de la transition } (n,n')$
 11. mettre le parent de n' à n
 12. si *closed* ou *open* contient un nœud n'' égal à n' avec $f(n') < f(n'')$
 13. enlever n'' de *closed* ou *open* et insérer n' dans *open* (ordre croissant selon $f(n)$)
 11. si n' n'est ni dans *open* ni dans *closed*
 15. insérer n' dans *open* en triant les nœuds en ordre croissant selon $f(n)$

Exemple A* avec recherche dans une ville

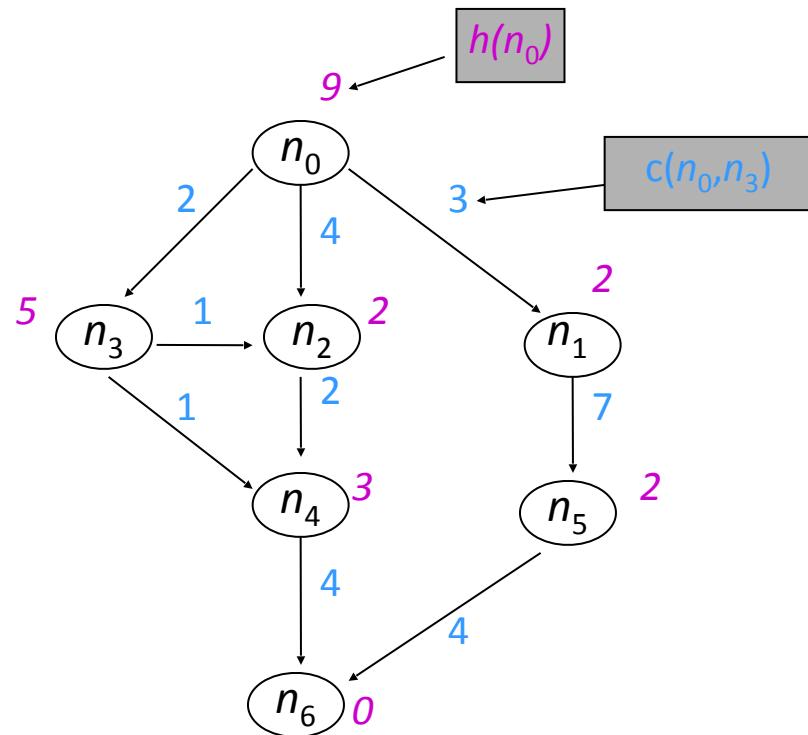
Routes entre les villes :

n_0 : ville de départ

n_6 : destination

h : distance à vol d'oiseau

c : distance réelle entre deux ville



Exemple A* avec recherche dans une ville

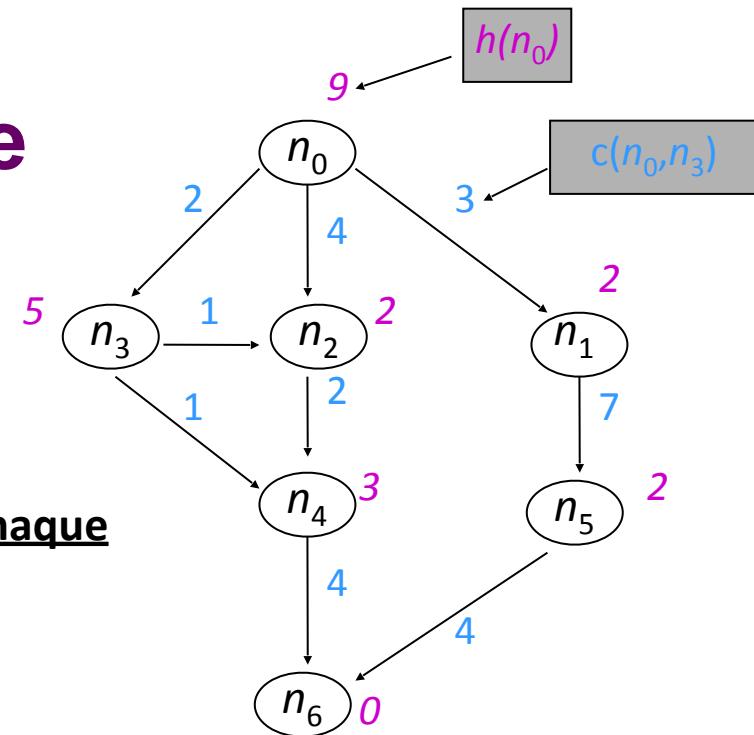
But: chemin menant à n_6

Contenu de *open* à chaque itération (état, f, parent) :

1. $(n_0, 9, \text{void})$
2. $(n_1, 5, n_0), (n_2, 6, n_0), (n_3, 7, n_0)$
3. $(n_2, 6, n_0), (n_3, 7, n_0), (n_5, 12, n_1)$
4. $(n_3, 7, n_0), (n_4, 9, n_2), (n_5, 12, n_1)$
5. $(n_2, 5, n_3), (n_4, 6, n_3), (n_5, 12, n_1)$
6. $(n_4, 6, n_3), (n_5, 12, n_1)$
7. $(n_6, 7, n_4), (n_5, 12, n_1)$
8. Solution : n_0, n_3, n_4, n_6

Contenu de *closed* à chaque itération :

1. Vide
2. $(n_0, 9, \text{void})$
3. $(n_0, 9, \text{void}), (n_1, 5, n_0)$
4. $(n_0, 9, \text{void}), (n_1, 5, n_0), (n_2, 6, n_0)$
5. $(n_0, 9, \text{void}), (n_1, 5, n_0), (n_3, 7, n_0)$
6. $(n_0, 9, \text{void}), (n_1, 5, n_0), (n_3, 7, n_0), (n_2, 5, n_3)$
7. $(n_0, 9, \text{void}), (n_1, 5, n_0), (n_3, 7, n_0), (n_2, 5, n_3), (n_4, 6, n_3)$
8. $(n_0, 9, \text{void}), (n_1, 5, n_0), (n_3, 7, n_0), (n_2, 5, n_3), (n_4, 6, n_3), (n_6, 7, n_4)$



Propriétés de A*

- Si le graphe est fini, A* termine toujours
- Si un chemin vers le but existe, A* va en trouver un
- Si la fonction heuristique h retourne toujours un **estimé inférieur ou égal au coût réel à venir**, on dit que h est **admissible** :

$$0 \leq h(n) \leq h^*(n)$$

- ◆ dans ce cas, A* **retourne toujours un chemin optimal**

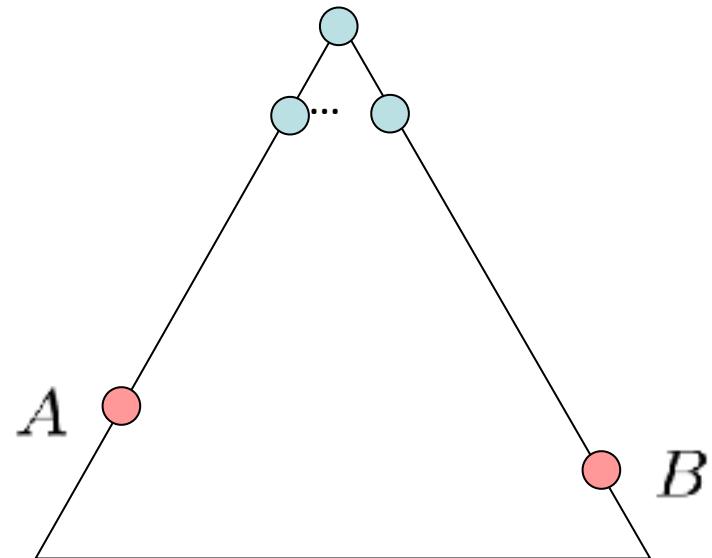
optimalité de A*

on suppose:

- A est un noeud but optimal
- B est un but non-optimal
- h est admissible

Affirmation:

- A sera exploré avant B



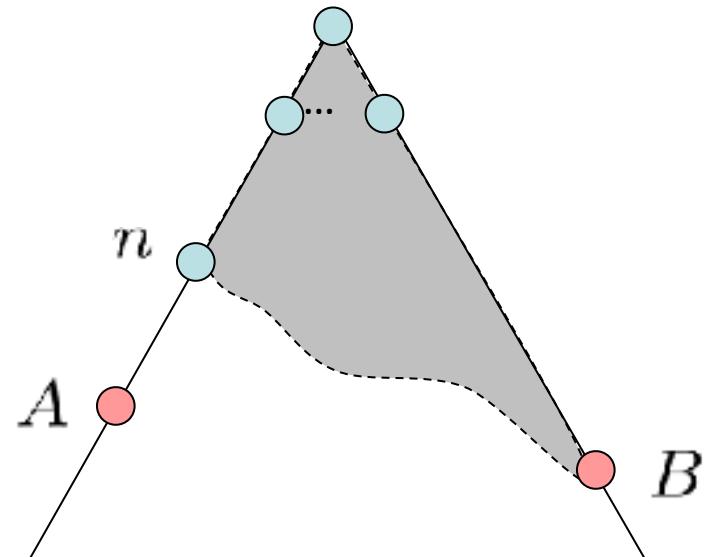
optimalité de A*

on suppose:

- B est dans Open
- n un ancêtre de A est dans Open

affirmation :

- n sera exploré avant B
 1. $f(n) \leq f(A)$
 2. $f(A) < f(B)$
 3. n est exploré avant B
- tous les ancêtres de A sont explorés avant B
- A est exploré avant B
- A* est optimal



$$f(n) \leq f(A) \leq f(B)$$

Autres Propriétés de A*

- Si quelque soit un nœud n_1 et son successeur n_2 , nous avons toujours

$$h(n_1) \leq c(n_1, n_2) + h(n_2)$$

où $c(n_1, n_2)$ est le coût de l'arc (n_1, n_2) .

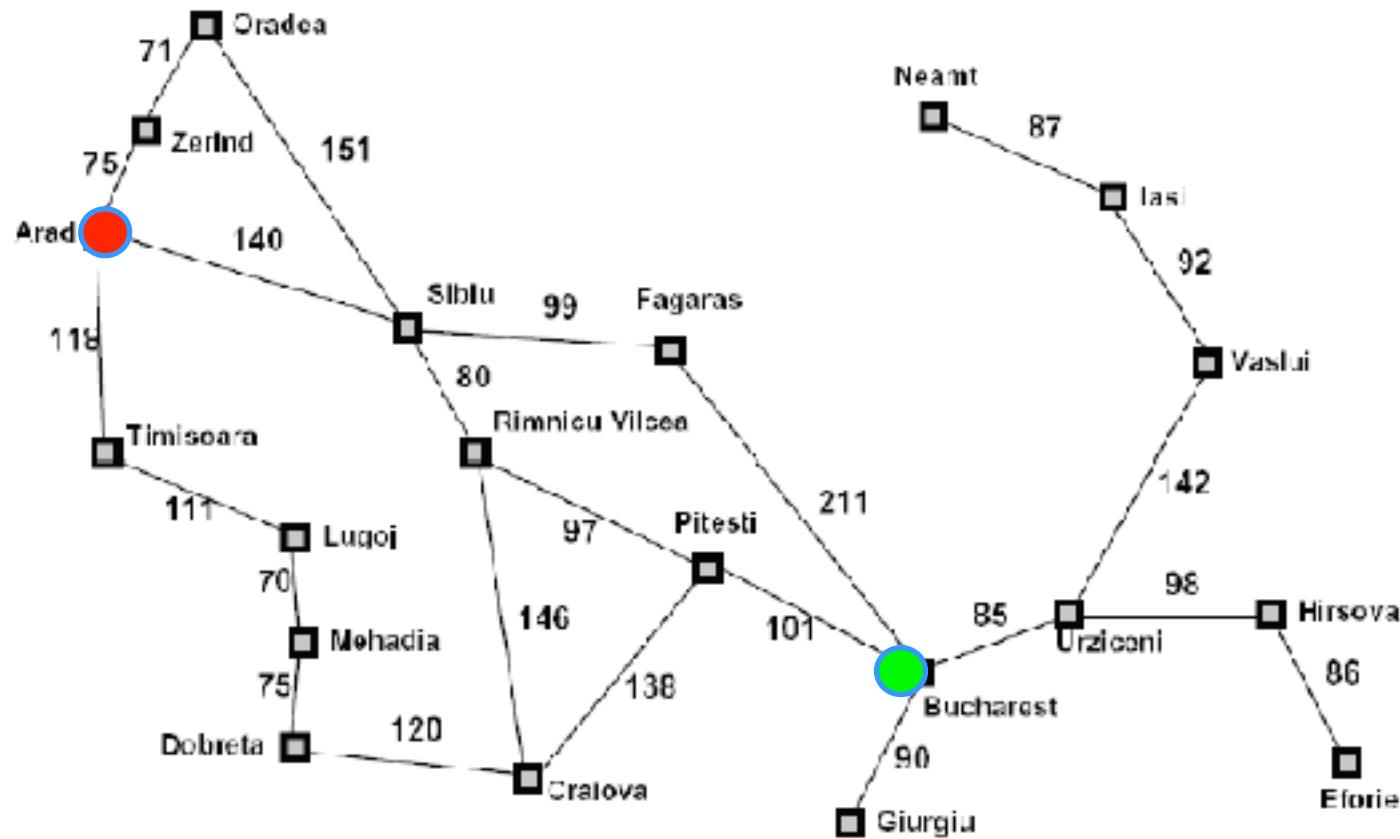
On dit alors que h est **cohérente** (on dit aussi parfois **monotone** – mais c'est en réalité f qui devient monotone). Dans ce cas :

- ◆ h est aussi admissible
- ◆ chaque fois que A* choisit un nœud au début de open, cela veut dire que A* a déjà trouvé un chemin optimal vers ce nœud : le nœud ne sera plus jamais revisité!

Autres Propriétés de A*

- Si on a deux heuristiques *admissibles* h_1 et h_2 , tel que $h_1(n) < h_2(n)$, alors $h_2(n)$ conduit plus vite au but : avec h_2 , A* explore moins ou autant de nœuds avant d'arriver au but qu'avec h_1
- Si h n'est pas admissible, soit b la borne supérieure sur la surestimation du coût, c-à-d. on a toujours $h(n) \leq h^*(n) + b$:
 - ◆ A* retournera une solution dont le coût est au plus b de plus que le coût optimal, c-à-d., A* ne se trompe pas plus que b sur l'optimalité.

Exemple



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

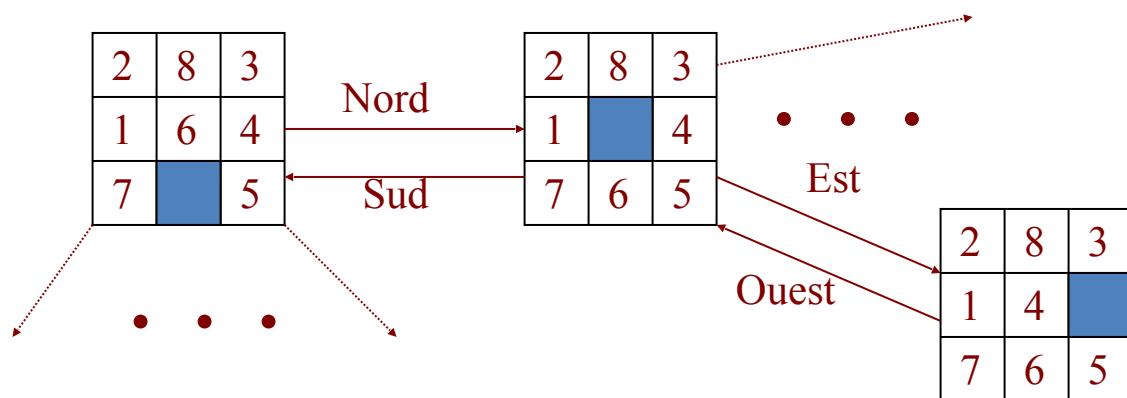
Exemples de fonctions heuristiques

- Chemin dans une ville
 - ◆ distance **Euclidienne** ou distance de **Manhattan** pour un chemin sur une carte
 - ◆ éventuellement pondéré par la qualité des routes, le prix du billet, etc.
- Probabilité d'atteindre l'objectif en passant par le nœud
- Qualité de la configuration d'un jeu par rapport à une configuration gagnante
- N-Puzzle
 - ◆ nombre de tuiles mal placées
 - ◆ somme des distances des tuiles

Exemple

- 8-puzzle

- ◆ *État* : configuration légale du jeu
- ◆ *État initial* : configuration initiale
- ◆ *État final (but)* : configuration gagnante
- ◆ *Transitions*



Exemple; Heuristique

2	8	3
1	6	4
7		5



1	2	3
8		4
7	6	5

Average nodes expanded when
the optimal path has...

...4 steps ...8 steps ...12 steps

UCS	112	6,300	3.6×10^6
TILES	13	39	227
MANHATTAN	12	25	73

Arbre de recherche pour les Jeux

Arbre de recherche pour les Jeux

- Checkers: 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!
- Chess: 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.
- Go: 2016: Alpha GO defeats human champion. Uses Monte Carlo Tree Search, learned evaluation function.

Type de jeux

- Déterministe ou stochastique?
- Un, Deux, ou plusieurs joueurs?
- Somme zero?
- Observable / partiellement observable?
- Des algorithmes pour calculer une **stratégie** qui recommande un coup pour un état donné.

Type de jeux

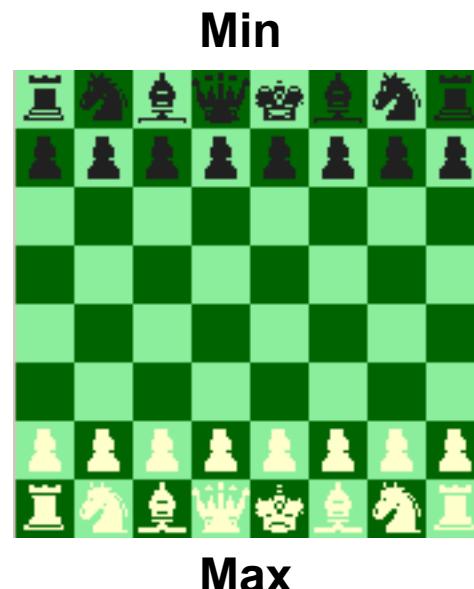
- Dans un jeu, des joueurs peuvent être :
- Coopératifs
 - ◆ ils veulent atteindre le même but
- Des adversaires en compétition
 - ◆ un gain pour les uns est une perte pour les autres
 - ◆ cas particulier : les jeux à somme nulle (zero-sum games)
 - » jeux d'échecs, de dame, tic-tac-toe, Go, etc.
- Mixte
 - ◆ il y a tout un spectre entre les jeux purement coopératifs et les jeux avec adversaires (ex. : alliances)

Type de jeux

- Dans ce cours, nous aborderons les :
 - ◆ jeux à deux adversaires
 - ◆ jeux à tour de rôle
 - ◆ jeux à somme nulle
 - ◆ jeux avec états complètement observés
 - ◆ jeux déterministes (sans hasard ou incertitude)

Jeux entre deux adversaires

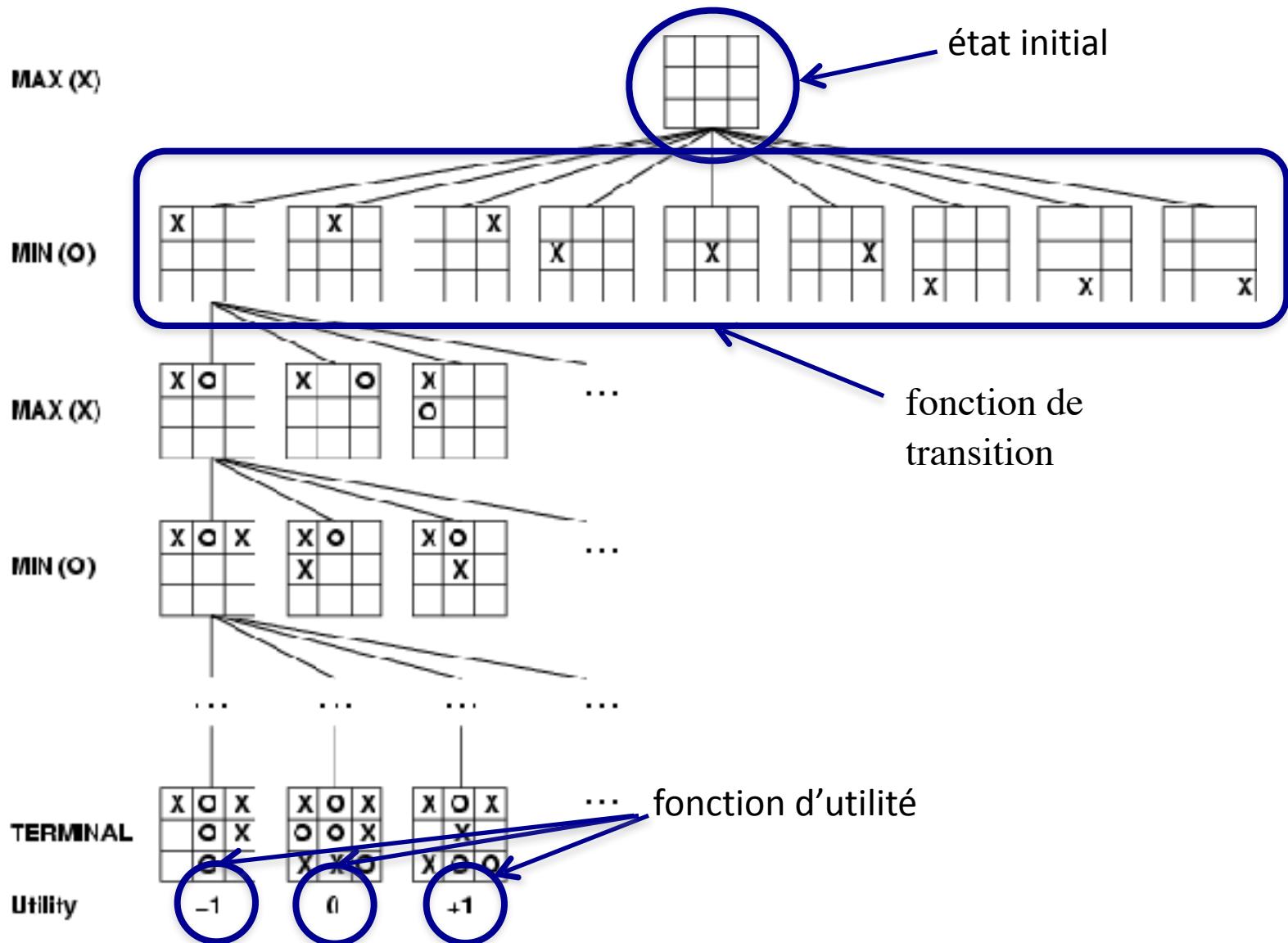
- Noms des joueurs : Max vs. Min
 - Max est le premier à jouer (notre joueur)
 - Min est son adversaire
-
- On va interpréter le résultat d'une partie comme la distribution d'une récompense
 - On peut voir cette récompense
 - ◆ comme le résultat d'un pari
 - ◆ Min reçoit l'opposé de ce que Max reçoit



Arbre de recherche

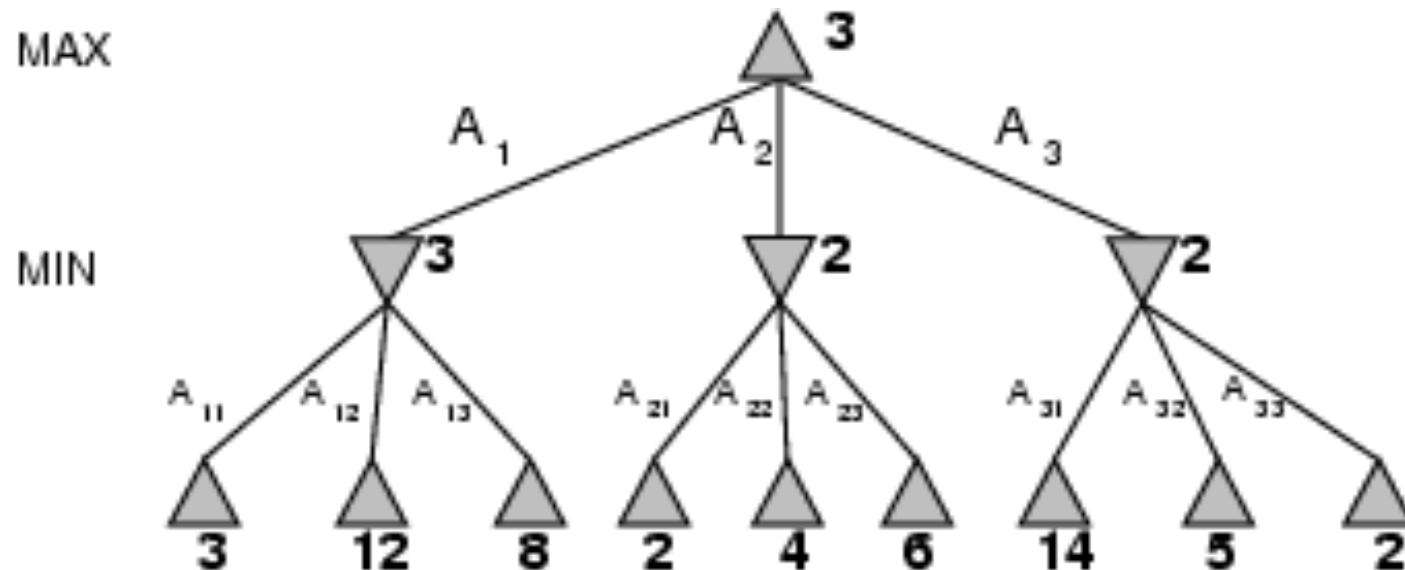
- Comme pour les problèmes que A* peut résoudre, on commence par déterminer la structure de notre espace de recherche
- Un problème de jeu peut être vu comme un problème de recherche dans un arbre :
 - ◆ Un noeud (état) initial : configuration initiale du jeu
 - ◆ des joueurs
 - ◆ Une fonction de transition :
 - » retournant un ensemble de paires (action, noeud successeur)
 - action possible (légale)
 - noeud (état) résultant de l'exécution de cette action
 - ◆ Un test de terminaison
 - » indique si le jeu est terminé
 - ◆ Une fonction d'utilité pour les états finaux (c'est la récompense reçue)

Arbre de recherche

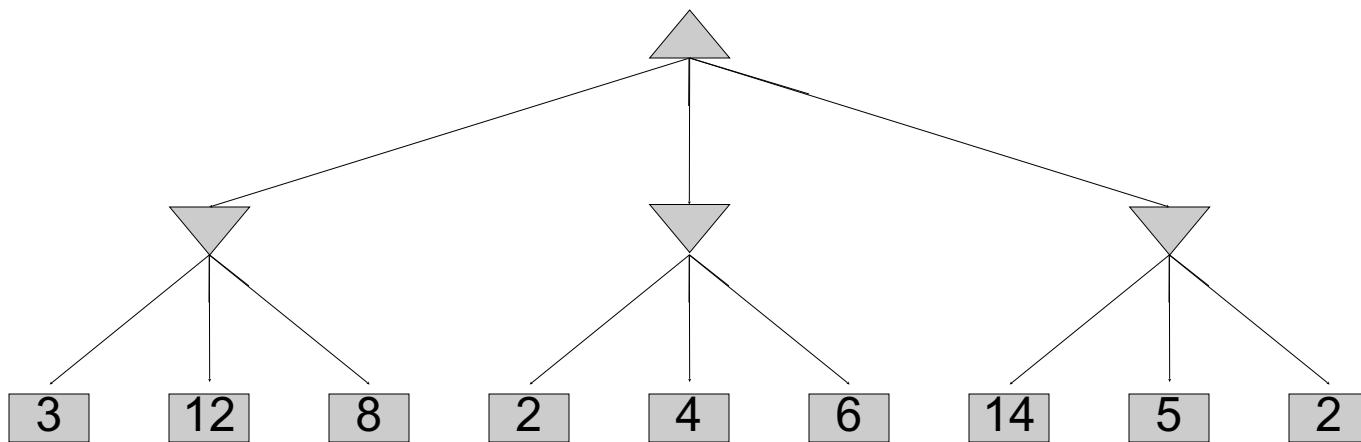


Algorithme MinMax

- Idée: À chaque tour, choisir l'action menant à la plus grande valeur minimax.
- Cela donne la meilleure action optimale (plus grand gain) contre un joueur optimal.
- Exemple simple:



Minimax Example



Algorithme MinMax

- Hypothèse: MAX et MIN jouent optimalement.
- Idée: À chaque tour, choisir l'action menant à la plus grande valeur minimax.
- Cela donne la meilleure action optimale (plus grand gain) contre un joueur optimal (rationnel).

MINIMAX-VALUE(n) =

UTILITY(n)

Si n est un nœud terminal

$\max_{s \in \text{successors}(n)} \text{MINIMAX-VALUE}(s)$

Si n est un nœud Max

$\min_{s \in \text{successors}(n)} \text{MINIMAX-VALUE}(s)$

Si n est un nœud Min

Ces équations donne la programmation récursive des valeurs jusqu'à la racine de l'arbre.

Algorithme MinMax

```
def value(state):
```

 if the state is a terminal state: return the state's utility
 if the next agent is MAX: return max-value(state)
 if the next agent is MIN: return min-value(state)

```
def max-value(state):
```

 initialize v = -∞

 for each successor of state:

 v = max(v, value(successor))

 return v

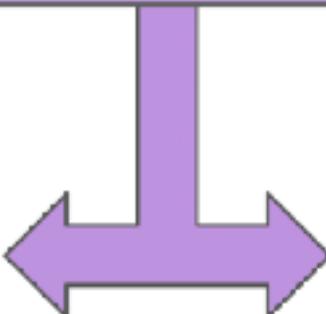
```
def min-value(state):
```

 initialize v = +∞

 for each successor of state:

 v = min(v, value(successor))

 return v



Propriétés de minimax

- Complet?
 - ◆ Oui (si l'arbre est fini)
- Optimal?
 - ◆ Oui (contre un adversaire qui joue optimalement)
- Complexité en temps?
 - ◆ $O(b^m)$:
 - ◆ b: le nombre maximum d'actions/coups légales à chaque étape
 - ◆ m: nombre maximum de coup dans un jeu (profondeur maximale de l'arbre).
- Complexité en espace?
 - ◆ $O(b^m)$, parce que l'algorithme effectue une recherche en profondeur.
- Pour le jeu d'échec: b ≈ 35 et m ≈ 100 pour un jeu « raisonnable »
- Il n'est pas réaliste d'espérer trouver une solution exacte en temps réel.

Comment accélérer la recherche

- Deux approches
 - ◆ la première maintient l'exactitude de la solution
 - ◆ la deuxième introduit une approximation
- Élagage alpha-bêta (alpha-beta pruning)
 - ◆ idée : identifier des chemins dans l'arbre qui sont explorés inutilement
- Couper la recherche et remplacer l'utilité par une fonction d'évaluation heuristique
 - ◆ idée : faire une recherche la plus profonde possible en fonction du temps à notre disposition et tenter de prédire le résultat de la partie si on n'arrive pas à la fin

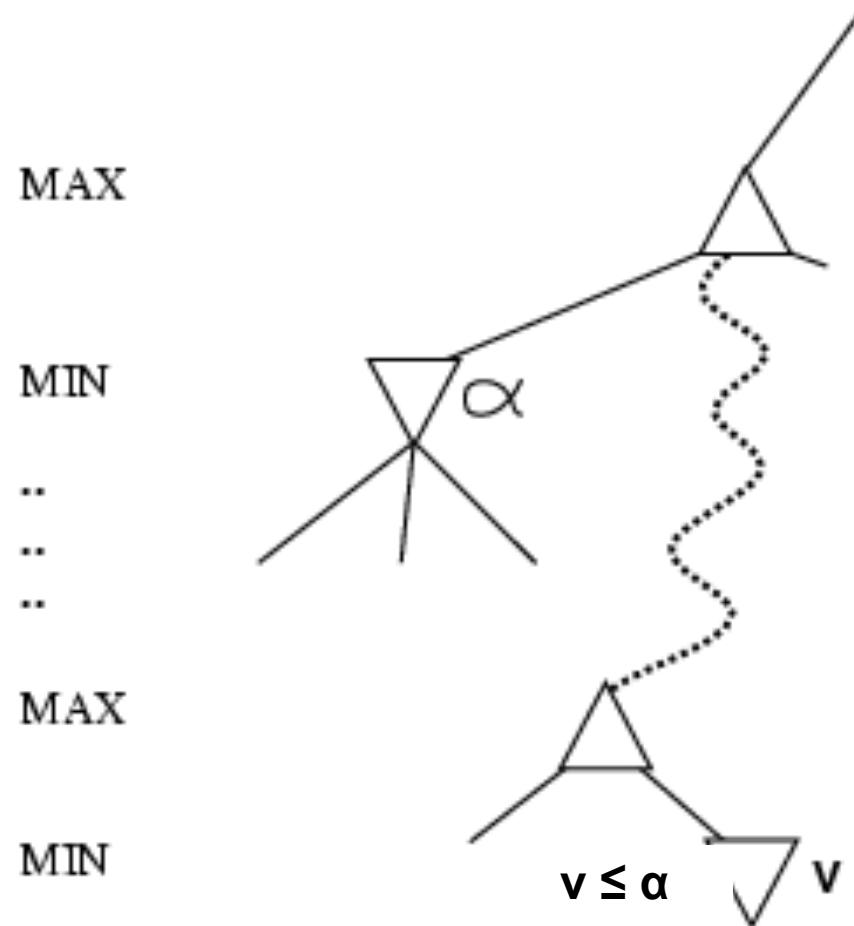
Alpha-beta pruning

- L'algorithme alpha-beta tire son nom des paramètres suivant décrivant les bornes des valeurs d'utilité enregistrée durant le parcourt.
- α est la valeur du meilleur choix pour Max (c.-à-d., plus grande valeur) trouvée jusqu'ici:
- β est la valeur du meilleur choix pour Min (c.-à-d., plus petite valeur) trouvée jusqu'ici.

Alpha-beta Pruning

Condition pour couper dans un nœud Min

- Sachant que α est la valeur du meilleur choix pour Max (c.-à-d., plus grande valeur) trouvée jusqu'ici:
- Si on est dans un nœud Min et que sa valeur v devient inférieure α (donc « pire que α » du point de vue de Max), il faut arrêter la recherche (couper la branche).



Alpha-beta Pruning

Condition pour couper dans un nœud Max

- Sachant que β est la valeur du meilleur choix pour Min (c.-à-d., plus petite valeur) trouvée jusqu'ici:
- Si on est dans un nœud Max et que sa valeur devient supérieur à β (donc « pire que β » du point de vue de Min), il faut arrêter la recherche (couper la branche).

Exemple d'Alpha-beta pruning

Faire une recherche en profondeur jusqu'à la première feuille

MAX

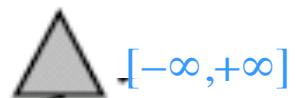
MIN

Entre croches [,]: Intervalle des valeurs possibles pour le nœud visité.

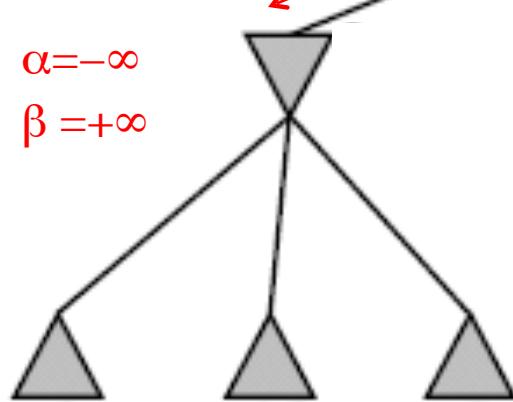
Valeur initial de α, β

$$\alpha = -\infty$$

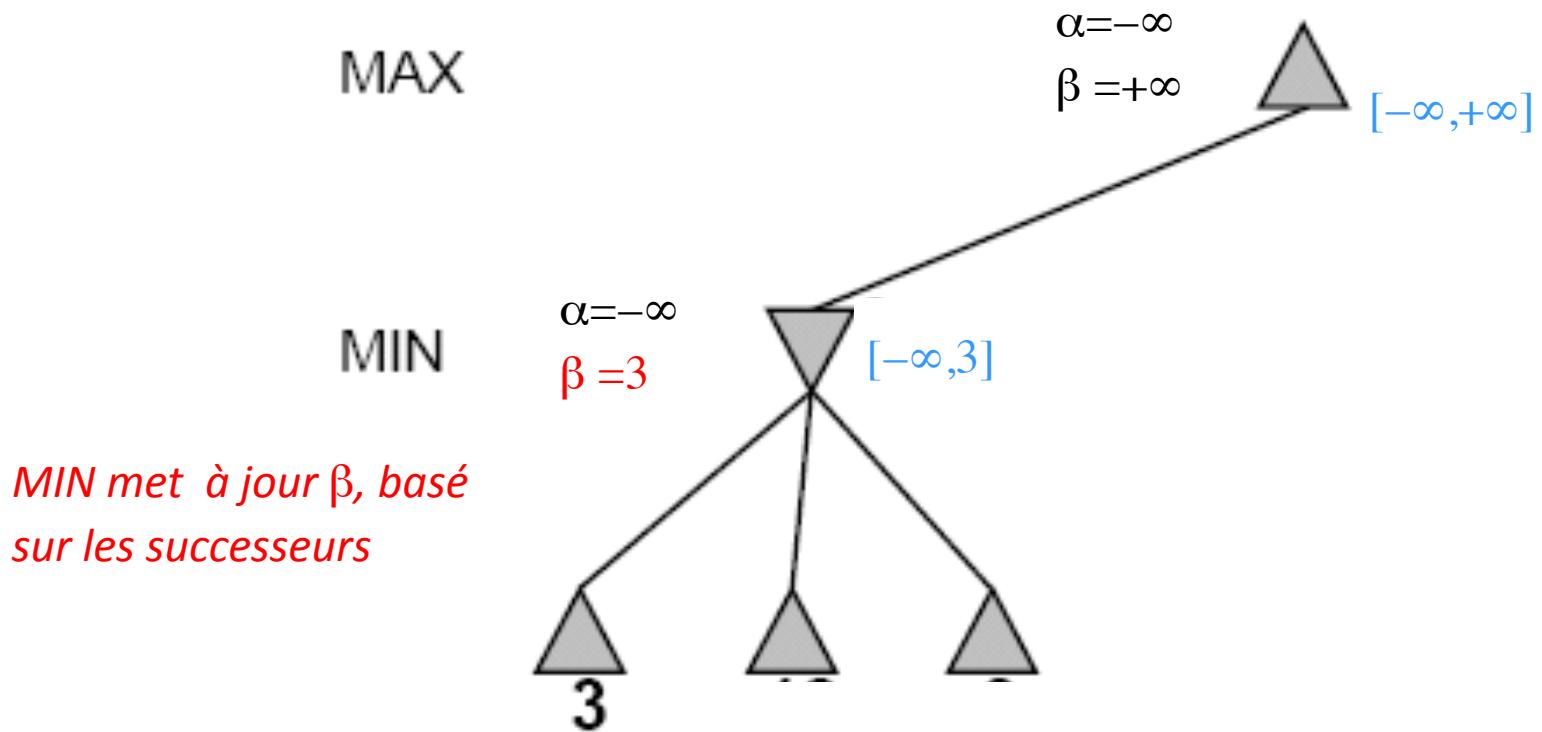
$$\beta = +\infty$$



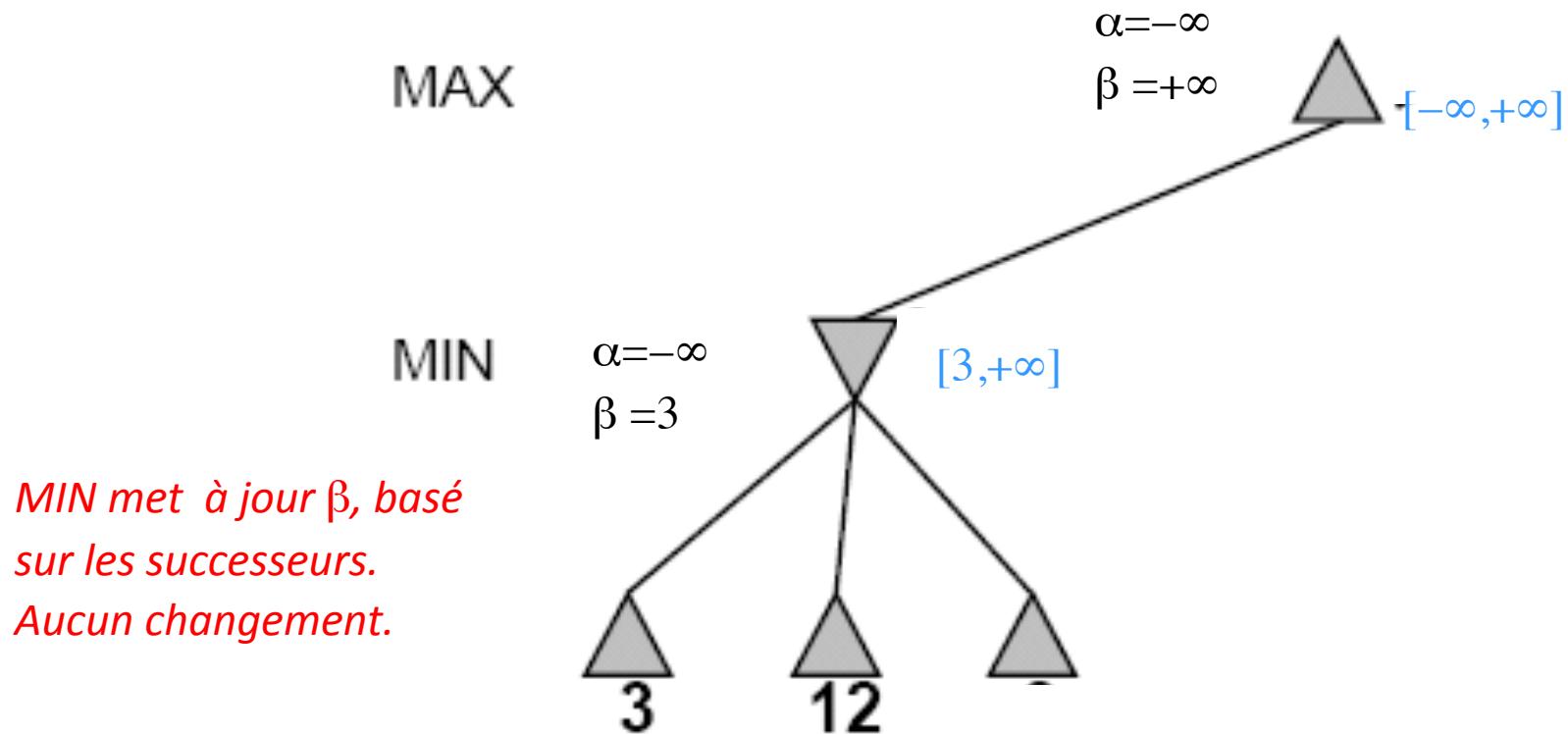
α, β , transmis aux successeurs



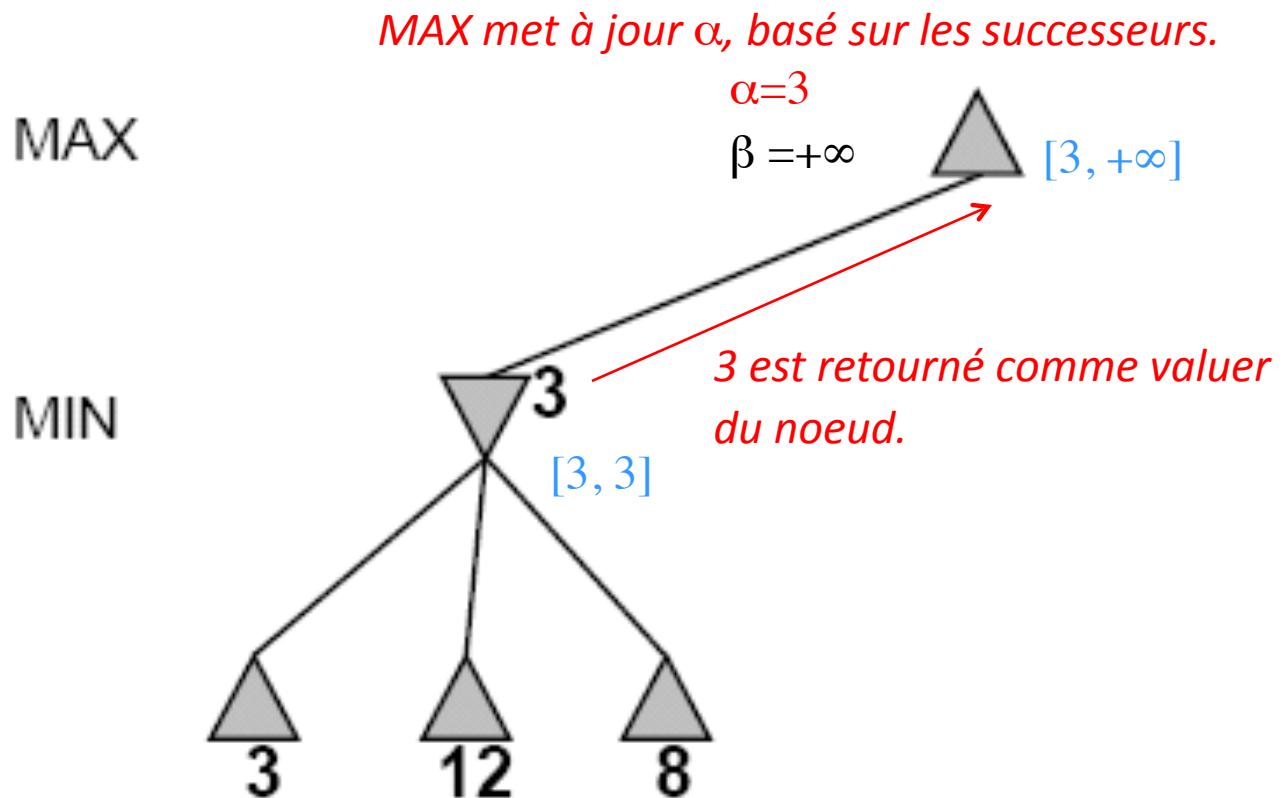
Exemple d'Alpha-beta pruning



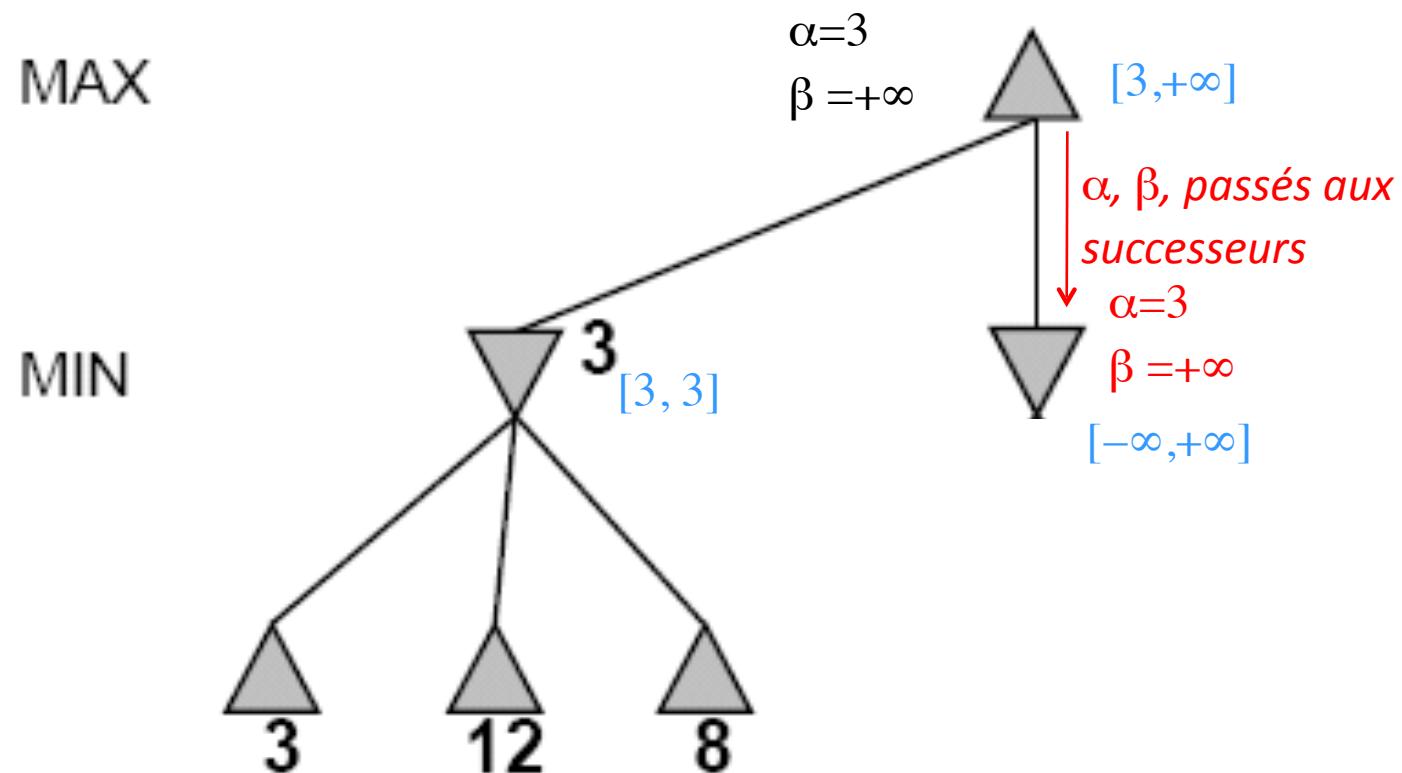
Exemple d'Alpha-beta pruning



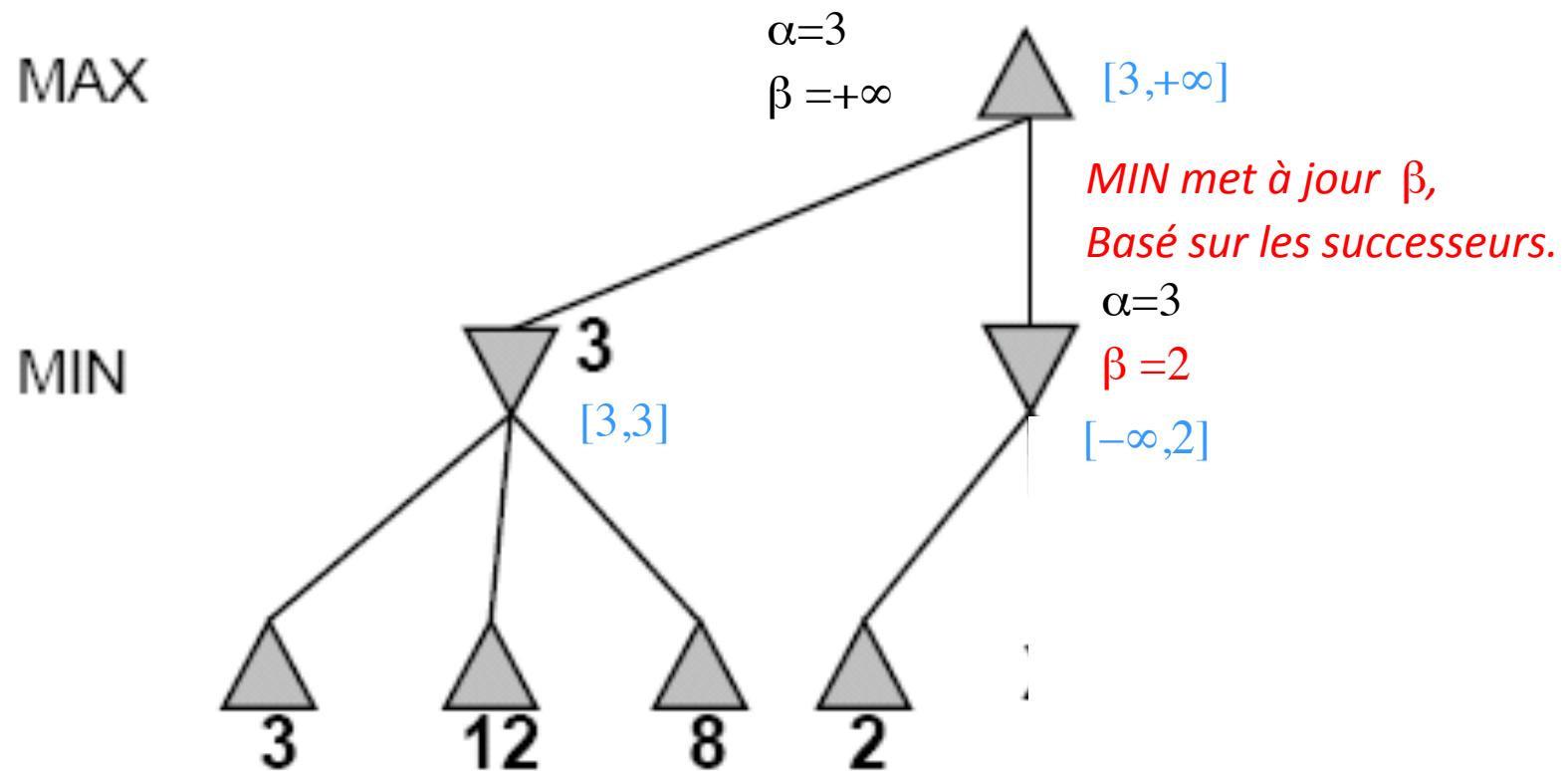
Exemple d'Alpha-beta pruning



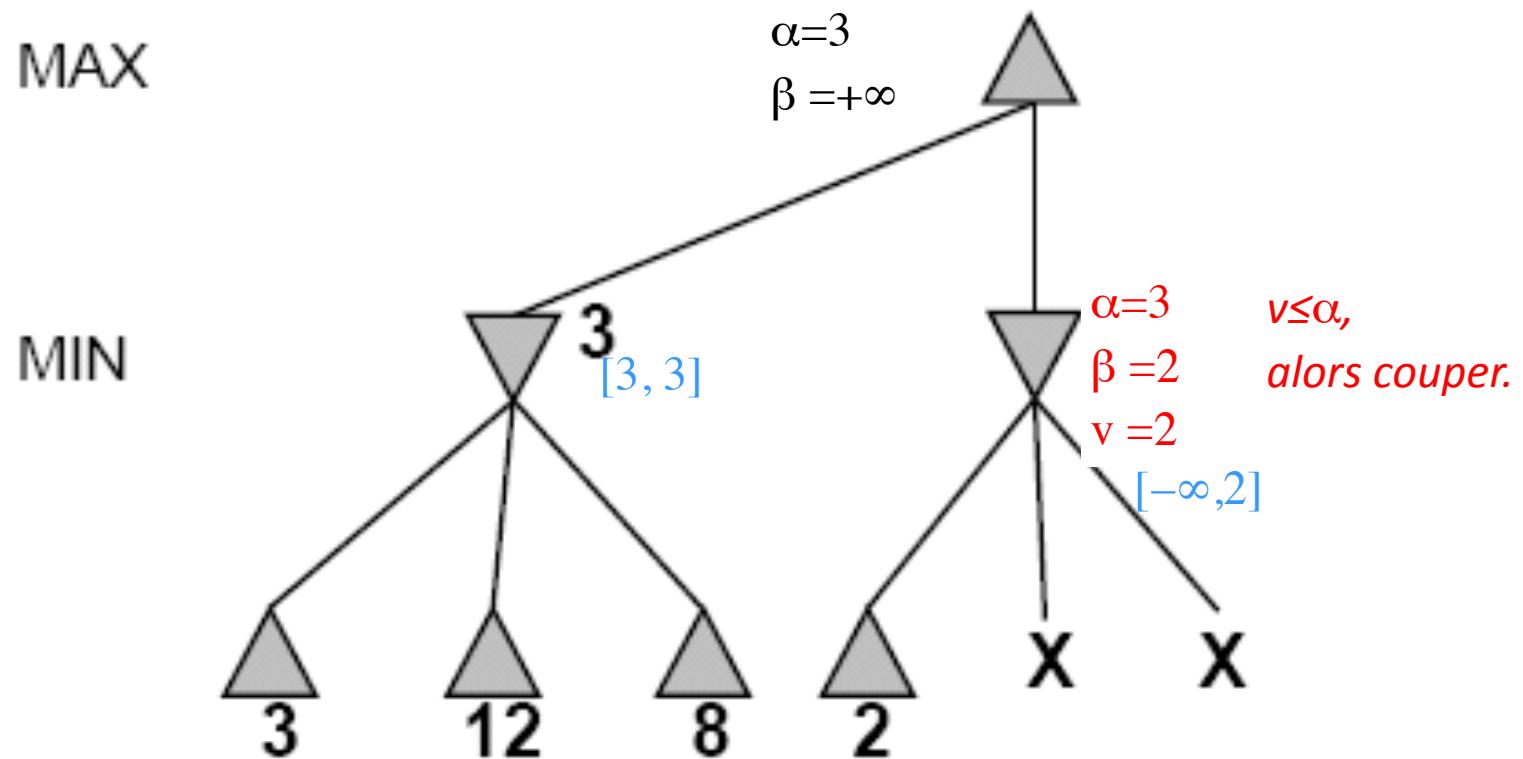
Exemple d'Alpha-beta pruning



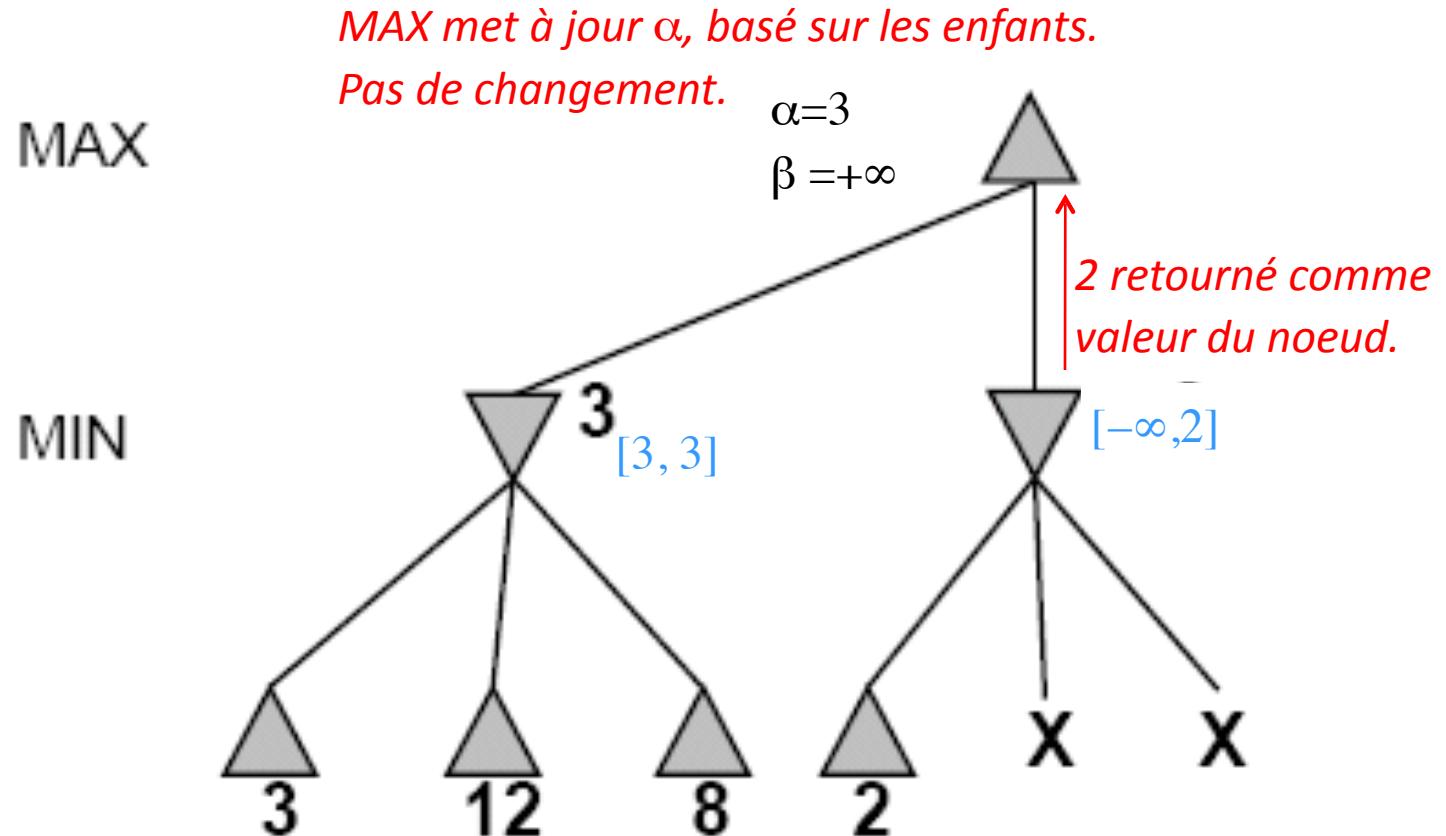
Exemple d'Alpha-beta pruning



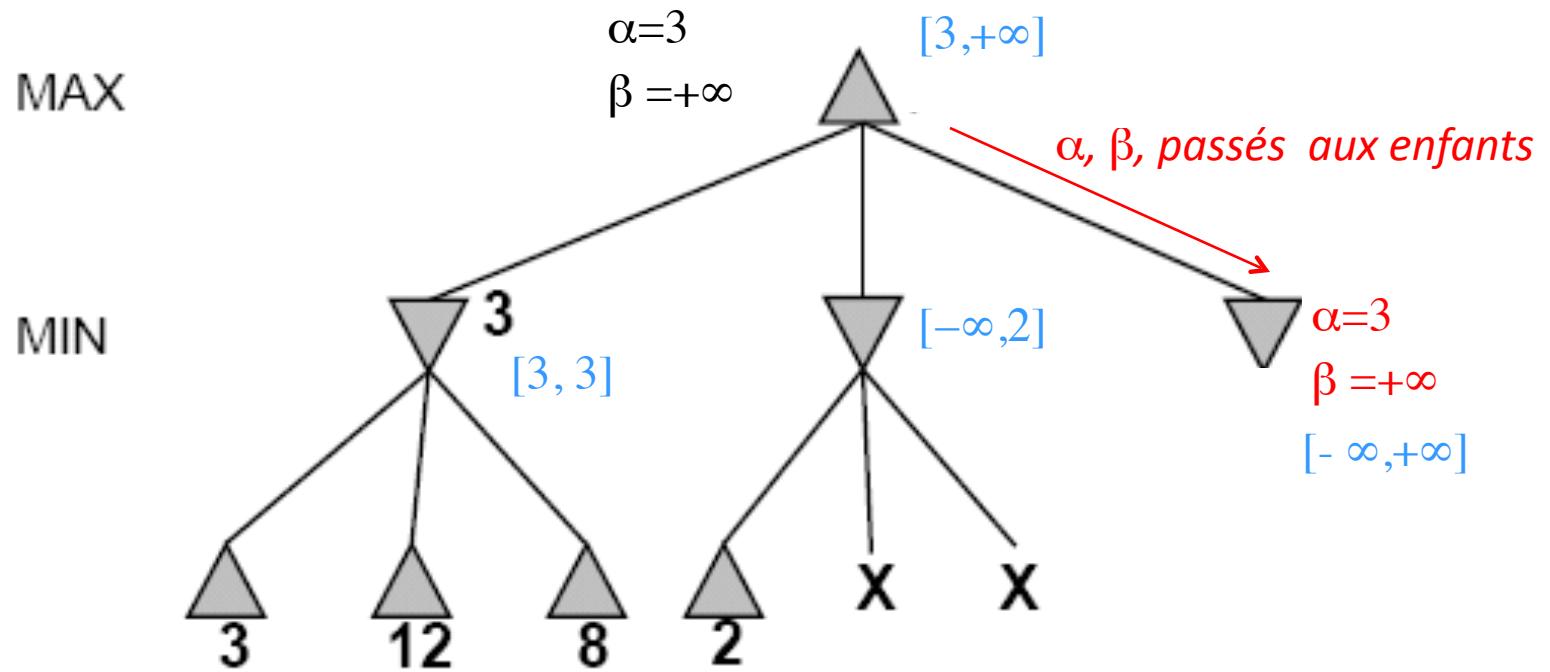
Exemple d'Alpha-beta pruning



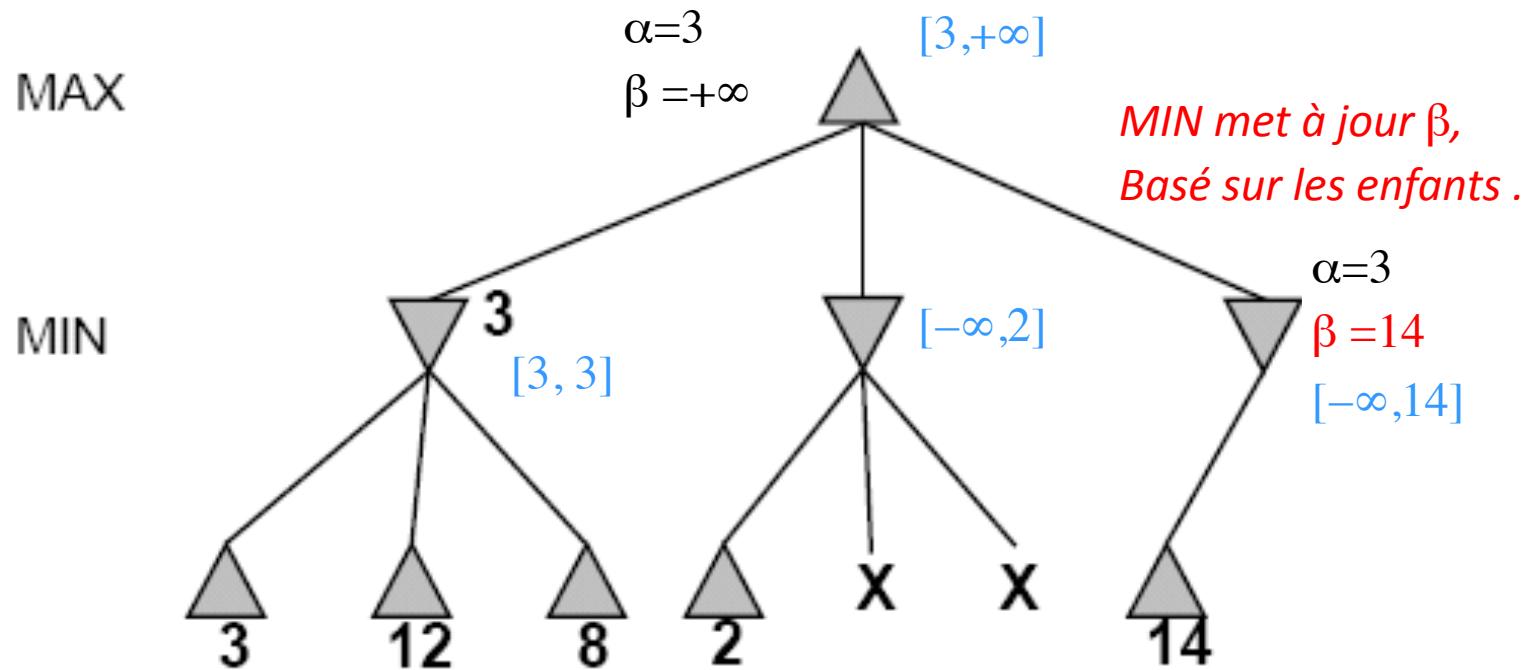
Exemple d'Alpha-beta pruning



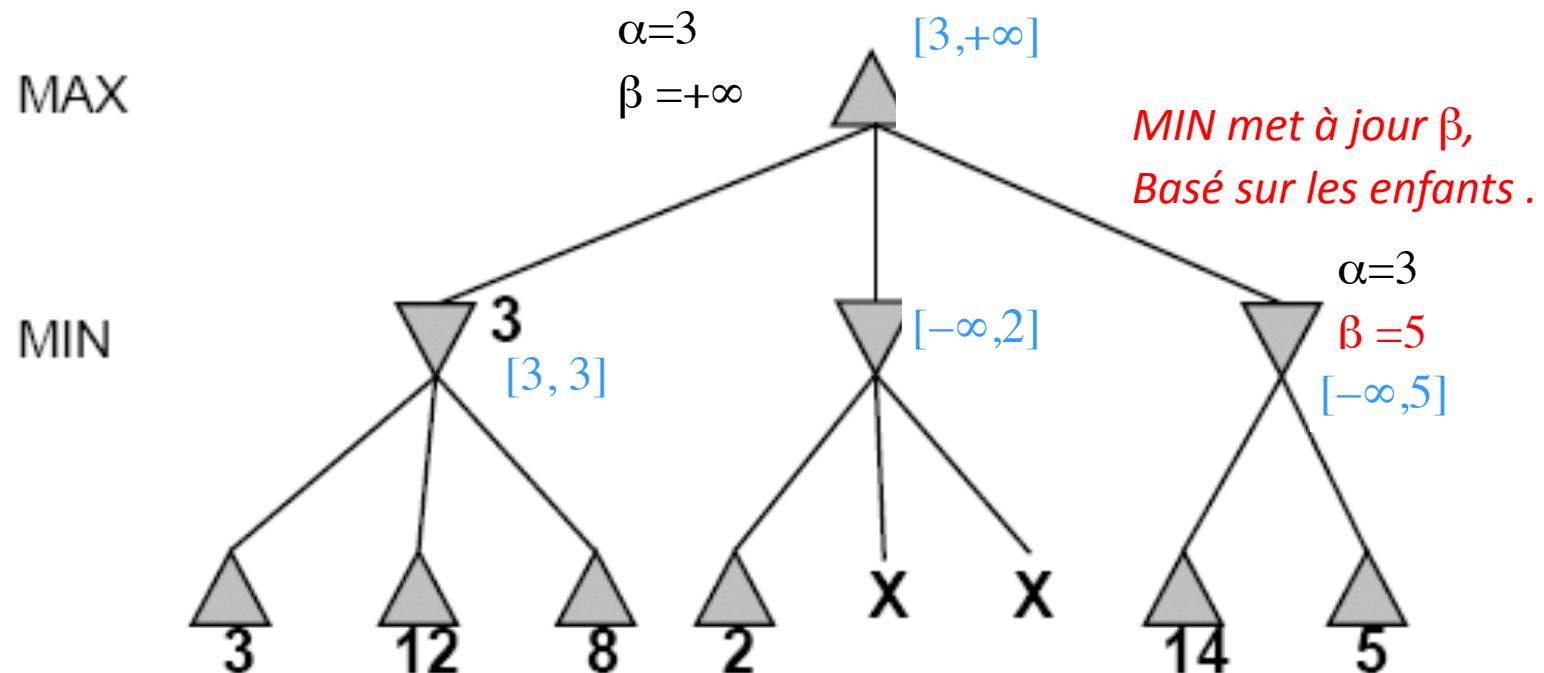
Exemple d'Alpha-beta pruning



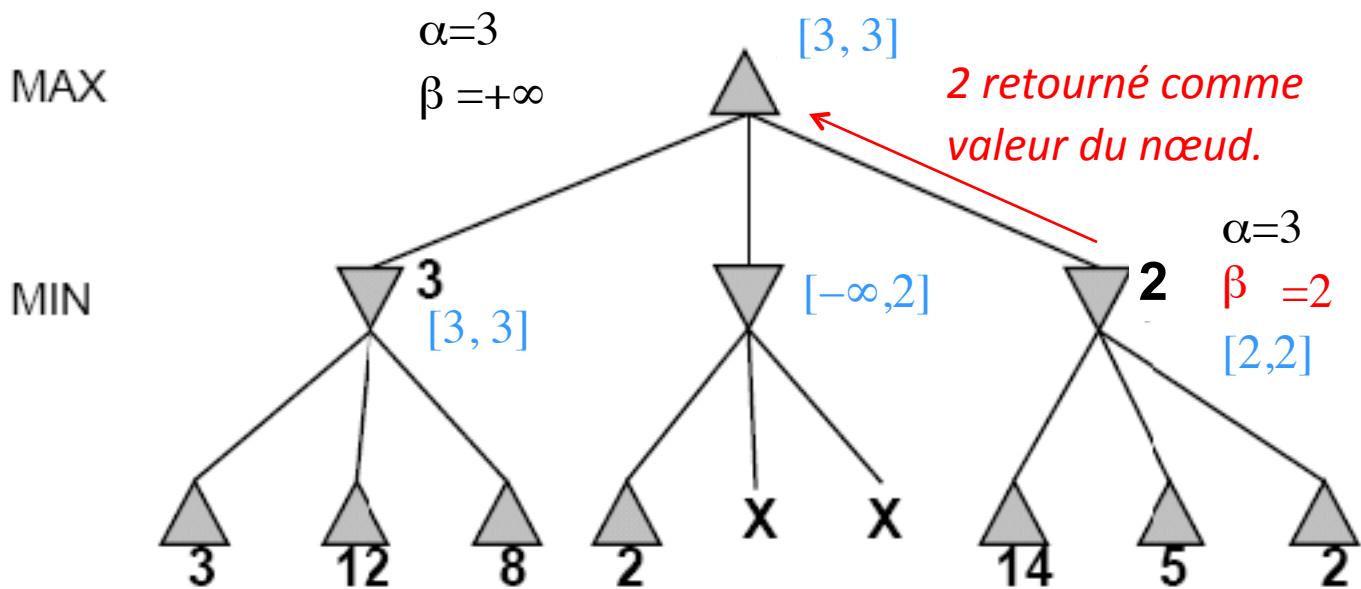
Exemple d'Alpha-beta pruning



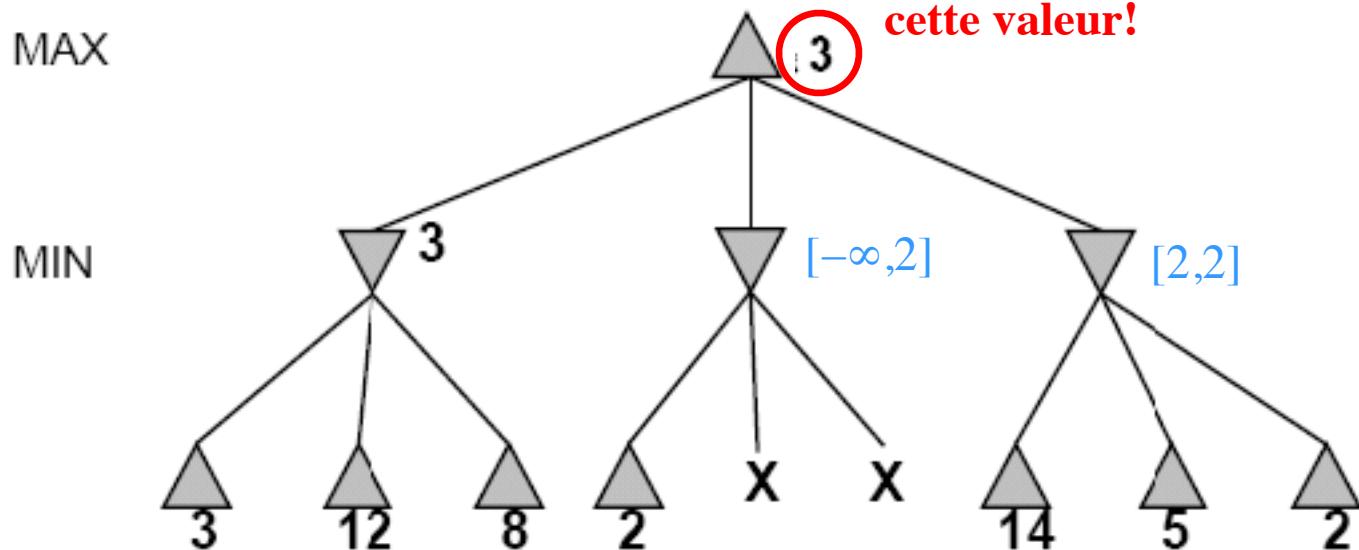
Exemple d'Alpha-beta pruning



Exemple d'Alpha-beta pruning



Exemple d'Alpha-beta pruning



Max calcule la valeur du nœud,
et se déplace vers le nœud ayant
cette valeur!

Alpha-beta pruning

α : MAX's best option on path to root
 β : MIN's best option on path to root

```
def max-value(state,  $\alpha$ ,  $\beta$ ):
```

 initialize $v = -\infty$

 for each successor of state:

$v = \max(v, \text{value}(\text{successor}, \alpha, \beta))$

 if $v \geq \beta$ return v

$\alpha = \max(\alpha, v)$

 return v

```
def min-value(state,  $\alpha$ ,  $\beta$ ):
```

 initialize $v = +\infty$

 for each successor of state:

$v = \min(v, \text{value}(\text{successor}, \alpha, \beta))$

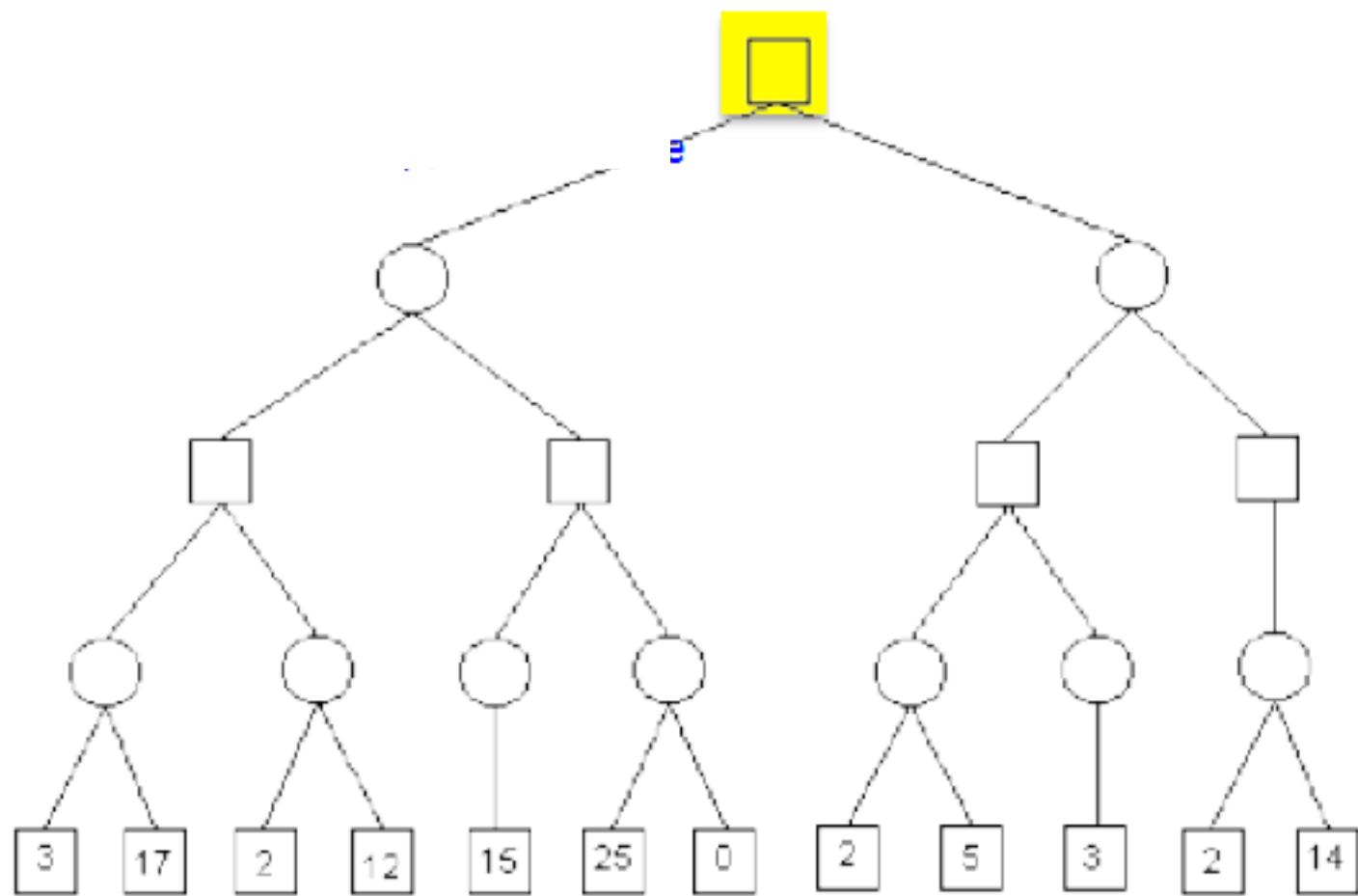
 if $v \leq \alpha$ return v

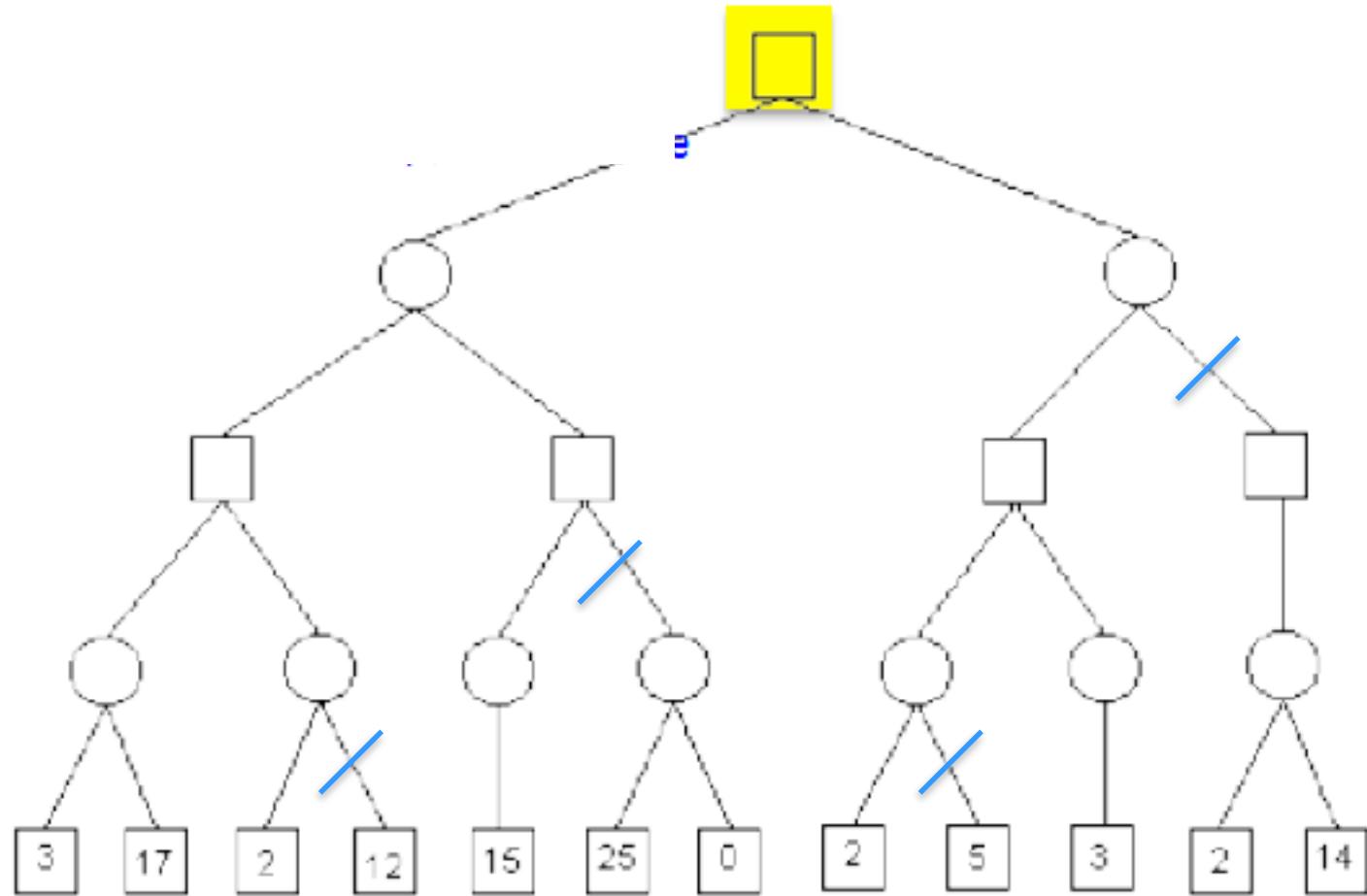
$\beta = \min(\beta, v)$

 return v

Propriétés de *alpha-beta pruning*

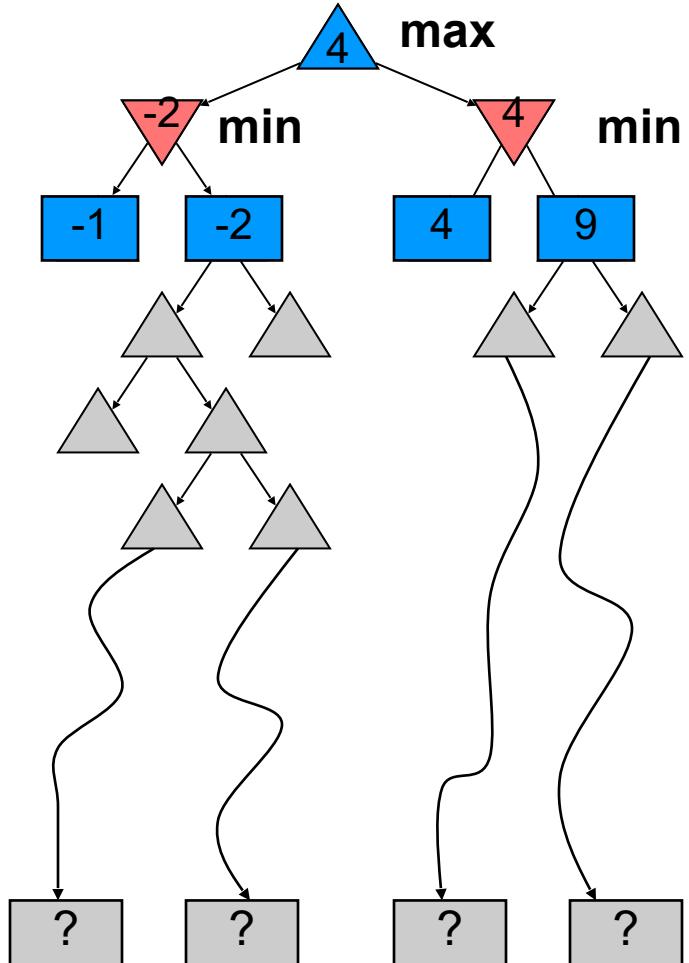
- L'élagage n'affecte pas le résultat final de *minimax*.
- Dans le pire des cas, alpha-beta *pruning* ne fait aucun élagage; il examine b^m nœuds terminaux comme l'algorithme *minimax*:
 - » b : le nombre maximum d'actions/coups légales à chaque étape
 - » m : nombre maximum de coup dans un jeu (profondeur maximale de l'arbre).
- Un bon ordonnancement des actions à chaque nœud améliore l'efficacité.
 - ◆ Dans le meilleur des cas (ordonnancement parfait), la complexité en temps est de $O(b^{m/2})$
 - » On peut faire une recherche deux fois moins profondément comparé à *minimax*!





Décisions en temps réel

- En général, des décisions imparfaites doivent être prises en temps réel :
 - ◆ Pas le temps d'explorer tout l'arbre de jeu
- Approche standard :
 - ◆ couper la recherche :
 - » par exemple, limiter la profondeur de l'arbre
 - ◆ fonction d'évaluation heuristique
 - » estimation de l'utilité qui aurait été obtenue en faisant une recherche complète
 - » on peut voir ça comme une estimation de la « chance » qu'une configuration mènera à une victoire
 - ◆ La solution optimale n'est plus garantie



Exemple de fonction d'évaluation

- Pour le jeu d'échec, une fonction d'évaluation typique est une somme (linéaire) pondérée de “métriques” estimant la qualité de la configuration:
- $$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$
- Par exemple:
 - ◆ w_i = poids du pion,
 - ◆ $f_i(s) = (\text{nombre d'occurrence d'un type de pion d'un joueur}) - (\text{nombre d'occurrence du même type de pion de l'opposant}),$
 - ◆ etc

Exemple de fonction d'évaluation

- Pour le *tic-tac-toe*, supposons que Max joue avec les X.

$\text{Eval}(s) =$

(nombre de ligne, colonnes et diagonales disponibles pour Max) - (nombre de ligne, colonnes et diagonales disponibles pour Min)

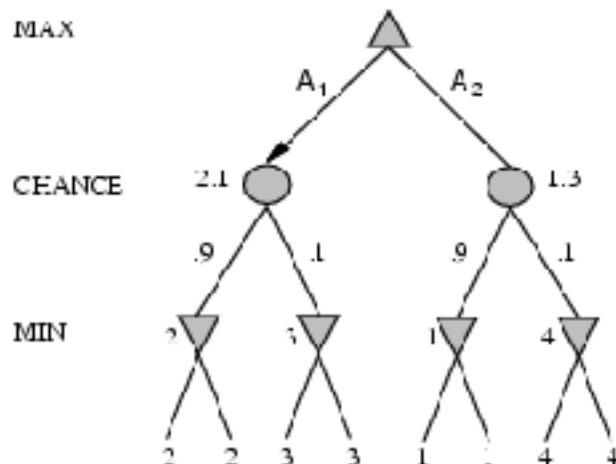
X		O

$$\text{Eval}(s) = 6 - 4 = 2$$

O	X	X
	O	

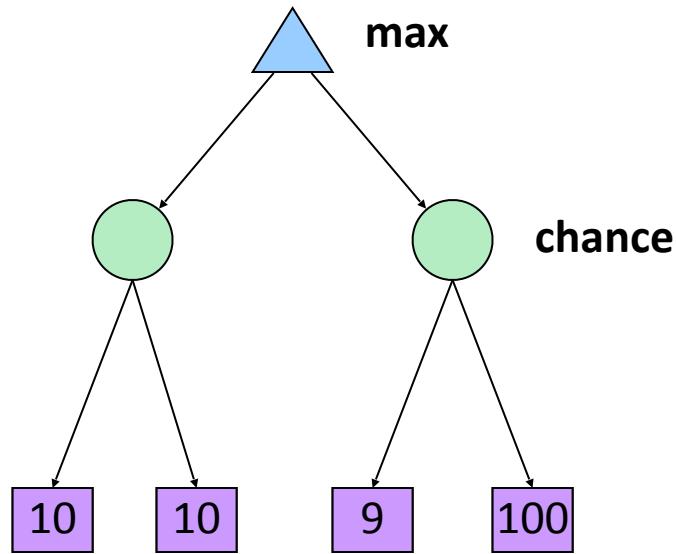
$$\text{Eval}(s) = 4 - 3 = 1$$

Généralisation aux actions aléatoires



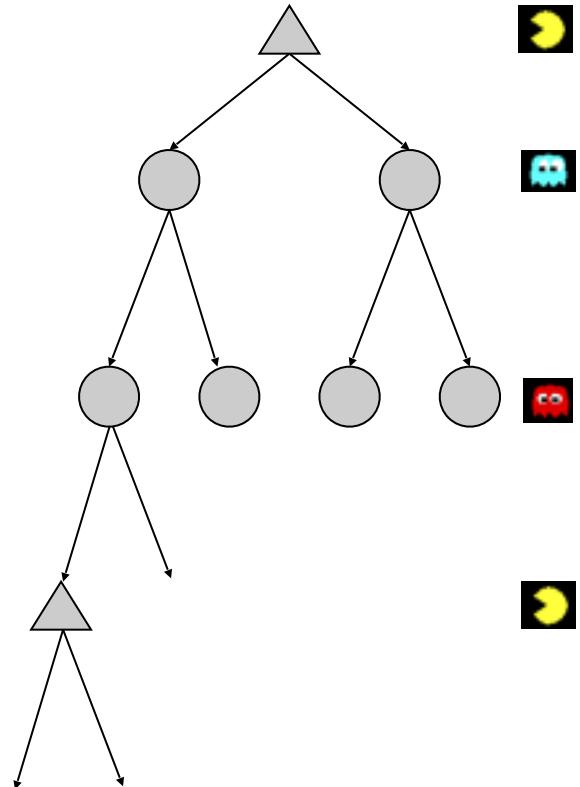
- Exemples :
 - ◆ Jeux où on lance un dé pour déterminer la prochaine action
 - ◆ Actions des fantômes dans Pacman
- **Solution :** On ajoute des nœuds chance, en plus des nœuds Max et Min
 - ◆ L'utilité d'un nœud chance est l'utilité espérée, c.-à-d., la moyenne pondérée de l'utilité de ses enfants

Généralisation aux actions aléatoires



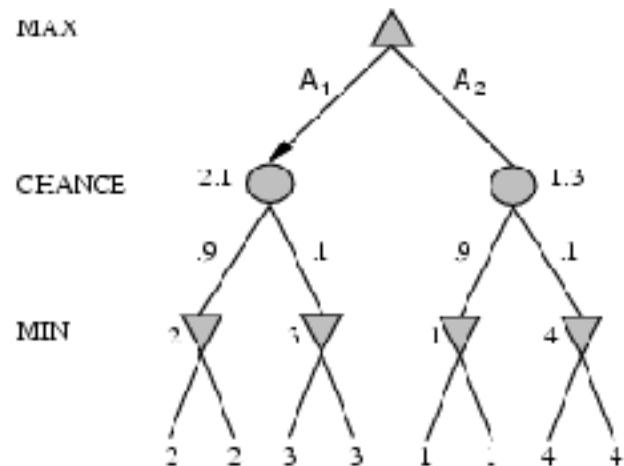
Algorithme Expectimax

- Un modèle probabiliste des comportement de l'opposant:
 - ◆ Le modèle peut être une simple distribution de probabilités
 - ◆ Le modèle peut être plus sophistiqué, demandant des inférences/calculs élaborés
 - ◆ Le modèle peut représenter des actions stochastiques/incontrôlables (à cause de l'opposant, l'environnement)
 - ◆ Le modèle pourrait signifier que des actions de l'adversaire sont probables
- nous supposons que nous avons une distribution de probabilités à associer aux actions de l'adversaire/environnement



Avoir une croyance probabiliste sur les actions d'un agent ne signifie pas que l'agent lance effectivement un dé!

Algorithme Expectimax



EXPECTIMINIMAX (n) =

UTILITY(n)

Si n est un nœud terminal

$\max_{s \in \text{successors}(n)} \text{EXPECTIMINIMAX}(s)$

Si n est un nœud Max

$\min_{s \in \text{successors}(n)} \text{EXPECTIMINIMAX}(s)$

Si n est un nœud Min

$\sum_{s \in \text{successors}(n)} P(s) * \text{EXPECTIMINIMAX}(s)$ Si n est un nœud chance

Ces équations donne la programmation récursive des valeurs jusqu'à la racine de l'arbre.
162

Constraint Satisfaction Problems

Constraint Satisfaction Problem

- Problème de recherche:
 - ◆ Un état est une boite noire: structure de données arbitraire
 - ◆ test du but fonction sur les états
 - ◆ fonction successeur n'importe quelle fonction sur les états
- Constraint satisfaction problems (CSPs):
 - ◆ un cas particulier de pb de recherche
 - ◆ un état est défini par : des variables V_i (features vector) ayants des valeurs du domaine D (D dépend de i)
 - ◆ test du but : ensemble de contraintes spécifiant les combinaisons valables de valeur pour les sous-ensembles de variables

formalisation CSP

- Formellement, un problème de satisfaction de contraintes (ou CSP pour Constraint Satisfaction Problem) est défini par:
 - ◆ Un ensemble fini de variables V_1, \dots, V_n .
 - » Chaque variable V_i a un domaine D_i de valeurs permises.
 - ◆ Un ensemble fini de contraintes C_1, \dots, C_m sur les variables.
 - » Une contrainte restreint les valeurs pour un sous-ensemble de variables (scope).
 - » $C_1(V_1, V_2, V_4)$ contrainte sur les variables V_1, V_2, V_4
 - ◆ Un état d'un problème CSP est défini par une assignation de valeurs à certaines variables ou à toutes les variables.
 - » $\{V_i=v_i, V_1=v_1, \dots\}$.
 - » une contrainte retourne
 - vrai si l'affectation satisfait la contrainte
 - faux si l'affectation ne satisfait pas la contrainte

formalisation CSP

- Formellement, un problème de satisfaction de contraintes (ou CSP pour Constraint Satisfaction Problem) est défini par:
 - ◆ Un ensemble fini de variables V_1, \dots, V_n .
 - » Chaque variable V_i a un domaine D_i de valeurs permises.
 - ◆ Un ensemble fini de contraintes C_1, \dots, C_m sur les variables.
 - » Une contrainte restreint les valeurs pour un sous-ensemble de variables (scope).
 - » $C_1(V_1, V_2, V_4)$ contrainte sur les variables V_1, V_2, V_4
 - ◆ Un état d'un problème CSP est défini par une assignation de valeurs à certaines variables ou à toutes les variables.
 - » $\{V_i=v_i, V_1=v_1, \dots\}$.
 - » une contrainte retourne
 - vrai si l'affectation satisfait la contrainte
 - faux si l'affectation ne satisfait pas la contrainte

formalisation CSP

- Constraintes
 - ◆ contrainte unaire : contrainte sur une variable
 - » $C(X): X=2; C(Y): Y>2$
 - ◆ contrainte binaire : contrainte entre 2 variables
 - » $C(X,Y) : X + Y < 6$
 - ◆ contrainte d'ordre supérieur : contrainte sur 3 ou plus de variables
- Solutions
 - ◆ une solution est une affectation de valeurs à toutes les variables de telle sorte que chaque contrainte est satisfaite.
 - ◆ un CSP est non-satisfait si aucune solution existe.

Exemple: Sudoku

- Variables: $V_{11}, V_{12}, \dots, V_{21}, V_{22}, \dots, V_{91}, \dots, V_{99}$
- Domaines:
 - ◆ $\text{Dom}[V_{ij}] = \{1-9\}$ pour les cellules vides
 - ◆ $\text{Dom}[V_{ij}] = \{k\}$ une valeur fixe k pour les cellules pleines

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

Exemple: Sodoku

- Contraintes:
 - ◆ contraintes ligne:
 - » All-Diff(V11, V12, V13, ..., V19)
 - » All-Diff(V21, V22, V23, ..., V29)
 - », All-Diff(V91, V92, ..., V99)

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

Exemple: Sodoku

- Contraintes:
 - ◆ Contraintes colonnes:
 - » All-Diff(V11, V21, V31, ..., V91)
 - » All-Diff(V21, V22, V13, ..., V92)
 - » ..., All-Diff(V19, V29, ..., V99)

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

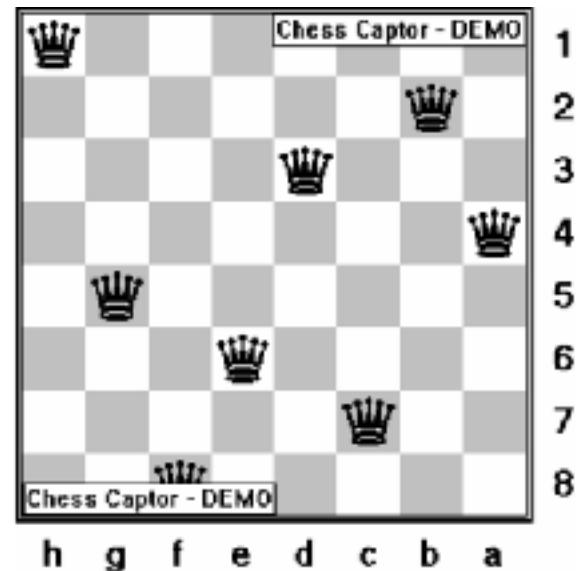
Exemple: Sudoku

- Contraintes:
 - ◆ Contraintes sous-carré:
 - » All-Diff(V11, V12, V13,
V21, V22, V23, V31,
V32, V33)
 - » ..., All-Diff(V77, V78,
V79,..., V97, V98, V99)
- Contraintes:
 - ◆ 3 x 9 all diff contraintes.

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

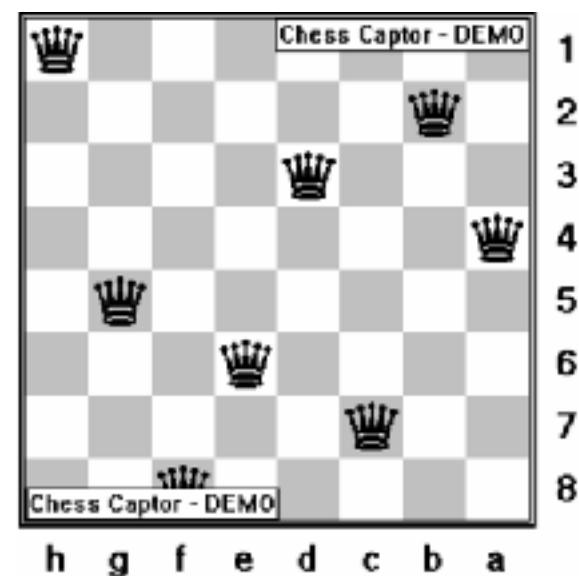
Exemple: N-Reines

- Placer N reines sur un échiquier N x N de telle sorte qu'aucune reine attaque une autre reine
- Une reine peut attaquer une autre si elles sont toutes les deux sur: la même ligne, la même colonne, ou la même diagonale.



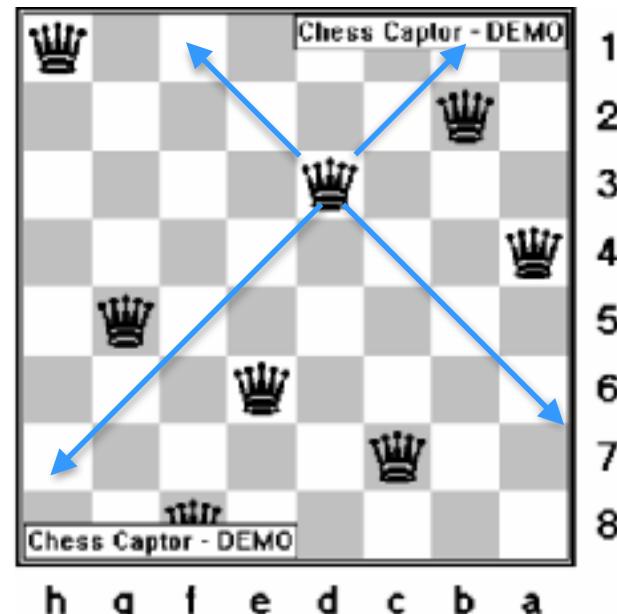
Exemple: N-Reines

- Variables : Q₁ ... Q_N correspondant aux colonnes 1, ..., N.
- Domaines :
 - ◆ Chaque variable a le domaine de valeurs {1, ..., N}
 - ◆ La colonne i a la valeur k si la reine (Queen) dans la colonne i est dans la rangée k.
 - ◆ cette représentation a N^N états:
 - ◆ pour 8-Queens: $8^8 = 16,777,216$



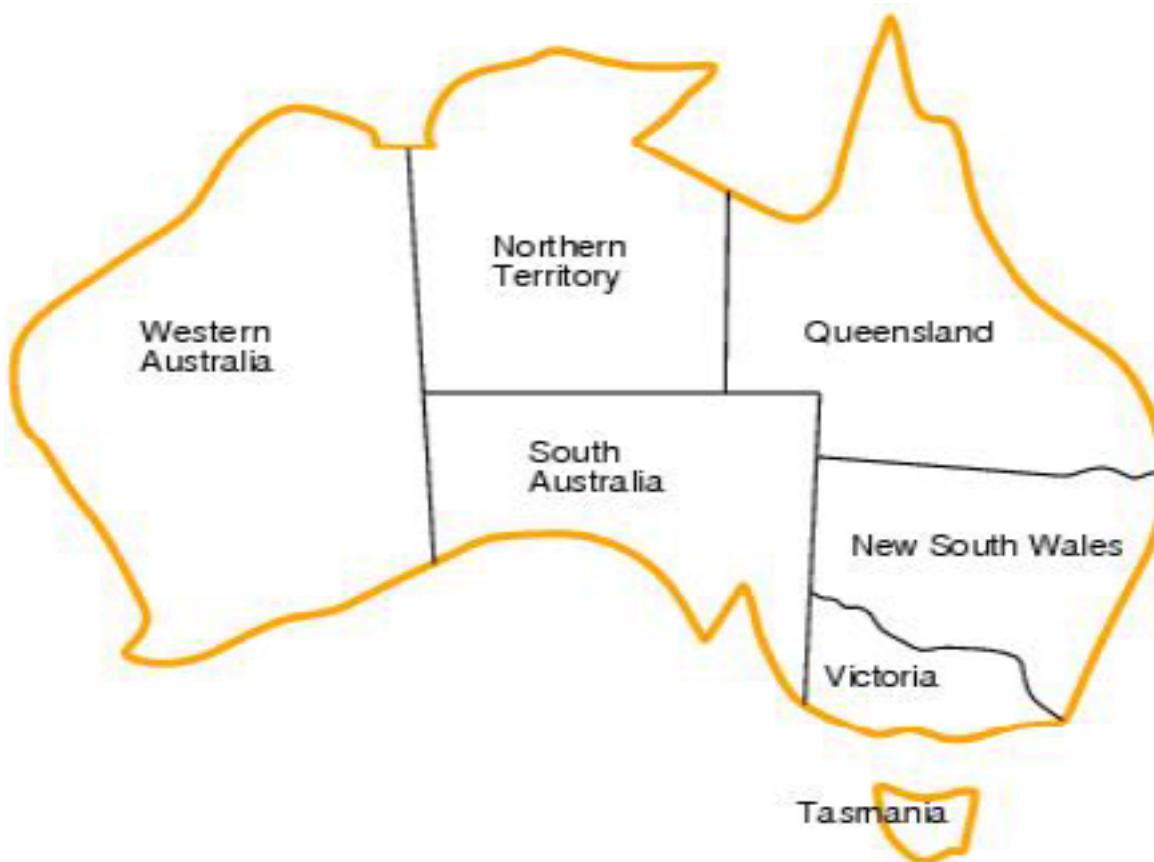
Exemple:N-Reines

- Contraintes
 - ◆ ne pas mettre 2 reines dans une même ligne
 - » $Q_i \neq Q_j$ for all $i \neq j$
 - ◆ contraintes diagonale
 - » $\text{abs}(Q_i - Q_j) \neq \text{abs}(i - j)$
 - » i.e., la difference des valeurs de Q_i et Q_j est différentes des valeurs i et j .



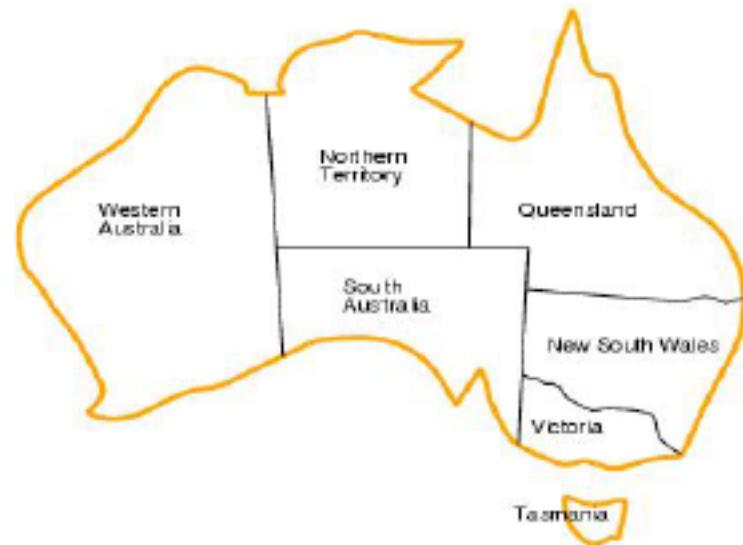
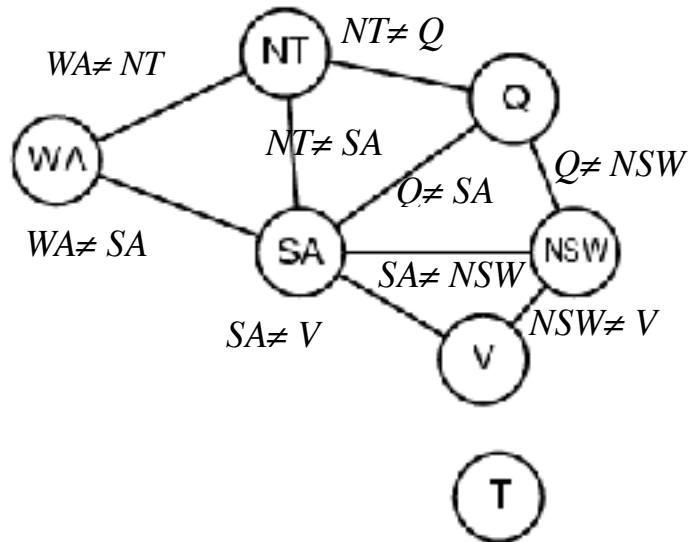
Exemple: map coloring

- Utiliser seulement trois couleurs (**rouge**, **vert** et **bleu**) de sorte que deux états frontaliers n'aient jamais les mêmes couleurs.



Exemple:map coloring

- Modelisation
 - ◆ Variables: WA, NT, Q, NSW, V, SA, T
 - ◆ Domaines: Di={rouge, vert, bleu}
 - ◆ Contraintes: des regions adjacente doivent avoir des couleurs différentes.
 - » E.g. $WA \neq NT$

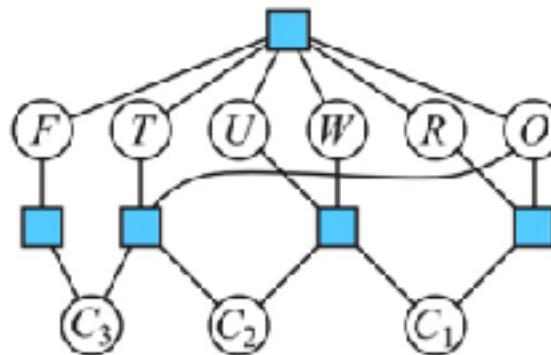


Exemple:map coloring



Cryptarithmetic

$$\begin{array}{r} T \ W \ O \\ + T \ W \ O \\ \hline F \ O \ U \ R \end{array}$$



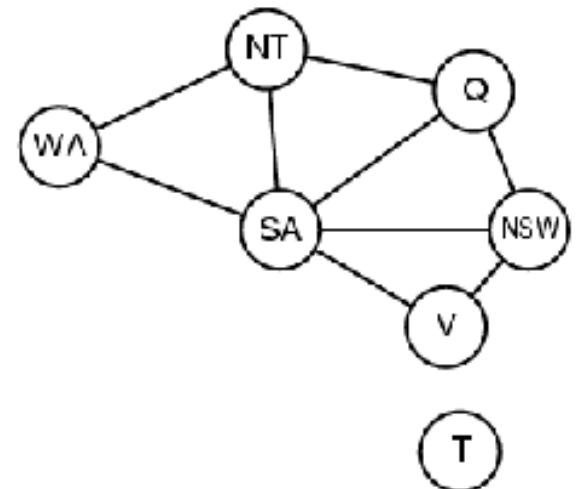
- Variables: $X = \{F, T, U, W, R, O, C1, C2, C3\}$
- Domain: $D = \{0, 1, 2, \dots, 9\}$
- Constraints:
 - $\text{Alldiff}(F, T, U, W, R, O)$
 - $O + O = R + 10 \leftarrow C1$
 - $C1 + W + W = U + 10 \leftarrow C2$
 - $C2 + T + T = O + 10 \leftarrow C3$
 - $C3 = F$

Recherche pour CSP

- Un état est une affectation.
- **État initial** : affectation vide { }
- **Fonction successeur** : affecte une valeur à une variable non encore affectée.
- **But** : affectation complète et consistante.
- Comme la solution doit être complète, elle apparaît à une profondeur n, si nous avons n variables.
- ici le chemin à la solution est sans importance.

DFS pour CSP

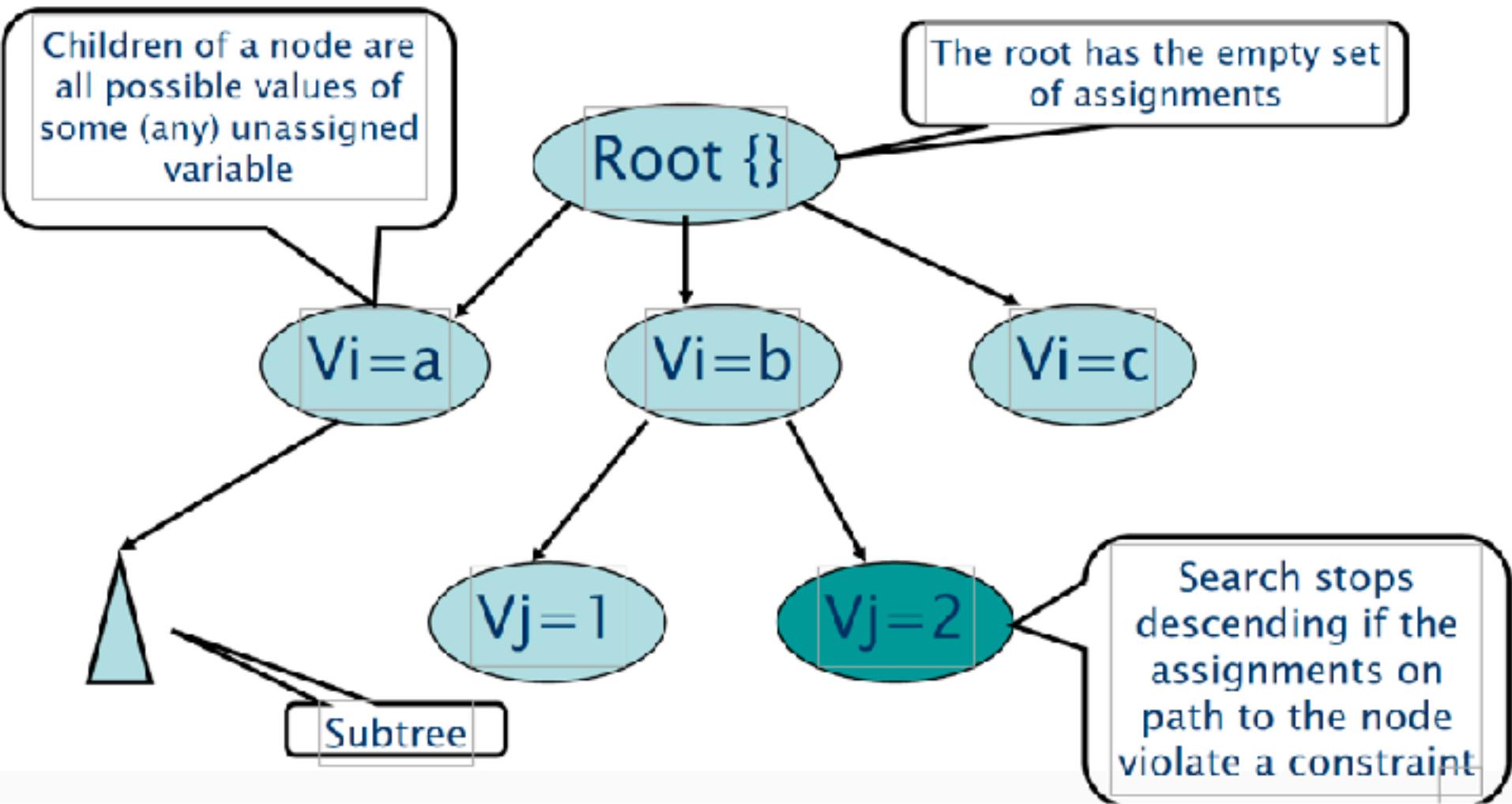
- Que fait BFS ?
 - ◆ on va explorer même des configurations non possibles
 - ◆ développer tout l'arbre
- Que fait DFS ?
 - ◆ oui mais complexité en temps



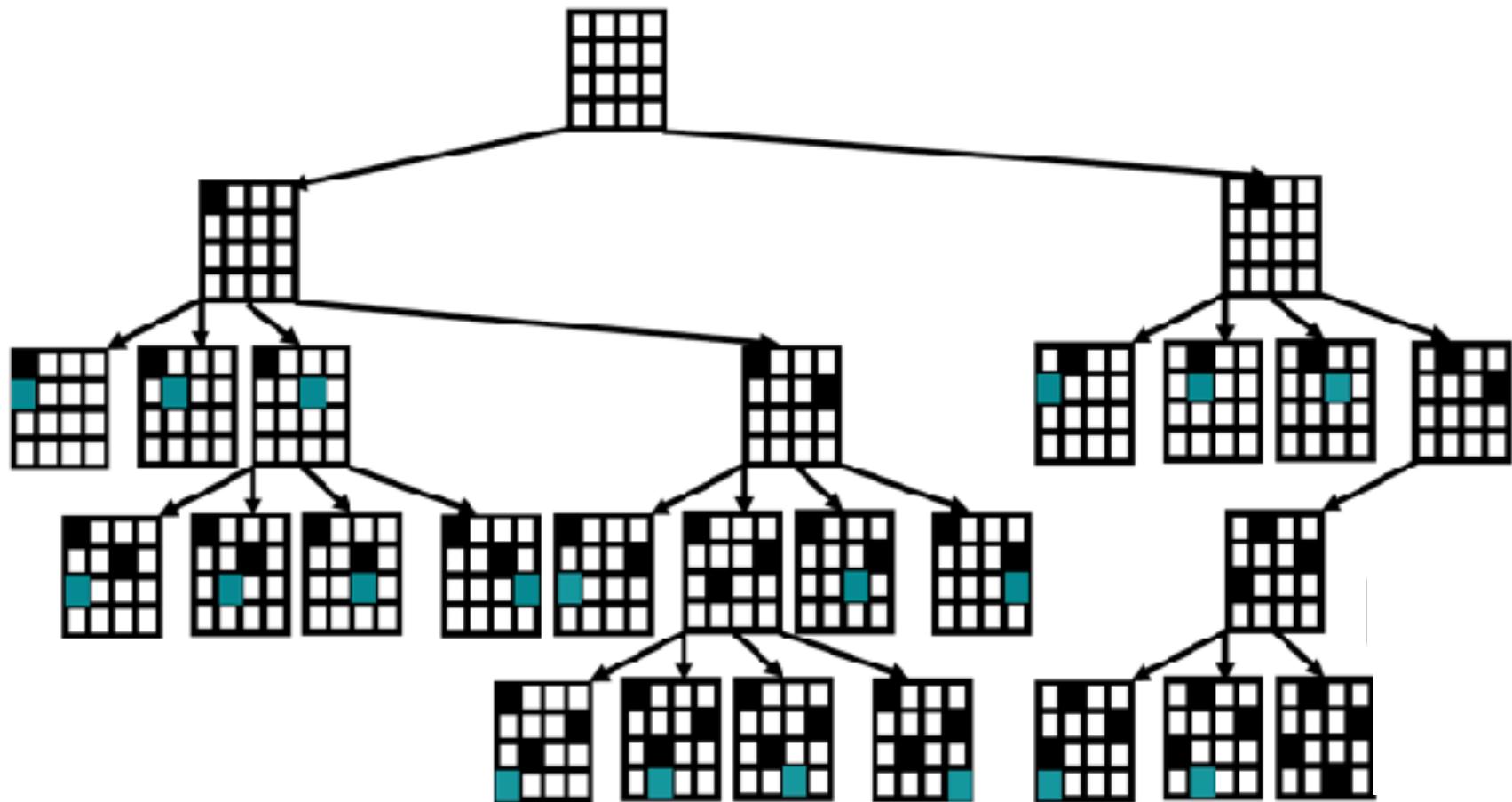
Backtracking Search

- CSPs sont le mieux résolu en utilisant un algorithme spécifique de recherche: Backtracking Search.
 - ◆ Version spécifique de depth first search
 - ◆ Explore des affectations partielles de variables.
 - ◆ Un noeud est abandonné s'il viole une contrainte
 - ◆ Un noeud spécifiant une affectation totale des variables est une solution.

Backtracking Search



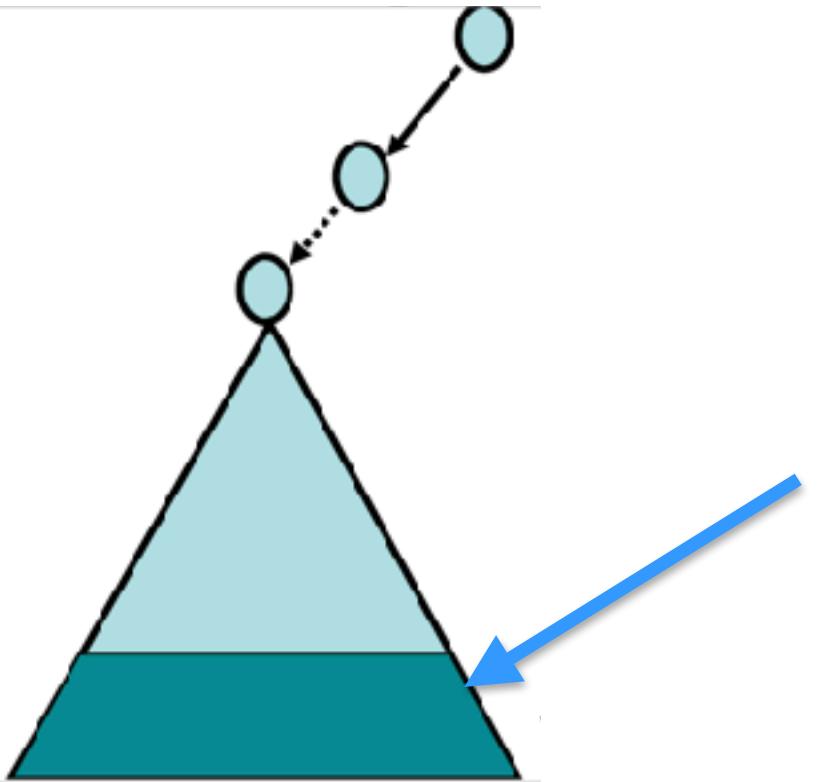
Backtracking Search



Backtracking Search

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)
function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment given CONSTRAINTS[csp] then
      add {var = value} to assignment
      result  $\leftarrow$  RECURSIVE-BACKTRACKING(assignment, csp)
      if result  $\neq$  failure then return result
      remove {var = value} from assignment
  return failure
```

Backtracking Search



Variables n'ont pas
d'affectations possible
mais nous détectons
cela que quand nous
voulons leur affecter
une valeur

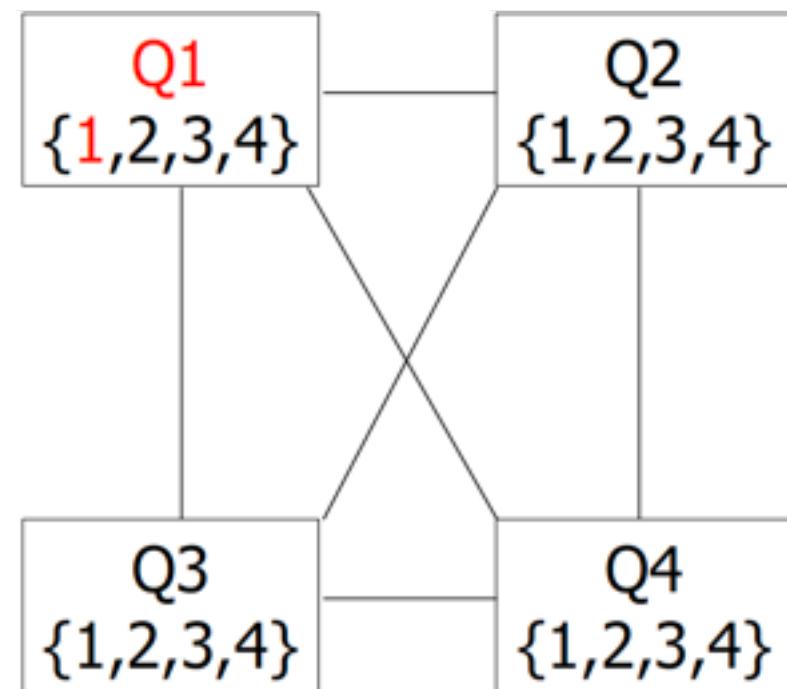
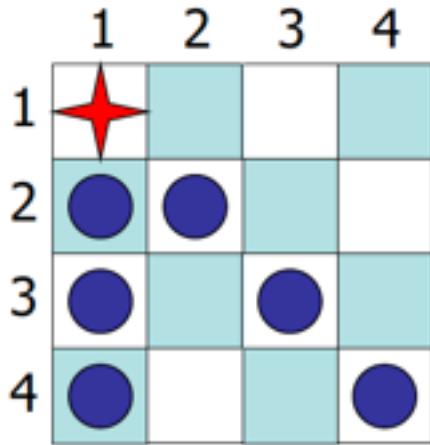
Backtracking Search

- Sans heuristiques, l'algorithme est limité.
- Des heuristiques générales peuvent améliorer l'algorithme significativement :
 - ◆ Choisir judicieusement la prochaine variable:
 - » SELECT-UNASSIGNED-VARIABLE
 - ◆ Choisir judicieusement la prochaine valeur à affecter:
 - » ORDER-DOMAIN-VALUES
- Faire des inférences pour détecter plus tôt les affectations conflictuelles:
 - ◆ INFERENCES (forward checking, arc consistency...)

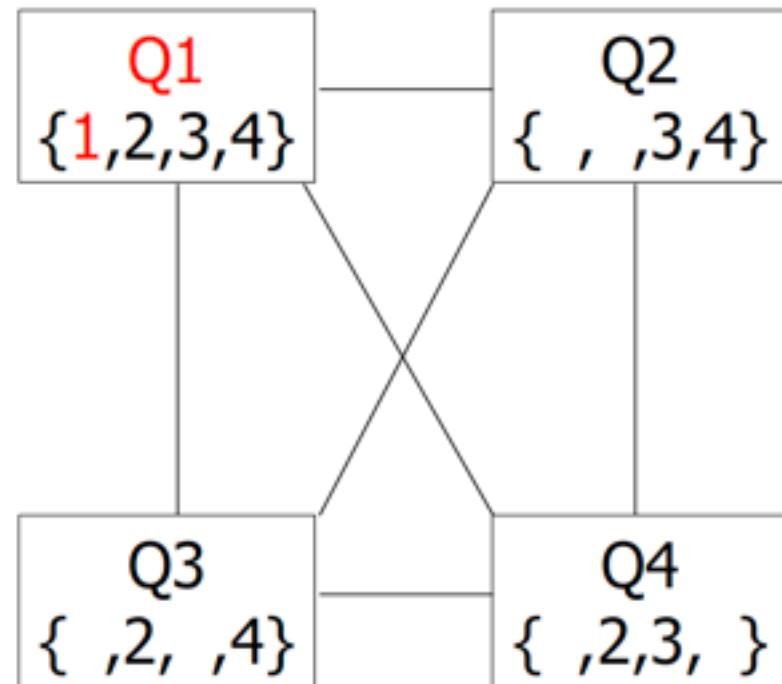
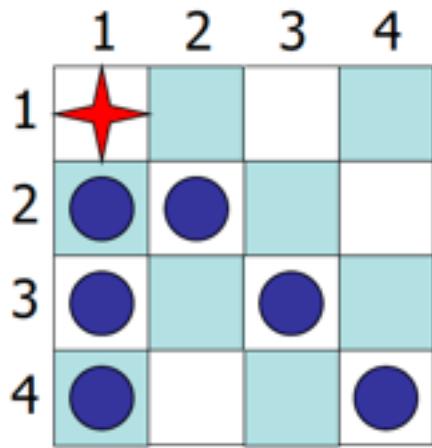
forward-checking

- L'idée de forward-checking (vérification anticipative) est :
 - ◆ vérifier les valeurs compatibles des variables non encore affectées
 - ◆ terminer la récursivité (conflit) lorsqu'une variable (non encore assignée) a son ensemble de valeurs compatibles qui devient vide

forward-checking

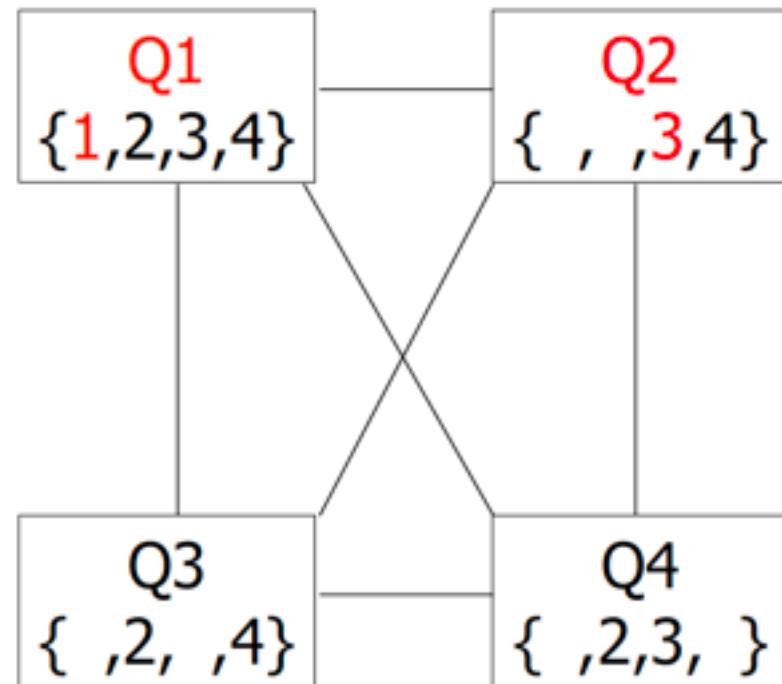


forward-checking

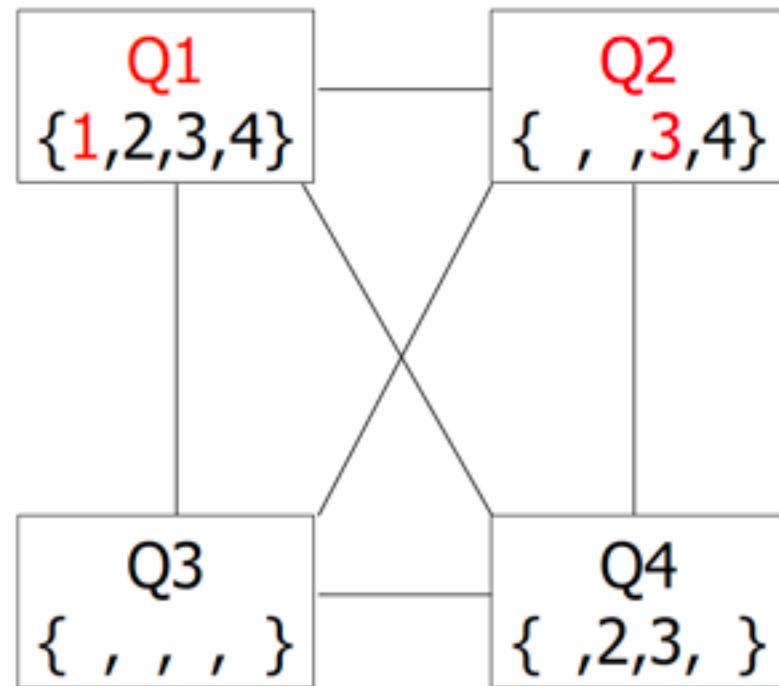
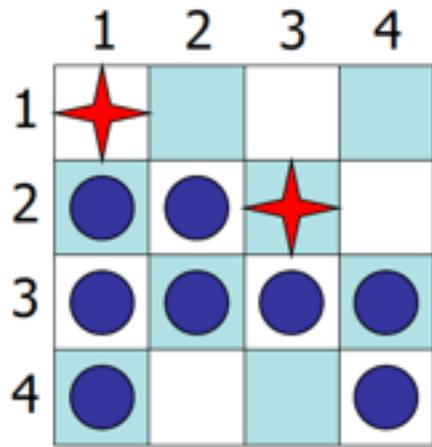


forward-checking

	1	2	3	4
1	★			
2	●	●	★	
3	●		●	
4	●			●

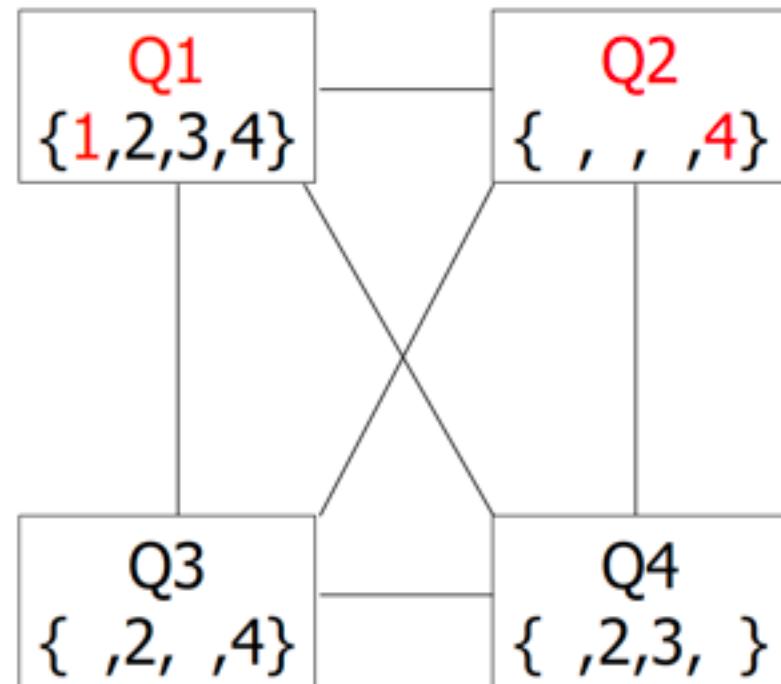


forward-checking

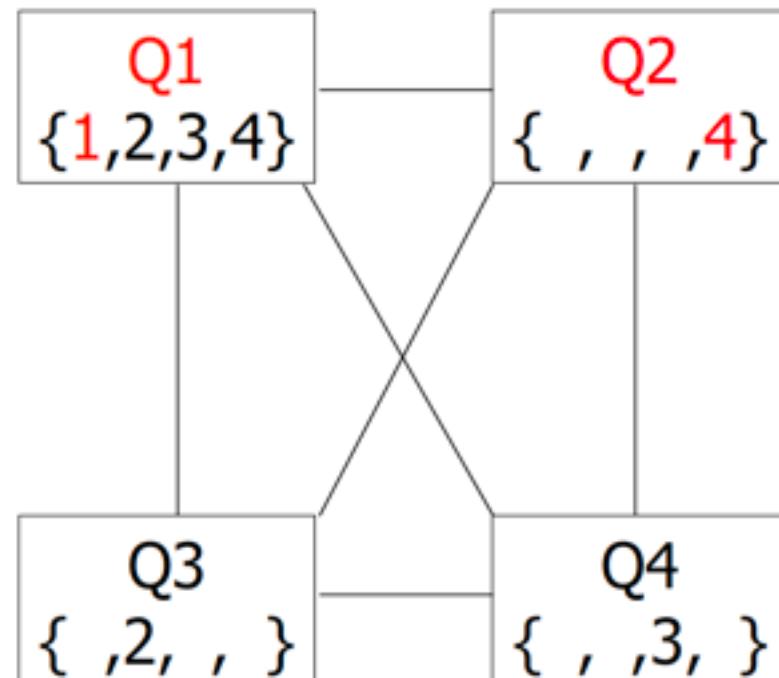
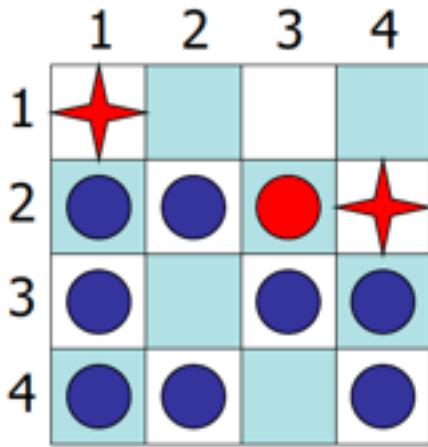


forward-checking

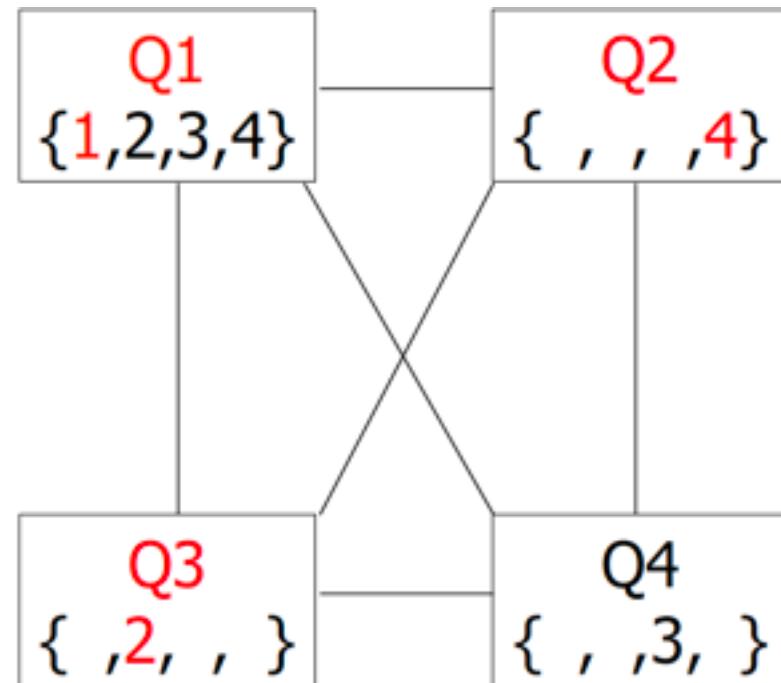
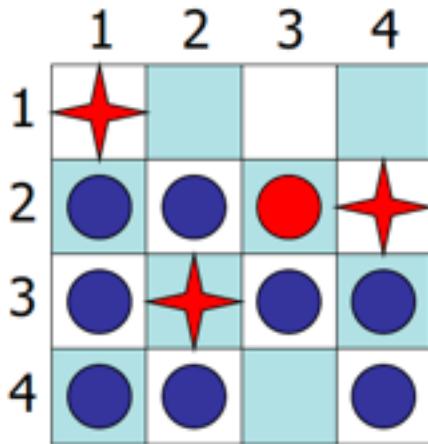
	1	2	3	4
1	★			
2	●	●	●	★
3	●		●	
4	●			●



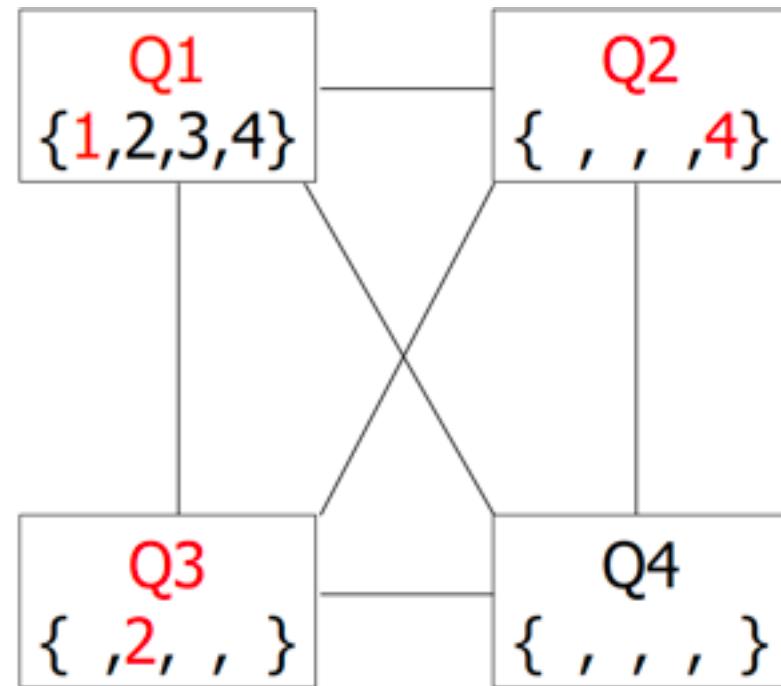
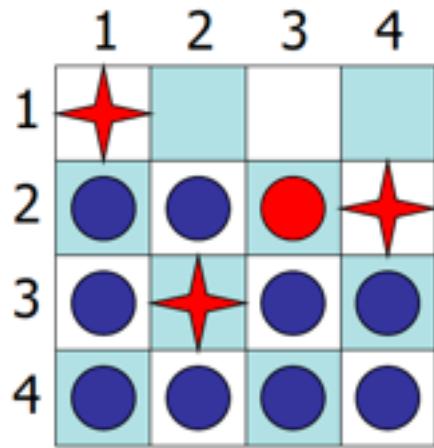
forward-checking



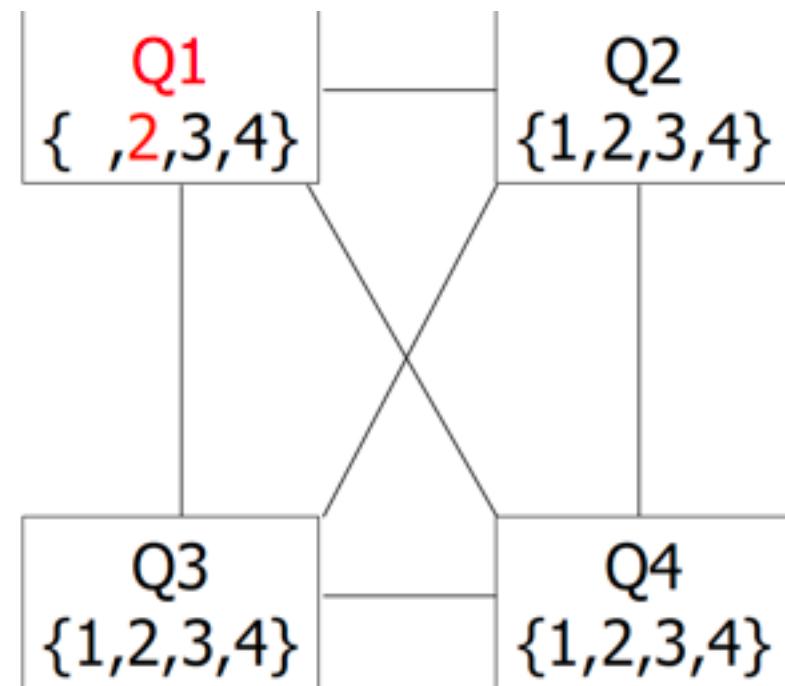
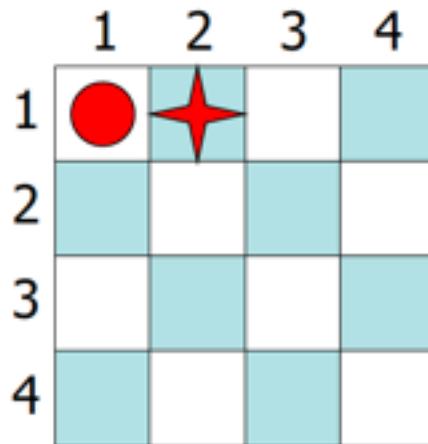
forward-checking



forward-checking

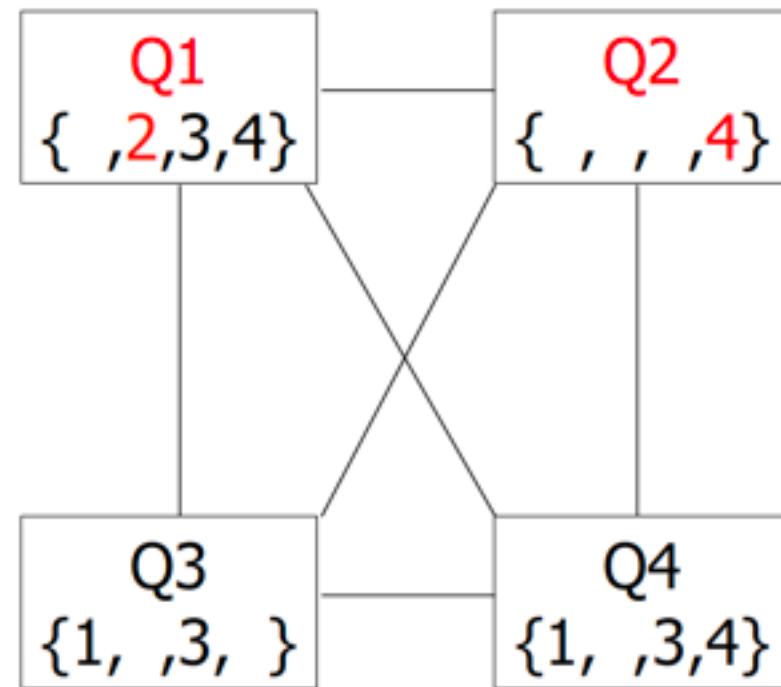


forward-checking

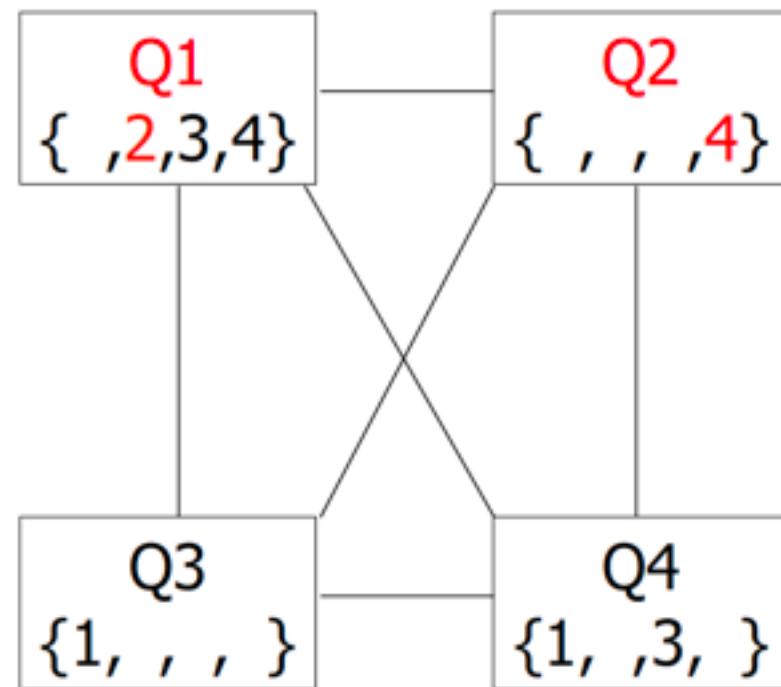
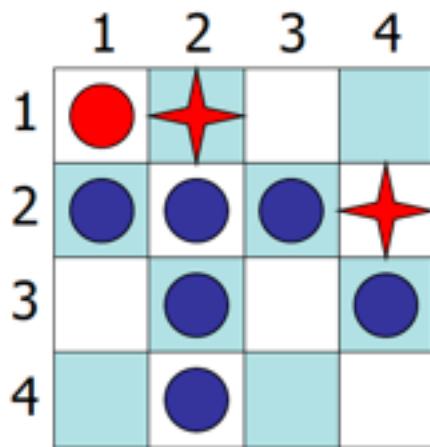


forward-checking

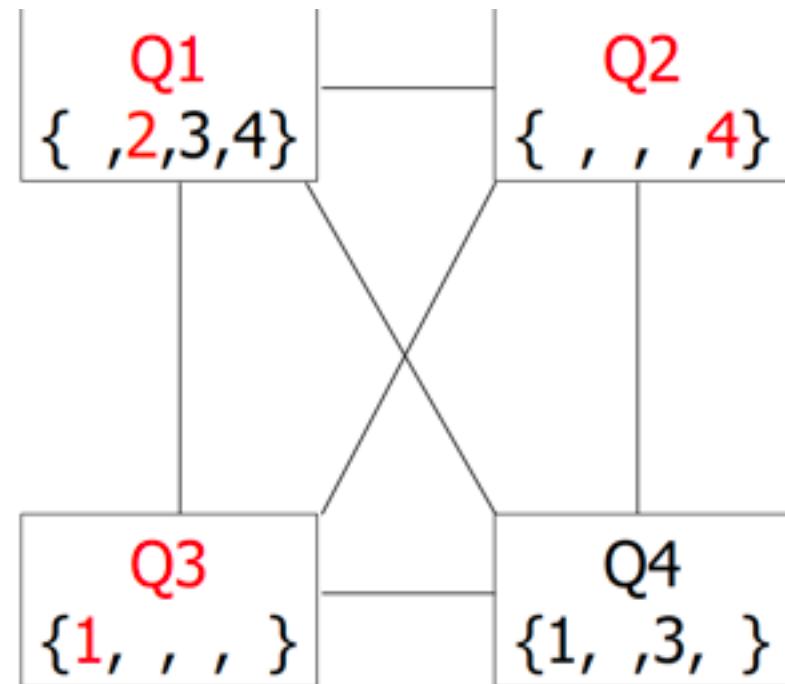
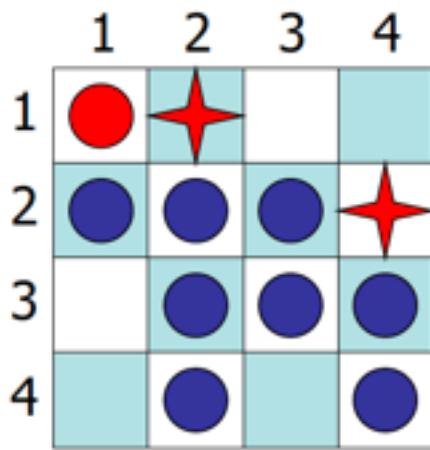
	1	2	3	4
1	●		★	
2	●	●	●	
3		●		●
4		●		



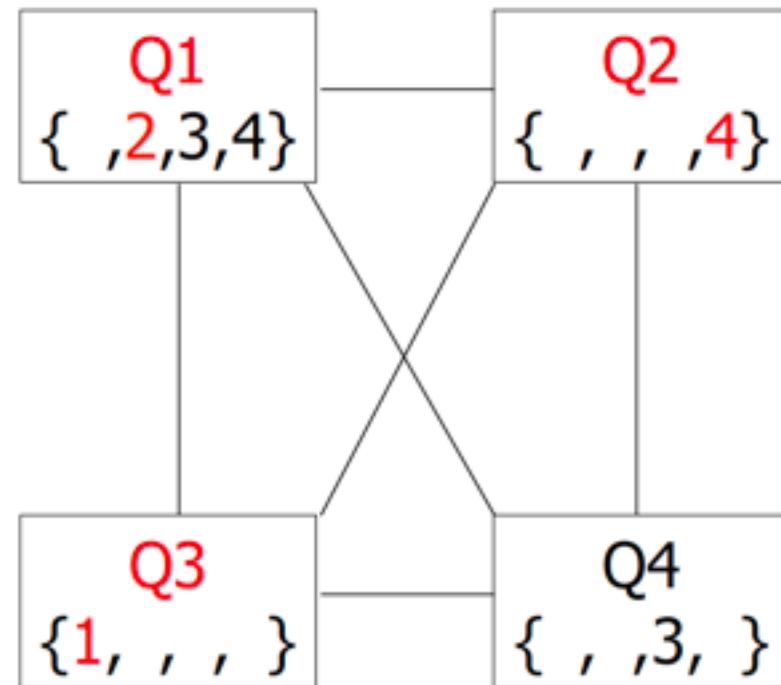
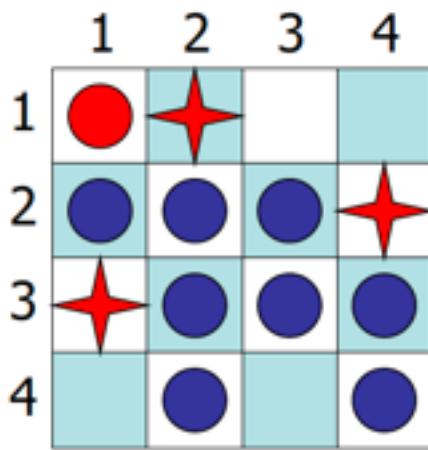
forward-checking



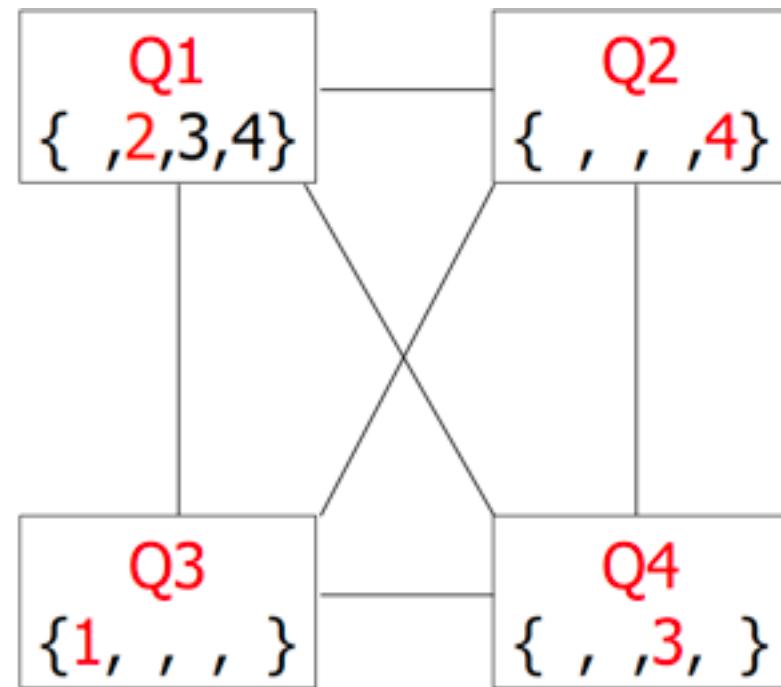
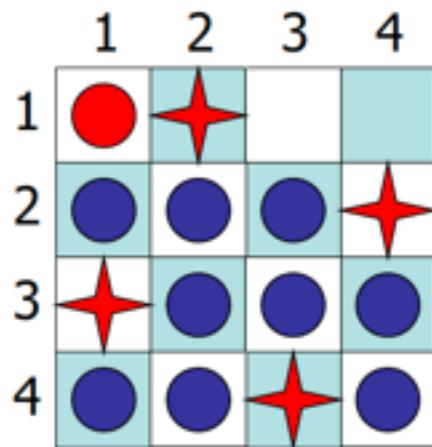
forward-checking



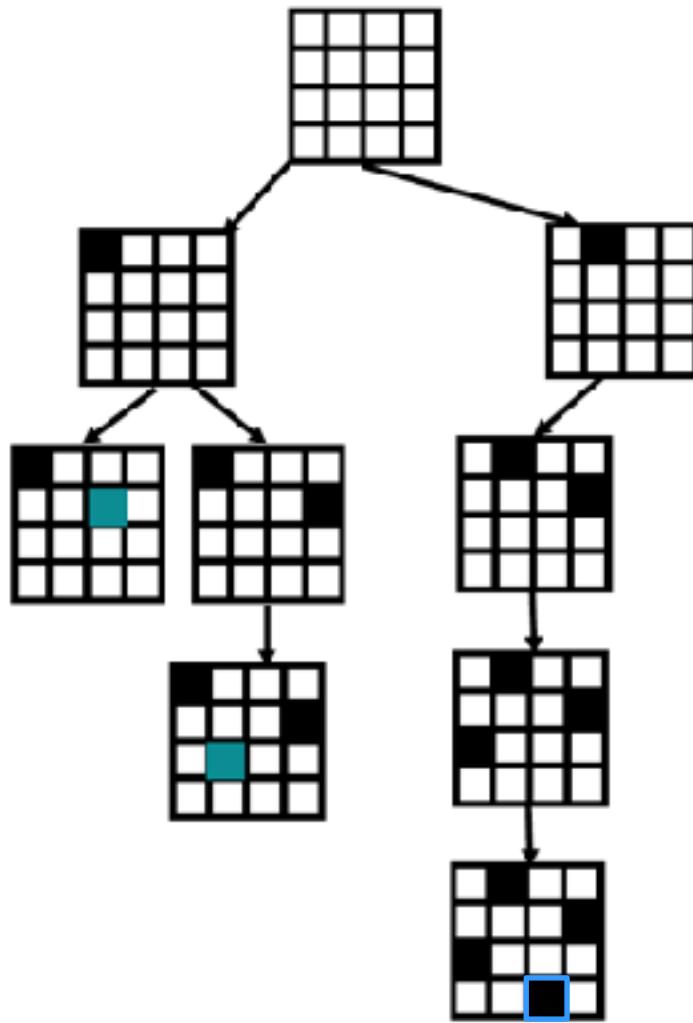
forward-checking



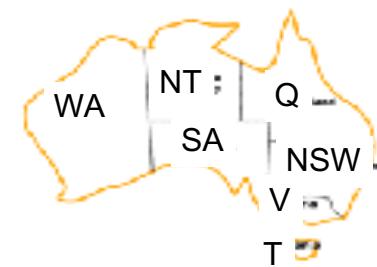
forward-checking



forward-checking



Forward-Checking: Exple2



WA

NT

Q

NSW

V

SA

T

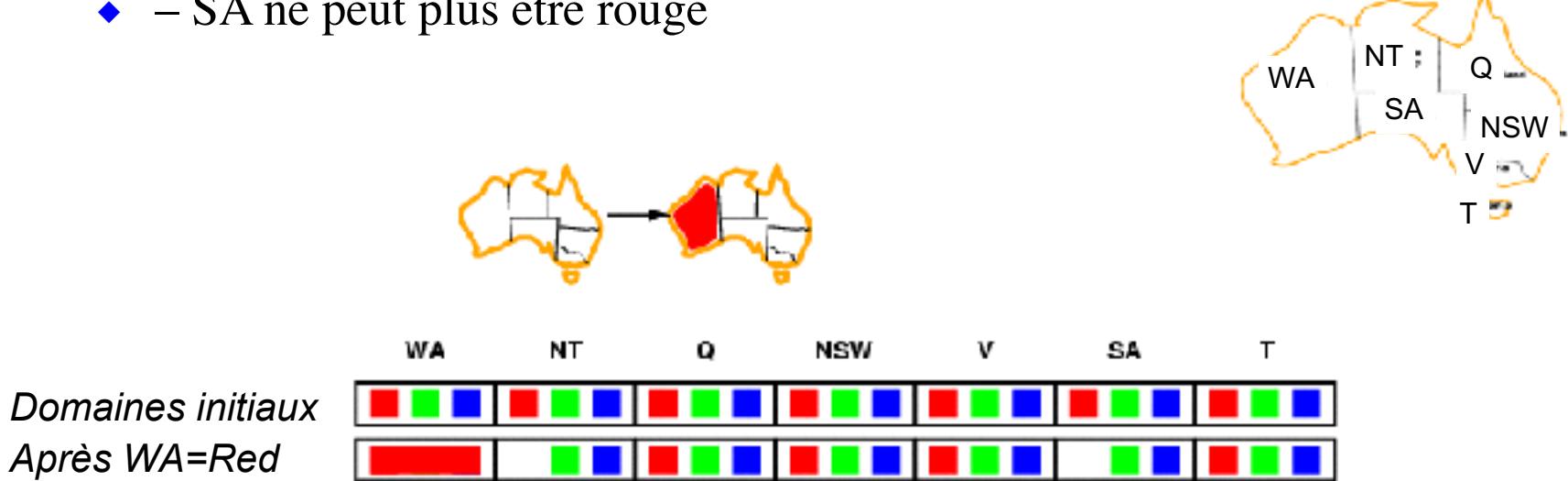
Domaines initiaux



- L'idée de *forward-checking* (*vérification anticipative*) est :
 - ◆ vérifier les valeurs compatibles des variables non encore assignées
 - ◆ terminer la récursivité (conflit) lorsqu'une variable (non encore assignée) a son ensemble de valeurs compatibles qui devient vide
- Attention: si la valeur est assignée à X, on vérifie juste la cohérence des variables Y telle que il y a une contrainte impliquant X et Y. Par contre, si ce faisant, Y est modifié, on ne vérifiera pas l'impact de cette modification sur les contraintes impliquant Y!

Forward checking

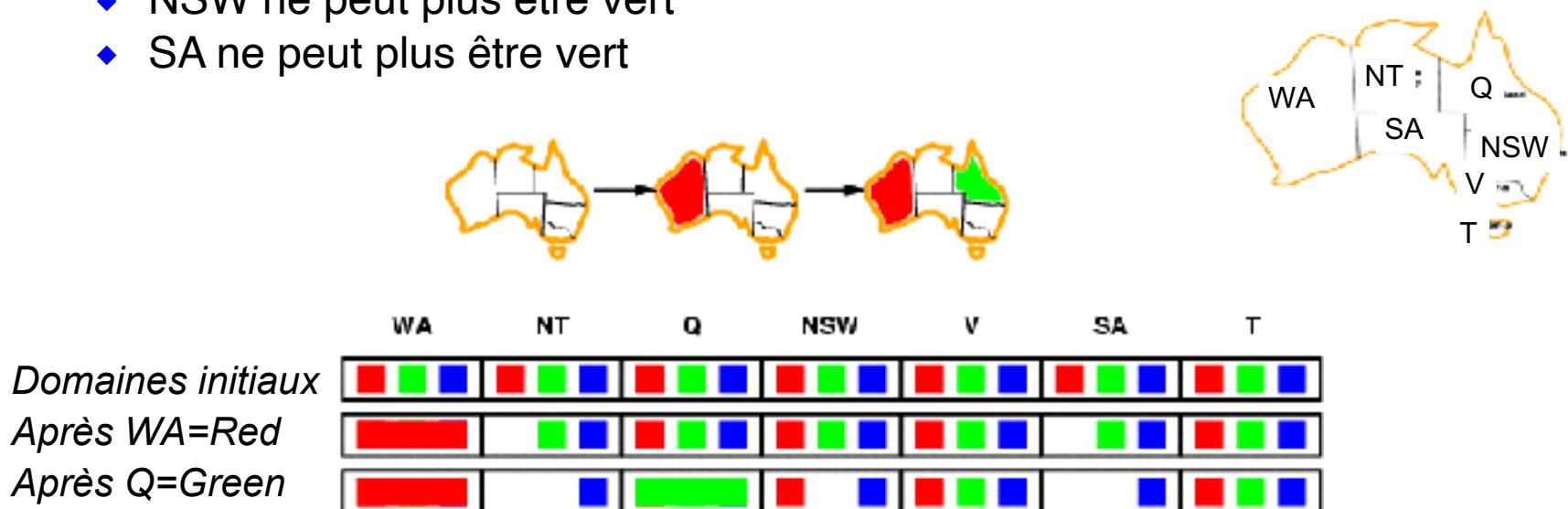
- Supposons que l'on choisisse au départ la variable WA (première étape de la récursivité de *backtracking-search*). Considérons l'assignation WA=Rouge. On voit ici le résultat de *forward-checking*.
 - NT ne peut plus être rouge
 - SA ne peut plus être rouge



Domaines initiaux
Après WA=Red

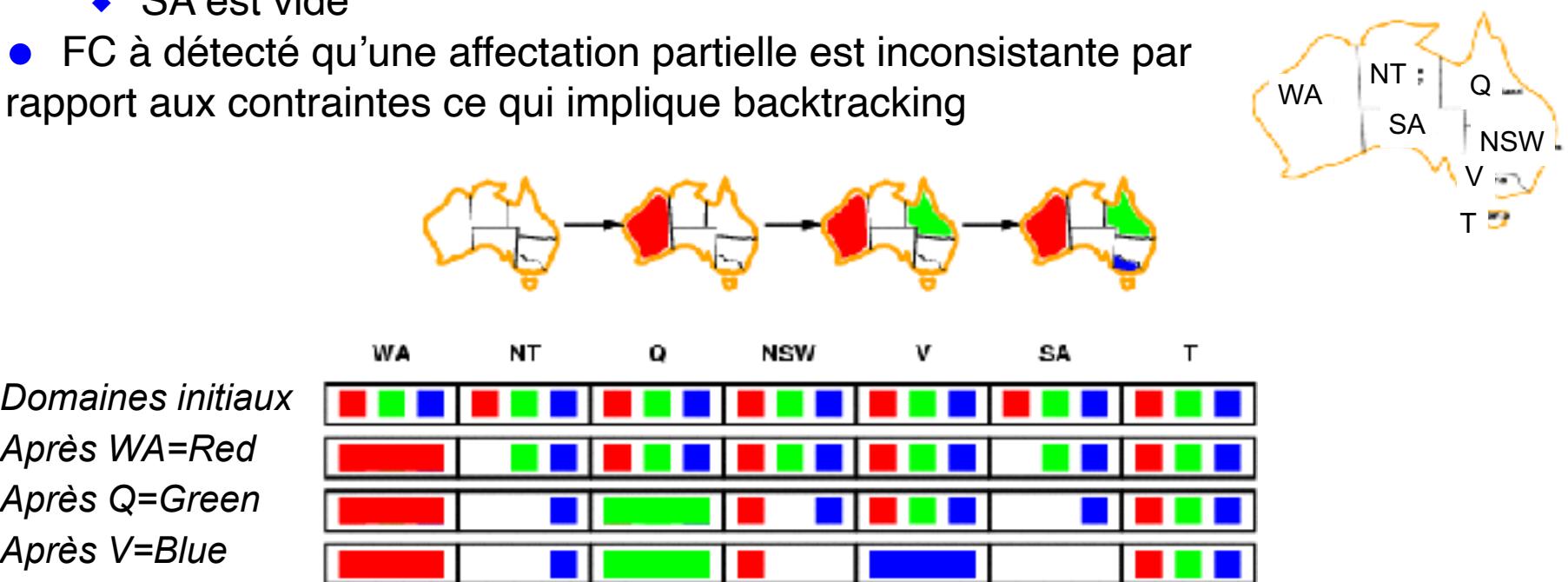
Forward checking

- Supposons maintenant que l'on choisisse la variable Q à la prochaine étape de la récursivité de *backtracking-search*. Considérons l'assignation Q=Vert. On voit ici le résultat de *forward-checking*.
 - NT ne peut plus être vert
 - NSW ne peut plus être vert
 - SA ne peut plus être vert



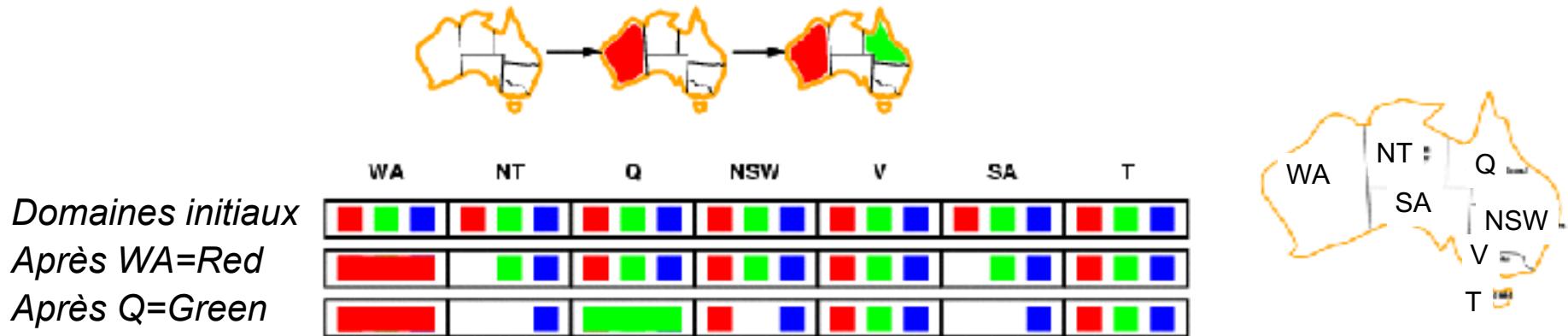
Forward checking

- Supposons maintenant que l'on choisisse la variable V à la prochaine étape de la récursivité de *backtracking-search*. Considérons l'assignation V=Bleu. On voit ici le résultat de *forward-checking*.
 - ◆ NSW ne peut plus être bleu
 - ◆ SA est vide
- FC a détecté qu'une affectation partielle est inconsistante par rapport aux contraintes ce qui implique backtracking



Propagation de contraintes

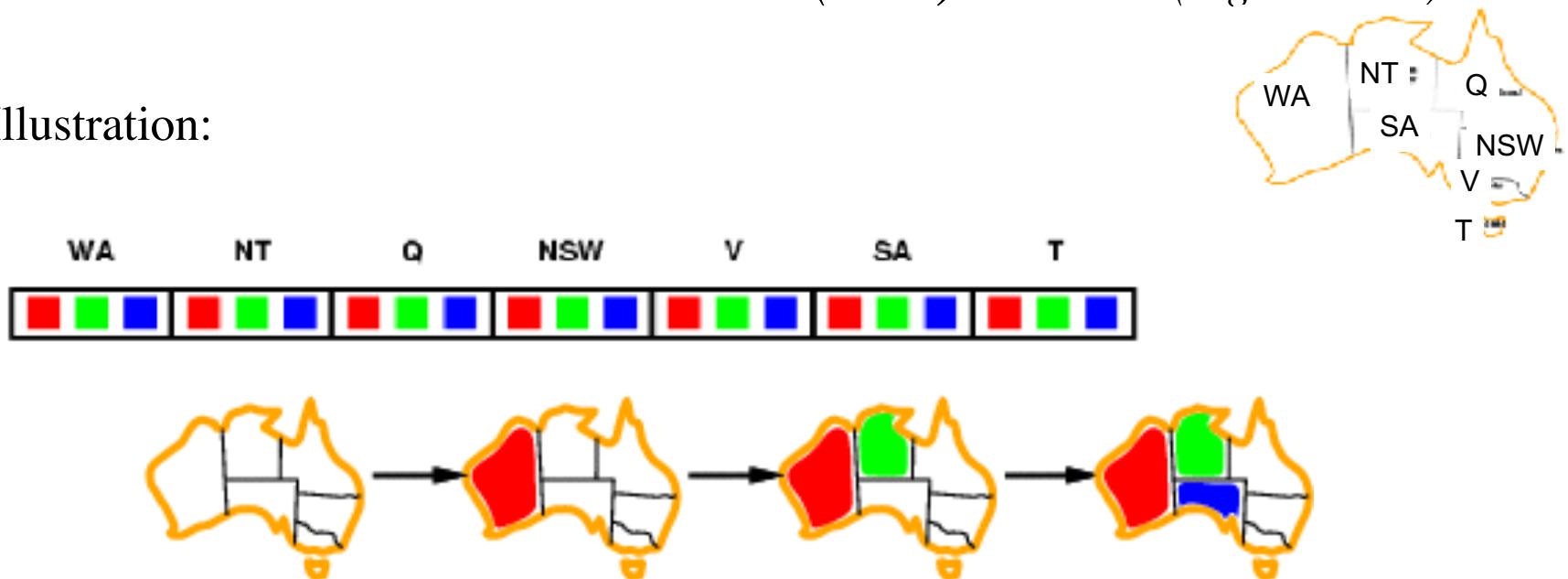
- *Forward checking* propage l'information d'une variables assignée vers les variables en contraintes avec elle, mais ne propage pas l'effet des modifications de ces dernières.



- Revenons à l'étape de backtracking-search, après que nous ayons choisi la variable Q et assigné la valeur 'Green'.
 - ◆ On voit ici le résultat de forward-checking
 - ◆ Forward-checking ne propage pas la modification du domaine SA vers NT pour constater que NT et SA ne peuvent pas être en bleu ensemble!
- La propagation des contraintes permet de vérifier ce type de conflits dans les assignations de variables.

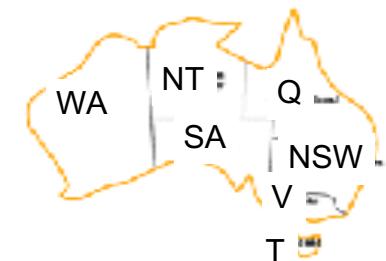
Choisir la prochaine variable

- À chaque étape, choisir la variable avec le moins de valeurs consistantes restantes.
 - ◆ C-à-d., la variable « posant le plus de restrictions ».
 - ◆ Appelé: *Minimum Remaining Value (MRV) Heuristic (smallest domain)*
 - ◆ ou *Most Constrained Variable (MCV) Heuristic. (highest arcs)*
- Illustration:



Choisir la prochaine valeur

- Pour une variable donnée, choisir une valeur qui invalide le moins de valeurs possibles pour les variables non encore assignées.



Laisse une seule valeur pour SA

Ne laisse aucune valeur pour SA

Propagation de contraintes

- FC est souvent 100 fois plus rapide que BS
- FC avec MRV (minimal remaining values) est souvent 10000 fois plus rapide.
- mais FC n'est pas aussi rapide. D'autres formes de propagation de contraintes sont utilisés en pratique.

Constraint Propagation: Generalized Arc Consistency

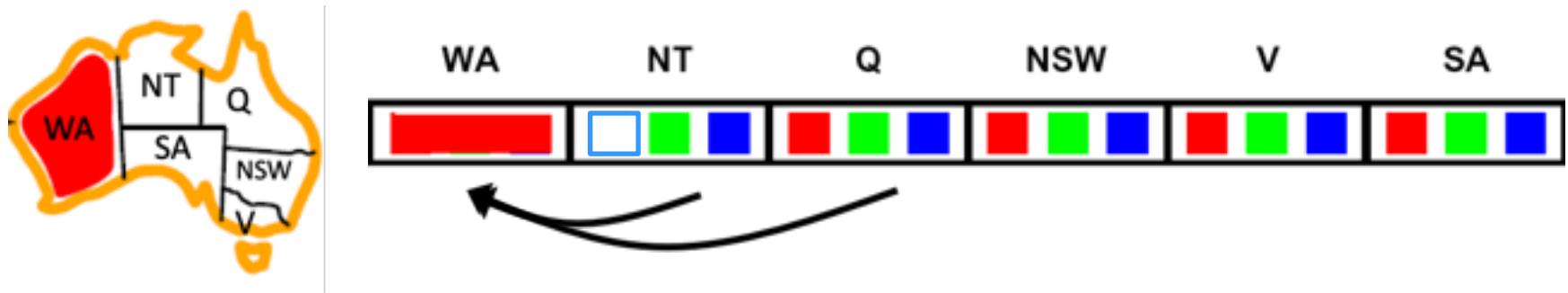
- Arc consistency est la forme de propagation de contraintes la plus utilisée
 - ◆ Vérifie la consistance entre les arcs.
 - ◆ C-à-d., la consistance des contraintes entre deux variables.
- L'arc $X \rightarrow Y$ est consistant si et seulement si
 - ◆ Pour chaque valeur x de X il existe au moins une valeur permise de y de Y et consistante avec x .
- Path-consistency (n -ary constraints): généralisation de Arc consistency de contraintes binaires au contraintes multiples.

Constraint Propagation: Generalized Arc Consistency

- Il est toujours possible de transformer une contrainte n-aire en des contraintes binaires. Les alg de résolution de CSP utilisent les contraintes binaires.
- Un CSP est arc-consistant si toutes ses contraintes sont arc-consistantes.
- Utiliser une queue de contraintes qui doivent être révisées quand le domaine d'une variable a changé seules les contraintes portant sur cette variable sont ajoutées à la queue.

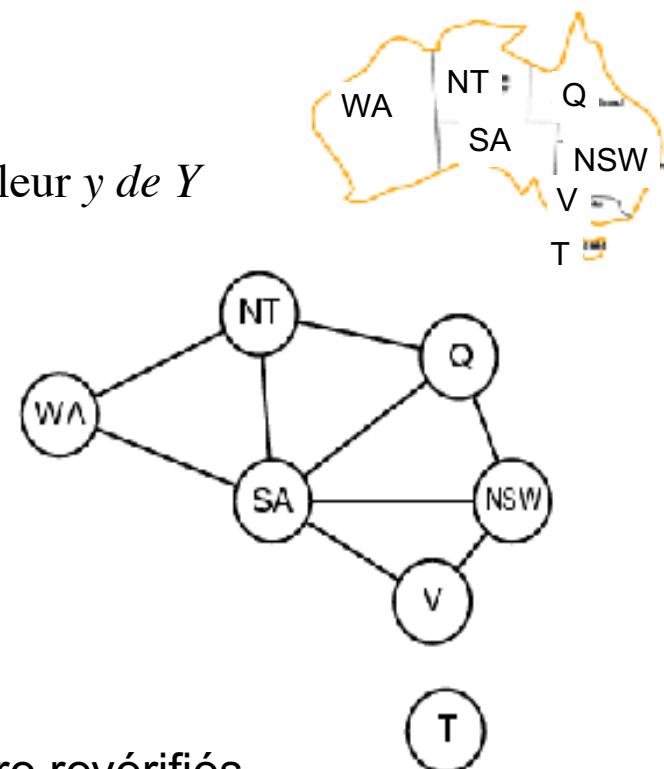
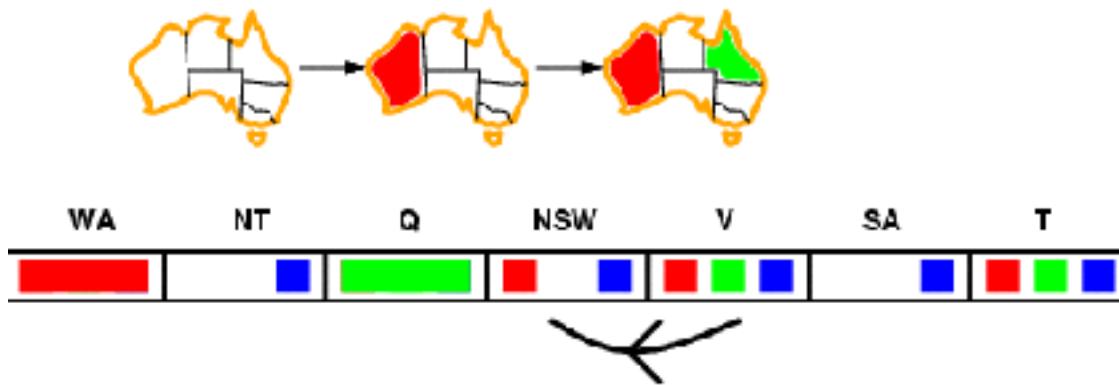
One arc consistency

- Un arc $X \rightarrow Y$ est **consistent**ssi pour chaque X (dans la queue de l'arc) il y au moins un Y (dans la tête de l'arc) qui peut être affecté à Y sans violer une contrainte



Generalized Arc consistency

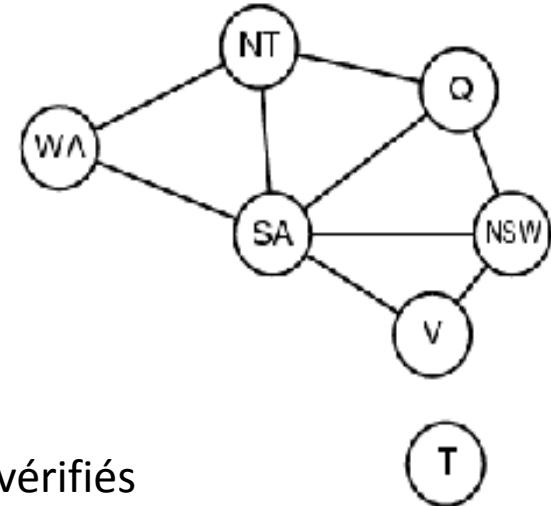
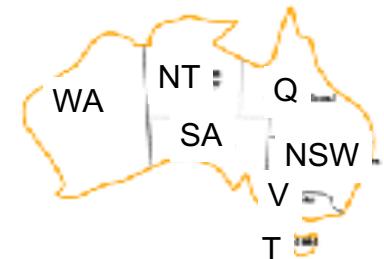
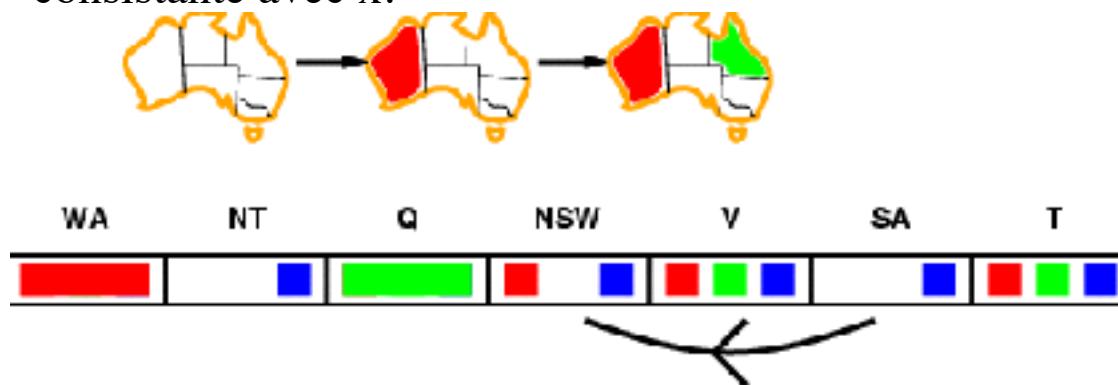
- *Arc consistency* est la forme de propagation de contraintes la plus simple
 - ◆ Vérifie la consistance entre les arcs.
 - ◆ C-à-d., la consistance des contraintes entre deux variables.
- L'arc $X \rightarrow Y$ est consistante si et seulement si
 - ◆ Pour chaque valeur x de X il existe au moins une valeur y de Y consistante avec x .



Si une variable perd une valeur, ses voisins doivent être revérifiés.

Generalized Arc consistency

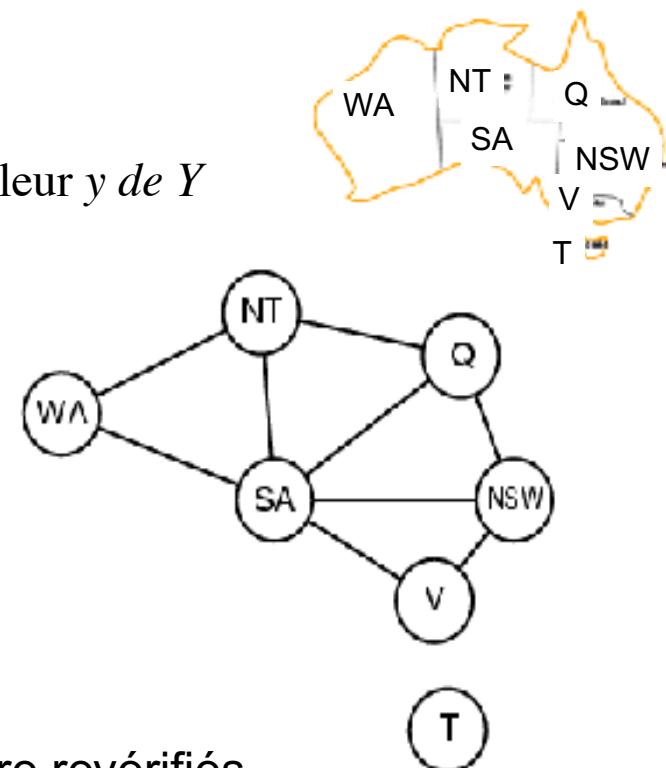
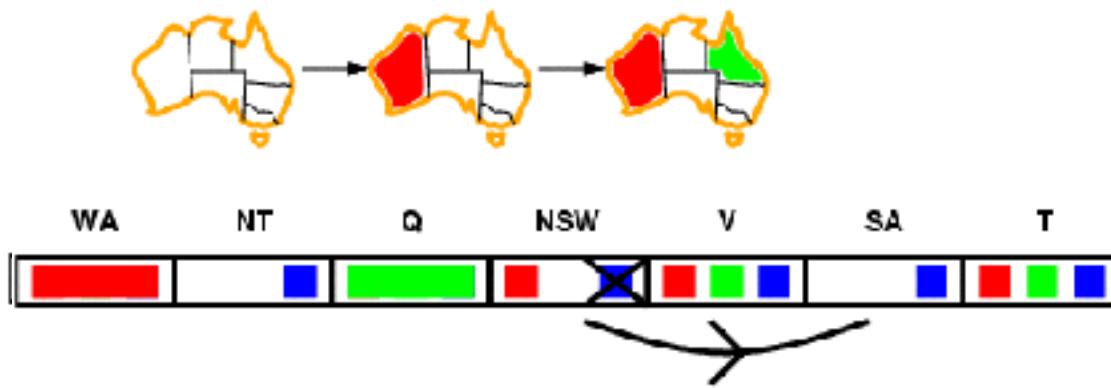
- *Arc consistency* est la forme de propagation de contraintes la plus simple
 - ◆ Vérifie la consistance entre les arcs.
 - ◆ C-à-d., la consistance des contraintes entre deux variables.
- L'arc $X \rightarrow Y$ est consistante si et seulement si
 - ◆ Pour chaque valeur x de X il existe au moins une valeur y de Y consistante avec x .



Si une variable perd une valeur, ses voisins doivent être revérifiés

Generalized Arc consistency

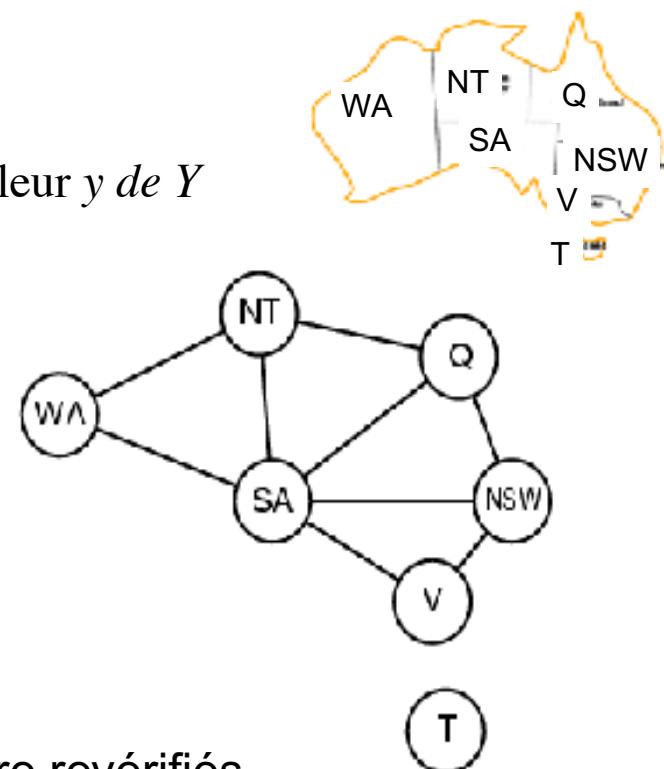
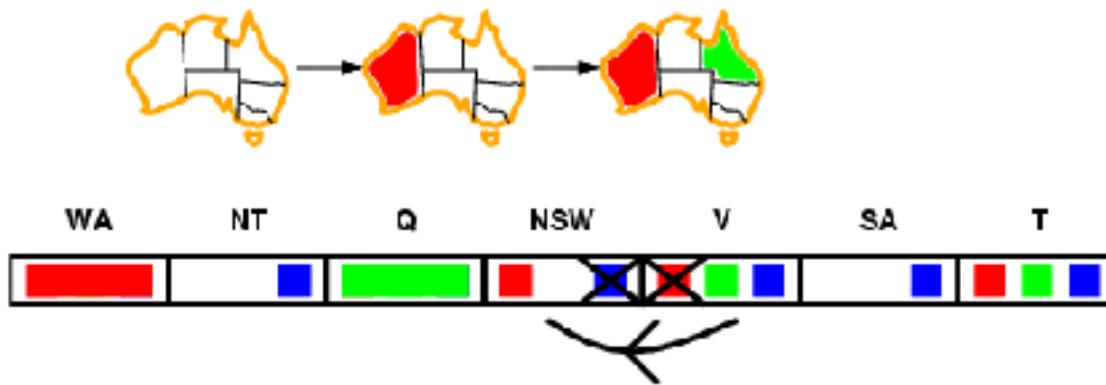
- *Arc consistency* est la forme de propagation de contraintes la plus simple
 - ◆ Vérifie la consistance entre les arcs.
 - ◆ C-à-d., la consistance des contraintes entre deux variables.
- L'arc $X \rightarrow Y$ est consistante si et seulement si
 - ◆ Pour chaque valeur x de X il existe au moins une valeur y de Y consistante avec x .



Si une variable perd une valeur, ses voisins doivent être revérifiés.

Generalized Arc consistency

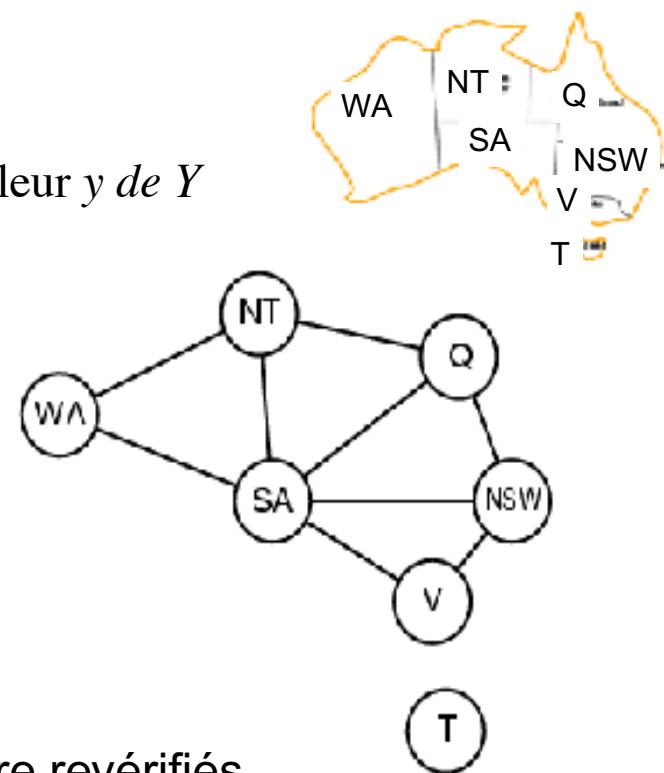
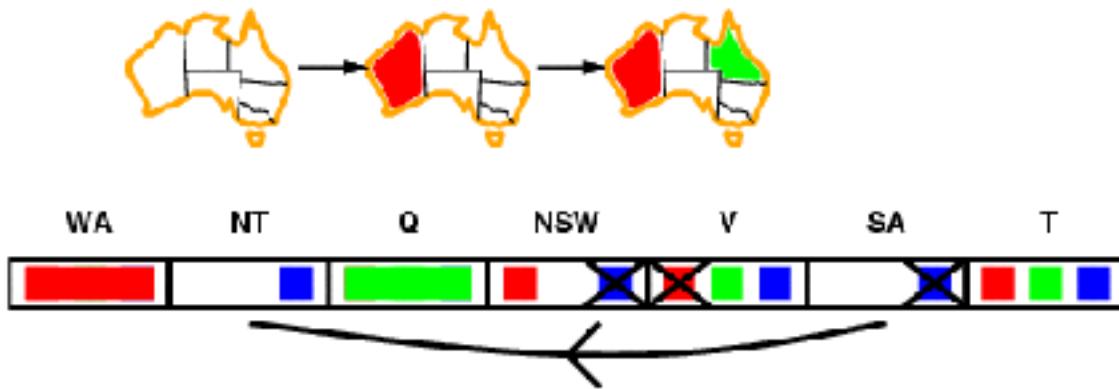
- *Arc consistency* est la forme de propagation de contraintes la plus simple
 - ◆ Vérifie la consistance entre les arcs.
 - ◆ C-à-d., la consistance des contraintes entre deux variables.
- L'arc $X \rightarrow Y$ est consistante si et seulement si
 - ◆ Pour chaque valeur x de X il existe au moins une valeur y de Y consistante avec x .



Si une variable perd une valeur, ses voisins doivent être revérifiés.

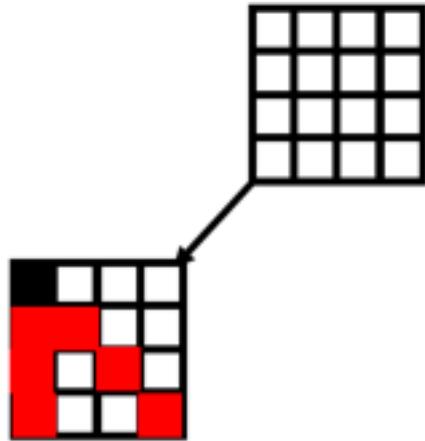
Generalized Arc consistency

- *Arc consistency* est la forme de propagation de contraintes la plus simple
 - ◆ Vérifie la consistance entre les arcs.
 - ◆ C-à-d., la consistance des contraintes entre deux variables.
- L'arc $X \rightarrow Y$ est consistante si et seulement si
 - ◆ Pour chaque valeur x de X il existe au moins une valeur y de Y consistante avec x .



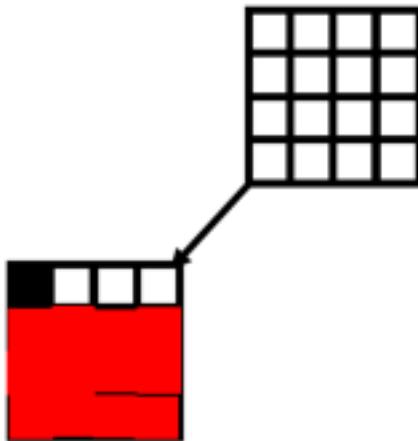
Si une variable perd une valeur, ses voisins doivent être revérifiés.

Generalized Arc Consistency



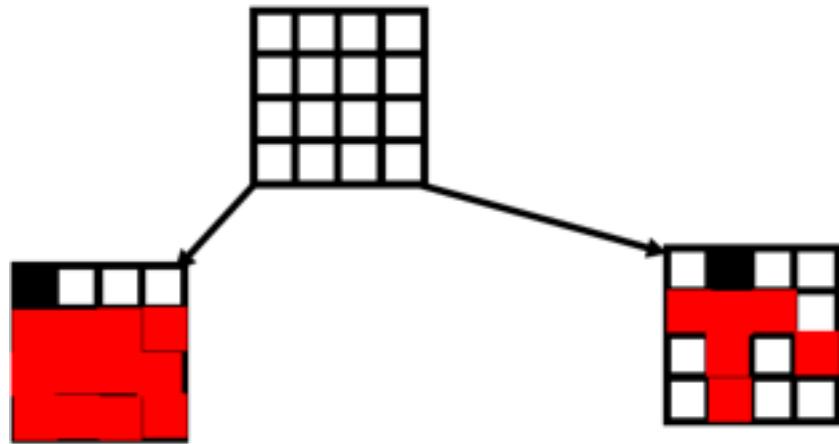
- Quand on affecte $V_1 = 1$ on réduit $\text{Dom}(V_1) = \{1\}$. la valeur de V_1 est **fixée**.
- élaguer $V_2=1, V_2=2, V_3=1, V_3=3, V_4=1, V_4=4$ —ces valeurs sont inconsistantes avec $\text{Dom}(V_1) = \{1\}$, $\text{Dom}(V_2) = \{3,4\}$, $\text{Dom}(V_3) = \{2,4\}$, $\text{Dom}(V_4) = \{2,3\}$

Generalized Arc Consistency



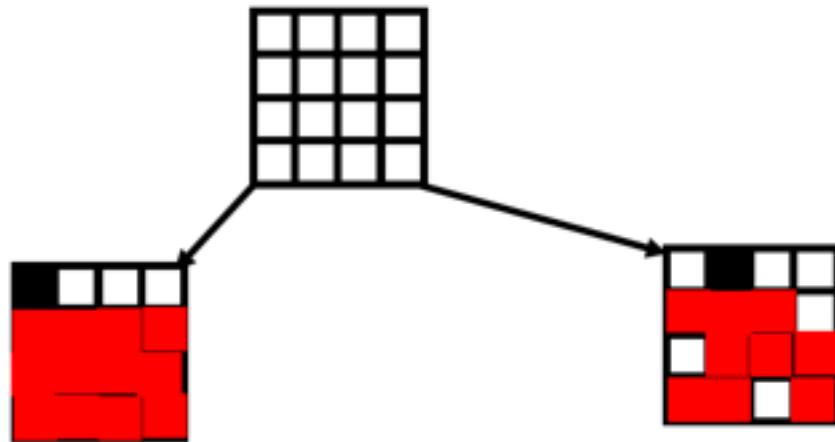
- $\text{Dom(V2)} = \{3,4\}$, $\text{Dom(V3)} = \{2, 4\}$, $\text{Dom(V4)} = \{2,3\}$;
 - ◆ $V2 = 3$ n'a pas de valeur compatible avec les valeurs de $V3$
 - ◆ $V3 = 4$ n'a pas de valeur compatible avec les valeurs de $V2$
 - ◆ $\text{Dom(V2)} = \{4\}$, $\text{Dom(V3)} = \{2\}$
- $\text{Dom(V3)} = \{2\}$, $\text{Dom(V4)} = \{2, 3\}$
 - ◆ $V3=2$ n'a pas de valeur compatible avec les valeurs de $V4$
 - ◆ $\text{Dom(V3)} = \{\}$

Generalized Arc Consistency



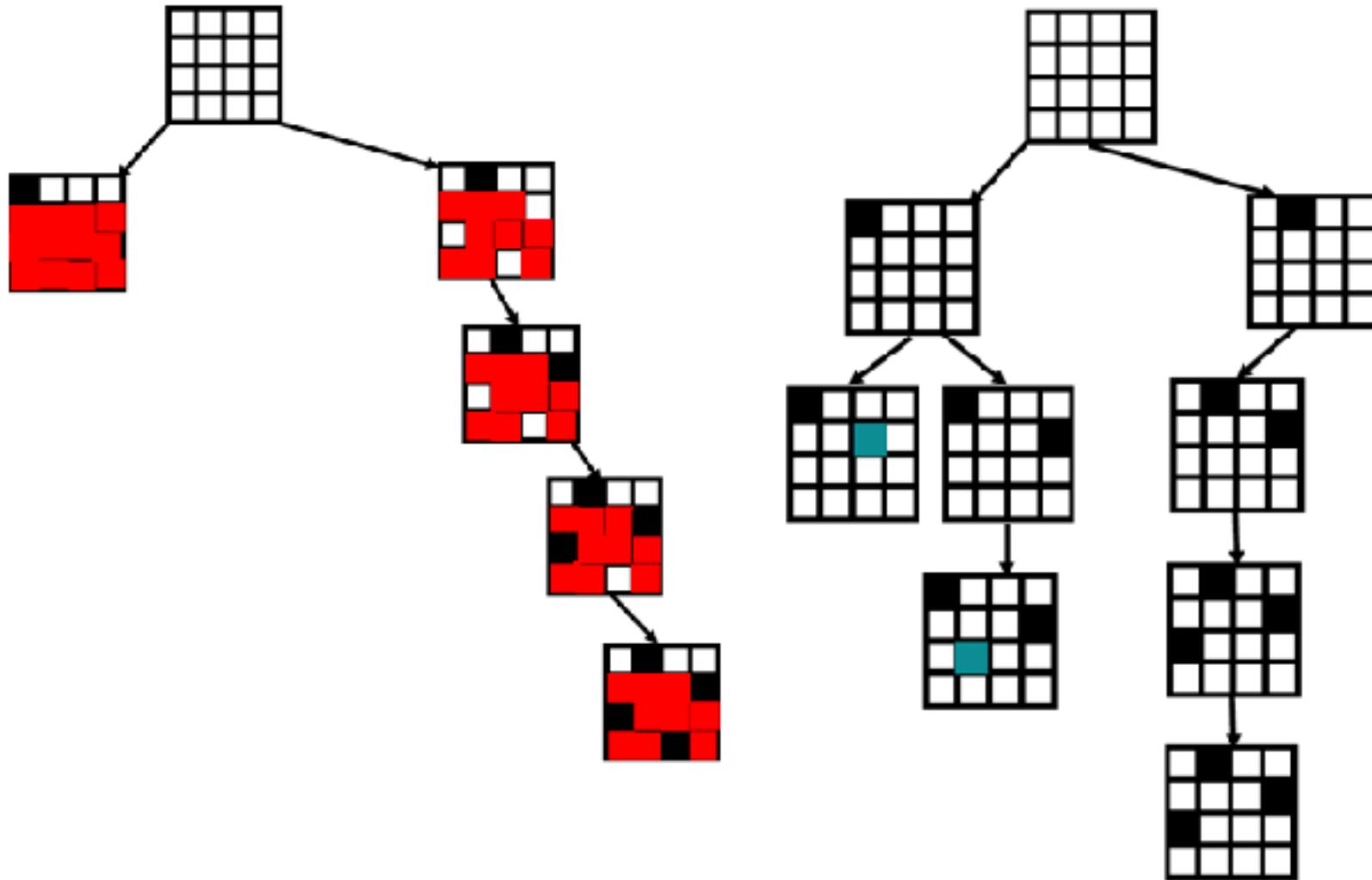
- Quand on affecte $V_1 = 2$ on réduit $\text{Dom}(V_1) = \{2\}$. la valeur de V_2 est **fixée**.
- élaguer $V_2=1, V_2=2, V_2=3, V_3=2, V_3=4, V_4=2$ —ces valeurs sont inconsistantes avec $\text{Dom}(V_1) = \{2\}$, $\text{Dom}(V_2) = \{4\}$, $\text{Dom}(V_3) = \{1,3\}$, $\text{Dom}(V_4) = \{1,3,4\}$

Generalized Arc Consistency



- $V_3=3$ n'a pas de valeur compatible avec les valeurs de V_2
 - ◆ $\text{Dom}(V_3) = \{1\}$
- $V_4=1$ n'a pas de valeur compatible avec les valeurs de V_3 .
- $V_4=4$ n'a pas de valeur compatible avec les valeurs de V_2 .
 - ◆ $\text{Dom}(V_4) = 3$

Generalized Arc Consistency



ARC consistency

```
function AC-3(csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables  $\{X_1, X_2, \dots, X_n\}$ 
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
     $(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\text{queue})$ 
    if REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) then
      for each  $X_k$  in NEIGHBORS[ $X_i$ ] do
        add  $(X_k, X_i)$  to queue
```

```
function REMOVE-INCONSISTENT-VALUES( $X_i, X_j$ ) returns true iff succeeds
  removed  $\leftarrow \text{false}$ 
  for each  $x$  in DOMAIN[ $X_i$ ] do
    H if no value  $y$  in DOMAIN[ $X_j$ ] allows  $(x, y)$  to satisfy the constraint  $X_i \leftrightarrow X_j$ 
      then delete  $x$  from DOMAIN[ $X_i$ ]; removed  $\leftarrow \text{true}$ 
  return removed
```

T

complexité de AC3

- Let n be the number of variables, and d be the domain size.
- If every node (variable) is connected to the rest of the variables, then we have $n(n - 1)$ arcs (constraints) ! $O(n^2)$
- Each arc can be inserted in the queue d times ! $O(d)$
- Checking the consistency of an arc costs ! $O(d^2)$.
- Overall complexity is $O(n^2d^3)$.

CSP

- problèmes d'affectations
 - ◆ qui donne un cours à une classe
- problems d'ordonancement
 - ◆ planning des examens
 - ◆ scheduling des transport
 - ◆ scheduling de tâches
- Configuration matérielles
 - ◆ ensemble de composants compatibles

CSP

- CSPs:

- Variables

- Domains

- Constraints

- Implicit (provide code to compute)
 - Explicit (provide a list of the legal tuples)
 - Unary / Binary / N-ary

- Goals:

- Here: find any solution

- Also: find all, find best, etc.

-

CSP

- General-purpose ideas give huge gains in speed
 - ... but it's all still NP-hard
- Filtering: Can we detect inevitable failure early?
- Ordering:
 - Which variable should be assigned next? (MRV)
 - In what order should its values be tried? (LCV)
- Structure: Can we exploit the problem structure?
 -

Representation de la connaissance

Representation de la connaissance

- Certains problèmes comme les jeux s’apprêtent à un modèle basé sur l’exploration d’un espace d’états.
- Pour des problèmes plus complexes (par exemple, la planification de tâches pour un robot, le diagnostic), il faut en plus des outils permettant de modéliser le raisonnement, c’est-à-dire, la capacité de déduire des connaissances à partir d’informations brutes.
- Le calcul des prédictats est la base de beaucoup de formalismes de représentation et de manipulation de la connaissance en intelligence artificielle.

Representation de la connaissance

- A l'origine, le calcul des prédictats est un modèle mathématique du raisonnement par déduction.
- Généralement, pour construire un modèle d'un objet réel, on détermine les caractéristiques principales de l'objet et on crée un modèle ayant ces caractéristiques.
- Ici, on veut modéliser le raisonnement, c'est-à-dire le processus permettant d'inférer des expressions “logiquement correctes” à partir des faits.
- La capacité de modéliser le raisonnement, si approximatif soit-il, nous permettra de le programmer dans des systèmes informatisés.

Representation de la connaissance

- En premier lieu, nous allons considérer la syntaxe, c-à-d., la forme des expressions qu'on peut écrire dans le calcul des prédictats.
- Ensuite, nous considérons la sémantique, c-à-d., comment déterminer si une expression est logiquement correcte.
- Plus tard, nous verrons une méthode de déduction, c-à-d, une méthode pour déterminer des expressions qui sont des conséquences logiques des autres.

Le Calcul Propositionnel ...

But du calcul propositionnel : donner un fondement formel à un ensemble restreint d'énoncés du langage en utilisant :

- des propositions élémentaires (« Il fait beau », ...).
- des connecteurs (et, ou, donc, équivalent à) et
- la négation

Exemple : phrase $p = \text{« Il fait beau et je suis en vacances »}$

p est vraie si les deux propositions q et r sont vraies :

$$q = \text{« Il fait beau »}$$

$$r = \text{« Je suis en vacances »}$$

Symboles du Calcul Propositionnel ...

- Symboles (ou atomes) propositionnels : P, Q, R, S, ... (dernières lettres majuscules de l'alphabet)

Dénotent des propositions ou des phrases, pouvant être vraies ou fausses, à propos d'un environnement (« la voiture est rouge », « l'eau est froide », ...).

- Symboles de vérité : vrai, faux
 - Connecteurs propositionnels : équivalence \equiv (ou \leftrightarrow)
implication \rightarrow
conjonction \wedge
disjonction \vee
négation \neg

Formules du Calcul Propositionnel ...

- Les formules (ou formules bien formées FBF) du calcul propositionnel, notées par les lettres majuscules de l'alphabet (A, B, C ...), sont définies par :
 - 1) Les atomes (vrai, P, Q, R, ...) sont des formules.
 - 2) Si A et B sont deux formules, alors $(A \rightarrow B)$, $(A \equiv B)$, $(A \wedge B)$ et $(A \vee B)$ sont des formules.
 - 3) Si A est une formule, alors $(\neg A)$ est une formule.
- Symboles () et [] : utilisés pour grouper des formules.

$$(P \vee Q) \equiv R \quad \neq \quad P \vee (Q \equiv R)$$

Formules du Calcul Propositionnel (suite) ...

- Une expression est une FBF si et seulement si elle a été formulée à partir de symboles légaux et des règles utilisant les connecteurs.
- **Exemple :** $((P \wedge Q) \rightarrow R) \equiv \neg P \vee \neg Q \vee R$ est une FBF puisque :
 P, Q, R sont des FBFs,
 $P \wedge Q$ est une FBF (conjonction de deux FBFs),
 $(P \wedge Q) \rightarrow R$ est une FBF (implication entre deux FBFs),
 $\neg P$ et $\neg Q$ sont deux FBFs (négation de FBFs),
 $\neg P \vee \neg Q$ est une FBF (disjonction de deux FBFs),
 $\neg P \vee \neg Q \vee R$ est une FBF (disjonction de 3 FBFs) et
 $((P \wedge Q) \rightarrow R) \equiv \neg P \vee \neg Q \vee R$ est une FBF (équivalence entre deux FBFs).

Règles d'inférence

- Une **règle d'inférence** est une manière de générer des FBFs à partir d'autres FBFs
 - La FBF B peut être inférer à partir des FBFs A et $A \rightarrow B$ (*modus ponens*)
 - La FBF $A \wedge B$ peut être inférer de A et B (introduction de \wedge)
 - La FBF $B \wedge A$ peut être inférer de $A \wedge B$ (commutativité)
- La FBF A peut être inférer de $\neg(\neg A)$ (élimination de \neg)

Définition de preuve

- La séquence de FBFs $\{A_1, A_2, \dots, A_n\}$ est appelée une **preuve** de A_n à partir d'un ensemble de FBFs W ssi chaque A_i de la séquence appartient à W ou bien il peut être inférer d'une FBF antérieure en utilisant les règles d'inférence.
- S'il existe une preuve de A_n à partir de W , nous disons que A_n est un **théorème** de l'ensemble W .

$$W \vdash_R A_n$$

On peut inférer A_n à partir de W en utilisant les règles d'inférence R

Sémantique du Calcul Propositionnel ...

- **But :** établir un mécanisme sémantique d'évaluation des FBFs.

- Interprétation

- donner un sens à nos symboles
- donner la valeur de vérité des atomes

Exemple

Associer l'atome Bat_OK a la proposition P « la batterie est chargée »

La valeur de vérité (vraie, faux) de Bat_OK dépend de la valeur de la proposition P. (qui a son tour dépend, par exemple, d'un senseur X qui mesure le voltage de la batterie)

Sémantique du Calcul Propositionnel ...

Symbol → Une valeur de vérité : vrai (V) ou faux (F)

Connecteur → Une table de vérité

- donner la valeur de vérité des connecteurs qui relient ces atomes

P	Q	P	P ∨ Q	P ∧ Q	P → Q	P ≡ Q
V	V	F	V	V	V	V
V	F	F	V	F	F	F
F	V	V	V	F	V	F
F	F	V	F	F	V	V

Sémantique du Calcul Propositionnel ...

- Une interprétation **satisfait** une FBF si la FBF à la valeur **T** pour cette interprétation.
- Une interprétation qui satisfait un ensemble de FBFs est appelée un **modèle** de cet ensemble.
- une FBF est **valide** si elle est *vraie* pour *toute* les interprétations des atomes qui la constituent

$$P \rightarrow P,$$

$$P \vee T,$$

$$\top (P \wedge \top P)$$

Sémantique du Calcul Propositionnel ...

- **FBFs équivalentes** : donnent la même valeur de vérité dans tous les cas possibles.

Exemple :

P → Q est équivalent à] P ∨ Q

P	Q]P]P v Q	P → Q
V	V	F	V	V
V	F	F	F	F
F	V	V	V	V
F	F	V	V	V

- ## ● Équivalences possibles :

$$J(J \cap P) = P$$

$$(P \vee Q) \equiv (\neg P \rightarrow Q)$$

$$(P \rightarrow Q) \equiv (\neg Q \rightarrow \neg P)$$

$\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$ et $\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$: Règle de Morgan

$(P \vee Q) \equiv (Q \vee P)$ et $(P \wedge Q) \equiv (Q \wedge P)$: Commutativité

$$((P \wedge Q) \wedge R) \equiv (P \wedge (Q \wedge R))$$

$$((P \vee Q) \vee R) \equiv (P \vee (Q \vee R))$$

$$(P \vee (Q \wedge R)) \equiv (P \vee Q) \wedge (P \vee R)$$

$$(P \wedge (Q \vee R)) \equiv (P \wedge Q) \vee (P \wedge R)$$

Sémantique du Calcul Propositionnel ...

Conséquence logique (entailment)

- si une FBF A a une valeur vraie pour toutes les interprétations pour lesquelles les FBFs de l'ensemble W sont vraies, alors nous disons que A est une **conséquence logique** de W.

$$W \vDash A$$

- $\{P, P \rightarrow Q\} \vDash Q$
- $F \vDash A$ (A n'importe quelle FBF)
- $P \wedge Q \vDash P$

Si certaines propositions sont vraies alors d'autres sont nécessairement vraies.

Sémantique du Calcul Propositionnel ...

Cohérence (soundness)

- Pour tout ensemble de FBFs W , et pour toute FBF A

$$W \vdash_R A \rightarrow W \vDash A$$

Complétude

- Pour tout ensemble de FBFs W , et pour toute FBF A

$$W \vDash A \rightarrow W \vdash_R A$$

Résolution

- La résolution est une règle d'inférence qui combine plusieurs autres règles d'inférence (incluant modus ponens)
- La règle de résolution pour le calcul propositionnel s'écrit:

De $A \vee B$ et $\neg A \vee C$ nous pouvons inférer $B \vee C$,

Où A, B et C sont FBFs

La résolution est une règle d'inférence sound

Résolution

Définitions

- Un **littéral** est un atome ou la négation d'un atome $P \mid \neg P$
- Une **clause** est une disjonction de littéraux, $P \vee \neg Q \vee \dots \vee R$

Généralisation de la résolution

De $P_1 \vee P_2 \vee \dots \vee P_n$ et $\neg P_1 \vee Q_1 \vee \dots \vee Q_k$
nous pouvons inférer $P_2 \vee \dots \vee P_n \vee Q_1 \vee \dots \vee Q_k$

Résolution

- **Exemples**

1. Résolution de $R \vee P$ et $\neg P \vee Q$ donne $R \vee Q$

- $R \vee P$ est équivalent à $\neg R \rightarrow P$
- $\neg P \vee Q$ est équivalent à $P \rightarrow Q$
- $R \vee Q$ est équivalent à $\neg R \rightarrow Q$

Ce qui est équivalent à la règle d'inférence par chaînage

2. Résolution de R et $\neg R \vee P$ donne P

Ce qui est équivalent à la règle d'inférence du modus ponens

Résolution

Conversion d'une FBF à une conjonction de clauses (FNC)

$$\lceil (P \rightarrow Q) \vee (R \rightarrow P)$$

1. Éliminer le signe d'implication

$$\lceil (\lceil P \vee Q) \vee (\lceil R \vee P)$$

2. Réduire la portée de la négation en utilisant la loi de DeMorgan et éliminer les doubles négations

$$(P \wedge \lceil Q) \vee (\lceil R \vee P)$$

3. Convertir à un ensemble de conjonction en utilisant les lois d'associativité et de distributivité

$$(P \vee \lceil R) \wedge (\lceil Q \vee \lceil R \vee P)$$

Ce qui donne l'ensemble des clauses (reliées par conjonction)
 $\{(P \vee \lceil R), (\lceil Q \vee \lceil R \vee P)\}$

Résolution

La résolution n'est pas une règle d'inférence complète

On ne peut pas trouver une inférence pour $P \wedge R \Rightarrow P \vee R$ en utilisant la résolution pour l'ensemble des clauses $\{P, R\}$

→ Réfutation

Utiliser la résolution pour montrer que la négation de $P \vee R$ est inconsistante avec l'ensemble des clauses $\{P, R\}$

1. $(P \vee R)$ donne $\neg P \wedge \neg R$ dans l'ensemble des clauses nous auront $\{P, R, \neg P, \neg R\}$

La résolution utilisant la réfutation est une procédure d'inférence complète (pour toute formule valide on peut trouver une inférence)

Résolution

1. Convertir les FBFs en une forme clausale (conjonction de clauses)
- 2 . Convertir la négation de la FBF, A, à prouver en une forme clausale
3. Combiner les clauses résultant des 2 étapes précédentes dans un ensemble W
4. Appliquer itérativement la résolution pour les clauses dans W
Jusqu'à il n'y pas de résultat à ajouter ou la clause vide est produite

Résolution

Exemple

Calcul de Prédicats (ou Logique de 1^{er} Ordre) ...

- **Calcul propositionnel** : chaque atome (P, Q, R, \dots) dénote une proposition d'une certaine complexité.

On ne peut pas accéder aux composantes de telles assertions.

- **Calcul de prédictats** : permet de généraliser les assertions moyennant les variables.

Exemple :

Calcul propositionnel : $p = \text{« It rained on yesterday »}$

Calcul de prédictats : $\text{weather}(\text{tuesday}, \text{rain})$

prédicat décrivant la relation entre une date et le temps.

Généralisation → pour toutes les valeurs de X , où X est un jour,
le prédicat $\text{weather}(X, \text{rain})$ est vrai.

- Le calcul de prédictats permet de s'intéresser à des énoncés relatifs à un ensemble d'objets ou certains éléments de cet ensemble.

Syntaxe du Calcul de Prédicats ...

- **Alphabets** utilisés pour construire les symboles :

a, b, ..., z, A, B, ..., Z (lettres majuscules et minuscules de l'alphabet)

0, 1, ..., 9 (les chiffres)

_ (caractère souligné)

Symboles légaux

George

fire3

tom_and_jerry

bill

XXXXX

friends_of

Symboles non légaux

3jack

no blanks allowed

ab%cd

***71

duck??

$$l(g, k) \xrightarrow{\text{Améliorer la lisibilité}} \text{likes}(\text{george}, \text{kate})$$

Syntaxe du Calcul de Prédicats (suite) ...

- **Constantes** : symboles commençant par une lettre minuscule.
george, tree, geORGE, ...
- **Variables** : symboles commençant par une lettre majuscule.
George, Bill, Kate, ...
- **Fonctions** : symboles commençant par une lettre minuscule.
Chaque fonction est d'arité positive ou nulle (nombre de paramètres).

plus(5, 2) : arité 2
father(george) : arité 1
price(bananas)
f(X, Y)

- **Connecteurs** : ceux du calcul propositionnel
- **Quantificateurs** : \forall (universel) et \exists (existential)

$\exists Y \text{ friends}(Y, \text{peter})$
 $\forall X \text{ likes}(X, \text{ice_cream})$

Syntaxe du Calcul de Prédicats (Suite) ...

● **Terme** : défini récursivement par :

- Une variable est un terme.
- Un symbole de constante est un terme.
- Si f est un symbole de fonction d'arité n et si t_1, t_2, \dots, t_n sont des termes, alors $f(t_1, t_2, \dots, t_n)$ est un terme.

Exemples : Cat X mother(jane)
 times(2, 3) Blue Kate

● **Prédicats** : symboles commençant par une lettre minuscule.

Chaque prédicat décrit une relation entre zéro ou plusieurs objets.

Exemples : likes Near on
 equals part_of

Description d'un ensemble de relation de parenté :

mother(eve, lebel) mother(eve, cain)
father(adam, abel) father(adam, cain)
 $\forall X \forall Y \text{father}(X, Y) \vee \text{mother}(X, Y) \rightarrow \text{parent}(X, Y)$
 $\forall X \forall Y \forall Z \text{parent}(X, Y) \wedge \text{parent}(X, Z) \rightarrow \text{sibling}(Y, Z)$

Syntaxe du Calcul de Prédicats (suite) ...

- Différentes relations entre la négation et le quantificateurs \exists (existentiel) et \forall (universel) :

$$\neg \exists X p(X) \equiv \forall X \neg p(X)$$

$$\neg \forall X p(X) \equiv \exists X \neg p(X)$$

$$\exists X p(X) \equiv \exists Y p(Y) \quad : \text{variable muette}$$

$$\forall X q(X) \equiv \forall Y q(Y)$$

$$\forall X (p(X) \wedge q(X)) \equiv \forall X p(X) \wedge \forall Y q(Y)$$

$$\exists X (p(X) \vee q(X)) \equiv \exists X p(X) \vee \exists Y q(Y)$$

- La logique de prédicat de 1^{er} ordre permet aux variables quantifiables de référer aux objets du domaine, mais pas aux prédicats et fonctions.

$\forall (\text{likes}) \text{ likes}(\text{george}, \text{kate})$ expression mal formée selon la logique de prédicat de 1^{er} ordre

- D'autres logiques de prédicats d'ordre supérieur permettent de telles expressions.

Syntaxe du Calcul de Prédicats (suite) ...

- **Atome** : Si p est symbole de prédicat d'arité n et si t_1, t_2, \dots, t_n sont des termes, alors $p(t_1, t_2, \dots, t_n)$ est un atome (ou formule atomique).

Exemples :

helps(bill, george)
friends(father_of(david), father_of(andrew))
likes(george, kate)
likes(X, george)
likes(X, X)
likes(george, sarah, tuesday)

- Un symbole de prédicat peut être utilisé avec différents nombres de paramètres (représente dans ce cas des relations différentes).

Syntaxe du Calcul de Prédicats (suite) ...

- **Formule bien formée (FBF)** : définie récursivement par :
 - Tout atome est une formule.
 - **Si** A et B sont deux formules, **alors** $(A \rightarrow B)$, $(A = B)$, $(A \wedge B)$ et $(A \vee B)$ sont des formules.
 - **Si** A est une formule, **alors** $\exists A$ est une formule.
 - **Si** A est une formule, et si X est une variable quelconque, **alors** $(\forall X A)$ est une formule.
(On dit que A est la portée du quantificateur $\forall X$)
 - **Si** A est une formule, et si X est une variable quelconque, **alors** $(\exists X A)$ est une formule.
(On dit que A est la portée du quantificateur $\exists X$)

Syntaxe du Calcul de Prédicats (suite) ...

● Exemples de FBFs :

times

plus } : Fonctions d'arité 2

equal

foo } : prédictats d'arité 2 et 3 respectivement

plus(two, three)

equal(plus(two, three), five) : Fonction mais pas un atome

equal(plus(2, 3), seven) } : Deux atomes

$\exists X \text{ foo}(X, \text{two}, \text{plus}(\text{two}, \text{three})) \wedge (\text{equal}(\text{plus}(\text{two}, \text{three}), \text{five}) = \text{true})$

: Formule (conjonction de formules)

$(\text{foo}(\text{two}, \text{two}, \text{plus}(\text{two}, \text{three}))) \rightarrow (\text{equal}(\text{plus}(\text{three}, \text{two}), \text{five}) = \text{true})$

: Formule (tous les composants sont des formules connectées par des opérateurs logiques)

Algorithme verify_sentence ...

```
function verify_sentence
begin
    case
        expression is atomic sentence: return SUCCESS;
        expression is of the form Q X s or ] s, where Q is either
             $\exists$  or  $\forall$  and X is a variable:
                if (verify_sentence(s) = SUCCESS)
                    then return SUCCESS
                else return FAIL;
        expression is of the form  $s_1 \text{ op } s_2$ , where op is a binary logical operator:
                if (verify_sentence( $s_1$ ) = SUCCESS and
                    verify_sentence( $s_2$ ) = SUCCESS)
                    then return SUCCESS
                else return FAIL;
        otherwise: return FAIL
    end
end.
```

permet de vérifier si une expression est une FBF ou non.

Règles d'Inférences : Définitions ...

- **Règle d'inférence** : moyen permettant de produire de nouvelles expressions du calcul de prédicats à partir d'autres expressions.

- Règles d'inférence utiles :

- **Modus ponens** : si les deux expressions P et $P \rightarrow Q$ sont vraies, alors Q est vraie.

P	$P \rightarrow Q$	Q
V	V	V
F	F	F
F	V	V
F	V	F

Règles d'Inférences Utiles (suite) ...

- **Modus ponens** (suite) :

Observations : 1) « If it is raining then the ground will be wet »
2) « it is raining »

$P = \text{« It is raining »}$

$Q = \text{« The ground is wet »}$

Observation 1) $\longrightarrow P \rightarrow Q$ est vraie

Ensemble d'axiomes étant donné qu'il pleut maintenant :

P
 $P \rightarrow Q$

} **Modus ponens** \longrightarrow On peut ajouter le fait que la terre
est mouillée (Q) à l'ensemble des
expressions ayant la valeur vraie.

Règles d'Inférences Utiles (suite) ...

- **Modus tollens** : si $P \rightarrow Q$ est vraie et Q est fausse, alors $\neg P$ est vraie (ou P est fausse).

P	$P \rightarrow Q$	Q
V	V	V
V	F	F
F	V	V
F	V	F

- **Élimination « et »** : si $P \wedge Q$ est vraie, alors P est vraie et Q est vraie.
- **Introduction « et »** : si P et Q sont vraies, alors $P \wedge Q$ est vraie.

Règles d'Inférences Utiles (suite) ...

- **Instantiation universelle** : si a est une constante appartenant au domaine de X et que $\forall X p(X)$ est vraie, alors $p(a)$ est vraie.

« All man are mortal » $\longrightarrow \forall X (\text{man}(X) \rightarrow \text{mortal}(X))$

« Socrates is a man » $\longrightarrow \text{man}(\text{socrates})$

$\text{man}(\text{socrates}) \xrightarrow[\text{universelle}]{\text{Instantiation}} \text{mortal}(\text{socrates})$

$\text{man}(\text{socrates}) \rightarrow \text{mortal}(\text{socrates})$
 $\text{man}(\text{socrates})$

$\xrightarrow{\text{Modus ponens}} \text{mortal}(\text{socrates})$

Cet axiome peut être ajouté à l'ensemble des expressions qui sont des **conséquences logiques** des assertions originales.

Sémantique du Calcul de Prédicats ...

- **Objectif** : Construire un modèle permettant de dégager une interprétation sémantique des formules.
- **Sémantique du calcul des prédictats** : fournit une base pour déterminer la valeur de vérité des expressions bien formées.

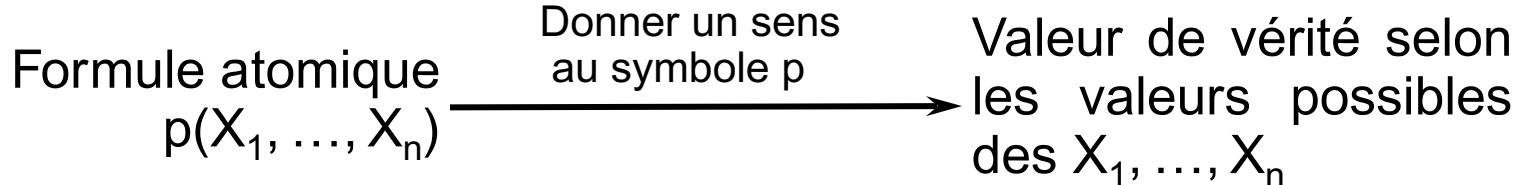
Exemple :

friends(george, susie)
friends(george, kate) } Valeur de vérité de
« *george est un ami de susie* » est vrai (V)

- Calcul propositionnel : utilisation des tables de vérité.
- Calcul de prédictats : différents domaines de variables
 - **problème** : Impossible d'utiliser la méthode de tables de vérité directement dérivée du calcul propositionnel.

Solution : Utiliser une fonction d'interprétation.

Sémantique du Calcul de Prédicats (suite) ...



● Exemple :

prédicat p d'arité un : est_bleu
domaine : {martien, Homme, schtroumfs}
Interprétation possible, notée $I(p)(X)$, de p est :

X	$I(p)(X)$
martien	F
Homme	F
schtroumfs	V

Sémantique du Calcul de Prédicats (suite) ...

- Interprétation d'une formule F est caractérisée par la donnée de :
 - Un ensemble D ($\neq \emptyset$), appelé *domaine de l'interprétation*.
 - Une fonction d'interprétation I qui associe à :

Chaque symbole de F \longrightarrow Un élément de D

Constante \longrightarrow un élément de D

Variable \longrightarrow un sous-ensemble de D

Chaque symbole de fonction f d'arité n \longrightarrow Graphe d'une fonction de $D^n \rightarrow D$ définissant f

Chaque symbole de prédicat p d'arité n \longrightarrow Graphe d'une fonction de $D^n \rightarrow \{V, F\}$ définissant p

Interprétation de Formules ...

- Soit : E une expression
I une interprétation de E sur le domaine D
La valeur de vérité de E est déterminée par :

- 1) Symbole de constante \longrightarrow Sa valeur selon I
- 2) Variable \longrightarrow Sous-ensemble de D affectée à cette variable par I
- 3) Symbole de fonction $f(t_1, \dots, t_n)$ \longrightarrow Terme $f'(\underbrace{t_1, \dots, t_n}_{\text{Interprétation de } f})$
Interprétations des t_1, \dots, t_n
- 4) Symbole « vrai »
de vérité « faux » \longrightarrow V
 \longrightarrow F
- 5) Atome $p(t_1, \dots, t_n)$ \longrightarrow Fonction $p'(\underbrace{t_1, \dots, t_n}_{\text{Interprétation de } p})$
Interprétations des t_1, \dots, t_n

Interprétation de Formules (suite) ...

**6-10) E est de la forme ($\neg A$),
 $(A \rightarrow B)$, $(A \equiv B)$,
 $(A \wedge B)$ ou $(A \vee B)$**  **I(E) est définie par les mêmes lois fonctionnelles que celles du calcul propositionnel**

11) E est de la forme $\rightarrow I(E)(a_1, \dots, a_n) = T$
 $\forall X g(X, Y_1, \dots, Y_n)$, pour tout si pour tout valeur $a \in D$, $I(g)(a, n\text{-uplet } (a_1, \dots, a_n) \in D^n a_1, \dots, a_n) = T$

12) E est de la forme $\rightarrow I(E)(a_1, \dots, a_n) = T$
 $\exists X g(X, Y_1, \dots, Y_n)$, pour tout $S'il existe une valeur $a \in D$, $I(g)(a, a_1, \dots, a_n) = T$$
 n -uplet $(a_1, \dots, a_n) \in D^n$
 $I(E)(a_1, \dots, a_n) = F$ sinon.

Calcul de Prédicats : Exemples ...

- If it doesn't rain on Monday, Tom will go to the mountains.
] **weather(rain, monday) → go(tom, mountains)**
- Emma is a Doberman pinscher and a good dog.
gooddog(emma) ∧ isa(emma, doberman)
- All basketball players are tall.
∀ X (basketball_player(X) → tall(X))
- Some people like anchovies.
(person(X) ∧ likes(X, anchovies))
- If wishes were horses, beggars would ride.
equal(wishes, horses) → ride(beggars)
- Nobody likes taxes.
] **∃ X likes(X, taxes)**

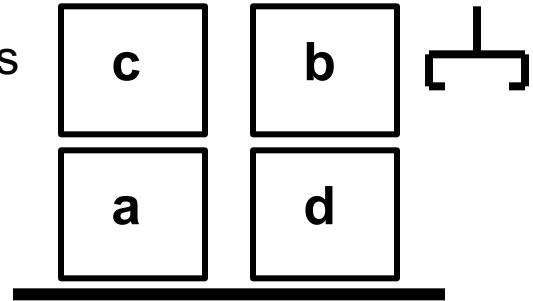
Calcul de Prédicats : Exemples (suite) ...

- Modéliser les blocs suivants afin de concevoir un algorithme de contrôle du bras d'un robot.

- Ensemble de prédicats représentant les relations qualitatives entre les blocs :

on(c, a) ontable(a)
on(b, d) ontable(d)
hand_empty

clear(b)
clear(c)



- Exemples de règles d'inférence :

$$\forall X (\exists Y \text{ on}(Y, X) \rightarrow \text{clear}(X))$$

- Vérifier si un bloc X n'a aucun autre bloc superposé sur lui.
- Déterminer quels sont les blocs qui sont superposés sur le bloc X.

$$\forall X \forall Y (\text{hand_empty} \wedge \text{clear}(X) \wedge \text{clear}(Y) \wedge \text{pickup}(X) \wedge \text{put_down}(X, Y) \rightarrow \text{stack}(X, Y))$$

Des actions du bras sont attachées à ces prédicats

Mettre un bloc sur un autre

- **Formule valide** : une formule F est dite valide si et seulement si, pour toute interprétation I , on a $I(F) = T$.

$p(X) \vee \neg p(X)$: formule valide

- **Formule satisfiable et formule inconsistante** :

une formule A sera dite satisfiable, ou sémantiquement consistante, s'il existe une interprétation I telle que $I(A) = T$.

Domaine d'interprétation = $\{1, 2, \dots\}$

$A = \text{inferieur_a}(X, 0)$ formule inconsistante (non satisfiable)

$B = \text{superieur_a}(X, 0)$ formule satisfiable

$C = \exists X (p(X) \wedge \neg p(X))$ formule inconsistante

- **Conséquence logique** : formule B , famille de n formules A_i

B est conséquence logique des A_i si pour toute interprétation I telle que $\forall A_i, I(A_i) = T$, on a aussi $I(B) = T$.

Domaine d'interprétation = $\{2, 4, 6, \dots\}$

$A_1 = \text{pair}(X)$

$A_2 = \text{superieur_a}(X, 0)$

$B = \neg \text{pair}(\text{plus}(X, 1)) \wedge \neg \text{superieur_a}(\text{moins}(X, \text{plus}(X, X)), 0)$

Unification ...

- Pour appliquer les règles d'inférence telle que modus ponens, un système d'inférence doit être capable de déterminer quand deux expressions sont identiques ou se correspondent.
- **Cas du calcul propositionnel** : deux expressions se correspondent SSI elles sont syntaxiquement identiques.
- **Cas du calcul des prédictats** : le processus de vérifier si deux expressions se correspondent est plus complexe (du fait de l'existence des variables dans les expressions).
- **Unification** : algorithme permettant de déterminer quelles substitutions sont nécessaires pour faire correspondre deux expressions.

Exemple 1 :

$$\begin{array}{c} \forall X \text{ man}(X) \rightarrow \text{mortal}(X) \\ \text{man}(\text{socrates}) \end{array} \quad \left. \begin{array}{l} \\ \end{array} \right\} \quad \begin{array}{l} \text{Substituer } X \text{ par socrates} \\ \text{et utiliser modus ponens} \\ \text{mortal}(\text{socrates}) \end{array}$$

Unification (suite)...

Exemple 2 : variables muettes

$p(X)$ et $p(Y)$ sont équivalentes



On peut substituer Y par X pour faire correspondre les deux expressions.

Unification +
Règles d'inférences



Faire des inférences sur un ensemble d'assertions logiques.

La base de données doit être exprimée sous une forme appropriée.

- **Cas du quantificateur \exists** : une variable quantifiée existentiellement est éliminée des expressions de la base de données en la remplaçant par une constante qui donne la valeur vraie à ces expressions.

$\exists X \text{ parent}(X, \text{tom})$



parent(bob, tom) ou parent(mary, tom)
Où bob et mary sont des parents de tom selon l'interprétation considérée.

Unification (suite)...

- **Problème** : Cas où la valeur des substitutions dépend de la valeur d'autres variables de l'expression :

$\forall X \exists Y \text{mother}(X, Y)$ —> La valeur de la variable quantifiée existentiellement Y dépend de la valeur de X

- **Solution** : Skolémisation

Construire une formule dont la consistance est équivalente à la consistance de la formule initiale.

- **Algorithme de Skolémisation** : permet de construire la forme de Skolem S_F d'une formule F :

1) Transformer la formule F en une forme prénexe.

2) Remplacer toutes les variables quantifiées existentiellement par un symbole de fonction dont les arguments sont les variables qui sont quantifiées universellement et qui précèdent notre variable.

3) Supprimer les quantificateurs existentiels devenus inutiles.

- Exemple.

Unification (suite)...

- Suppression des quantificateurs existentiels.

L'unification peut être utilisée pour trouver une correspondance entre deux formules afin d'appliquer les règles d'inférence telle que modus ponens.

- Exemple d'unification : soit l'expression $\text{foo}(X, a, \text{goo}(Y))$

1) $\text{foo}(\text{fred}, a, \text{goo}(Z)) \longrightarrow \{\text{fred}/X, Z/Y\}$

2) $\text{foo}(W, a, \text{goo}(jack)) \longrightarrow \{W/X, \text{jack}/Y\}$

3) $\text{foo}(Z, a, \text{goo}(\text{moo}(Z))) \longrightarrow \{Z/X, \text{moo}(Z)/Y\}$

Exemples d'unification permettant de rendre l'expression initiale identique à l'une des trois expressions

- Une variable ne peut être unifiée avec un terme contenant cette variable.



Faux, parce que cette substitution crée une expression infinie : $p(p(p(\dots X \dots)))$

Substitution ...

- **Substitution par renommage** : si X_1/X_2 est plus tard X_1 est remplacée par une constante, alors il faut que la variable X_2 reflète cette substitution.

$$\textcircled{1} \quad p(Y, Z) \rightarrow q(Y, Z)$$

$$\textcircled{2} \quad q(W, b) \rightarrow r(W, b)$$

Substitution 1 = {a/Y, X/Z} dans clause $\textcircled{1} \longrightarrow q(a, X)$

Substitution 2 = {a/W, b/X} dans clause $\textcircled{2} \longrightarrow r(a, b)$

- **Composition de substitutions** : soit S et S' deux substitutions définies par :

$$S = \{t_1/X_1, \dots, t_n/X_n\}$$

$$S' = \{u_1/Y_1, \dots, u_n/Y_n\}$$

Substitution (suite)...

- Composition de substitutions : la composition de S et S' est la substitution SS' définie à partir de l'ensemble

$$\underbrace{\{S'(t_1)/X_1, \dots, S'(t_n)/X_n,}_{\text{Appliquer la substitution } S' \text{ aux éléments de } S} u_1/Y_1, \dots, u_n/Y_n\}}_{S'}$$

dont on a supprimé tous les $S'(t_i)/X_i$ pour lesquels $X_i = S'(t_i)$ et tous les u_i/Y_i tel que $Y_i \in \{X_1, \dots, X_n\}$

- Exemples.

- Unificateur : soient deux expressions E_1 et E_2 .

On appelle unificateur de E_1 et E_2 toute substitution S telle que $E_1S = E_2S$.

Exemple : $E_1 = p(X)$ $E_2 = p(Y)$

Unificateur 1 : $\{\text{fred}/X, \text{fred}/Y\}$

Unificateur 2 : $\{Z/X, Z/Y\}$

Unificateur le Plus Général ...

- **Unificateur le Plus Général** d'un ensemble d'expressions E, un unificateur g de E (s'il existe) tel que pour tout unificateur S de E il existe un autre unificateur S' tel que $E S = E g S'$.

Exemple : $E_1 = Y$ $E_2 = f(X)$

Unificateur le plus général : $\{f(X)/Y\}$

- **Algorithme d'unification :**
Simplification de la syntaxe :

Syntaxe du calcul de prédictats

$p(a, b)$
 $p(f(a), g(X, Y))$
 $\text{equal}(\text{eve}, \text{mother}(\text{cain}))$

Syntaxe sous forme de listes

$(p\ a\ b)$
 $(p\ (f\ a)\ (g\ X\ Y))$
 $(\text{equal}\ \text{eve}\ (\text{mother}\ \text{cain}))$

Algorithme d'Unification ...

function unify(E_1, E_2) permet d'unifier deux expressions

begin

case

both E_1 and E_2 are constants or the empty list:

if ($E_1 = E_2$)

then return {}

else return FAIL;

E_1 is a variable:

if (E_1 occurs in E_2)

then return FAIL

else return $\{E_2 / E_1\}$;

E , is a variable:

if (E_2 occurs in E_1)

then return FAIL

else return {E₁ / E₂};

either E_1 or E_2 is the empty list: return FAIL

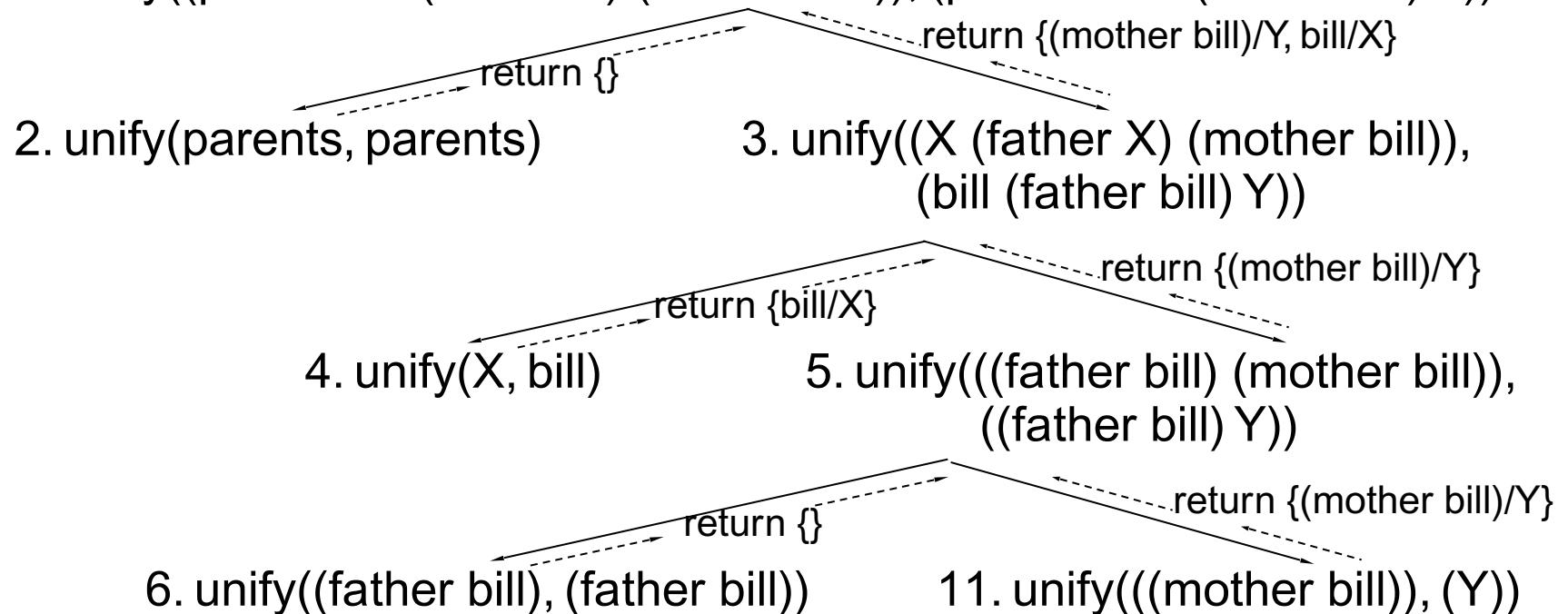
Algorithme d'Unification (suite) ...

otherwise:

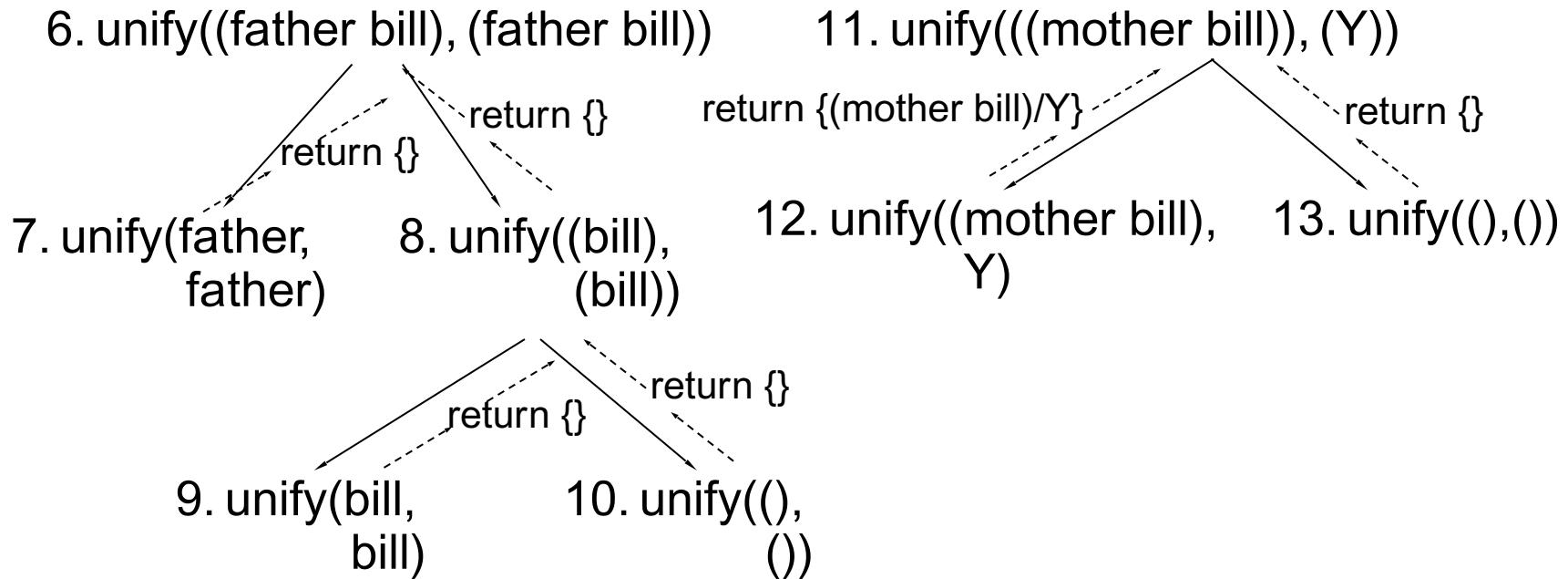
```
begin
    HE1 := first element of E1;
    HE2 := first element of E2;
    SUBS1 := unify(HE1, HE2);
    if (SUBS1 = FAIL) then return FAIL;
    TE1 := apply (SUBS1, rest of E1);
    TE2 := apply (SUBS1, rest of E2);
    SUBS2 := unify(TE1, TE2);
    if (SUBS2 = FAIL)
        then return FAIL;
        else return composition (SUBS1, SUBS2);
end
end
end.
```

Exemple d'Unification ...

1. unify((parents X (father X) (mother bill)), (parents bill (father bill) Y))



Exemple d'Unification (suite) ...



- Exemples.

Résolution par réfutation

Cas du calcul des propositions

1. Convertir les FBFs en une forme clausale (conjonction de clauses)
- 2 . Convertir la négation de la FBF, A, à prouver en une forme clausale
3. Combiner les clauses résultant des 2 étapes précédentes dans un ensemble W (*les clauses combinées sont appelées **clauses parentes***)
4. Appliquer itérativement la résolution pour les clauses dans W jusqu'à il n'y pas de résultat à ajouter ou la clause vide est produite

*La clause résultante est appelée clause **résolvante**,*

Résolution

- **Pour le calcul des prédictats :**
 1. Un littéral et sa négation s'éliminent s'il existe une substitution S qui unifie les deux littéraux. S est ensuite appliquée aux résultat de la résolution. S doit être l'unificateur le plus général.
 2. Les substitutions utilisées dans la preuve par résolution constituent l'ensemble des littéraux/variables pour lesquels la clause à prouver est vraie.

Résolution

Forme clausale

1. Éliminer les implications

Utiliser l'équivalence $P \rightarrow Q \equiv \neg P \vee Q$ pour éliminer les implications

2. Réduire la portée de la négation

Utiliser les lois de DeMorgan

$$\neg(P \vee Q) \equiv (\neg P \wedge \neg Q) \text{ et } \neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$$

et d'autres transformations

$$1.(\neg P) \equiv P$$

$$\neg \exists X p(X) \equiv \forall X \neg p(X)$$

$$\neg \forall X p(X) \equiv \exists X \neg p(X)$$

Résolution

3. Standardiser les variables

Renommer les variables de façon à avoir des variables différentes pour chaque quantificateur.

$$\forall X q(X) \vee \forall X q(X) \equiv \forall X q(X) \vee \forall Y q(Y)$$

4. Mettre sous forme prenexe

Déplacer tous les quantificateurs au début de l'expression en gardant leur ordre d'apparition.

Résolution

5. Éliminer les quantificateurs existentiels (skolémization)

1. Remplacer toutes les variables quantifiées existentiellement par un symbole de fonction dont les arguments sont les variables qui sont quantifiées universellement qui précèdent notre variable.
2. Supprimer les quantificateurs existentiels devenus inutiles.

6. Éliminer les quantificateurs universels

7. Mettre la formules sous forme de conjonctions de disjonctions

Utiliser les propriétés de distributivité

$$(P \vee (Q \wedge R)) \equiv (P \vee Q) \wedge (P \vee R)$$

Résolution

8. Éliminer les conjonctions

On obtient un ensemble de clauses

9. Standardiser les variables

Renommer les variables de façon que chaque 2 clauses ont des variables différentes

Exemples

1. $\text{person}(\text{Marcus}) \Rightarrow \text{person}(\text{Marcus})$
2. $\text{pompeian}(\text{Marcus}) \Rightarrow \text{pompeian}(\text{Marcus})$
3. $\forall x (\text{pompeian}(x) \rightarrow \text{roman}(x)) \Rightarrow \neg \text{pompeian}(x) \vee \text{roman}(x)$
4. $\text{ruler}(\text{Caesar}) \Rightarrow \text{ruler}(\text{Caesar})$
5. $\forall x (\text{roman}(x) \rightarrow \text{loyalto}(x, \text{Caesar}) \vee \text{hate}(x, \text{Caesar})) \Rightarrow$
 $\neg \text{roman}(z) \vee \text{loyalto}(z, \text{Caesar}) \vee \text{hate}(z, \text{Caesar})$
6. $\forall x \exists y \text{loyalto}(x, y) \Rightarrow \text{loyalto}(w, f(w))$
7. $\forall x \forall y (\text{person}(x) \wedge \text{ruler}(y) \wedge \text{tryAssassinate}(x, y) \rightarrow \neg \text{loyalto}(x, y)) \Rightarrow$
 $\neg \text{person}(u) \vee \neg \text{ruler}(y) \vee \neg \text{tryAssassinate}(u, y) \vee \neg \text{loyalto}(u, y)$
8. $\text{tryAssassinate}(\text{Marcus}, \text{Caesar}) \Rightarrow \text{tryAssassinate}(\text{Marcus}, \text{Caesar})$

Règles de résolution pour les prédictats

- Étant donné 2 clauses $\bigcup_i \{L_i\}$ et $\bigcup_j \{M_j\}$, la clause résolvante est calculée comme suit:
- Trouver un littéral L_k et un littéral M_l pour lesquels il existe un unificateur le plus général θ tel que

$$\theta(L_k) = \neg \theta(M_l)$$

- Alors la clause résolvante serait

$$\bigcup_i \{\theta(L_i)\} \setminus \{\theta(L_k)\} \bigcup_j \{\theta(M_j)\} \setminus \{\theta(M_l)\}$$

Factorisation

- Soit C une clause contenant 2 littéraux ayant un unificateur le plus général θ , la clause obtenue par $\theta(C)$ et après avoir éliminé les littéraux identiques est appelé facteur de C et elle est une conséquence logique de C .
- La Factorisation consiste à générer un facteur à partir d'une clause.
- La règle de résolution est sound mais pas complète.
- La résolution et la factorisation représente ensemble un système de preuve complet.

Preuve par Résolution

Pour prouver que la formule f implique la formule g

- Transformer f en un ensemble de clauses
- Transformer $\neg g$ en un autre ensemble de clauses
- Combiner les 2 ensembles de clauses pour obtenir un seul ensemble
- Appliquer répétitivement la règle de *Résolution* et la *factorisation* à cet ensemble jusqu'à générer la clause vide, notée aussi [].

À chaque étape appliquer la règle de résolution à 2 clauses de l'ensemble courant ou la factorisation à une clause de l'ensemble courant. Ajouter la clause résolvante résultat ou le facteur à l'ensemble courant pour obtenir un nouvel ensemble, qui devient l'ensemble courant de la prochaine étape. Arrêter quand l'ensemble contient la clause vide.

Exemple

Étant donné les faits, observations et règles suivantes

1. Marcus is a person
2. Marcus was a Pompeian
3. All Pompeians were Romans
4. Caesar was a ruler
5. All Romans are loyal to Caesar or hate him
6. Everyone is loyal to somebody.
7. The only rulers people try to assassinate are those to whom they are not loyal
8. Marcus tried to assassinate Caesar

Prouver que Marcus hated Caesar

Exemple

Formellement, étant donné la conjonction des formules suivantes:

1. $\text{person}(\text{Marcus})$
2. $\text{pompeian}(\text{Marcus})$
3. $\forall x (\text{pompeian}(x) \rightarrow \text{roman}(x))$
4. $\text{ruler}(\text{Caesar})$
5. $\forall x (\text{roman}(x) \rightarrow \text{loyalto}(x, \text{Caesar}) \vee \text{hate}(x, \text{Caesar}))$
6. $\forall x \exists y \text{loyalto}(x, y)$
7. $\forall x \forall y (\text{person}(x) \wedge \text{ruler}(y) \wedge \text{tryAssassinate}(x, y) \rightarrow \neg \text{loyalto}(x, y))$
8. $\text{tryAssassinate}(\text{Marcus}, \text{Caesar})$

Prouver que $\text{hate}(\text{Marcus}, \text{Caesar})$ en est une conséquence logique

Exemple

Nous prouvons que la conjonction des formules suivantes est inconsistante

1. $\text{person}(\text{Marcus})$
2. $\text{pompeian}(\text{Marcus})$
3. $\forall x (\text{pompeian}(x) \rightarrow \text{roman}(x))$
4. $\text{ruler}(\text{Caesar})$
5. $\forall x (\text{roman}(x) \rightarrow \text{loyalto}(x, \text{Caesar}) \vee \text{hate}(x, \text{Caesar}))$
6. $\forall x \exists y \text{loyalto}(x, y)$
7. $\forall x \forall y (\text{person}(x) \wedge \text{ruler}(y) \wedge \text{tryAssassinate}(x, y) \rightarrow \neg \text{loyalto}(x, y))$
8. $\text{tryAssassinate}(\text{Marcus}, \text{Caesar})$
9. $\neg \text{hate}(\text{Marcus}, \text{Caesar})$

Exemple

Convertir les formules en un ensemble de clauses

1. $\text{person}(\text{Marcus})$
2. $\text{pompeian}(\text{Marcus})$
3. $\neg \text{pompeian}(x) \vee \text{roman}(x)$
4. $\text{ruler}(\text{Caesar})$
5. $\neg \text{roman}(z) \vee \text{loyalto}(z, \text{Caesar}) \vee \text{hate}(z, \text{Caesar})$
6. $\text{loyalto}(w, f(w))$
7. $\neg \text{person}(u) \vee \neg \text{ruler}(y) \vee \neg \text{tryAssassinate}(u, y) \vee \neg \text{loyalto}(u, y)$
8. $\text{tryAssassinate}(\text{Marcus}, \text{Caesar})$
9. $\neg \text{hate}(\text{Marcus}, \text{Caesar})$

Si l'ensemble des clauses est inconsistante la preuve par résolution donnera la clause vide

Extraction de réponse

- Une trace de preuve par résolution peut être utiliser pour répondre à une question.
- L'idée est lorsqu'on prouve que g est une conséquence logique de f ,
 - ajouter le littéral $\text{answer}(x_1, \dots, x_n)$ à chaque clause de la forme clausale de g , avec x_1, \dots, x_n les variables apparaissant dans la clause.
 - ensuite appliquer la procédure de résolution, mais s'arrêter quand nous obtenons une clause composée des littéraux answer (au lieu de la clause vide).

Exemple

1. Marcus is a person
2. Marcus was a Pompeian
3. All Pompeians were Romans
4. Caesar was a ruler
5. All Romans are loyal to Caesar or hate him
6. The only rulers people try to assassinate are those to whom they are not loyal
7. Marcus tried to assassinate Caesar

Is there someone who hates Caesar ?

Exemple

Formellement, étant donné la conjonction des formules suivantes:

1. $person(Marcus)$
2. $pompeian(Marcus)$
3. $\forall x (pompeian(x) \rightarrow roman(x))$
4. $ruler(Caesar)$
5. $\forall x (roman(x) \rightarrow loyalto(x,Caesar) \vee hate(x,Caesar))$
6. $\forall x \exists y loyalto(x,y)$
7. $\forall x \forall y (person(x) \wedge ruler(y) \wedge tryAssassinate(x,y) \rightarrow \neg loyalto(x,y))$
8. $tryAssassinate(Marcus,Caesar)$

prouver que $\exists x hate(x,Caesar)$ est une conséquence logique et donner la valeur de x .

Exemple

Ajouter la négation de $\exists x \text{ hate}(x, Caesar)$ aux autres formules:

1. $\text{person}(\text{Marcus})$
2. $\text{pompeian}(\text{Marcus})$
3. $\forall x (\text{pompeian}(x) \rightarrow \text{roman}(x))$
4. $\text{ruler}(\text{Caesar})$
5. $\forall x (\text{roman}(x) \rightarrow \text{loyalto}(x, Caesar) \vee \text{hate}(x, Caesar))$
6. $\forall x \exists y \text{ loyalto}(x, y)$
7. $\forall x \forall y (\text{person}(x) \wedge \text{ruler}(y) \wedge \text{tryAssassinate}(x, y) \rightarrow \neg \text{loyalto}(x, y))$
8. $\text{tryAssassinate}(\text{Marcus}, \text{Caesar})$
9. $\neg \exists x \text{ hate}(\text{Marcus}, \text{Caesar})$ (equivalent to $\forall x \neg \text{hate}(\text{Marcus}, \text{Caesar})$)

Exemple

Convertir la formule en forme clausale, ajouter le littéral *answer* aux clauses résultant de $\neg \exists x \text{ hate}(Marcus, Caesar)$, puis appliquer la résolution jusqu'à l'obtention de la clause *answer*.

1. $\text{person}(\text{Marcus})$
2. $\text{pompeian}(\text{Marcus})$
3. $\neg \text{pompeian}(x) \vee \text{roman}(x)$
4. $\text{ruler}(\text{Caesar})$
5. $\neg \text{roman}(z) \vee \text{loyalto}(z, \text{Caesar}) \vee \text{hate}(z, \text{Caesar})$
6. $\text{loyalto}(w, f(w))$
7. $\neg \text{person}(u) \vee \neg \text{ruler}(y) \vee \neg \text{tryAssassinate}(u, y) \vee \neg \text{loyalto}(u, y)$
8. $\text{tryAssassinate}(\text{Marcus}, \text{Caesar})$
9. $\neg \text{hate}(v, \text{Caesar}) \vee \text{answer}(v)$

Planning

- The Planning problem
- Planning with State-space search
- Partial-order planning

What is Planning

- Generate sequences of actions to perform tasks and achieve objectives.
 - ◆ States, actions and goals
- Search for solution over abstract space of plans.
- Classical planning environment: fully observable, deterministic, finite, static and discrete.
- Assists humans in practical applications
 - ◆ design and manufacturing
 - ◆ military operations
 - ◆ games
 - ◆ space exploration

Difficulty of real world problems

- Assume a problem-solving agent
- using some search method ...
 - ◆ Which actions are relevant?
 - ◆ Exhaustive search vs. backward search
 - ◆ What is a good heuristic functions?
 - ◆ Good estimate of the cost of the state?
 - ◆ Problem-dependent vs, -independent
 - ◆ How to decompose the problem?
 - ◆ Most real-world problems are nearly decomposable.

Planning language

- What is a good language?
 - ◆ Expressive enough to describe a wide variety of problems.
 - ◆ Restrictive enough to allow efficient algorithms to operate on it.
 - ◆ Planning algorithm should be able to take advantage of the logical structure of the problem.
- STRIPS and ADL

General language features

- Representation of states
 - ◆ Decompose the world in logical conditions and represent a state as a conjunction of positive literals.
 - ◆ Propositional literals: Poor ∪ Unknown
 - ◆ FO-literals (grounded and function-free): $At(Plane1, Melbourne) \wedge At(Plane2, Sydney)$
- Closed world assumption
- Representation of goals
 - ◆ Partially specified state and represented as a conjunction of positive ground literals
 - ◆ A goal is satisfied if the state contains all literals in goal.

General language features

- Representations of actions
- Action = PRECOND + EFFECT

Action(Fly(p,from, to),

PRECOND: At(p,from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)

EFFECT: \neg AT(p,from) \wedge At(p,to))

- = action schema (p, from, to need to be instantiated)
 - ◆ Action name and parameter list
 - ◆ Precondition (conj. of function-free literals)
 - ◆ Effect (conj of function-free literals and P is True and not P is false)
- Add-list vs delete-list in Effect

Language semantics?

How do actions affect states?

- ◆ An action is applicable in any state that satisfies the precondition.
- ◆ For FO action schema applicability involves a substitution θ for the variables in the PRECOND.

$At(P1, JFK) \wedge At(P2, SFO) \wedge Plane(P1) \wedge Plane(P2) \wedge Airport(JFK) \wedge Airport(SFO)$

Satisfies : $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$

With $\theta = \{p/P1, from/JFK, to/SFO\}$

Thus the action is applicable.

Language semantics?

The result of executing action a in state s is the state s'

- s' is same as s except
 - Any positive literal P in the effect of a is added to s'
 - Any negative literal $\neg P$ is removed from s'
- EFFECT: $\neg AT(p, \text{from}) \wedge At(p, \text{to})$:
 $\text{At}(P1, SFO) \wedge \text{At}(P2, SFO) \wedge \text{Plane}(P1) \wedge \text{Plane}(P2) \wedge \text{Airport}(JFK) \wedge \text{Airport}(SFO)$
- STRIPS assumption: (avoids representational frame problem)
every literal NOT in the effect remains unchanged

Expressiveness and extensions

STRIPS is simplified

- ◆ Important limit: function-free literals
 - » Allows for propositional representation
 - » Function symbols lead to infinitely many states and actions

Recent extension: Action Description language (ADL)

*Action(Fly(p :Plane, from: Airport, to: Airport),
PRECOND: At(p ,from) \wedge (from \neq to)
EFFECT: \neg At(p ,from) \wedge At(p ,to))*

Standardization : *Planning domain definition language (PDDL)*

Example: air cargo transport

$Init(At(C1, SFO) \wedge At(C2, JFK) \wedge At(P1, SFO) \wedge At(P2, JFK) \wedge Cargo(C1) \wedge Cargo(C2) \wedge Plane(P1) \wedge Plane(P2) \wedge Airport(JFK) \wedge Airport(SFO))$

$Goal(At(C1, JFK) \wedge At(C2, SFO))$

$Action(Load(c, p, a))$

PRECOND: $At(c, a) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$

EFFECT: $\neg At(c, a) \wedge In(c, p)$

$Action(Unload(c, p, a))$

PRECOND: $In(c, p) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$

EFFECT: $At(c, a) \wedge \neg In(c, p)$

$Action(Fly(p, from, to))$

PRECOND: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$

EFFECT: $\neg At(p, from) \wedge At(p, to)$

$[Load(C1, P1, SFO), Fly(P1, SFO, JFK), Load(C2, P2, JFK), Fly(P2, JFK, SFO)]$

Example: Spare tire problem

Init(At(Flat, Axle) \wedge At(Spare,trunk))

Goal(At(Spare,Axle))

Action(Remove(Spare,Trunk))

 PRECOND: *At(Spare,Trunk)*

 EFFECT: \neg *At(Spare,Trunk) \wedge At(Spare,Ground)*)

Action(Remove(Flat,Axle))

 PRECOND: *At(Flat,Axle)*

 EFFECT: \neg *At(Flat,Axle) \wedge At(Flat,Ground)*)

Action(PutOn(Spare,Axle))

 PRECOND: *At(Spare,Groundp) \wedge \neg At(Flat,Axle)*

 EFFECT: *At(Spare,Axle) \wedge \neg At(Spare,Ground)*)

Action(LeaveOvernight)

 PRECOND:

 EFFECT: \neg *At(Spare,Ground) \wedge \neg At(Spare,Axle) \wedge \neg At(Spare,trunk) \wedge \neg At(Flat,Ground) \wedge \neg At(Flat,Axle))*

This example goes beyond STRIPS: negative literal in pre-condition (ADL description)

Example: Blocks world

$Init(On(A, Table) \wedge On(B, Table) \wedge On(C, Table) \wedge Block(A) \wedge Block(B) \wedge Block(C) \wedge Clear(A) \wedge Clear(B) \wedge Clear(C))$

$Goal(On(A,B) \wedge On(B,C))$

$Action(Move(b,x,y))$

PRECOND: $On(b,x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b) \wedge (b \neq x) \wedge (b \neq y) \wedge (x \neq y)$

EFFECT: $On(b,y) \wedge Clear(x) \wedge \neg On(b,x) \wedge \neg Clear(y))$

$Action(MoveToTable(b,x))$

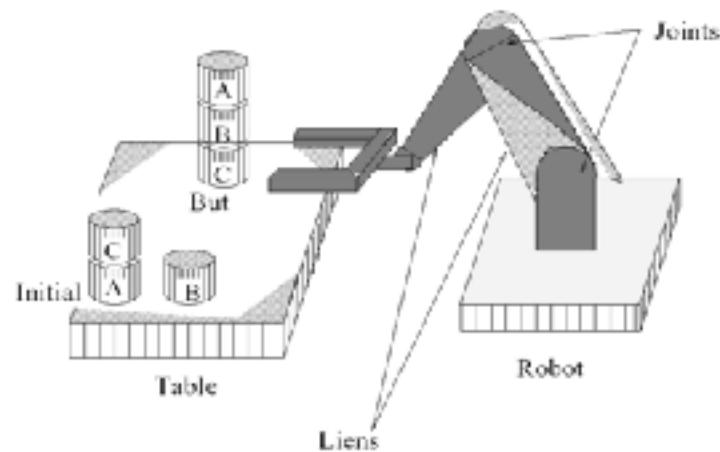
PRECOND: $On(b,x) \wedge Clear(b) \wedge Block(b) \wedge (b \neq x)$

EFFECT: $On(b,Table) \wedge Clear(x) \wedge \neg On(b,x))$

Spurious actions are possible: $Move(B,C,C)$

Example

Monde des blocks (*Blocksworld* en anglais)



Micro-environnement didactique, couramment utilisé en IA.

Example

```
(define (domain blocksworld)
  (:requirements :strips)
  (:predicates (clear ?x) (on-table ?x) (arm-empty) (holding ?x)(on ?x ?y))

  (:action pickup
    :parameters (?ob)
    :precondition (and (clear ?ob) (on-table ?ob) (arm-empty))
    :effect (and (holding ?ob) (not (clear ?ob)) (not (on-table ?ob))
      (not (arm-empty)))))

  (:action putdown
    :parameters (?ob)
    :precondition (holding ?ob)
    :effect (and (clear ?ob) (arm-empty) (on-table ?ob)
      (not (holding ?ob)))))
```

Example

```
(:action stack
  :parameters (?ob ?underob)
  :precondition (and (clear ?underob) (holding ?ob))
  :effect (and (arm-empty) (clear ?ob) (on ?ob ?underob)
    (not (clear ?underob)) (not (holding ?ob)))))

(:action unstack
  :parameters (?ob ?underob)
  :precondition (and (on ?ob ?underob) (clear ?ob) (arm-empty))
  :effect (and (holding ?ob) (clear ?underob)
    (not (on ?ob ?underob)) (not (clear ?ob)) (not (arm-
empty)))))
```

Example

```
(define (problem BWexample)
  (:domain blocksworld)
  (:objects a b c )
  (:init
    (arm-empty) (on-table a) (on-table b) (clear b) (clear c) (on c a)
  )
  (:goal
    (and (on-table a) (on b a) (on c b) (clear c) (arm-empty)))
  )
)
```

Planning with state-space search

Both forward and backward search possible

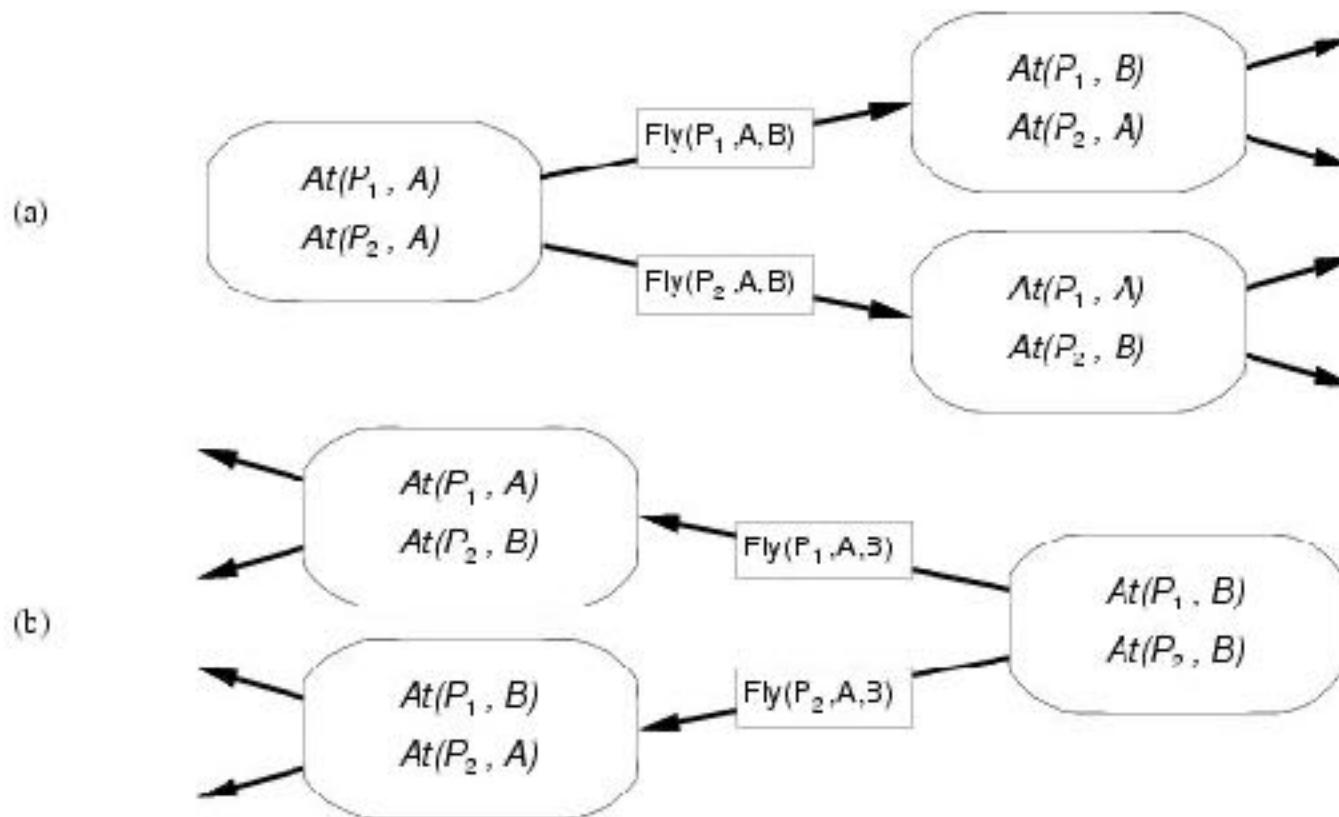
Progression planners

- forward state-space search
- Consider the effect of all possible actions in a given state

Regression planners

- backward state-space search
- To achieve a goal, what must have been true in the previous state.

Progression and regression



Progression algorithm

Formulation as state-space search problem:

- Initial state = initial state of the planning problem
 - Literals not appearing are false
- Actions = those whose preconditions are satisfied
 - Add positive effects, delete negative
- Goal test = does the state satisfy the goal
- Step cost = each action costs 1

No functions ... any graph search that is complete is a complete planning algorithm.

- E.g. A*

Inefficient:

- (1) irrelevant action problem
- (2) good heuristic required for efficient search

Regression algorithm

How to determine predecessors?

- What are the states from which applying a given action leads to the goal?

Goal state = $At(C1, B) \wedge At(C2, B) \wedge \dots \wedge At(C20, B)$

Relevant action for first conjunct: *Unload(C1,p,B)*

Works only if pre-conditions are satisfied.

Previous state= *In(C1, p) \wedge At(p, B) \wedge At(C2, B) \wedge ... \wedge At(C20, B)*

Subgoal $At(C1, B)$ should not be present in this state.

Actions must not undo desired literals (consistent)

Main advantage: only relevant actions are considered.

- Often much lower branching factor than forward search.

Regression algorithm

General process for predecessor construction

- Give a goal description G
- Let A be an action that is relevant and consistent
- The predecessors is as follows:
 - Any positive effects of A that appear in G are deleted.
 - Each precondition literal of A is added , unless it already appears.

Any standard search algorithm can be added to perform the search.

Termination when predecessor satisfied by initial state.

- In FO case, satisfaction might require a substitution.

Heuristics for state-space search

Neither progression or regression are very efficient without a good heuristic.

- How many actions are needed to achieve the goal?
- Exact solution is NP hard, find a good estimate

Two approaches to find admissible heuristic:

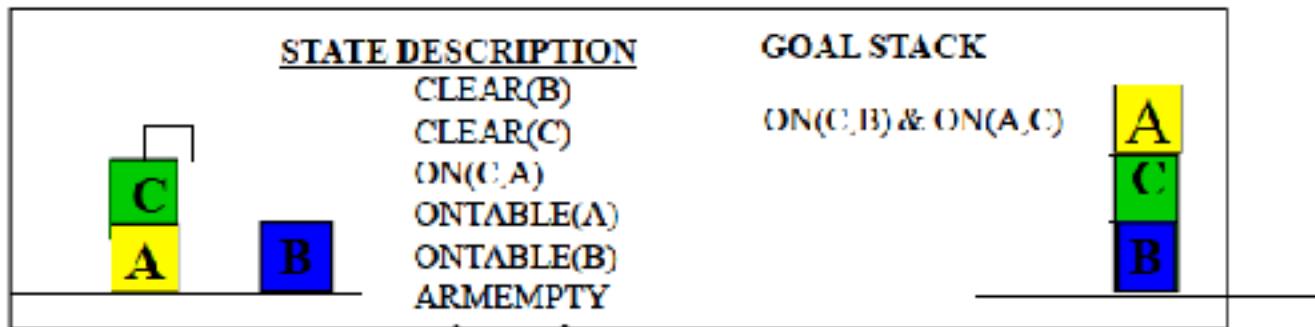
- The optimal solution to the relaxed problem.
 - Remove all preconditions from actions
- The subgoal independence assumption:

The cost of solving a conjunction of subgoals is approximated by the sum of the costs of solving the subproblems independently.

Strips Planning

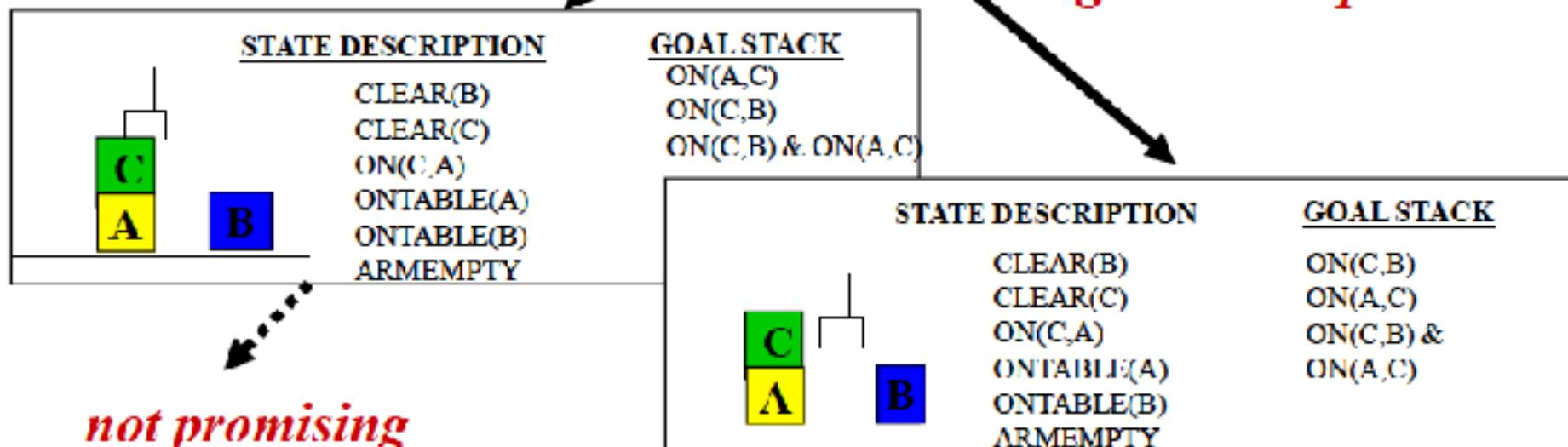
If the top item on the goal stack is:

1. empty(the goal stack is empty) , return the actions executed
 - they form the plan to achieve the goal
2. a goal, and it is satisfied in the current state, remove it from the stack (no replacement necessary)
3. a complex goal ,break it into subgoals, placing all subgoals on the goal stack (the original goal is pushed down in the goal stack)
4. a predicate, find an action that will make it true, then place that action (with variables bound appropriately) and its preconditions on the goal stack (preconditions first)
5. an action and its preconditions are satisfied, perform the action, updating the world state using the delete and add lists of the action (if the pre-conditions are not satisfied, add them to the goal list without removing the action). Add this action to the partial plan

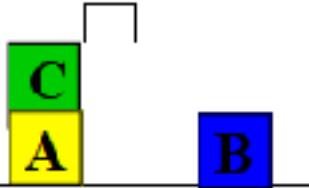


goal decomposition

goal decomposition



*not promising
(why is this?)*

STATE DESCRIPTIONGOAL STACK

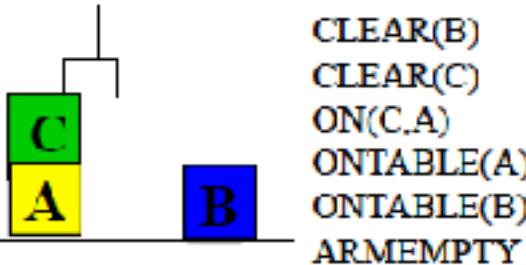
CLEAR(B)
CLEAR(C)
ON(C,A)
ONTABLE(A)
ONTABLE(B)
ARMEMPTY

ON(C,B)
ON(A,C)
ON(C,B) & ON(A,C)

production rule

● **stack(x,y)**

- P&D: HOLDING(x),
CLEAR(y)
- A: ARMEMPTY, ON(x,y),
CLEAR(x)

STATE DESCRIPTIONGOAL STACK

CLEAR(B) & HOLDING(C)
stack(C,B) ←
ON(A,C)
ON(C,B) & ON(A,C)

F-rule

Solution = {}

STATE DESCRIPTION

CLEAR(B)
CLEAR(C)
ON(C,A)
ONTABLE(A)
ONTABLE(B)
ARMEMPTY

GOAL STACK

HOLDING(C)
CLEAR(B)
CLEAR(B) & HOLDING(C)
stack(C,B)
ON(AC.)
ON(C,B) & ON(A,C)

production rule

STATE DESCRIPTION

CLEAR(B)
CLEAR(C)
ON(C,A)
ONTABLE(A)
ONTABLE(B)
ARMEMPTY

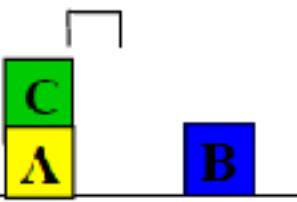
GOAL STACK

!AREMPTY & CLEAR(C) &
ON(C,y)
unstack(C,y)
CLEAR(B)
CLEAR(B) & HOLDING(C)
stack(C,B)
ON(AC.)
ON(C,B) & ON(A,C)

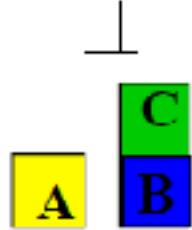
Solution = {}

♦ **unstack(x,y)**

- P&D:
AREMPTY,
CLEAR(x), ON(x,y)
- A: **HOLDING(x),**
CLEAR(y)

STATE DESCRIPTION	GOAL STACK
	HANDEMPTY & CLEAR(C) & ON(C, y) unstack(C, y) CLEAR(B) CLEAR(B) & HOLDING(C) stack(C,B) ON(AC.) ON(C,B) & ON(A,C)
CLEAR(B) CLEAR(C) ON(C,A) ONTABLE(A) ONTABLE(B) ARMEMPTY	

↓
*Substitute {A/y}, then apply
unstack(C,A) then stack(C,B)*

STATE DESCRIPTION	GOAL STACK
	ON(A,C)
CLEAR(C) CLEAR(A) ON(C,B)	ON(C,B) & ON(A,C)
ONTABLE(A) ONTABLE(B) ARMEMPTY	

↓

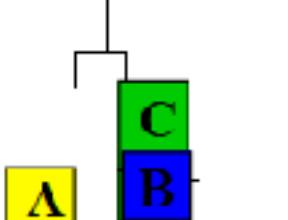
Solution = {unstack(C,A), stack(C,B)}

• unstack(x,y)

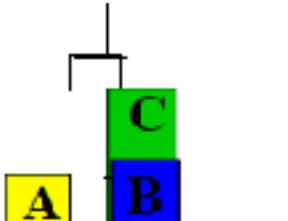
- P&D: ARMEMPTY,
CLEAR(x), ON(x,y)
- A: HOLDING(x),
CLEAR(y)

• stack(x,y)

- P&D: HOLDING(x),
CLEAR(y)
- A: ARMEMPTY,
ON(x,y), CLEAR(x)

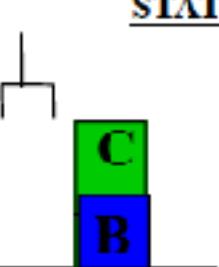
<u>STATE DESCRIPTION</u>	<u>GOAL STACK</u>
	ON(A,C)
CLEAR(C)	ON(C,B) & ON(A,C)
CLEAR(A)	
ON(C,B)	
ONTABLE(A)	
ONTABLE(B)	
ARMEMPTY	

production rule

<u>STATE DESCRIPTION</u>	<u>GOAL STACK</u>
	CLEAR(C) & HOLDING(A)
CLEAR(C)	stack(A,C)
CLEAR(A)	ON(C,B) & ON(A,C)
ON(C,B)	
ONTABLE(A)	
ONTABLE(B)	
ARMEMPTY	

- ◆ **stack(x,y)**
 - ⌘ P&D: HOLDING(x), CLEAR(y)
 - ⌘ A: ARMEMPTY, ON(x,y), CLEAR(x)

Solution = {unstack(C,A), stack(C,B)}



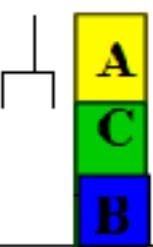
STATE DESCRIPTION

CLEAR(C)
CLEAR(A)
ON(C,B)
ONTABLE(A)
ONTABLE(B)
ARMEMPTY

GOAL STACK

ONTABLE(A) & CLEAR(A) &
ARMEMPTY
pickup(A)
CLEAR(C) & HOLDING(A)
stack(A,C)
ON(C,B) & ON(A,C)

*Apply pickup(A)
and then
stack(A,C)*



STATE DESCRIPTION

ON(A,C)
ON(C,B)
CLEAR(A)
ONTABLE(B)
ARMEMPTY

GOAL STACK

NIL



**Solution plan = {unstack(C,A), stack(C,B),
pickup(A), stack(A,C)}**

◆ **pickup(x)**

- P&D: ONTABLE(x),
CLEAR(x), HANDEMPTY
- A: HOLDING(x)

◆ **stack(x,y)**

- P&D: HOLDING(x),
CLEAR(y)
- A: HANDEMPTY,
ON(x,y), CLEAR(x)

Initial state:

clear(a)

clear(b)

clear(c)

ontable(a)

ontable(b)

ontable(c)

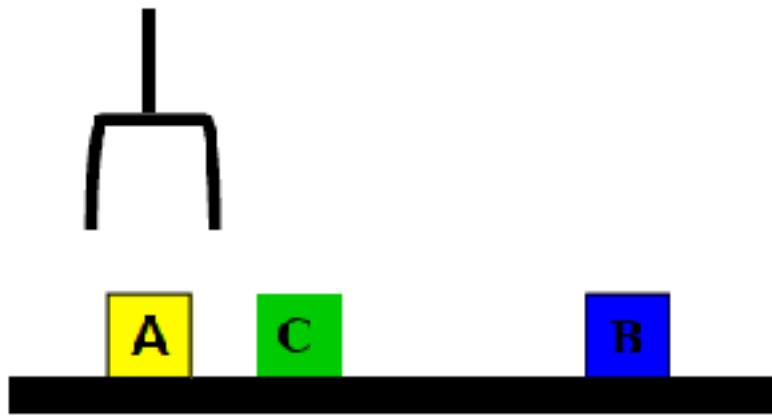
handempty

Goal:

on(b,c)

on(a,b)

ontable(c)



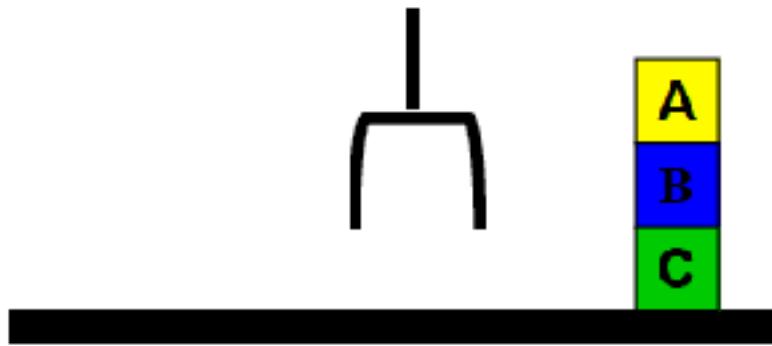
A plan:

pickup(b)

stack(b,c)

pickup(a)

stack(a,b)



Partial-order planning

Progression and regression planning are *totally ordered plan search* forms.

- They cannot take advantage of problem decomposition.
 - Decisions must be made on how to sequence actions on all the subproblems

Least commitment strategy:

- Delay choice during search

Shoe example

Goal(RightShoeOn \wedge LeftShoeOn)

Init()

Action(RightShoe, PRECOND: RightSockOn
EFFECT: RightShoeOn)

Action(RightSock, PRECOND:
EFFECT: RightSockOn)

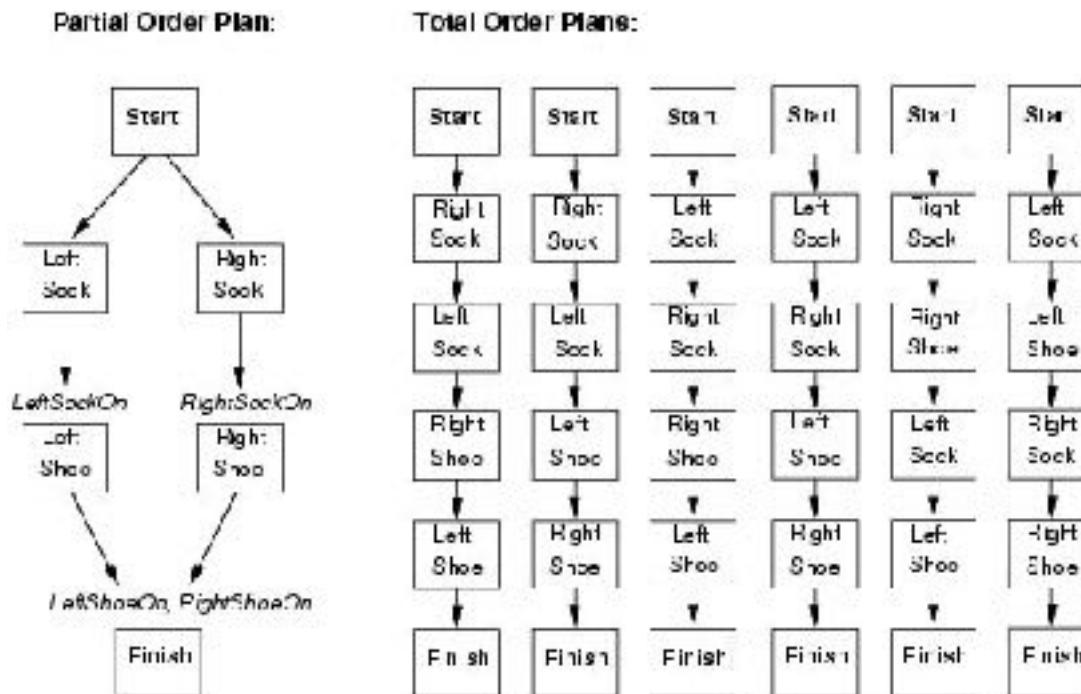
Action(LeftShoe,
 EFFECT: LeftShoeOn) PRECOND: LeftSockOn

Action(LeftSock, PRECOND:
EFFECT: LeftSockOn)

Planner: combine two action sequences (1)leftsock, leftshoe (2)rightsock, rightshoe

Partial-order planning(POP)

Any planning algorithm that can place two actions into a plan without which comes first is a PO plan.



POP as a search problem

States are (mostly unfinished) plans.

- The empty plan contains only start and finish actions.

Each plan has 4 components:

- A set of actions (steps of the plan)
- A set of ordering constraints: $A < B$ (A before B)
 - Cycles represent contradictions.
- A set of causal links $A \xrightarrow{p} B$
 - The plan may not be extended by adding a new action C that conflicts with the causal link. (if the effect of C is $\neg p$ and if C could come after A and before B)
- A set of open preconditions.
 - If precondition is not achieved by action in the plan.

Actions={Rightsock, Rightshoe, Leftsock,
Leftshoe, Start, Finish}

Orderings={Rightsock < Rightshoe; Leftsock
< Leftshoe}

Links={Rightsock->Rightsockon ->
Rightshoe, Leftsock->Leftsockon-> Leftshoe,
Rightshoe->Rightshoeon->Finish, ...}

Open preconditions={}

POP as a search problem

A plan is *consistent* iff there are no cycles in the ordering constraints and no conflicts with the causal links.

A consistent plan with no open preconditions is a *solution*.

A partial order plan is executed by repeatedly choosing *any* of the possible next actions.

- This flexibility is a benefit in non-cooperative environments.

Solving POP

Assume propositional planning problems:

- The initial plan contains *Start* and *Finish*, the ordering constraint $\textit{Start} < \textit{Finish}$, no causal links, all the preconditions in *Finish* are open.
- Successor function :
 - picks one open precondition p on an action B and
 - generates a successor plan for every possible consistent way of choosing action A that achieves p .
- Test goal

Enforcing consistency

When generating successor plan:

- The causal link $A \rightarrow p \rightarrow B$ and the ordering constraint $A < B$ is added to the plan.
 - If A is new also add $\text{start} < A$ and $A < B$ to the plan
- Resolve conflicts between new causal link and all existing actions
- Resolve conflicts between action A (if new) and all existing causal links.

Process summary

Operators on partial plans

- Add link from existing plan to open precondition.
- Add a step to fulfill an open condition.
- Order one step w.r.t another to remove possible conflicts

Gradually move from incomplete/vague plans
to complete/correct plans

Backtrack if an open condition is
unachievable or if a conflict is irresolvable.

Example: Spare tire problem

Init(At(Flat, Axle) \wedge At(Spare, trunk))

Goal(At(Spare, Axle))

Action(Remove(Spare, Trunk))

 PRECOND: *At(Spare, Trunk)*

 EFFECT: \neg *At(Spare, Trunk) \wedge At(Spare, Ground)*)

Action(Remove(Flat, Axle))

 PRECOND: *At(Flat, Axle)*

 EFFECT: \neg *At(Flat, Axle) \wedge At(Flat, Ground)*)

Action(PutOn(Spare, Axle))

 PRECOND: *At(Spare, Groundp) \wedge \neg At(Flat, Axle)*

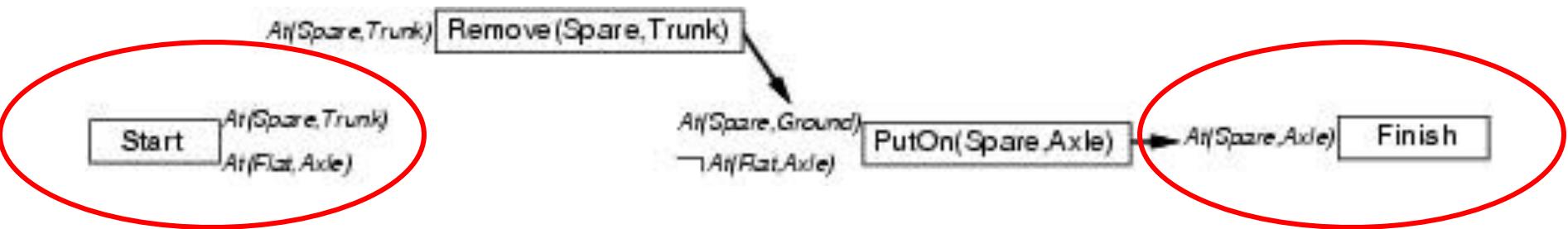
 EFFECT: *At(Spare, Axle) \wedge \neg Ar(Spare, Ground)*)

Action(LeaveOvernight)

 PRECOND:

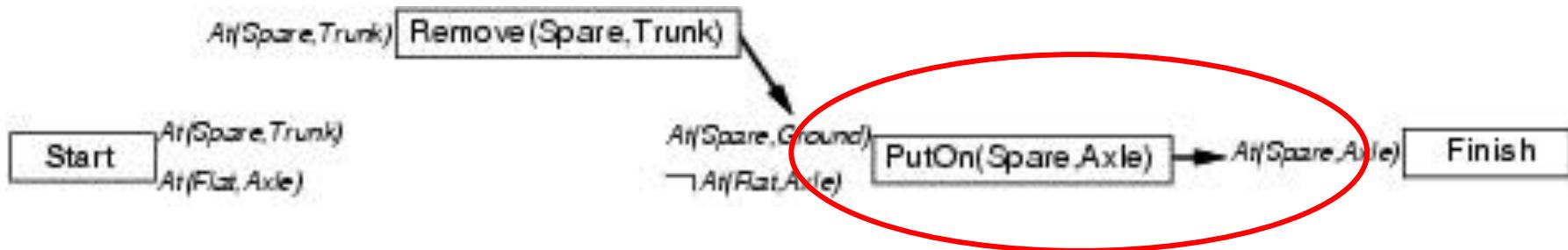
 EFFECT: \neg *At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, trunk) \wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle))*

Solving the problem



Initial plan: Start with EFFECTS and Finish with PRECOND.

Solving the problem



Initial plan: Start with EFFECTS and Finish with PRECOND.

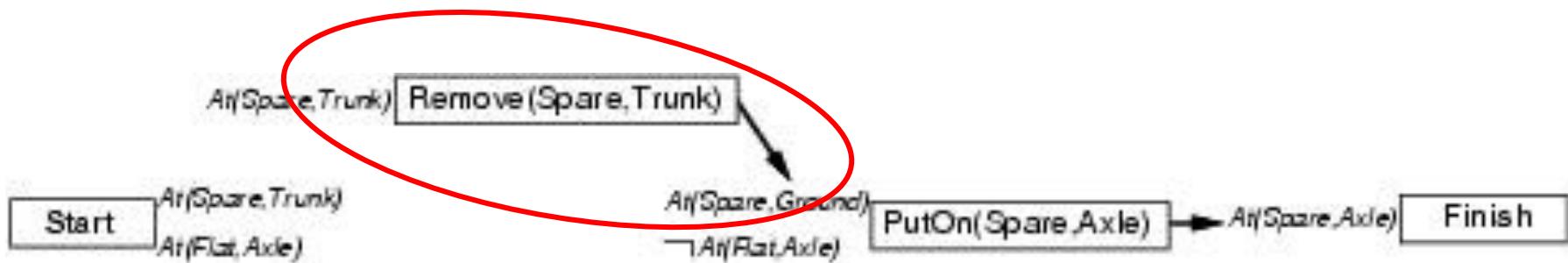
Pick an open precondition: *At(Spare, Axle)*

Only *PutOn(Spare, Axle)* is applicable

Add causal link: $\text{PutOn}(\text{Spare}, \text{Axle}) \xrightarrow{\text{At}(\text{Spare}, \text{Axle})} \text{Finish}$

Add constraint : $\text{PutOn}(\text{Spare}, \text{Axle}) < \text{Finish}$

Solving the problem



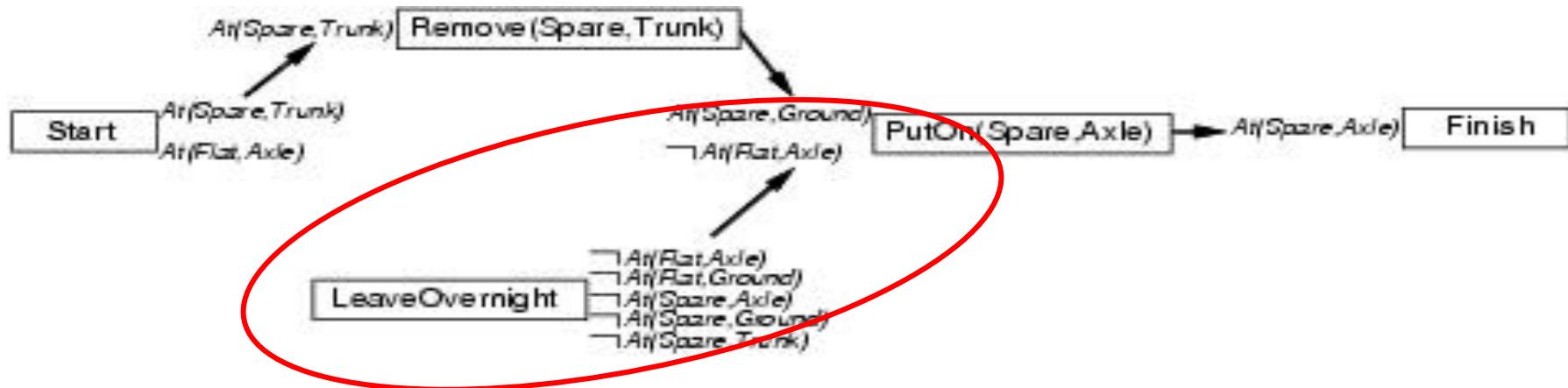
Pick an open precondition: *At(Spare, Ground)*

Only *Remove(Spare, Trunk)* is applicable

Add causal link: *Remove(Spare,Trunk)* $\xrightarrow{\text{At(Spare,Ground)}}$ *PutOn(Spare,Axle)*

Add constraint : *Remove(Spare, Trunk)* < *PutOn(Spare, Axle)*

Solving the problem



Pick an open precondition: $\neg At(Flat, Axle)$

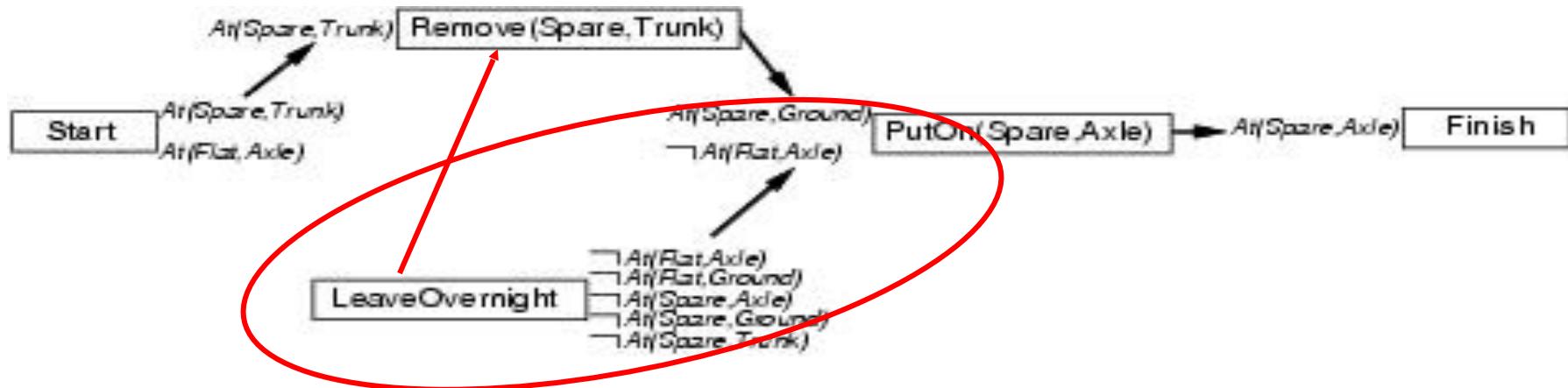
LeaveOverNight is applicable

conflict: *LeaveOverNight* also has the effect $\neg At(Spare, Ground)$

$Remove(Spare,Trunk) \xrightarrow{At(Spare,Ground)} PutOn(Spare,Axle)$

To resolve, add constraint : $LeaveOverNight < Remove(Spare, Trunk)$

Solving the problem



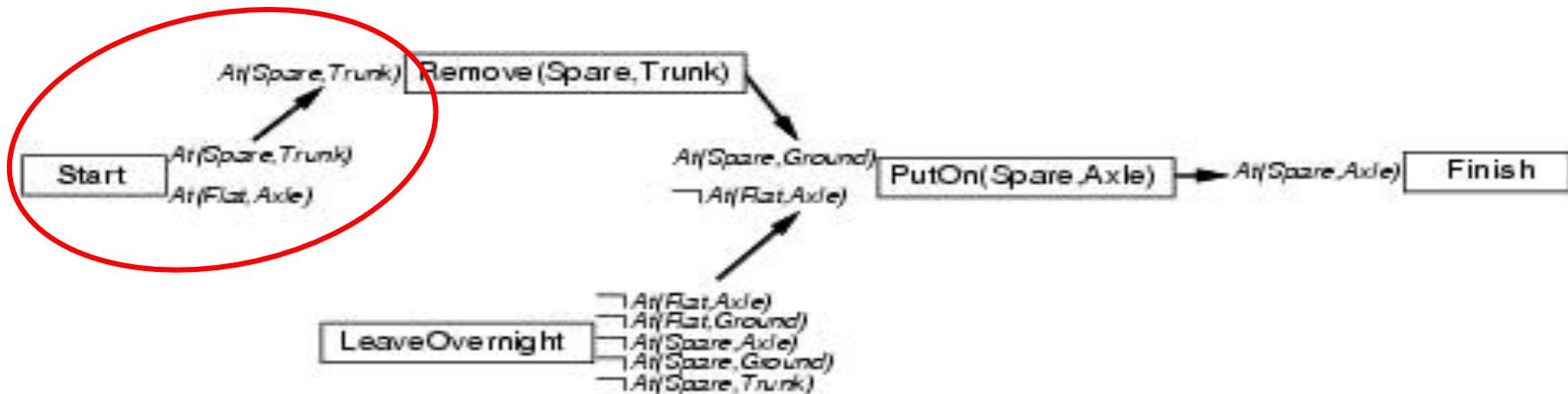
Pick an open precondition: *At(Spare, Ground)*
LeaveOverNight is applicable

conflict: $\text{Remove}(\text{Spare}, \text{Trunk}) \xrightarrow{\text{At}(\text{Spare}, \text{Ground})} \text{PutOn}(\text{Spare}, \text{Axle})$

To resolve, add constraint : $\text{LeaveOverNight} < \text{Remove}(\text{Spare}, \text{Trunk})$

Add causal link: $\text{LeaveOverNight} \xrightarrow{\neg \text{At}(\text{Spare}, \text{Ground})} \text{PutOn}(\text{Spare}, \text{Axle})$

Solving the problem



Pick an open precondition: $At(Spare, Trunk)$

Only *Start* is applicable

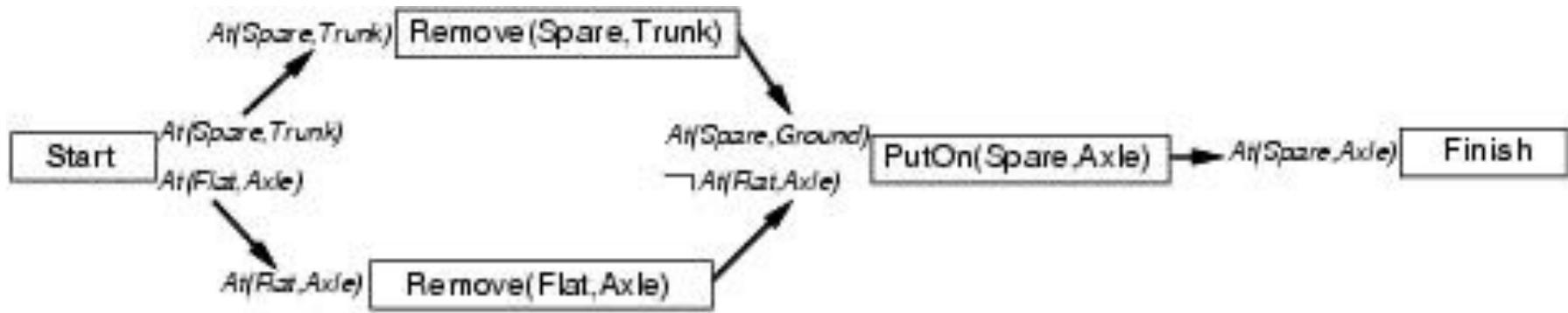
Add causal link: $Start \xrightarrow{At(Spare, Trunk)} Remove(Spare, Trunk)$

Conflict: of causal link with effect $At(Spare, Trunk)$ in *LeaveOverNight*

- No re-ordering solution possible.

backtrack

Solving the problem



Remove *LeaveOverNight*, *Remove(Spare, Trunk)* and causal links

Repeat step with *Remove(Spare, Trunk)*

Add also *RemoveFlatAxle* and finish

Some details ...

What happens when a first-order representation that includes variables is used?

- Complicates the process of detecting and resolving conflicts.
- Can be resolved by introducing inequality constraint.

CSP's most-constrained-variable constraint can be used for planning algorithms to select a PRECOND.

Planners

<i>Planner</i>	<i>Reference</i>	<i>Applications</i>
STRIPS	Fikes & Nilsson 1971	Mobile Robot Control, etc.
HACKER	Sussman 1973	Simple Program Generation
NOAH	Sacerdoti 1977	Mechanical Engineers Apprentice Supervision
NONLIN	Tate 1977	Electricity Turbine Overhaul, etc.
NASL	McDermott 1978	Electronic Circuit Design
OPM	Hayes-Roth & Hayes-Roth 1979	Journey Planning
ISIS-II	Fox et. al. 1981	Job Shop Scheduling (Turbine Production)
MOLGEN	Stefik 1981	Experiment Planning in Molecular Genetics
DEVISER	Vere 1983	Spacecraft Mission Planning
FORBIN	Miller et al. 1985	Factory Control
SIPE-2	Wilkins 1988	Oil Spill Response, Military Planning, etc.
SHOP/SHOP-2	Nau et al. 1999	Evacuation Planning, Forest Fires, Bridge Baron, etc.
I-X/I-Plan	Tate et al. 2000	Emergency Response, etc.

Analysis of planning approach

Planning is an area of great interest within AI

- Search for solution
- Constructively prove existence of solution

Biggest problem is the combinatorial explosion in states.

Raisonnement Incertain
Réseaux bayésiens

Uncertainty

- In search we viewed actions as being deterministic.
 - ◆ If you are in state S1 and you execute action A you arrive at state S2.
- Furthermore, there was a fixed initial state S0. So with deterministic actions after executing any sequence of actions we know exactly what state we have arrived at.
 - ◆ Always know what state one is in.
- But in many domains they are not true.

Uncertainty

- We might not know exactly what state we start off in
 - ◆ E.g., we can't see our opponents cards in a poker game
- We might not know all of the effects of an action
 - ◆ The action might have a random component, like rolling dice.
- We might not know all of the long term effects of a drug.
- We might not know the status of a road when we choose the action of driving down it

Uncertainty

- In such domains we still need to act, but we can't act solely on the basis of known true facts. We have to “gamble”.
- E.g., we don't know for certain what the traffic will be on a trip to the airport.

Uncertainty

- But how do we gamble **rationally**?
 - ◆ If we must arrive at the airport at 9pm on a week night we could “safely” leave for the airport 1 hour before.
 - Some probability of the trip taking longer, but the probability is low.
 - ◆ If we must arrive at the airport at 4:30pm on Friday we most likely need 1 hour or more to get to the airport.
 - » Relatively high probability of it taking 1.5 hour

Uncertainty

- To act rationally under uncertainty we must be able to evaluate how likely certain things are.
- By weighing likelihoods of events (probabilities) we can develop mechanisms for acting rationally under uncertainty

Uncertainty

- General situation:
 - ◆ **Observed variables (evidence):** Agent knows certain things about the state of the world (e.g., sensor readings or symptoms)
 - ◆ **Unobserved variables:** Agent needs to reason about other aspects (e.g. where an object is or what disease is present)
 - ◆ **Model:** Agent knows something about how the known variables relate to the unknown variables
- Probabilistic reasoning gives us a framework for managing our beliefs and knowledge

Probability

- Given a set \mathbf{U} (universe), a probability function is a function defined over the subsets of \mathbf{U} that maps each subset to the real numbers and that satisfies the Axioms of Probability
- $\Pr(\mathbf{U}) = 1$
- $\Pr(A) \in [0,1]$
- $\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B)$

» $P(A) = P(A \cap B) + P(A \cap \sim B)$

Properties and Sets

- Any set of events A can be interpreted as a property: the set of events with property A.
 - ◆ Hence, we often write
 - ◆ $A \vee B$ to represent the set of events with either property A or B: the set $A \cup B$
 - ◆ $A \wedge B$ to represent the set of events both property A and B: the set $A \cap B$
 - ◆ $\sim A$ to represent the set of events that do not have property A: the set $U - A$
(i.e., the complement of A wrt the universe of events U)

Conditional probabilities

- With probabilities one can capture conditional information by using conditional probabilities.
- Conditional probabilities are essential for both representing and reasoning with probabilistic information.

Conditional Probability

- $P(A|B)$ = the probability of an event A occurring given that the event B has already occurred.

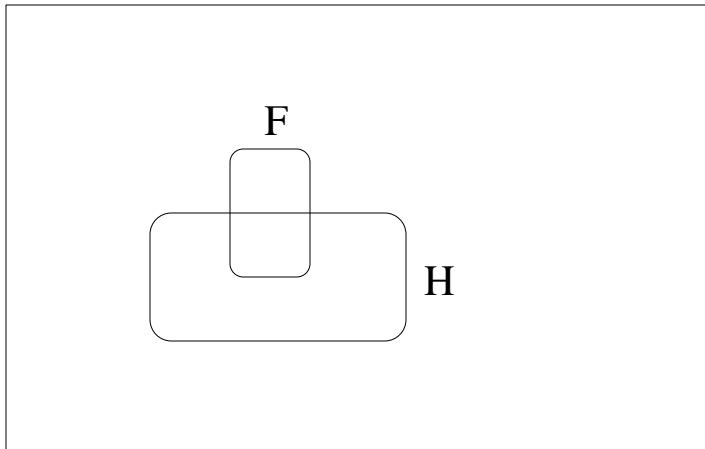
H = “Have a headache”

F = “Coming down with Flu”

$$P(H) = 1/10$$

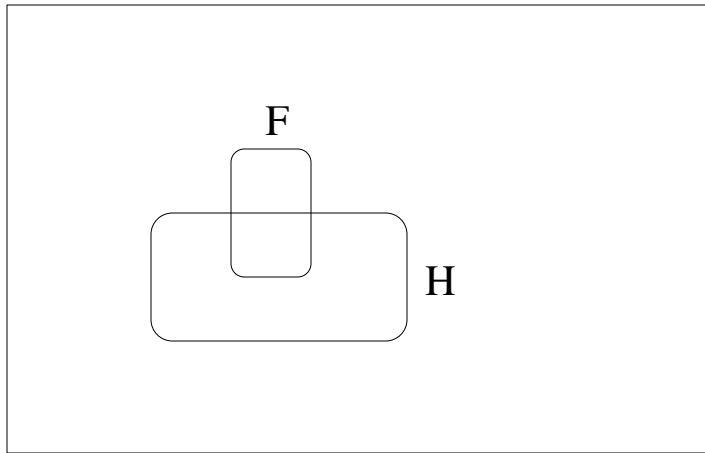
$$P(F) = 1/40$$

$$P(H|F) = 1/2$$



“Headaches are rare and flu is rarer, but if you’re coming down with ‘flu there’s a 50-50 chance you’ll have a headache.”

Conditional Probability



H = “Have a headache”

F = “Coming down with Flu”

$$P(H) = 1/10$$

$$P(F) = 1/40$$

$$P(H|F) = 1/2$$

$P(H|F)$ = Fraction of flu-inflicted worlds
in which you have a headache

= Area of “H and F” region

Area of “F” region

$$= P(H \wedge F)$$

P(F)

Definition of Conditional Probability

$$P(A \wedge B) \\ P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

Corollary: The Chain Rule

$$P(A \wedge B) = P(A|B) P(B)$$

$$P(X_1, X_2, \dots, X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1, X_2)\dots \\ = \prod_{i=1}^n P(X_i|X_1, \dots, X_{i-1})$$

Random variable

- a variable in probability theory with a domain of possible values it can take on
- weather
 - {sunny, windy, rainy}
- flight
 - {on time, delayed, canceled}

Probability distribution

- $P(\text{flight} = \text{on time}) = 0.6$
- $P(\text{flight} = \text{delayed}) = 0.3$
- $P(\text{flight} = \text{canceled}) = 0.1$
- $P(\text{flight}) = <0.6, 0.3, 0.1>$

Independence

- The knowledge that one event occurs does not affect the probability of the other event to occur.
 - $P(a \wedge b) = P(a)P(a|b) = P(a)P(b)$

Bayes Rule

$$P(A \wedge B) = P(B|A) P(A)$$

$$P(A|B) = \frac{P(A \wedge B)}{P(B)} = \frac{P(B|A) P(A)}{P(B)}$$

This is Bayes Rule

Bayes, Thomas (1763) An essay towards
solving a problem in the doctrine of chances.
*Philosophical Transactions of the Royal
Society of London*, 53:370-418



More General Forms of Bayes Rule

$$P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|\sim A)P(\sim A)}$$

$$P(A|B \wedge X) = \frac{P(B|A \wedge X)P(A \wedge X)}{P(B \wedge X)}$$

$$P(A|B) + P(\neg A|B) = 1$$

Bayes rules

- Knowing
 - ◆ $P(\text{visible effect} \mid \text{unknown cause})$
- we calculate
 - ◆ $P(\text{unknown cause} \mid \text{visible effect})$

Bayes Rule Example

- Example: Diagnostic probability from causal probability:

$$P(\text{cause}|\text{effect}) = \frac{P(\text{effect}|\text{cause})P(\text{cause})}{P(\text{effect})}$$

- Example:

- M: meningitis, S: stiff neck

$$\left. \begin{array}{l} P(+m) = 0.0001 \\ P(+s|m) = 0.8 \\ P(+s|-m) = 0.01 \end{array} \right\} \text{Example givens}$$

$$P(+m|+s) = \frac{P(+s|m)P(+m)}{P(+s)} = \frac{P(+s|m)P(+m)}{P(+s|m)P(+m) + P(+s|-m)P(-m)} = \frac{0.8 \times 0.0001}{0.8 \times 0.0001 + 0.01 \times 0.999}$$

- Note: posterior probability of meningitis still very small
 - Note: you should still get stiff necks checked out! Why?

Bayes Rule Example

- Disease $\in \{\text{malaria, cold, flu}\}$; Symptom = fever
 - ◆ Must compute $\Pr(D \mid \text{fever})$ to prescribe treatment
- Why not assess this quantity directly?
 - ◆ $\Pr(\text{mal} \mid \text{fever})$ is not natural to assess;
 - ◆ $\Pr(\text{mal} \mid \text{fever})$ does not reflect the underlying “causal” mechanism fever \rightarrow malaria
 - ◆ $\Pr(\text{mal} \mid \text{fever})$ is not “stable”: a malaria epidemic changes this quantity (for example)
- So we use Bayes rule: $\Pr(\text{mal} \mid \text{fever}) = \Pr(\text{fever} \mid \text{mal}) \Pr(\text{mal}) / \Pr(\text{fever})$

Bayes Rule Example

- $\Pr(\text{mal} \mid \text{fever}) = \Pr(\text{fever} \mid \text{mal})\Pr(\text{mal})/\Pr(\text{fever})$
- $\Pr(\text{mal})?$
 - ◆ This is the prior probability of Malaria, i.e., before you exhibited a fever, and with it we can account for other factors, e.g., a malaria epidemic, or recent travel to a malaria risk zone.
 - ◆ E.g., The center for disease control keeps track of the rates of various diseases.
- $\Pr(\text{fever} \mid \text{mal})$ and $\Pr(\text{fever} \mid \neg\text{mal})?$
 - ◆ This is the probability a patient with malaria exhibits a fever.
 - ◆ Again this kind of information is available from people who study malaria and its effects.

The Joint Distribution

*Example: Boolean variables
A, B, C*

Recipe for making a joint distribution of
M variables:

The Joint Distribution

Recipe for making a joint distribution of M variables:

1. Make a truth table listing all combinations of values of your variables (if there are M Boolean variables then the table will have 2^M rows).

Example: Boolean variables A, B, C

A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

The Joint Distribution

Recipe for making a joint distribution of M variables:

1. Make a truth table listing all combinations of values of your variables (if there are M Boolean variables then the table will have 2^M rows).
2. For each combination of values, say how probable it is.

Example: Boolean variables A, B, C

A	B	C	Prob
0	0	0	0.30
0	0	1	0.05
0	1	0	0.10
0	1	1	0.05
1	0	0	0.05
1	0	1	0.10
1	1	0	0.25
1	1	1	0.10

The Joint Distribution

Recipe for making a joint distribution of M variables:

1. Make a truth table listing all combinations of values of your variables (if there are M Boolean variables then the table will have 2^M rows).
2. For each combination of values, say how probable it is.
3. If you subscribe to the axioms of probability, those numbers must sum to 1.

Example: Boolean variables A, B, C

A	B	C	Prob
0	0	0	0.30
0	0	1	0.05
0	1	0	0.10
0	1	1	0.05
1	0	0	0.05
1	0	1	0.10
1	1	0	0.25
1	1	1	0.10

Using the Joint

gender	hours_worked	wealth	
Female	v0:40.5-	poor	0.253122
		rich	0.0245895
	v1:40.5+	poor	0.0421768
		rich	0.0116293
Male	v0:40.5-	poor	0.331313
		rich	0.0971295
	v1:40.5+	poor	0.134106
		rich	0.105933

Once you have the JD you can ask for the probability of any logical expression involving your attribute

$$P(E) = \sum_{\text{rows matching } E} P(\text{row})$$

Using the Joint

gender	hours_worked	wealth	
Female	v0:40.5-	poor	0.253122
		rich	0.0245895
	v1:40.5+	poor	0.0421768
		rich	0.0116293
Male	v0:40.5-	poor	0.331313
		rich	0.0971295
	v1:40.5+	poor	0.134106
		rich	0.105933

$$P(\text{Poor} \wedge \text{Male}) = 0.4654$$

$$P(E) = \sum_{\text{rows matching } E} P(\text{row})$$

Using the Joint

gender	hours_worked	wealth	
Female	v0:40.5-	poor	0.253122
		rich	0.0245895
Male	v1:40.5+	poor	0.0421768
		rich	0.0116293
Male	v0:40.5-	poor	0.331313
		rich	0.0971295
Male	v1:40.5+	poor	0.134106
		rich	0.105933

$$P(\text{Poor}) = 0.7604$$

$$P(E) = \sum_{\text{rows matching } E} P(\text{row})$$

Inference with the Joint

gender	hours_worked	wealth	
Female	v0:40.5-	poor	0.253122
		rich	0.0245895
	v1:40.5+	poor	0.0421768
		rich	0.0116293
Male	v0:40.5-	poor	0.331313
		rich	0.0971295
	v1:40.5+	poor	0.134106
		rich	0.105933

$$P(E_1 | E_2) = \frac{P(E_1 \wedge E_2)}{P(E_2)} = \frac{\sum_{\text{rows matching } E_1 \text{ and } E_2} P(\text{row})}{\sum_{\text{rows matching } E_2} P(\text{row})}$$

Inference with the Joint

gender	hours_worked	wealth	
Female	v0:40.5-	poor	0.253122
		rich	0.0245895
v1:40.5+		poor	0.0421768
		rich	0.0116293
Male	v0:40.5-	poor	0.331313
		rich	0.0971295
v1:40.5+		poor	0.134106
		rich	0.105933

$$P(E_1 | E_2) = \frac{P(E_1 \wedge E_2)}{P(E_2)} = \frac{\sum_{\text{rows matching } E_1 \text{ and } E_2} P(\text{row})}{\sum_{\text{rows matching } E_2} P(\text{row})}$$

$$P(\text{Male} | \text{Poor}) = 0.4654 / 0.7604 = 0.612$$

Inference with the Joint : Normalization trick

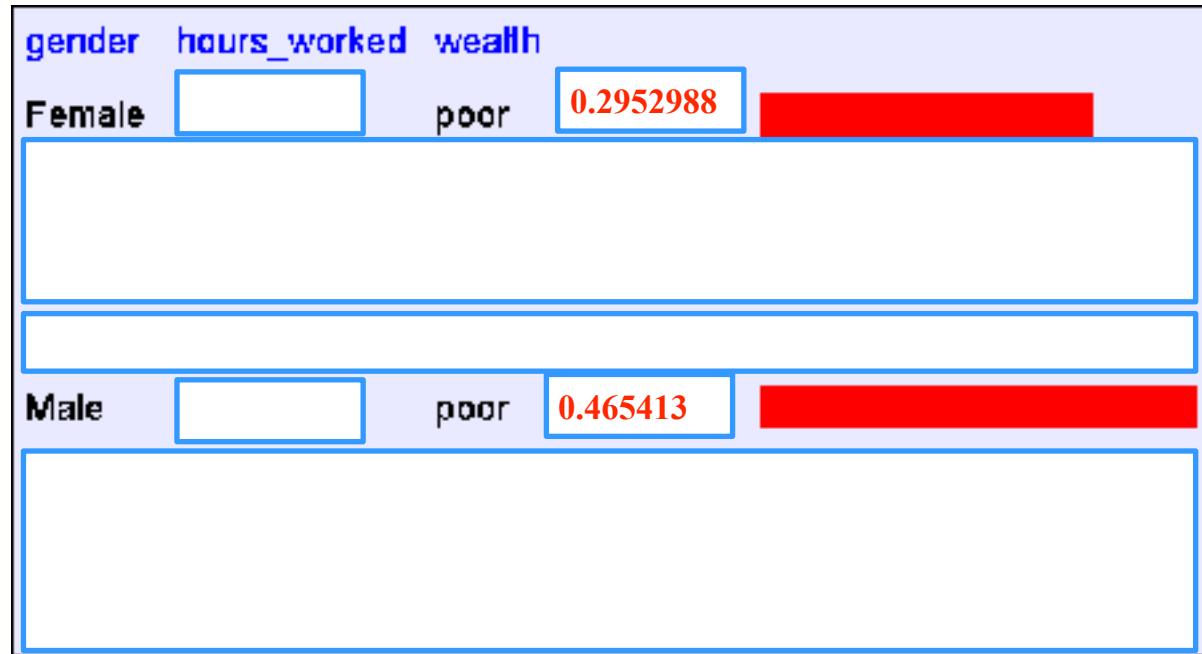
SELECT the joint
probabilities
matching the
evidence E2



gender	hours_worked	wealth	
Female	v0:40.5-	poor	0.253122
	v1:40.5+	poor	0.0421768
Male	v0:40.5-	poor	0.331313
	v1:40.5+	poor	0.134106

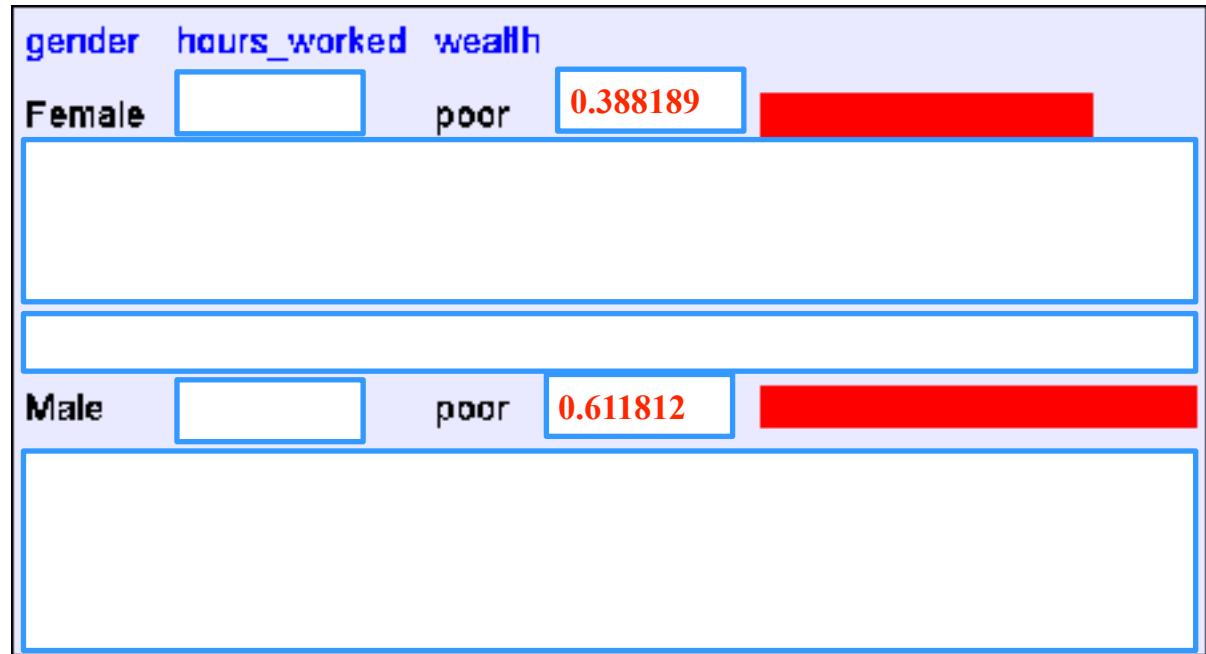
Inference with the Joint : Normalization trick

Sum Out the
Hidden (don't care)
variables



Inference with the Joint : Normalization trick

NORMALIZE the selection
(make it sum to one)



$$P(\text{Male} \mid \text{Poor}) = 0.612$$

Example:

- $P(W)$?
- $P(W \mid \text{winter})$?
- $P(W \mid \text{winter, hot})$?

S	T	W	P
summer	hot	sun	0.30
summer	hot	rain	0.05
summer	cold	sun	0.10
summer	cold	rain	0.05
winter	hot	sun	0.10
winter	hot	rain	0.05
winter	cold	sun	0.15
winter	cold	rain	0.20

How to produce joint distribution

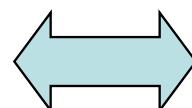
- Sometimes have conditional distributions but want the joint

$$P(y)P(x|y) = P(x, y)$$

- Example:

$P(W)$	
R	P
sun	0.8
rain	0.2

D	W	P
wet	sun	0.1
dry	sun	0.9
wet	rain	0.7
dry	rain	0.3



D	W	P
wet	sun	
dry	sun	
wet	rain	
dry	rain	

How to produce joint distribution

- more generally

$$\begin{aligned} P(X_1, X_2, \dots, X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_1, X_2)\dots \\ &= \prod_{i=1}^n P(X_i|X_1, \dots, X_{i-1}) \end{aligned}$$

Joint distributions

- Good news

Once you have a joint distribution, you can ask important questions about stuff that involves a lot of uncertainty

- Bad news

Impossible to create for more than about ten attributes because there are so many numbers needed.

Using fewer numbers

- Suppose there are two events:
 - ◆ M: Manuela teaches the class (otherwise it's Andrew)
 - ◆ S: It is sunny
- The joint p.d.f. for these events contain four entries.

Independence

“The sunshine levels do not depend on and do not influence who is teaching.”

This can be specified very simply:

$$P(S \mid M) = P(S)$$

This is a powerful statement!

It required extra domain knowledge. A different kind of knowledge than numerical probabilities. It needed an understanding of causation.

Independence

- A and B are independent properties

$$\Pr(B|A) = \Pr(B)$$

- A and B are dependent.

$$\Pr(B|A) \neq \Pr(B)$$

Independence

- From $P(S \mid M) = P(S)$, the rules of probability imply
 - $P(\sim S \mid M) = P(\sim S)$
 - $P(M \mid S) = P(M)$
 - $P(M \wedge S) = P(M) P(S)$
 - $P(\sim M \wedge S) = P(\sim M) P(S)$,
 - $P(M \wedge \sim S) = P(M) P(\sim S)$,
 - $P(\sim M \wedge \sim S) = P(\sim M) P(\sim S)$

chain rule : $P(A \wedge B) = P(A|B) P(B)$

Independence

We've stated:

$$P(M) = 0.6$$

$$P(S) = 0.3$$

$$P(S | M) = P(S)$$

From these 2 statements, we can derive the full joint pdf.

M	S	Prob
T	T	0.18
T	F	0.42
F	T	0.12
F	F	0.28

And since we now have the joint pdf, we can make any queries we like using the bayes theorem.

Independence (2)

- ◆ M : Manuela teaches the class
- ◆ S : It is sunny
- ◆ L : The lecturer arrives slightly late.

Assume both lecturers are sometimes delayed by bad weather. Andrew is more likely to arrive late than Manuela.

Independence (2)

- ◆ M : Manuela teaches the class
- ◆ S : It is sunny
- ◆ L : The lecturer arrives slightly late.

Assume both lecturers are sometimes delayed by bad weather.
Andrew is more likely to arrive late than Manuela.

$$P(S \mid M) = P(S), \quad P(S) = 0.3, \quad P(M) = 0.6$$

Lateness is not independent of the weather and is not
independent of the lecturer.

Independence (2)

- ◆ M : Manuela teaches the class
- ◆ S : It is sunny
- ◆ L : The lecturer arrives slightly late.

Assume both lecturers are sometimes delayed by bad weather.
Andrew is more likely to arrive late than Manuela.

Let's begin with writing down knowledge we're happy about:

$$P(S \mid M) = P(S), \quad P(S) = 0.3, \quad P(M) = 0.6$$

Lateness is not independent of the weather and is not
independent of the lecturer.

We already know the Joint of S and M, so all we need now is

$$P(L \mid S=u, M=v)$$

in the 4 cases of u/v = True/False.

Independence (2)

- M : Manuela teaches the class
- S : It is sunny
- L : The lecturer arrives slightly late.

Assume both lecturers are sometimes delayed by bad weather.
Andrew is more likely to arrive late than Manuela.

$$\begin{aligned}P(S \mid M) &= P(S) \\P(S) &= 0.3 \\P(M) &= 0.6\end{aligned}$$

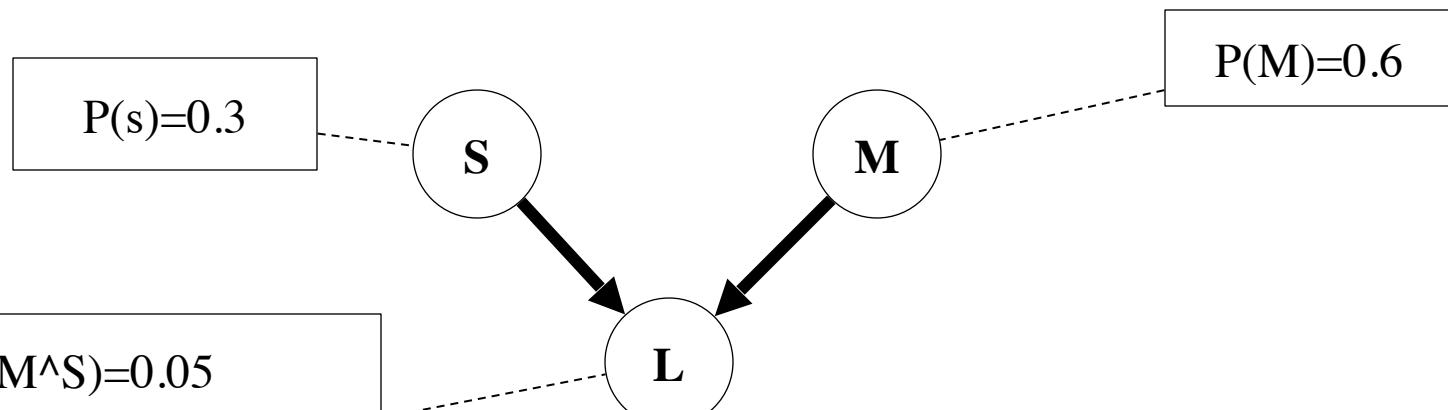
$$\begin{array}{l|l}P(L \mid M \wedge S) = 0.05 \\P(L \mid M \wedge \sim S) = 0.1 \\P(L \mid \sim M \wedge S) = 0.1 \\P(L \mid \sim M \wedge \sim S) = 0.2\end{array}$$

Now we can derive a full joint p.d.f.
with a “mere” six numbers instead
of eight

Independence (2)

$$\begin{aligned} P(S \mid M) &= P(S) \\ P(S) &= 0.3 \\ P(M) &= 0.6 \end{aligned}$$

$$\begin{aligned} P(L \mid M \wedge S) &= 0.05 \\ P(L \mid M \wedge \sim S) &= 0.1 \\ P(L \mid \sim M \wedge S) &= 0.1 \\ P(L \mid \sim M \wedge \sim S) &= 0.2 \end{aligned}$$



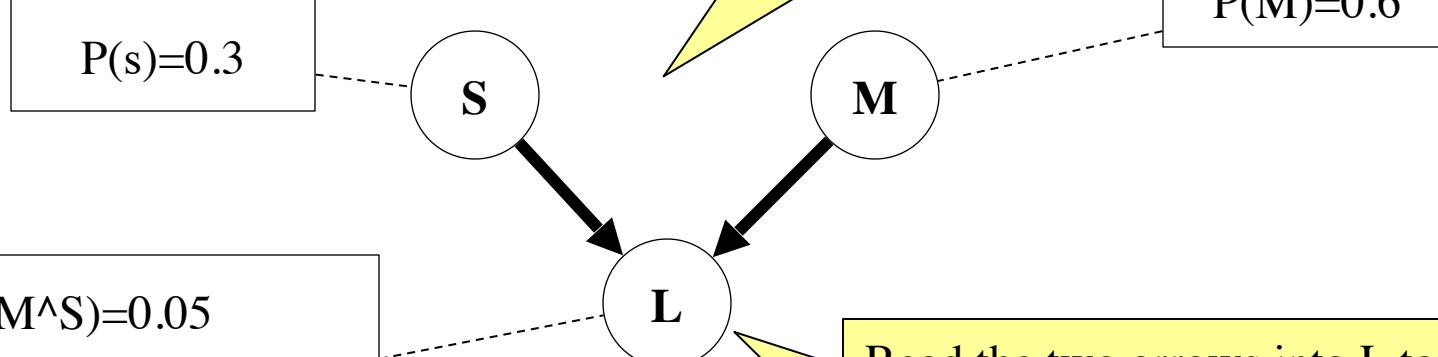
$$\begin{aligned} P(L \mid M \wedge S) &= 0.05 \\ P(L \mid M \wedge \sim S) &= 0.1 \\ P(L \mid \sim M \wedge S) &= 0.1 \\ P(L \mid \sim M \wedge \sim S) &= 0.2 \end{aligned}$$

Independence (2)

$$\begin{aligned} P(S \mid M) &= P(S) \\ P(S) &= 0.3 \\ P(M) &= 0.6 \end{aligned}$$

$$\begin{array}{l|l} P(L \mid M \wedge S) = 0.05 \\ P(L \mid M \wedge \sim S) = 0.1 \\ P(L \mid \sim M \wedge S) = 0.1 \\ P(L \mid \sim M \wedge \sim S) = 0.2 \end{array}$$

Read the absence of an arrow between S and M to mean “it would not help me predict M if I knew the value of S”



$$\begin{aligned} P(L \mid M \wedge S) &= 0.05 \\ P(L \mid M \wedge \sim S) &= 0.1 \\ P(L \mid \sim M \wedge S) &= 0.1 \\ P(L \mid \sim M \wedge \sim S) &= 0.2 \end{aligned}$$

Read the two arrows into L to mean that if I want to know the value of L it may help me to know M and to know S.

Conditional independence

Suppose we have these three events:

- M : Lecture taught by Manuela (otherwise Andrew)
- L : Lecturer arrives late
- R : Lecture concerns robots

Suppose:

- Andrew has a higher chance of being late than Manuela.
- Andrew has a higher chance of giving robotics lectures.

What kind of independence can we find?

L depends M, R depends M L depends R

How about:

- ◆ $P(L \mid M) = P(L)$? false
- ◆ $P(R \mid M) = P(R)$? false
- ◆ $P(L \mid R) = P(L)$? false

Conditional independence

Once you know who the lecturer is, then whether they arrive late doesn't affect whether the lecture concerns robots.

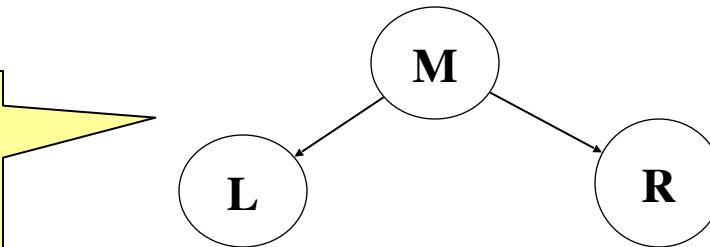
$$P(R \mid M, L) = P(R \mid M) \text{ and}$$

$$P(R \mid \sim M, L) = P(R \mid \sim M)$$

We express this in the following way:

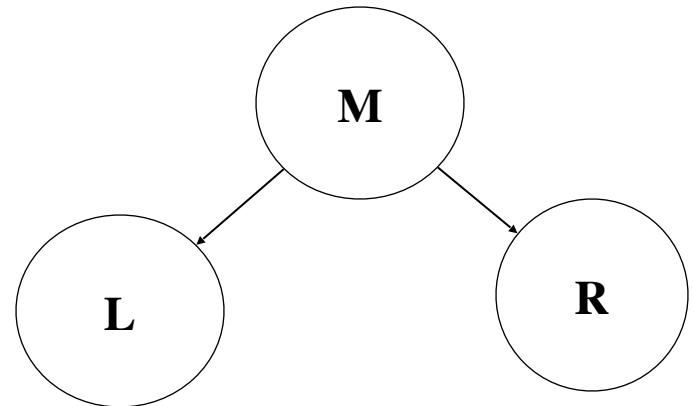
“R and L are conditionally independent given M”

..which is also notated by
the following diagram.



Given knowledge of M,
knowing anything else in the
diagram won't help us with
L, etc.

Conditional independence



We can write down $P(M)$. And then, since we know L is only directly influenced by M , we can write down the values of $P(L \mid M)$ and $P(L \mid \sim M)$ and know we've fully specified L 's behavior. Ditto for R .

$$P(M) = 0.6$$

$$P(L \mid M) = 0.085$$

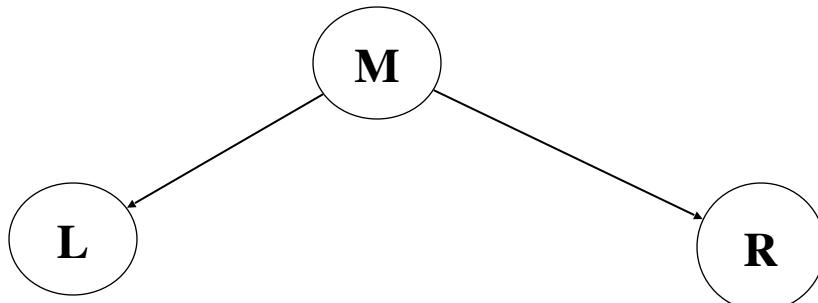
$$P(L \mid \sim M) = 0.17$$

$$P(R \mid M) = 0.3$$

$$P(R \mid \sim M) = 0.6$$

‘ R and L conditionally independent given M ’

Conditional independence



$$P(M) = 0.6$$

$$P(L \mid M) = 0.085$$

$$P(L \mid \sim M) = 0.17$$

$$P(R \mid M) = 0.3$$

$$P(R \mid \sim M) = 0.6$$

Conditional Independence:

$$P(R \mid M, L) = P(R \mid M),$$

$$P(R \mid \sim M, L) = P(R \mid \sim M)$$

Again, we can obtain any member of the Joint prob dist that we desire:

$$P(R \wedge L \wedge M) =$$

Assume five variables

T: The lecture started by 10:35

L: The lecturer arrives late

R: The lecture concerns robots

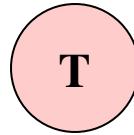
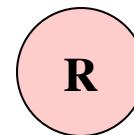
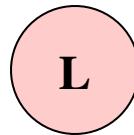
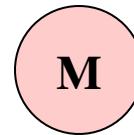
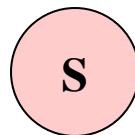
M: The lecturer is Manuela

S: It is sunny

- T only directly influenced by L (i.e. T is conditionally independent of R,M,S given L)
- L only directly influenced by M and S (i.e. L is conditionally independent of R given M & S)
- R only directly influenced by M (i.e. R is conditionally independent of L,S, given M)
- M and S are independent

Making a Bayes net

T: The lecture started by 10:35
L: The lecturer arrives late
R: The lecture concerns robots
M: The lecturer is Manuela
S: It is sunny

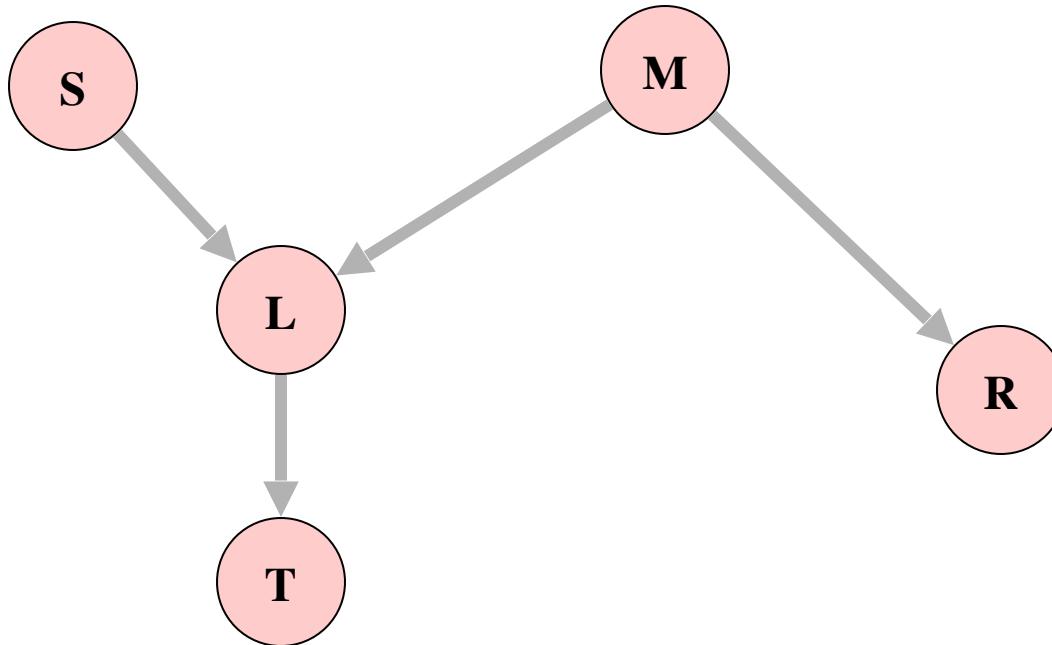


Step One: add variables.

- Just choose the variables you'd like to be included in the net.

Making a Bayes net

T: The lecture started by 10:35
L: The lecturer arrives late
R: The lecture concerns robots
M: The lecturer is Manuela
S: It is sunny

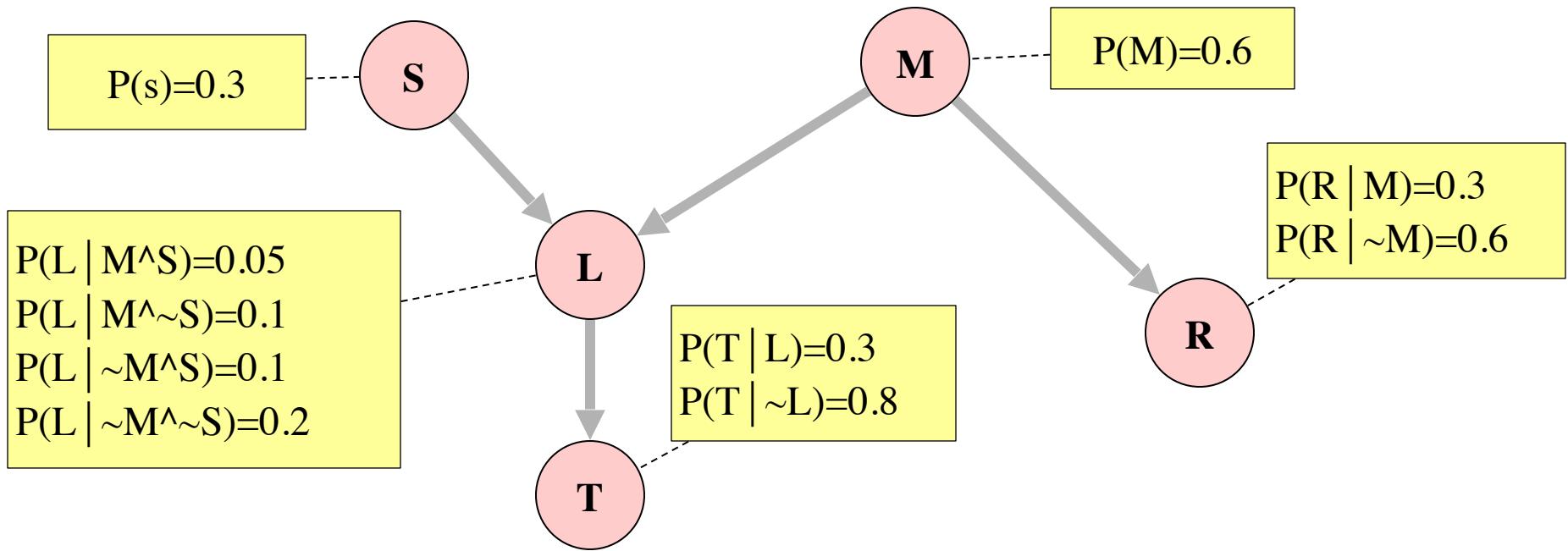


Step Two: add links.

- The link structure must be acyclic.
- If node X is given parents Q_1, Q_2, \dots, Q_n you are promising that any variable that's a non-descendent of X is conditionally independent of X given $\{Q_1, Q_2, \dots, Q_n\}$

Making a Bayes net

T: The lecture started by 10:35
L: The lecturer arrives late
R: The lecture concerns robots
M: The lecturer is Manuela
S: It is sunny

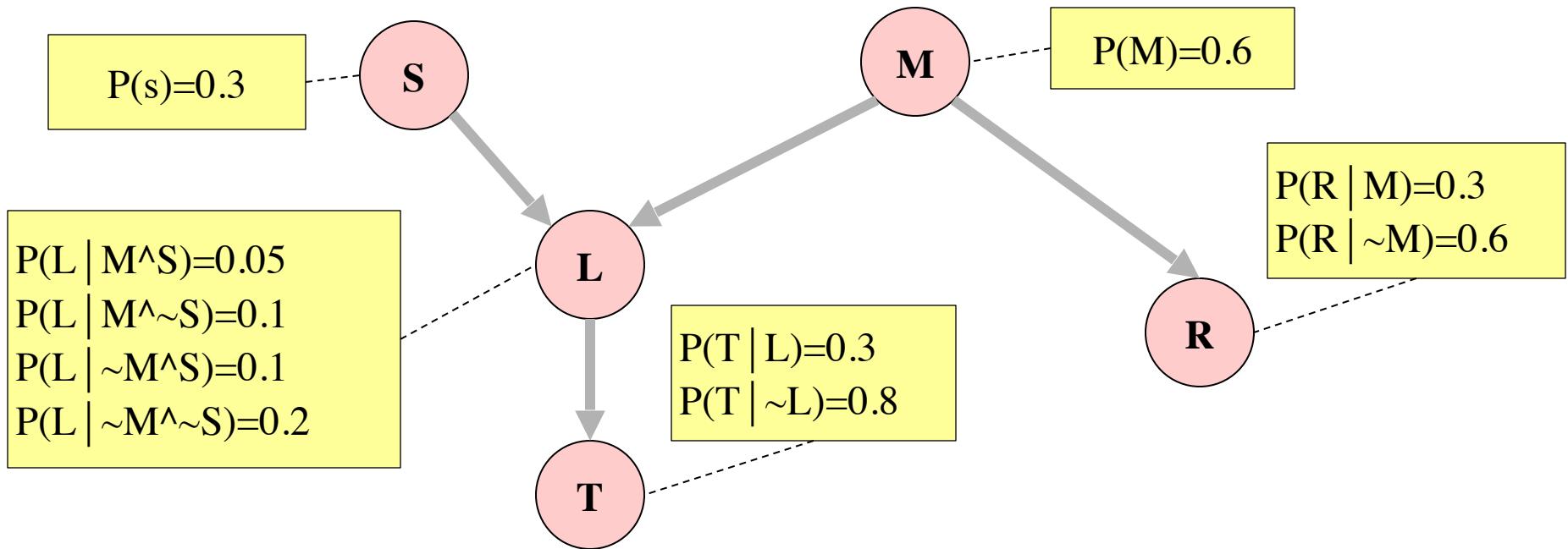


Step Three: add a probability table for each node.

- The table for node X must list $P(X|\text{Parent Values})$ for each possible combination of parent values

Making a Bayes net

T: The lecture started by 10:35
L: The lecturer arrives late
R: The lecture concerns robots
M: The lecturer is Manuela
S: It is sunny



- Each node is conditionally independent of all non-descendants in the tree, given its parents.

Bayesian network

- data structure that represents dependencies among random variables
 - ◆ directed graph
 - ◆ each node represents a random variable
 - ◆ arrow from X to Y means X is a parent of Y
 - ◆ each node X has probability distribution $P(X|\text{Parents}(X))$

Bayes Nets Formalized

- data structure that represents dependencies among random variables
- A Bayes net (also called a belief network) is an augmented directed acyclic graph, represented by the pair V, E where:
 - ◆ V is a set of vertices.
 - ◆ E is a set of directed edges joining vertices. No loops of any length are allowed. Edge from X to Y means X is a parent of Y
- Each vertex in V contains the following information:
 - ◆ The name of a random variable
 - ◆ A probability distribution table indicating how the probability of this variable's values depends on all possible combinations of parental values.
 - ◆ $P(V|Parents(V))$

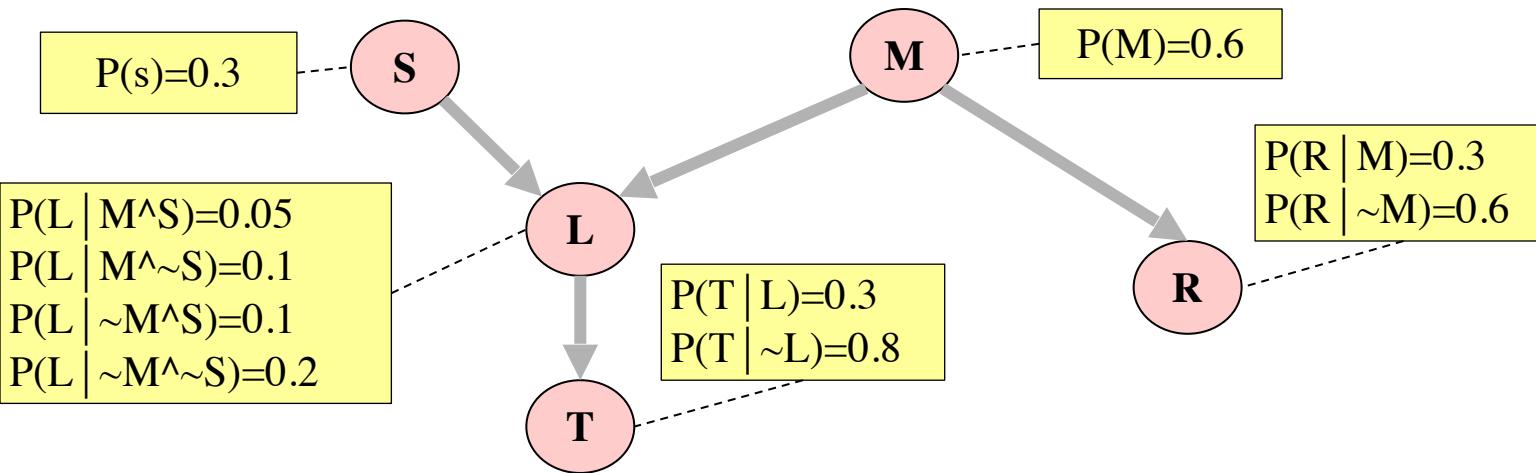
Building a Bayes Net

1. Choose a set of relevant variables.
2. Choose an ordering for them
3. Assume they're called $X_1 \dots X_m$ (where X_1 is the first in the ordering, X_2 is the second, etc)
4. For $i = 1$ to m :
 1. Add the X_i node to the network
 2. Set $Parents(X_i)$ to be a minimal subset of $\{X_1 \dots X_{i-1}\}$ such that we have conditional independence of X_i and all other members of $\{X_1 \dots X_{i-1}\}$ given $Parents(X_i)$
 3. Define the probability table of $P(X_i = k \mid \text{Assignments of } Parents(X_i))$.

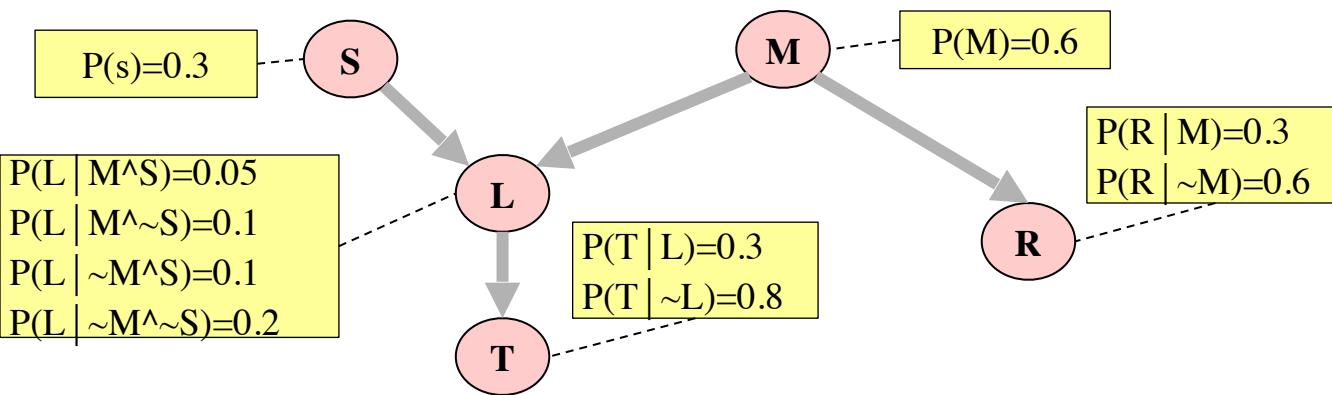
Computing a Joint Entry

How to compute an entry in a joint distribution?

E.G: What is $P(S \wedge \sim M \wedge L \wedge \sim R \wedge T)$?



Computing with Bayes Net



$$P(T \wedge \neg R \wedge L \wedge \neg M \wedge S) =$$

$$P(T | \neg R \wedge L \wedge \neg M \wedge S) * P(\neg R \wedge L \wedge \neg M \wedge S) =$$

$$P(T | L) * P(\neg R \wedge L \wedge \neg M \wedge S) =$$

$$P(T | L) * P(\neg R | L \wedge \neg M \wedge S) * P(L \wedge \neg M \wedge S) =$$

$$P(T | L) * P(\neg R | \neg M) * P(L \wedge \neg M \wedge S) =$$

$$P(T | L) * P(\neg R | \neg M) * P(L | \neg M \wedge S) * P(\neg M \wedge S) =$$

$$P(T | L) * P(\neg R | \neg M) * P(L | \neg M \wedge S) * P(\neg M | S) * P(S) =$$

$$P(T | L) * P(\neg R | \neg M) * P(L | \neg M \wedge S) * P(\neg M) * P(S).$$

The general case

$$P(X_1=x_1 \wedge X_2=x_2 \wedge \dots \wedge X_{n-1}=x_{n-1} \wedge X_n=x_n) =$$

$$P(X_n=x_n \wedge X_{n-1}=x_{n-1} \wedge \dots \wedge X_2=x_2 \wedge X_1=x_1) =$$

$$P(X_n=x_n \mid X_{n-1}=x_{n-1} \wedge \dots \wedge X_2=x_2 \wedge X_1=x_1) * P(X_{n-1}=x_{n-1} \wedge \dots \wedge X_2=x_2 \wedge X_1=x_1) =$$

$$P(X_n=x_n \mid X_{n-1}=x_{n-1} \wedge \dots \wedge X_2=x_2 \wedge X_1=x_1) * P(X_{n-1}=x_{n-1} \mid \dots \wedge X_2=x_2 \wedge X_1=x_1) *$$

$$P(X_{n-2}=x_{n-2} \wedge \dots \wedge X_2=x_2 \wedge X_1=x_1) =$$

:

:

$$\prod_{i=1}^n P((X_i = x_i) \mid ((X_{i-1} = x_{i-1}) \wedge \dots \wedge (X_1 = x_1)))$$

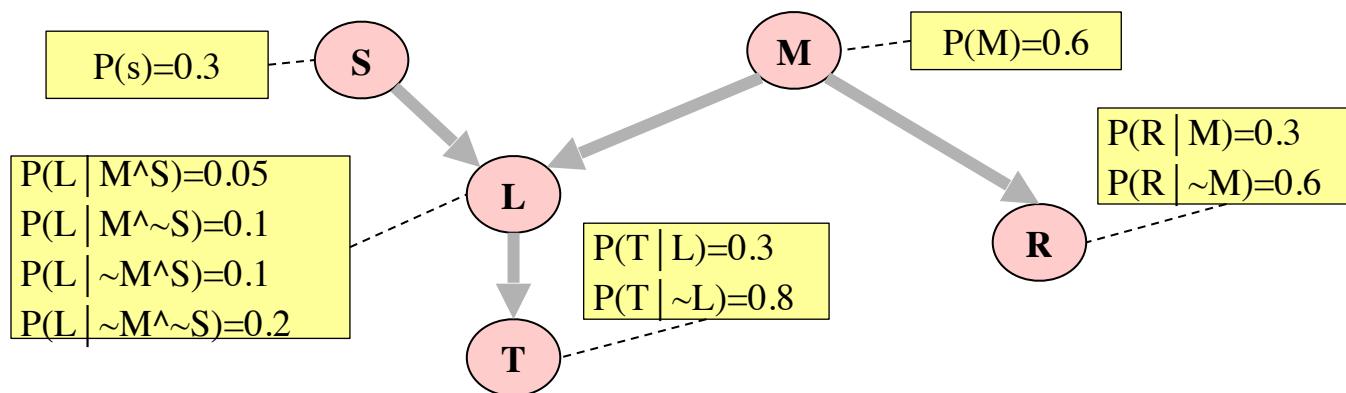
=

$$\prod_{i=1}^n P((X_i = x_i) \text{Assignment of Parents}(X_i))$$

So any entry in joint pdf table can be computed. And so **any conditional probability** can be computed.

Where are we now?

- We have a methodology for building Bayes nets.
- We don't require exponential storage to hold our probability table. Only exponential in the maximum number of parents of any node.
- We can compute probabilities of any given assignment of truth values to the variables. And we can do it in time linear with the number of nodes.
- So we can also compute answers to any questions.



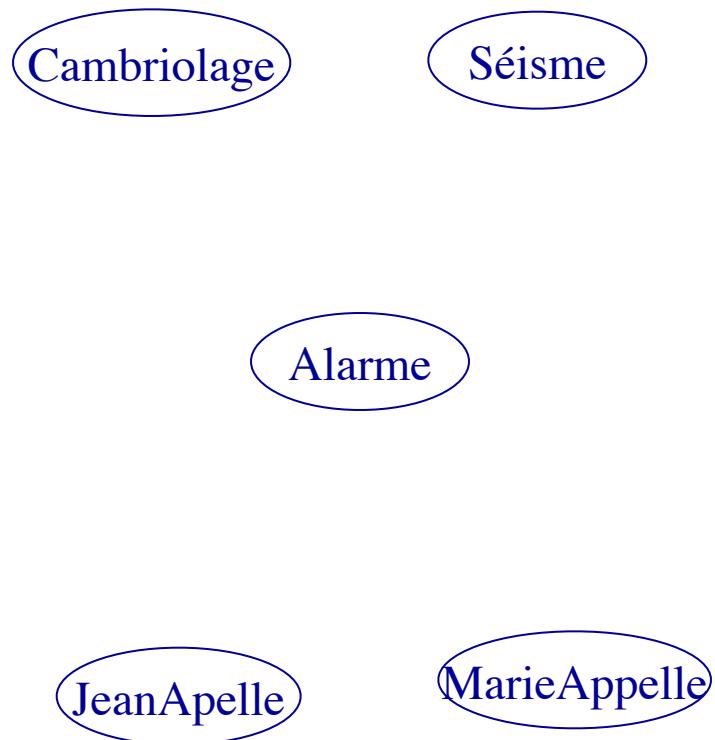
Exemple

- Considérons la situation suivante :
 - ◆ Je suis au travail. Mes voisins Marie et Jean m'ont promis de m'appeler chaque fois que mon alarme sonne.
 - ◆ Parfois mon alarme se met à sonner lorsqu'il y a de légers séismes.
 - ◆ Mon voisin Jean m'appelle pour me dire que mon alarme sonne.
 - » Parfois il confond l'alarme avec la sonnerie du téléphone.
 - ◆ Par contre ma voisine Marie ne m'appelle pas.
 - » Parfois elle met la musique trop fort.
 - ◆ Dois-je conclure qu'il y a un cambriolage chez moi ?
- On peut représenter cette situation par un RB.

Exemple

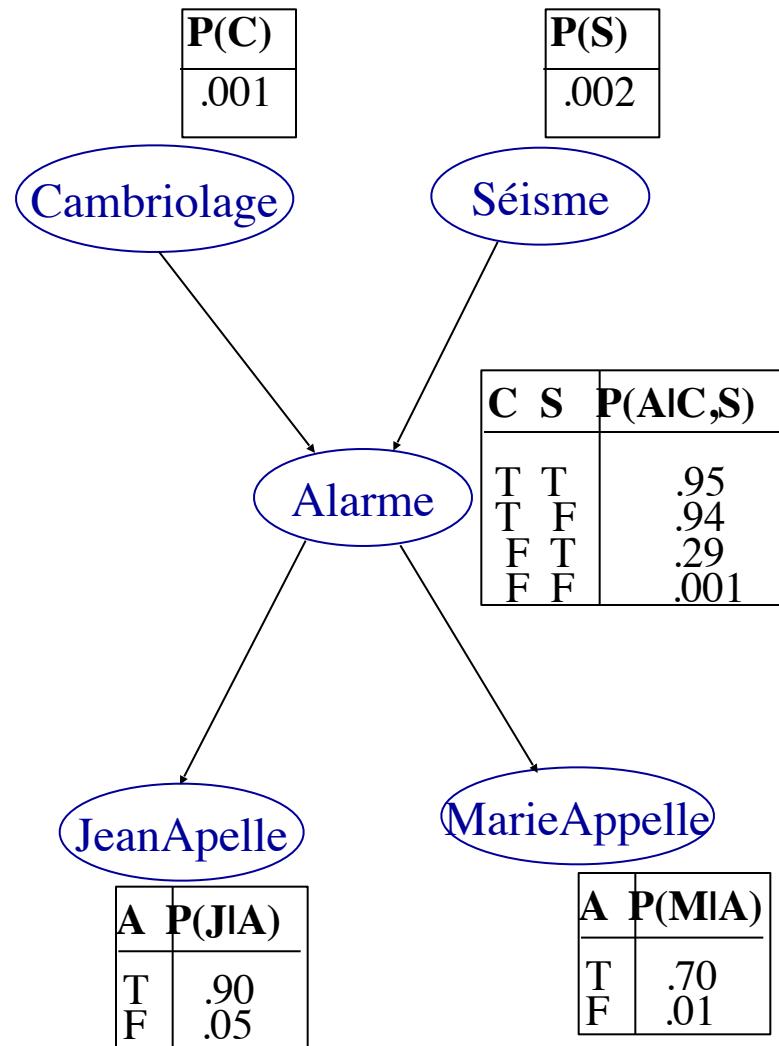
- Variables aléatoires :

- ◆ *Cambriolage*
- ◆ *Séisme*
- ◆ *Alarme*
- ◆ *Jean appelle*
- ◆ *Marie appelle.*



Exemple

- La topologie du RB modélise la connaissance causale.
- Un arc d'un nœud X vers un nœud Y signifie que la variable X influence la variable Y .
 - Un cambriolage peut déclencher l'alarme.
 - Un séisme aussi.
 - L'alarme peut inciter Jean à appeler.
 - Idem pour Marie à appeler.
- Pour chaque nœud, une table de probabilité conditionnelle (TPC) donne la probabilité pour chaque valeur du nœud étant donné les combinaisons des valeurs des parents du nœud.



Exemple

- $P(X_1, \dots, X_n) =$

$$\pi_{i=1}^n P(X_i | \text{Parents}(X_i))$$

- $P(j, m, a, \neg c, \neg s)$

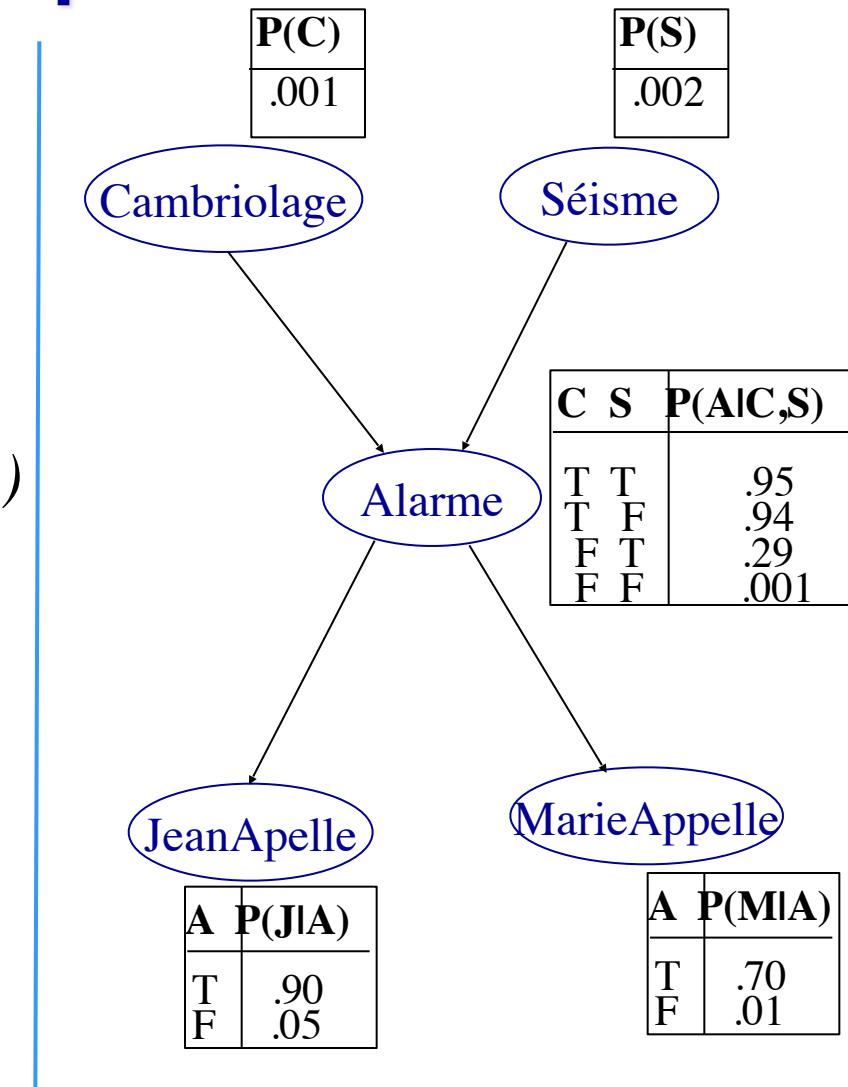
- $= P(j|a) P(m|a) P(a| \neg c, \neg s)$

$$P(\neg c) P(\neg s)$$

- $= .90 \times .70 \times .001 \times$

- $.999 \times .998$

- $= .00062$



Exemple

$$P(\neg c, a) = \sum_m \sum_j \sum_s P(J=j, M=m, a, \neg c, S=s)$$

$$= \sum_m \sum_s P(j|a) P(m|a) P(a|\neg c, s) P(\neg c) P(s)$$

$$= \sum_s \sum_m P(j|a) P(m|a) P(a|\neg c, s) P(\neg c) P(s)$$

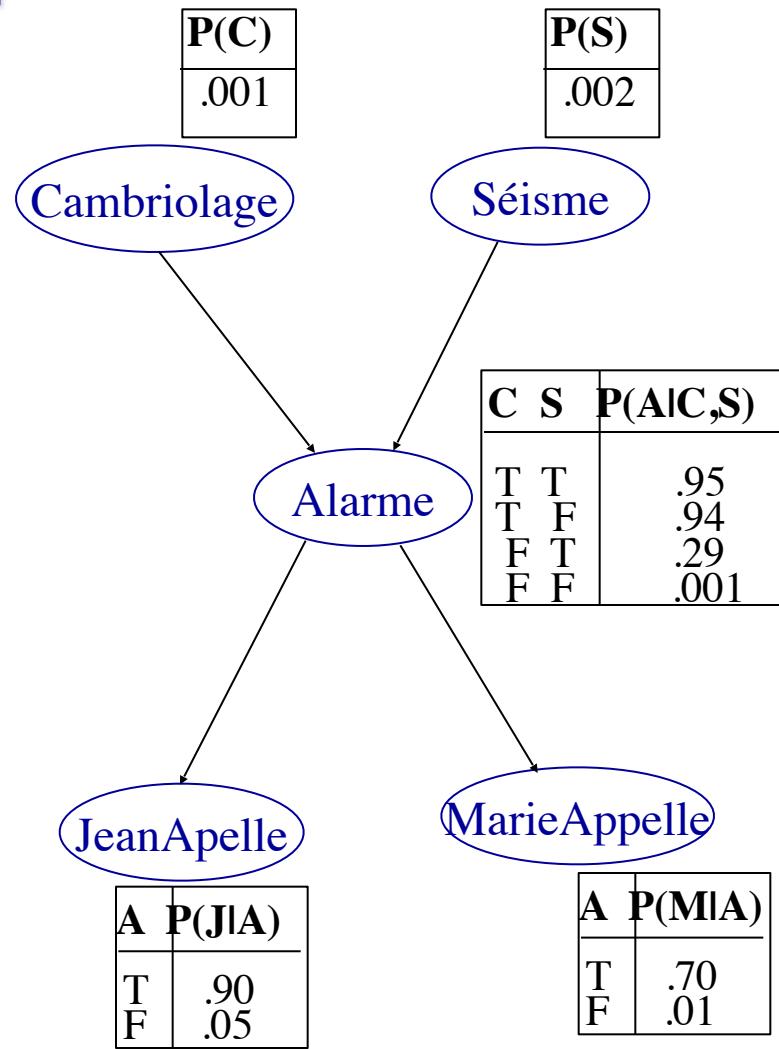
$$= \sum_s \sum_j P(j|a) P(a|\neg c, s) P(\neg c) P(s) \underbrace{\sum_m P(m|a)}_{=1}$$

$$= \sum_s P(a|\neg c, s) P(\neg c) P(s) \underbrace{\sum_j P(j|a)}_{=1}$$

$$= P(a|\neg c, s) P(\neg c) P(s) + P(a|\neg c, \neg s) P(\neg c) P(\neg s)$$

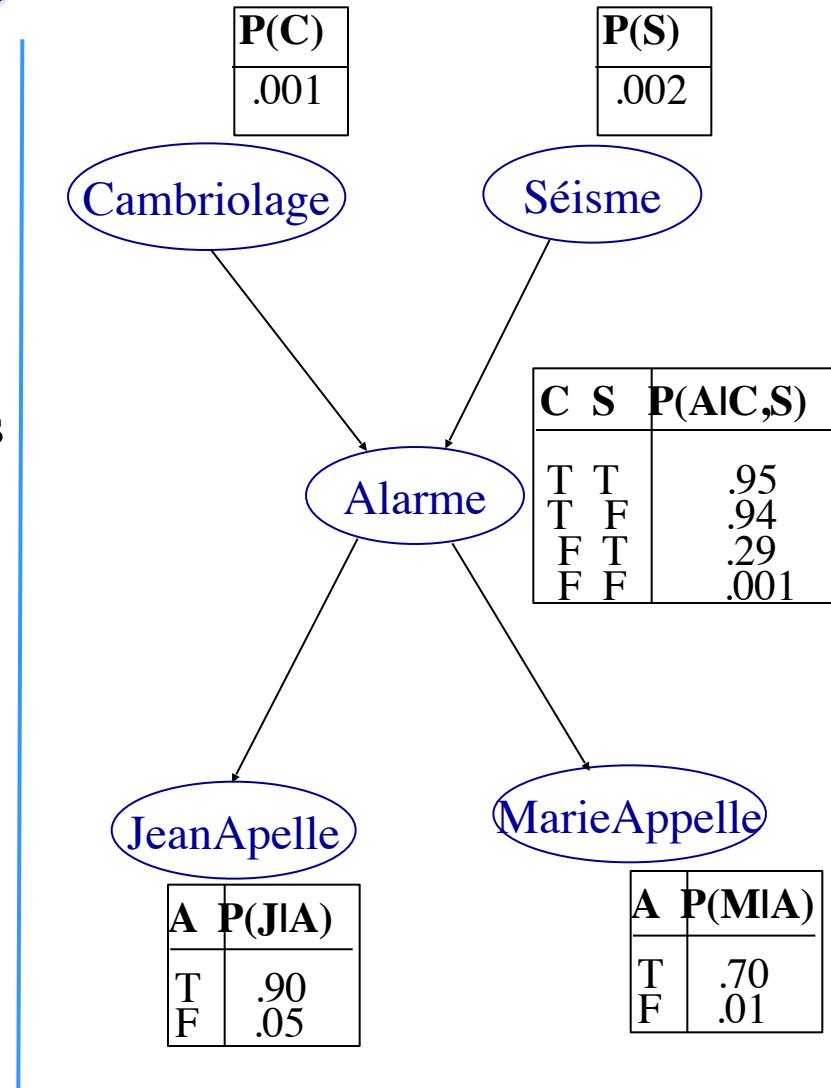
$$= .29 * .999 * .002 + .001 * .999 * .998$$

$$\approx 0.0016$$



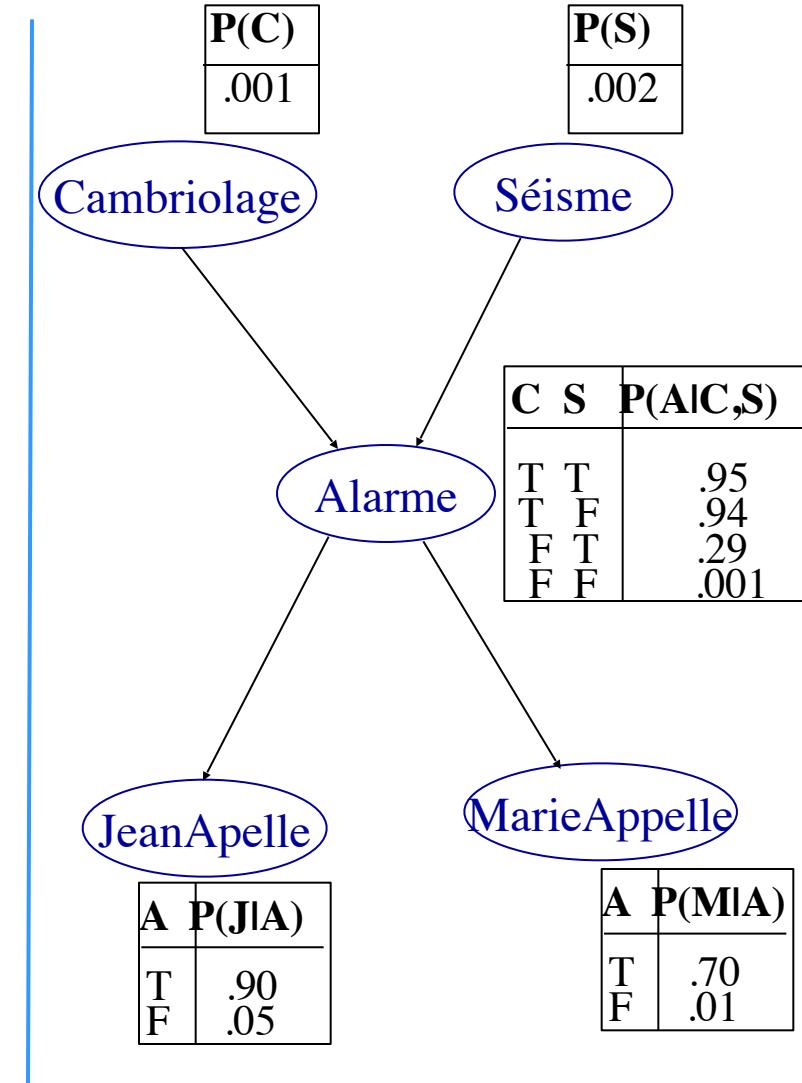
Types d'interrogations d'un RB

- **Prédiction** (étant donné les causes, quels sont les effets)
 - ◆ $P(\text{JeanAppelle}|\text{Cambriolage}=T)$
 - ◆ Garder à l'esprit qu'on a des arcs « causes → effets ».
- **Diagnostique** (on connaît les effets, on cherche les causes)
 - ◆ $P(\text{Cambriolage}|\text{JeanAppelle}=T)$.
- Probabilité jointe
 - ◆ $P(\text{Alarme})$



Exemple

- $P(\text{Cambriolage} \mid \text{JeanApelle} = T, \text{MarieAppelle} = T)$
- Noté $P(C \mid j, m)$



Exemple

- $P(Cambrilage \mid JeanAppelle = vrai, MarieAppelle = vrai)$

◆ noté $P(C \mid j, m)$

- Les variables cachées sont *Séisme* et *Alarme*

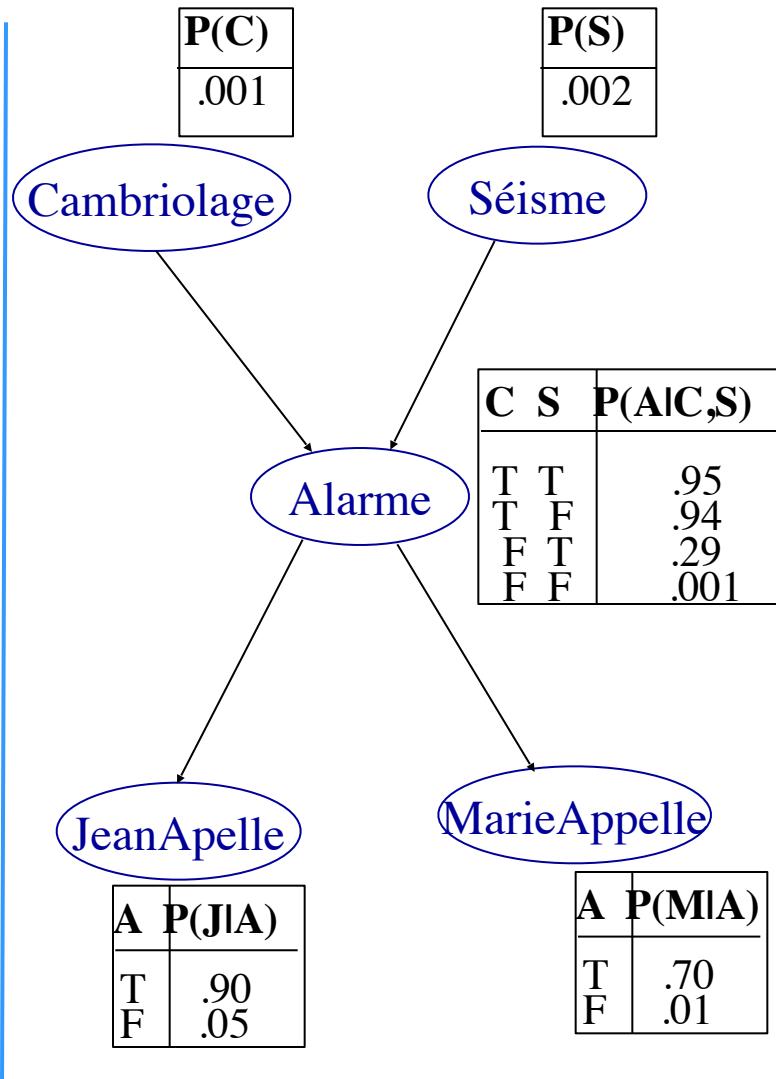
$$\begin{aligned} P(C \mid j, m) &= \alpha P(C, j, m) \\ &= \alpha \sum_s \sum_a P(C, s, a, j, m) \end{aligned}$$

$$= P(C) \sum_s \sum_a P(s) P(a \mid C, s) P(j \mid a) P(m \mid a)$$

$$= P(C) \sum_s P(s) \sum_a P(a \mid C, s) P(j \mid a) P(m \mid a)$$

Note :

- ◆ s et a prennent toutes les valeurs possibles pour $S=s$ et $A=a$
- ◆ ne pas confondre avec j et m qui sont des observations fixes ($J=vrai$ et $M=vrai$)



Exemple

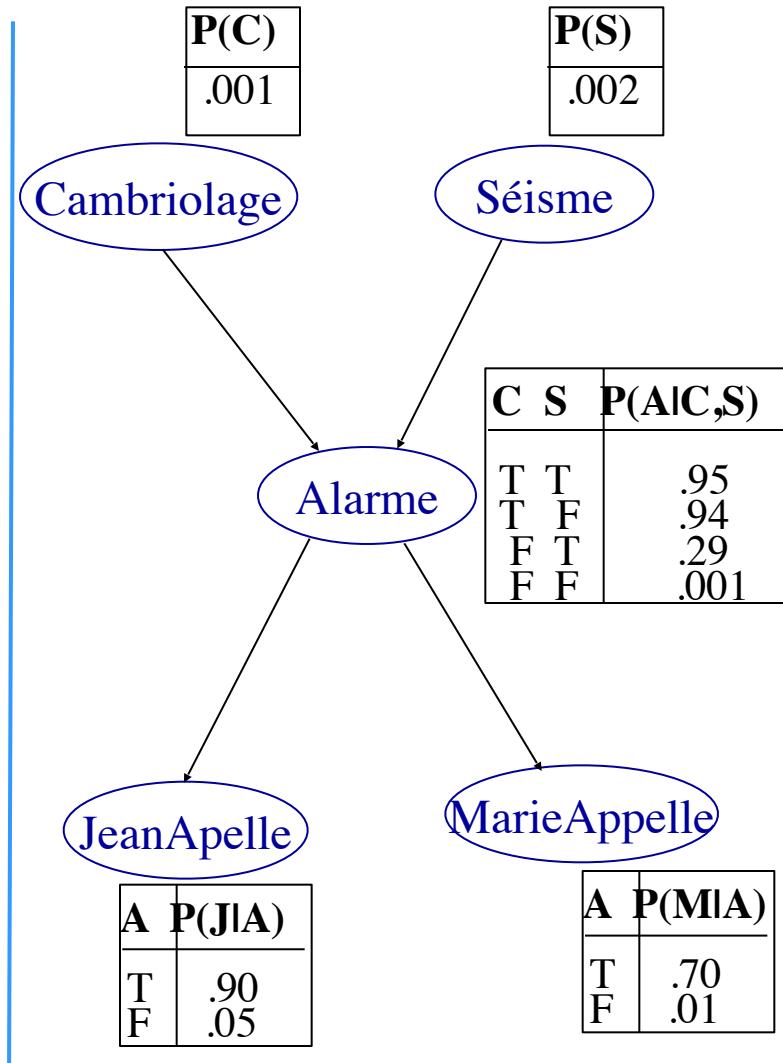
- On calcule pour $C = \text{vrai}$

$$\begin{aligned}
 P(c | j, m) &= \alpha P(c) \sum_s P(s) \sum_a P(a | c, s) P(j | a) P(m | a) \\
 &= \alpha * 0.001 * (0.002 * (0.95 * 0.90 * 0.70 + \\
 &\quad 0.05 * 0.05 * 0.01) + \\
 &\quad 0.998 * (0.94 * 0.90 * 0.70 + \\
 &\quad 0.06 * 0.05 * 0.01)) \\
 &= \alpha * 0.00059224
 \end{aligned}$$

- Et $C = \text{faux}$

$$\begin{aligned}
 P(\neg c | j, m) &= \alpha P(\neg c) \sum_s P(s) \sum_a P(a | \neg c, s) P(j | a) P(m | a) \\
 &= \alpha * 0.0014919 \\
 \alpha &= 1 / (0.00059224 + 0.0014919)
 \end{aligned}$$

- Donc, $P(C | j, m) = [0.284, 0.716]$



What you should know

- The meanings and importance of independence and conditional independence.
- The definition of a Bayes net.
- Computing probabilities of assignments of variables (i.e. members of the joint p.d.f.) with a Bayes net.
- The slow (exponential) method for computing arbitrary, conditional probabilities.