Informatisation du droit

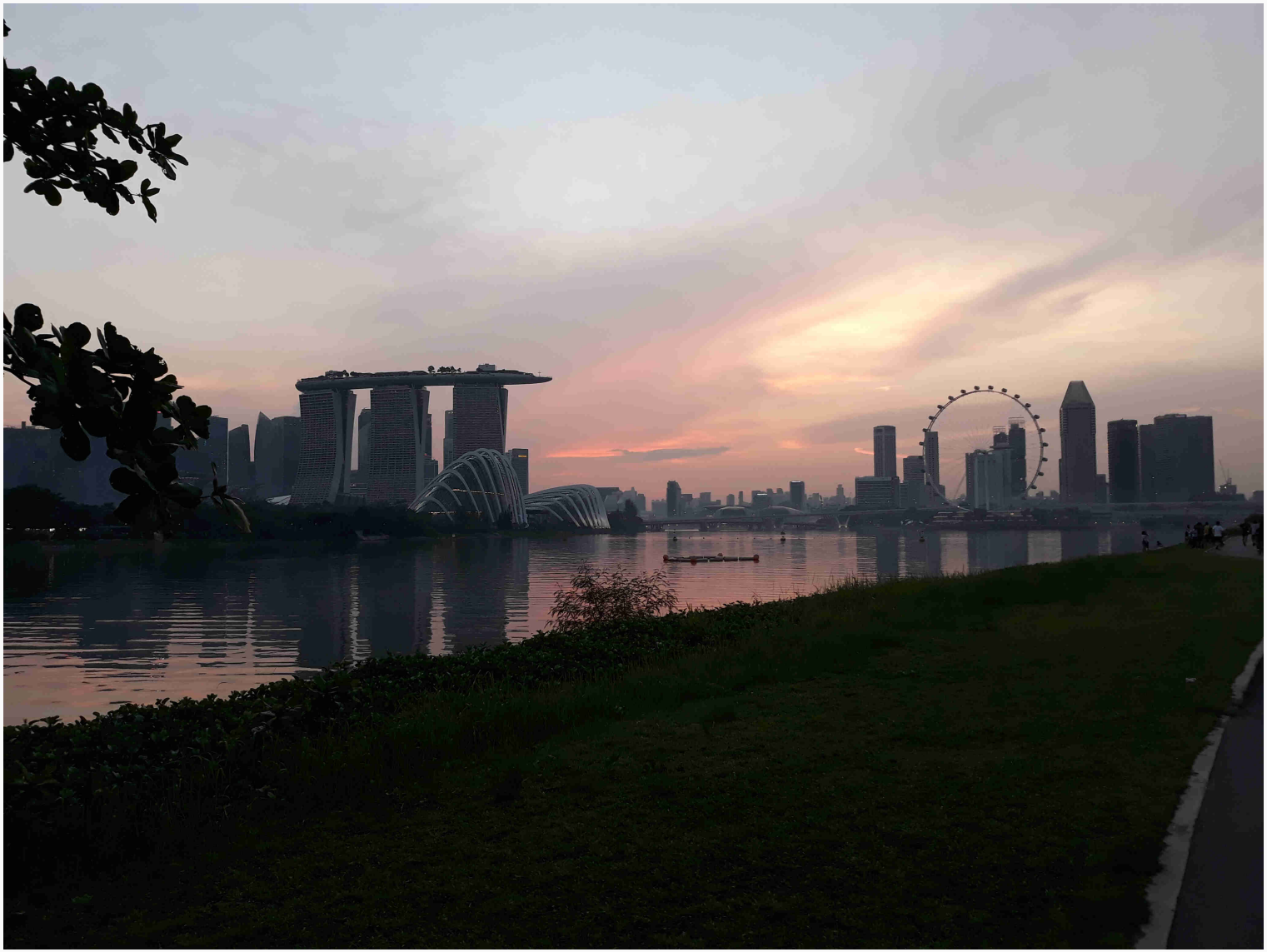
Martin Strecker

Mathématiques et Philosophie Contemporaines XII, 26 juin 2025

Université de Toulouse

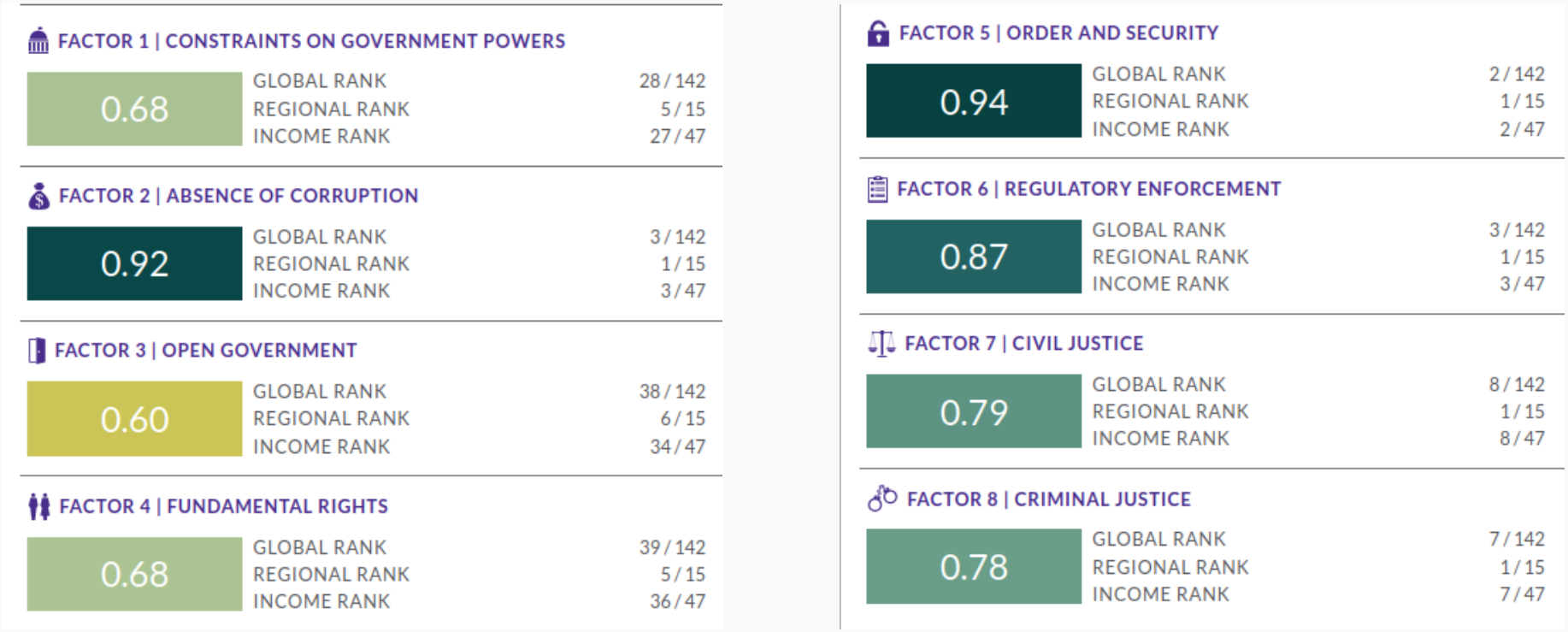
Plan

Singapour



Système de justice

rigoureux, efficace, bien organisé, voir World Justice Project

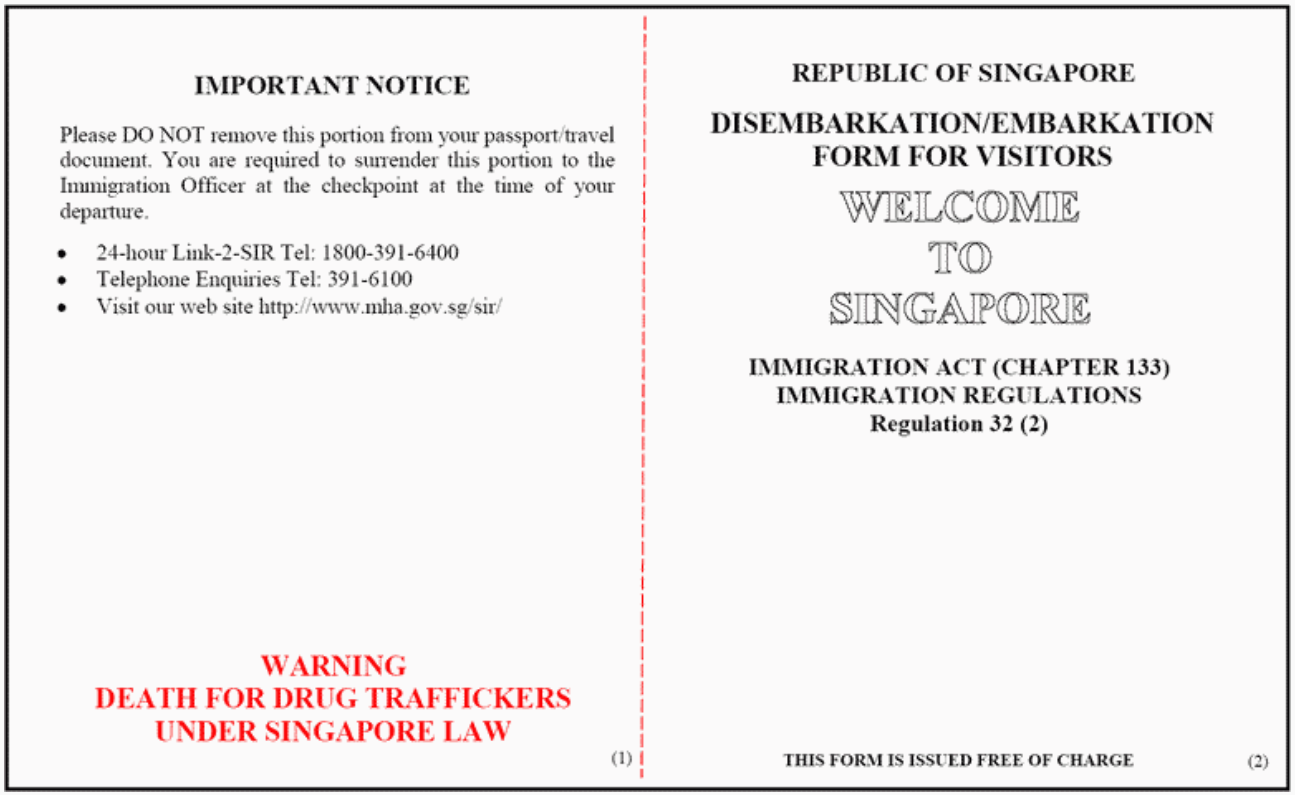


Système de justice

parfois un peu intrusif

Welcome

. . . avec des peines draconiennes



IMPORTANT NOTICE

Please DO NOT remove this portion from your passport/travel document. You are required to surrender this portion to the Immigration Officer at the checkpoint at the time of your departure.

- 24-hour Link-2-SIR Tel: 1800-391-6400
- Telephone Enquiries Tel: 391-6100
- Visit our web site http://www.mha.gov.sg/sir/

WARNING
DEATH FOR DRUG TRAFFICKERS
UNDER SINGAPORE LAW

(1)

REPUBLIC OF SINGAPORE

DISEMBARKATION/EMBARKATION FORM FOR VISITORS

WELCOME
TO
SINGAPORE

IMMIGRATION ACT (CHAPTER 133)
IMMIGRATION REGULATIONS
Regulation 32 (2)

THIS FORM IS ISSUED FREE OF CHARGE

(2)

WILLIAM GIBSON THE BIG STORY APR 1, 1993 12:00 PM

Welcome to 'Disneyland With the Death Penalty'

WIRED sent William Gibson to Singapore to see whether that clean dystopia represents our techno future.

Projet “Computational Law”

Contexte

- Projet de développement et de recherche (Centre for Computational Law)
- hébergé par la Singapore Management University
- financé à hauteur de 15M S$ ($\approx$ 10M €)
- sur une durée de 5 ans (fin juin 2025)

But

- Développer un DSL (*Domain Specific Language*)
- . . . pour formaliser la loi

Pourquoi un DSL ?

. . . à cause d'histoires comme : The commas that cost companies millions (BBC)

A dairy company in the US city of Portland, Maine settled a court case for $5m earlier this year because of a missing comma.

Three lorry drivers for Oakhurst Dairy claimed that they were owed years of unpaid overtime wages, all because of the way commas were used in legislation governing overtime payments.

The state's laws declared that overtime wasn't due for workers involved in "the canning, processing, preserving, freezing, drying, marketing, storing, packing for shipment or distribution of : 1) agricultural produce ; 2) meat and fish products ; and 3) perishable foods".

The drivers managed to successfully argue that because there was no comma after "shipment" and before "or distribution", they were owed overtime pay. If a comma had been there, the law would have explicitly ruled out those who distribute perishable foods.

Pourquoi pas de l'IA générative ?

BENJ EDWARDS, ARS TECHNICA BUSINESS APR 19, 2025 11:47 AM

An AI Customer Service Chatbot Made Up a Company Policy—and Created a Mess

When an AI model for code-editing company Cursor hallucinated a new rule, users revolted.

https://www.wired.com/story/cursor-ai-hallucination-policy-customer-service

. . . When the user contacted Cursor support, an agent named "Sam" told them it was expected behavior under a new policy. But no such policy existed, and Sam was a bot. The AI model made the policy up, sparking a wave of complaints and cancellation threats.

Plan

Plan

Règles si . . . alors

Des normes sont souvent énoncées en forme d'un ensemble de règles *si* . . . *alors*.

Exemple (hypothétique) du temps de Covid :

On est autorisé à utiliser le train

- si on est guéri : $g \longrightarrow ut$
- si on est vacciné : $v \longrightarrow ut$
- si on n'est pas testé positif : $\neg p \longrightarrow ut$

Une conclusion est établie sur la base d'une déduction classique :

- Si v et $v \longrightarrow ut$, on peut déduire ut

Règles et Prolog

Les premiers codages informatiques de lois se font dans le langage de programmation PROLOG (par exemple : Kowalski : *Legislation as logic programs*)

```
v.
ut :- g.
ut :- v.
ut :- not p.
```

L'évaluation se fait par *chaînage en arrière*. Essayons d'établir `ut` :

- avec la règle `ut :- g.`
  - essayons d'établir `g`. Échec.
- avec la règle `ut :- v.`
  - essayons d'établir `v`. Succès.

  Ceci montre `ut`.

Raisonnement par défaut et Negation as failure

L'absence d'information est interprétée comme confirmation du contraire :
Sans connaissance de `p`, la proposition `not p` est supposée vraie "par défaut".

Opérationnellement, `not p` produit un succès $\Leftrightarrow$ `p` produit un échec.

Exemple : Pour le programme

```
ut :- g.
ut :- v.
ut :- not p.
```

Essayons d'établir `ut` :

- avec la règle `ut :- g.` Échec.
- avec la règle `ut :- v.` Échec.
- avec la règle `ut :- not p.`
  - essayons d'établir `not p`.
    - essayons d'établir `p`. Échec.

    Ceci montre `not p`.

  Ceci montre `ut`.

Non-monotonie et autres effets

Le raisonnement par défaut produit une logique non monotone : augmenter l'ensemble des hypothèses n'augmente pas l'ensemble des conclusions.

Exemple précédent continué : Après rajout de `p`, on ne peut plus déduire `ut`.

Autres problèmes du raisonnement par défaut :

- Absence de sémantique déclarative
- Effets contraires à des équivalences / conséquences logiques traditionnelles : La formule $a \vee b$
  - écrite comme $\neg a \longrightarrow b$ permet de déduire b
  - écrite comme $\neg b \longrightarrow a$ permet de déduire a

Conclusion préliminaire :

- Ne pas confondre "règle" et "formule logique"
- Traiter des règles comme entités à part entière
- Est-ce qu'on a besoin d'une nouvelle logique ?

Inversion de règles : Idée

L'idée qui sous-tend le raisonnement par défaut est que les règles énoncées sont la seule manière de faire une inférence :

Exemple : On est autorisé à utiliser le train

- si on est guéri : $g \longrightarrow ut$
- si on est vacciné : $v \longrightarrow ut$
- si on n'est pas testé positif : $\neg p \longrightarrow ut$

et *uniquement sous ces conditions*.

L'inversion des règles permet ce raisonnement dans le cadre d'une logique classique (comparable à *Clark completion* en programmation logique).

Inversion de règles : Démarche

Soit $\mathcal{R}[P]$ l'ensemble des règles

$$\{\forall x_1, \ldots, x_n. Pre_1[P](x_1, \ldots x_n) \longrightarrow P(x_1, \ldots, x_n), \ldots,$$
$$\forall x_1, \ldots, x_n. Pre_k[P](x_1, \ldots x_n) \longrightarrow P(x_1, \ldots, x_n)\}$$

où les $Pre_i[P]$ contiennent P positivement.

La clôture inductive de $\mathcal{R}[P]$ est la formule de second ordre :

$$P^*(x_1, \ldots, x_n) = \forall P. \left(\bigwedge \mathcal{R}[P]\right) \longrightarrow P(x_1, \ldots x_n)$$

qui satisfait :

$$P^*(x_1, \ldots, x_n) \longrightarrow Pre_1[P^*](x_1, \ldots x_n) \vee \cdots \vee Pre_k[P^*](x_1, \ldots x_n)$$

approximée par Inv_p définie comme :

$$\forall x_1, \ldots, x_n.\ P(x_1, \ldots, x_n) \longrightarrow Pre_1[P](x_1, \ldots x_n) \vee \cdots \vee Pre_k[P](x_1, \ldots x_n)$$

Inversion de règles : Exemples

Exemple

- L'inversion de $\{g \longrightarrow ut, v \longrightarrow ut, \neg p \longrightarrow ut\}$ relative à ut est : $Inv_{ut} = ut \longrightarrow (g \vee v \vee \neg p)$
- L'inversion de $\{\}$ relative à p est : $Inv_p = p \longrightarrow \bot$ (disjonction vide), donc $Inv_p = \neg p$
- L'inversion de $\{p\}$ relative à p est : $Inv_p = p \longrightarrow \top$, donc $Inv_p = \top$

Utilisation A partir d'un ensemble de règles :

- Conjonction des règles et de l'inversion
- Vérification avec un SAT solveur (satisfiabilité)
- ... sans utiliser un moteur comme PROLOG

Plan

Structuration d'ensemble de règles

Un ensemble de règles peut mener à des conclusions contradictoires si on ne prend pas en compte leurs dépendances

- implicites : hiérarchie de lois (*non traité ici*)
- explicites : priorités déclarées par mots clés comme *despite*, *subject to*

Exemple : (voir détails)

> **(1)** A *data breach is a notifiable* data breach if the data breach results in . . . significant harm to an affected individual.
>
> **(4)** Despite subsections (1), (2) and (3), a data breach that relates to the unauthorised access . . . of personal data only within an organisation *is deemed not to be a notifiable data breach*.

Lecture : On peut tirer la conclusion *notifiable data breach* de (1) sauf si la condition de (4) est satisfaite.

Les modificateurs “despite” et “subject to”

despite rajoute la négation de la précondition de la règle *locale* à la règle *distante*
subject to rajoute la négation de la précond. de la règle *distante* à la règle *locale*

```
rule <db_notifiable_1>
  for b: Breach
  if significant_harm b then notifiable b

rule <db_notifiable_4>
   {restrict: {despite: db_notifiable_1}}
  for b: Breach
  if access_in_org b then not (notifiable b)
```

a pour effet :

```
rule <db_notifiable_1>
  for b: Breach
  if significant_harm b && not (access_in_org b)
  then notifiable b
```

Commpilation d'un ensemble de règles

Besoin partagé entre l'informatique et le droit :

- *Modularité* et *lisibilité*
- *Hiérarchisation*

Pourtant, une modification locale peut avoir un effet global
(par ex. modificateur `despite`)

On utilise un processus de compilation pour convertir un ensemble de règles en une règle qui rassemble les informations locales.

Opérateurs sur des règles

Les transformations se font avec des opérateurs de transformation comme :

- normalisation d'une règle
  - $(\exists x.P(x)) \Longrightarrow Q(x) \rightsquigarrow (\forall x.P(x) \Longrightarrow Q(x))$
  - $P \Longrightarrow Q(c) \rightsquigarrow (\forall x.P \wedge x = c \Longrightarrow Q(x))$
- élimination de *despite* / *subject to* d'un ensemble de règles
  (tri topologique des règles, modification des préconditions par ordre croissant des dépendances)
- inversion de règles

Plan

Raisonner avec et sur des règles

Exemple : vitesse maximale d'une voiture

```
decl maxSpeed : Vehicle -> Day -> Road -> Integer -> Boolean

rule <maxSpeedCarWorkday>
   for v: Vehicle, d: Day, r: Road
   if isCar v && isWorkday d
   then maxSpeed v d r 90

rule <maxSpeedCarHighway>
   {restrict: {subjectTo: maxSpeedCarWorkday}}
   for v: Vehicle, d: Day, r: Road
   if isCar v && isHighway r
   then maxSpeed v d r 130
```

Raisonner avec et sur des règles

Raisonner avec des règles :

- Obtenir un jugement pour une situation particulière

```
assert <maxSpeedPorscheFriday>    {SMT: valid}
isHighway instRoad --> maxSpeed porsche friday instRoad 90
```

Résultat : formule valide

- Savoir sous quelles conditions on obtient un résultat :

```
assert <maxSpeedPorsche200>  {SMT: consistent}
maxSpeed porsche instDay instRoad instSpeed && instSpeed >= 200
```

Résultat : formule insatisfiable

```
assert <maxSpeedPorsche100>  {SMT: consistent}
maxSpeed porsche instDay instRoad instSpeed && instSpeed >= 100
```

Résultat : formule satisfiable si :
`isHoliday instDay` et `isHighway instRoad`

Raisonner avec et sur des règles

Raisonner sur des règles :

- Savoir si un ensemble de règles est cohérent, *par ex. :*
  - Est-ce qu'une relation définit une fonction ?
  - Est-ce qu'on a en même temps une obligation et interdiction de faire . . . ?

```
assert <maxSpeedFunctional>  {SMT: valid}
maxSpeed instCar instDay instRoad instSpeed1 -->
maxSpeed instCar instDay instRoad instSpeed2 -->
instSpeed1 == instSpeed2
```

Résultat :

- Formule valide (pour les règles données)
- Contre-modèle (`instSpeed1 = 90, instSpeed2 = 130`)
  si on enlève `subjectTo` de `<maxSpeedCarHighway>`

Derrière la scène

De la formalisation, on construit un ensemble de formules qui sont envoyées à un SAT / SMT solveur (*Satisfiability Modulo Theories*)

```
(set-logic LIA )
(declare-sort Vehicle 0 )
(declare-fun isCar (Vehicle ) Bool )
(assert (and (and (maxSpeed porsche instDay instRoad instSpeed) (>= instSpeed 100))
  (and (=> true (isCar porsche ) ) (and (forall ((c Day ) )
       (=> true (not (and (isWorkday c ) (isHoliday c ) ) ) ) ) )
  (and (forall ((c Vehicle ) ) (=> true (not (and (isCar c ) (isTruck c ) ) ) ) ) )
  (and (forall ((v Vehicle ) ) (forall ((d Day ) ) (forall ((r Road ) )
       (=> (and (isCar v ) (isWorkday d ) ) (maxSpeed v d r 90 ) ) ) ) )
....
```

Derrière la scène

Le solveur réfute la formule ou renvoie un contre-modèle

```
((declare-fun Vehicle!val!0 () Vehicle ) (forall ((x Vehicle ) ) (= x Vehicle!val!0
(declare-fun Road!val!0 () Road ) (forall ((x Road ) ) (= x Road!val!0 ) )
(declare-fun Day!val!0 () Day ) (forall ((x Day ) ) (= x Day!val!0 ) )
(define-fun instDay () Day Day!val!0 )
(define-fun instRoad () Road Road!val!0 )
(define-fun instSpeed () Int 130 ) (define-fun porsche () Vehicle Vehicle!val!0 )
(define-fun maxSpeed ((x!0 Vehicle ) (x!1 Day ) (x!2 Road ) (x!3 Int ) ) Bool (= x!3
...
```

qu'on essaie d'afficher de manière "lisible" :

Synonymes: `porsche instCar instVeh`
Integer variables: `instSpeed = 130, instSpeed2 = 0, instSpeed1 = 0`
Predicates:

```
maxSpeed porsche instDay instRoad intArg3 = (= intArg3 130)
isHighway instRoad = true
isHoliday instDay = true
```

Plan

Plan

Logiques Modales - positionnement

Idée Les logiques modales sont une extension d'une logique de base (propositionnelle ; de 1er ordre) avec des opérateurs modaux telles que :

- possible $\Diamond$ / nécessaire $\Box$ (*aléthique*)
- permis P / obligatoire O (*déontique*)
- quelque temps / toujours (*temporelle*)
- (logiques épistémiques : x sait que ...)

Logiques Modales - Syntaxe

Syntaxe de la logique propositionnelle modale (*ici :* aléthique) :

- Formules de base : variables propositionnelles ; $\top$, $\bot$
- Négation $\neg\phi$
- Conjonction, disjonction, implication : $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \longrightarrow \psi$
- Possiblement : $\Diamond\phi$
- Nécessairement : $\Box\phi$

Logiques Modales - Vérité

Deux approches pour définir quand une formule est "vraie" :

1. Syntaxique, par un système déductif $\vdash \phi$
2. Sémantique (validité dans un ensemble de modèles) $\models \phi$

Système déductif contient des règles telles que :

$$\frac{\vdash A}{\vdash \Box A}\ (Necc)$$

$$\vdash \Box(A \longrightarrow B) \longrightarrow (\Box A \longrightarrow \Box B)$$

et selon le système :

$$\vdash \Box A \longrightarrow A$$

$$\vdash \Box A \longrightarrow \Box\Box A$$

$$\vdash \Diamond A \longrightarrow \Box\Diamond A$$

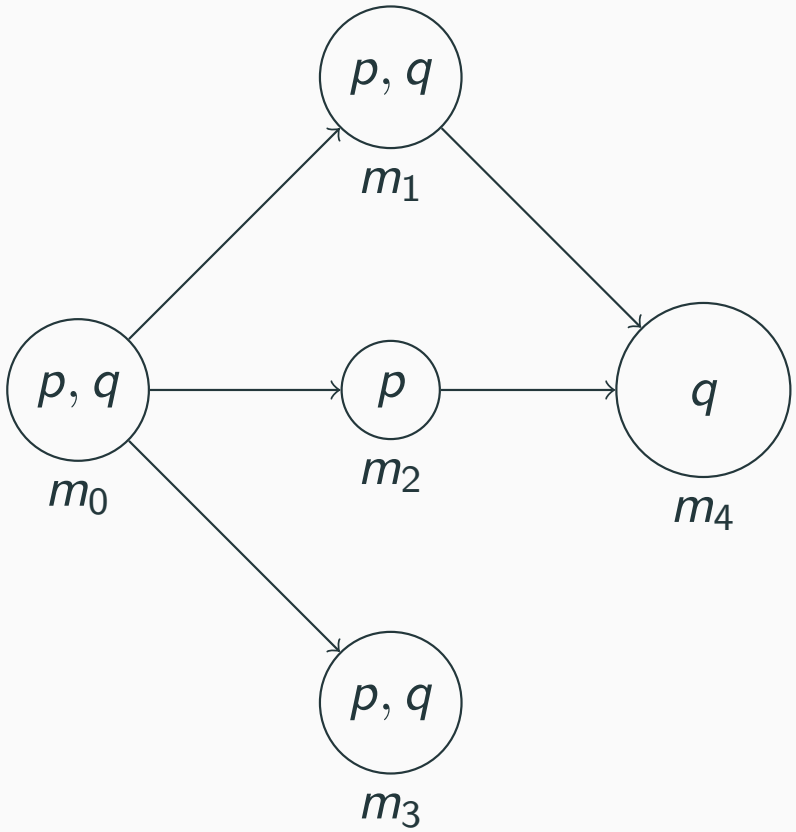
(Note : $(Necc)$ n'est pas équivalent à $\vdash A \longrightarrow \Box A$)

Logiques Modales - Sémantique

Sémantique des mondes possibles
dans la tradition de Leibniz - Carnap - Kripke

Un modèle $\mathcal{M} = (W, R, P)$ ("structure de Kripke") est donné par :

- W : ensemble de mondes
- $R \subseteq W \times W$: relation d'accessibilité
- $P : W \to \mathcal{P}(Var)$: propositions élémentaires validées dans les mondes



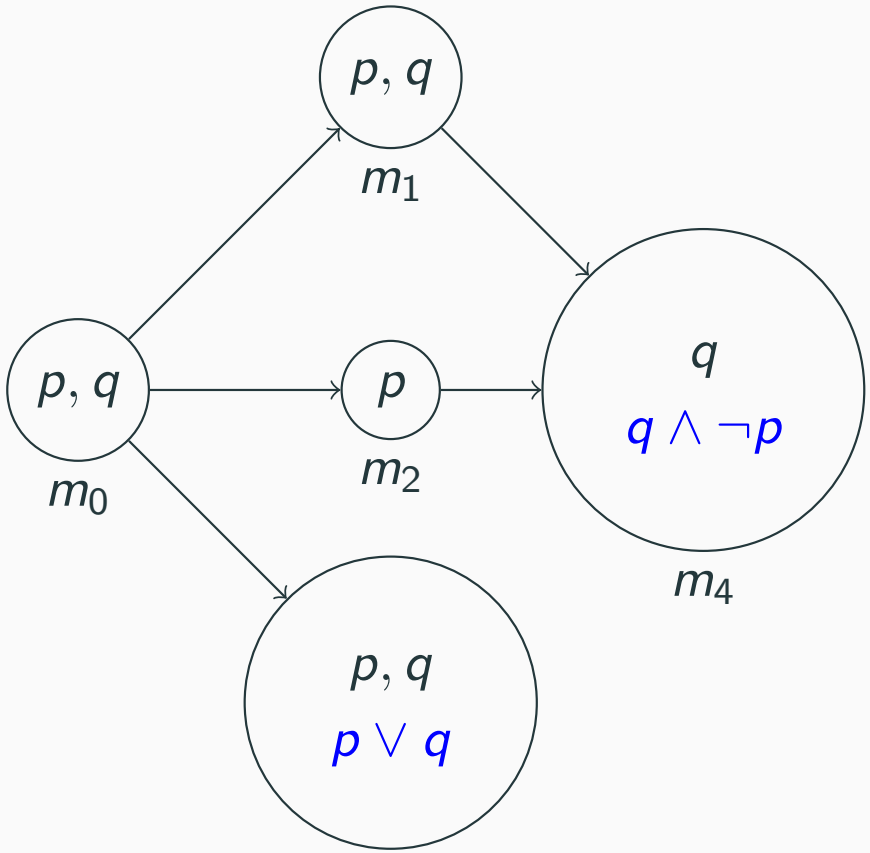
Logiques Modales - Sémantique

Sémantique des mondes possibles
dans la tradition de Leibniz - Carnap - Kripke

Validité d'une formule dans un monde m :

Logique propositionnelle :

- $\models_m v$ si $v \in P(v)$
- $\models_m (\phi \wedge \psi)$ si $\models_m \phi$ et $\models_m \psi$
- etc.



Logiques Modales - Sémantique

Sémantique des mondes possibles
dans la tradition de Leibniz - Carnap - Kripke

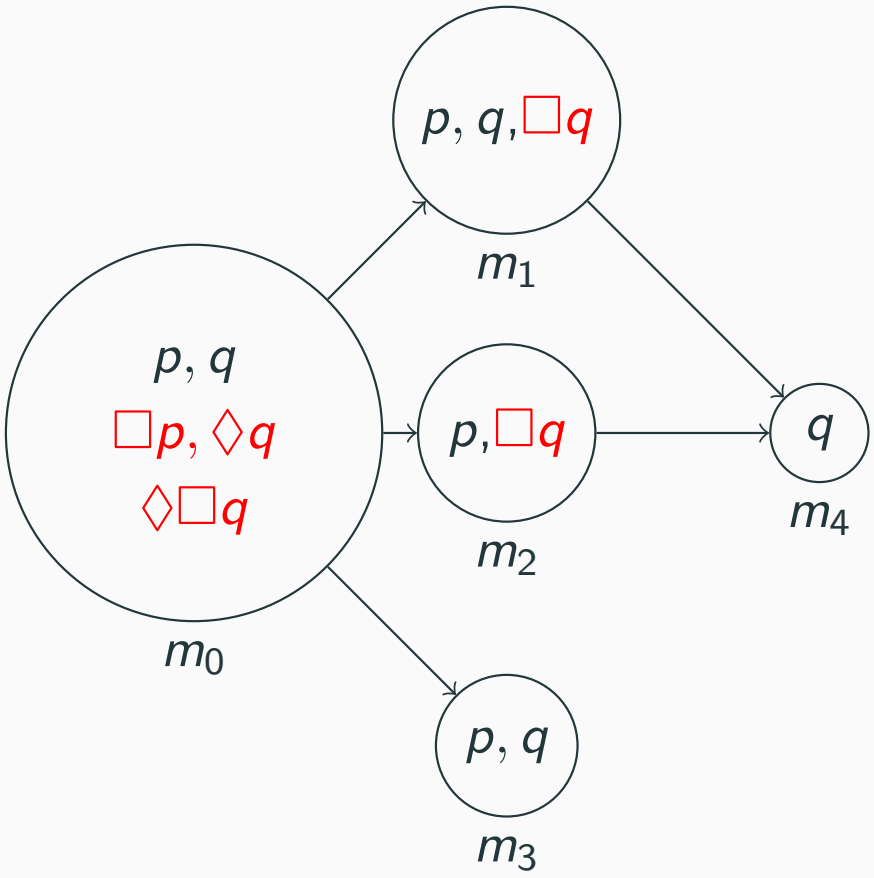
Validité d'une formule dans un monde m :

Logique propositionnelle :

- $\models_m v$ si $v \in P(v)$
- $\models_m (\phi \wedge \psi)$ si $\models_m \phi$ et $\models_m \psi$
- etc.

Opérateurs modaux :

- $\models_m \Box\phi$ si
  pour tout n tq. $R(m,n)$: $\models_n \phi$
- $\models_m \Diamond\phi$ si
  il existe n tq. $R(m,n)$ et $\models_n \phi$



Logiques Modales - Sémantique

Validité dans un modèle :

$\mathcal{M} \models \phi$ si $\models_m \phi$ pour tout $m \in W$

On peut s'intéresser à différentes classes de modèles :

- Relation R est réflexive $\Leftrightarrow$
  axiome $\Box A \longrightarrow A$ est satisfait
- Relation R est transitive $\Leftrightarrow$
  axiome $\Box A \longrightarrow \Box\Box A$ est satisfait

Dualité

- $\Box$ / $\Diamond$ est un quantificateur universel / existentiel sur les mondes
- $\neg\Box\phi \quad \Leftrightarrow \quad \Diamond\neg\phi$

Plan

Logiques Déontiques : Définitions

1. Définition par réduction à la logique aléthique selon Leibniz :
   - *permis* est ce qui est *possible* pour un homme bon ("vir bonus") à faire
   - *obligatoire* est ce qui est *nécessaire* à faire
2. Classe de logiques définies en analogie avec la logique aléthique
   - Sémantique de "mondes alternatifs" avec relation R de préférence morale.
   - $\models_m O\phi$ si pour tout n tq. $R(m,n) : \models_n \phi$
   - $\models_m P\phi$ s'il existe n tq. $R(m,n)$ et $\models_n \phi$
   - Dualité : $\neg O\phi \quad \Leftrightarrow \quad P\neg\phi$

Propriétés discutables de logiques déontiques :

- Généralement rejeté : $O\phi \longrightarrow \phi$ (tout ce qui est obligatoire est effectivement réalisé)
- Souhaitable : $O\phi \longrightarrow P\phi$ (tout ce qui est obligatoire est permis)
  équivalent à $\neg(O\phi \wedge O\neg\phi)$ (consistence des normes)
  équivalent à $\neg(O(\phi \wedge \neg\phi)) \equiv \neg(O\bot)$ (absence obligation d'idiocies)

Obligation d'être vs. obligation de faire

Strictu sensu, une proposition décrit une *situation* et non une *action*.

Des normes ou lois peuvent spécifier des obligations d'être :

- ... présent à un examen
- ... en possession d'un billet

mais énoncent souvent des obligations de faire :

- obligation d'effectuer un paiement jusqu'à une date donnée
- permission de voter

Souvent pas de séparation claire en logique ; inacceptable pour une informatisation.

Paradoxes des logiques déontiques

Constat : on peut affaiblir des obligations, la règle

$$\frac{A \longrightarrow B}{OA \longrightarrow OB}$$

est dérivable. Surtout :

$$Op \longrightarrow O(p \vee q)$$

est valide, et pareil

$$Pp \longrightarrow P(p \vee q)$$

Est-ce qu'on approuve les raisonnements suivants ? (Paradoxe de Ross) :

- Obligation :
  - Pierre doit envoyer la lettre.
  - *donc :* Pierre doit envoyer la lettre, ou la brûler.
- Permission :
  - Pierre est autorisé à boire de l'eau.
  - *donc :* Pierre est autorisé à boire de l'eau ou du whisky.

Paradoxes des logiques déontiques

Vers les obligations *contrary to duty* : le paradoxe de Chisholm

1. Jones doit aider ses voisins : Oa
2. Jones doit informer ses voisins de sa venue s'il les aide : $O(a \longrightarrow i)$
3. Si Jones n'aide pas ses voisins, il doit ne pas les informer de sa venue : $\neg a \longrightarrow O(\neg i)$
4. Jones n'aide pas ses voisins $\neg a$

De (1) et (2), on obtient Oi, de (3) et (4), on obtient $O\neg i$
donc $O(i \wedge \neg i)$ en contradiction avec le postulat $\neg(O(\phi \wedge \neg\phi))$

Contrary-to-duty

Des contraventions aux normes occupent une place importante des règlements :

- amendes et punitions
- restructuration d'une dette en cas de défaut de paiement

sans induire des incohérences.

Conclusion

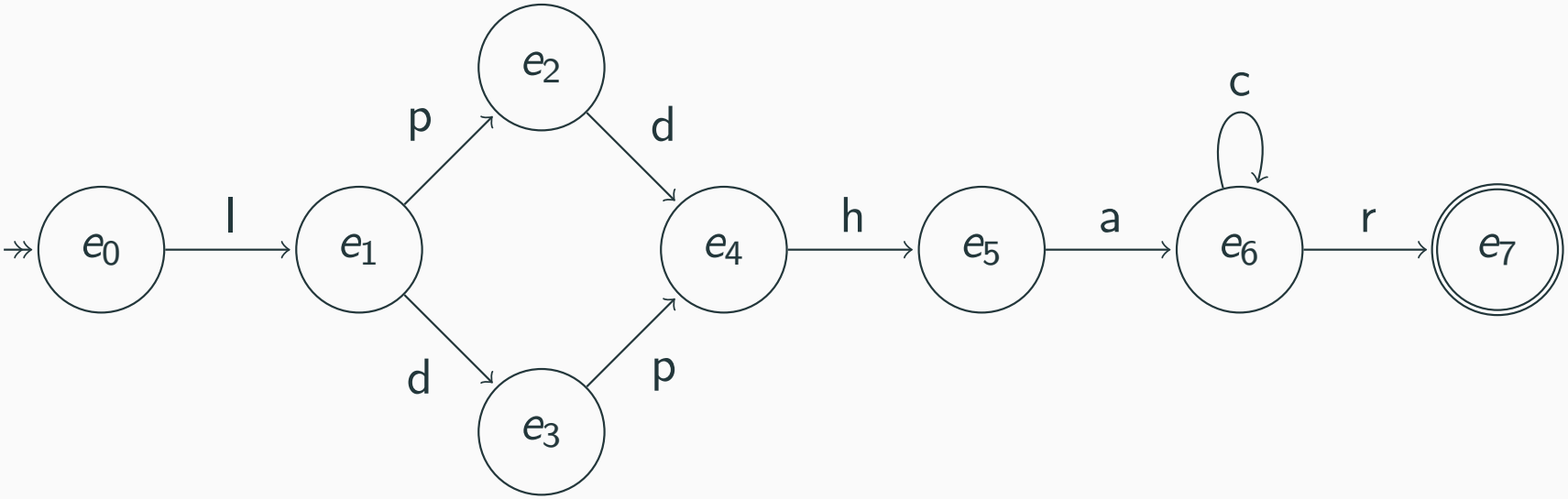
- Modélisation inadéquate d'actions, contraventions dans des logiques déontiques
- $\rightsquigarrow$ logiques temporelles

Plan

Plan

Exemple : La journée d'un étudiant médiocre

Automates Un modèle de calcul ubiquitaire en informatique

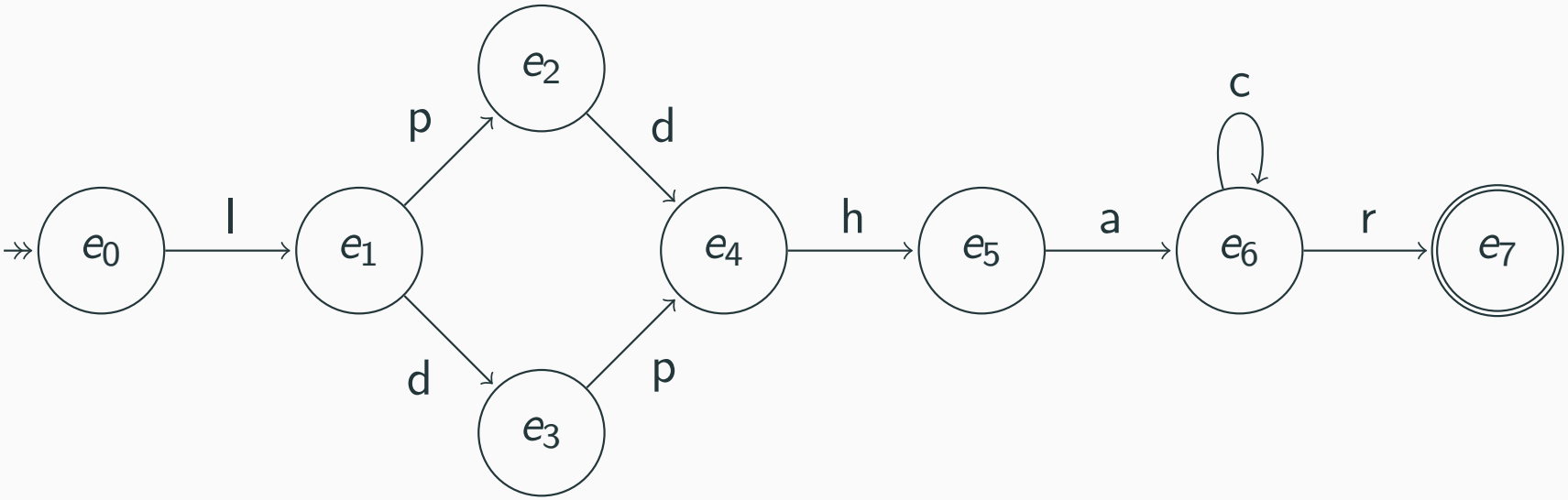


Sigles :

- l : se lever
- p : prendre le petit déjeuner
- d : prendre une douche
- h : s'habiller
- a : aller à l'université
- c : assister aux cours
- r : rentrer chez soi

Exemple : La journée d'un étudiant médiocre

Automates Un modèle de calcul ubiquitaire en informatique



Questions : est-il possible

- de prendre le petit déjeuner avant la douche ?
- de s'habiller avant de prendre la douche ?
- d'assister aux cours sans avoir pris une douche ?
- de rentrer chez soi sans avoir assisté à des cours ?

Ingrédients d'un automate

Un automate est composé de :

- un ensemble d'états Q
  - dont un état initial
  - un ou plusieurs états finaux
- un ensemble de transitions $\delta \subseteq Q \times \Sigma \times Q$ étiquetées par des symboles Σ

Traditionnellement, on s'intéresse aux mots reconnus par un automate :

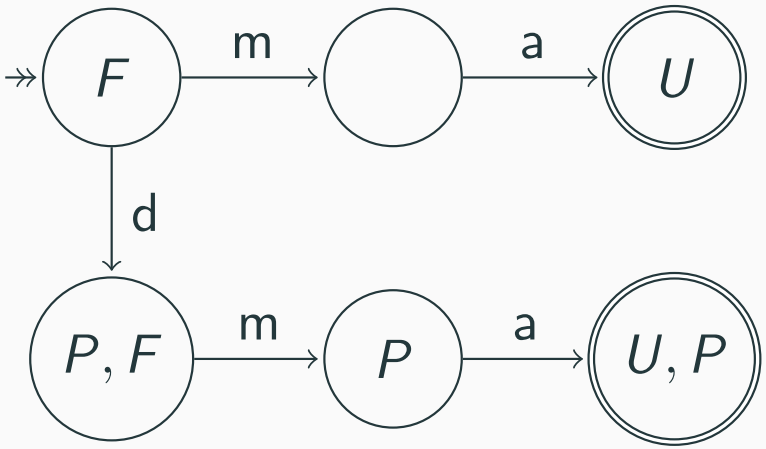
- mot : séquence de symboles menant de l'état initial à un état final
- exemple : *lpdhar* journée passée à l'université sans assiter aux cours

Automates et structures de Kripke

Les états peuvent être annotés avec des propositions, avec des équivalences :

- automate $\approx$ structure de Kripke
- état $\approx$ monde
- transition $\approx$ passage à un monde futur (temps discret)

Exemple : une autre situation estudiantine :
Actions : m : manger ; d : douche ; a : aller à l'université
Propriétés : F : avoir faim ; P : être propre ; U : être à l'université



Logiques temporelles : Computation Tree Logic

CTL permet de raisonner sur les évolutions d'un automate, avec des

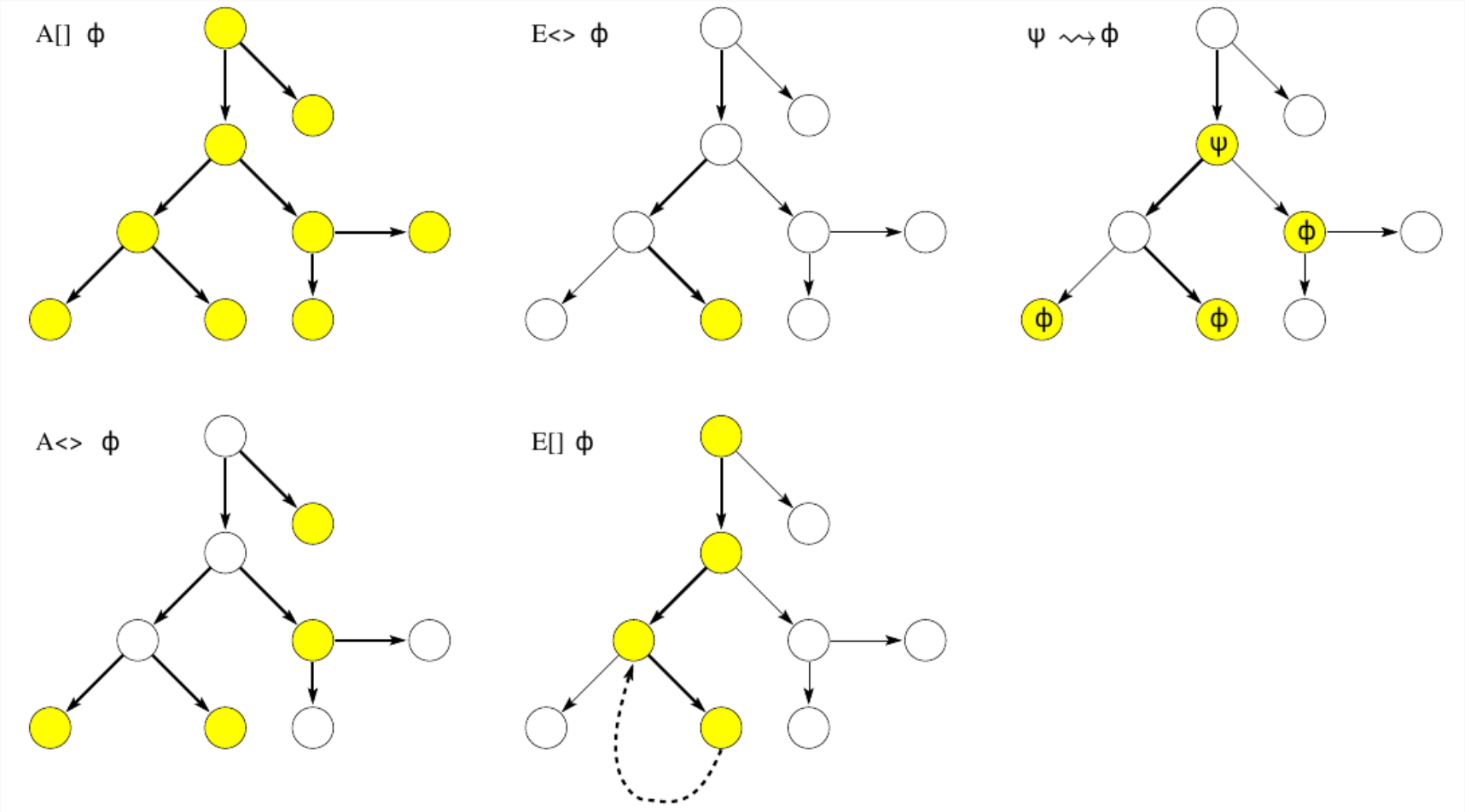
- Quantificateurs sur des chemins (exécutions possibles)
  - $A\phi$ pour toutes les exécutions, ϕ est satisfait
  - $E\phi$ il existe une exécution tq. ϕ est satisfait
- Quantificateurs sur des états sur un chemin :
  - $\Box\phi$: ϕ est toujours satisfait
  - $\Diamond\phi$: ϕ est éventuellement satisfait

Exemples :

- Dans tous les cas, l'étudiant sera finalement à l'université (*vrai*) : $A\Diamond U$
- Généralement, quand l'étudiant est à l'université, il est propre (*faux*) : $A\Box(U \longrightarrow P)$
- Généralement, quand l'étudiant est à l'université, il n'a pas faim (*vrai*) : $A\Box(U \longrightarrow \neg F)$
- L'étudiant a le choix d'être toujours propre quand il est à l'univ. (*vrai*) : $E\Box(U \longrightarrow P)$

Quantification sur des chemins et des états

Illustration graphique (*Note :* $\psi \rightsquigarrow \phi$ raccourci pour $A\Box(\psi \longrightarrow A\Diamond\phi)$)



Automates temporisés

Le modèle d'automate et de CTL est à *temps discret*, avec changement d'état initié par des actions.

Extensions temporisées pour modélisation et vérification de *temps réel*

- *Sémantique :* deux types de transitions :
  - Transitions discrètes sur événements (comme avant)
  - Passage de *temps continu*
- *Automates :* conditions temporelles entre horloges h et instants t :
  - *états :* contraintes $h \leq t$; sinon obligation de quitter l'état
  - *transitions* doit être franchie selon contrainte $t_1 \leq h \leq t_2$
- *Logique :* Extensions de la quantification de la forme : $A\Box^I \phi$:
  "la propriété ϕ est toujours satisfaite dans l'intervalle I, pour tout chemin"

CTL et raisonnement sur des normes

CTL est approprié pour exprimer des règles et normes telles que des

- *invariants* : conditions à toujours ($A\Box$) satisfaire dans un contrat
- *échéances :* situations à atteindre à un moment donné ($A\Diamond$)
- *réparations* après violation d'un contrat ; raisonnement *contrary-to-duty* de la forme $A\Box(\mathit{breach} \longrightarrow A\Diamond \mathit{repair})$

Difficultés : Maintenir la correspondance entre actions et propriétés.
Ex. : "avoir faim" change de valeur

- explicitement après des actions pertinentes ("manger")
- implicitement après un certain temps
- implicitement comme effet collatéral d'actions sans rapport évident ("voir nourriture appétissante") $\rightsquigarrow$ *frame problem*

Plan

Etude de cas : PDPA

Modélisation du Personal Data Protection Act de Singapour

But :

- Protection de la vie privée et des données personnelles
- En cas d'un soupçon de vol de données (*data breach*) :
  - Évaluer la gravité de l'atteinte à la vie privée et, selon le cas :
  - informer les individus concernés
  - informer une commission (PDPC) du vol

Le PDPA est composé :

- de *règles statiques* déterminant si une atteinte à la vie privée a eu lieu, et de sa gravité (*notifiable data breach*)
- d'un *processus* (règles dynamiques) à suivre en cas d'infraction notifiable

PDPA : Processus d'information

Procédure à suivre en cas d'infraction

Acteurs : Organisation ; Commission ; Individu lésé

Règles (extrait, voir détails)

(0) L'Organisation doit évaluer pendant au maximum 30 jours s'il y a infraction.

(1) L'Organisation doit informer la Commission au plus 3 jours après le constat d'une infraction.

(2) Sujet à (6), l'Organisation doit aussi informer l'Individu lors de la notification de la Commission ou après.

(6) L'Organisation ne doit pas informer l'Individu si la Commission l'ordonne ainsi.

Modélisation en Uppaal

Modélisation

- Traduction manuelle des règles en un système d'automates temporisés synchronisés
- Simulation et vérification avec l'outil Uppaal

Model Checking pour vérifier $\mathcal{M} \models \phi$

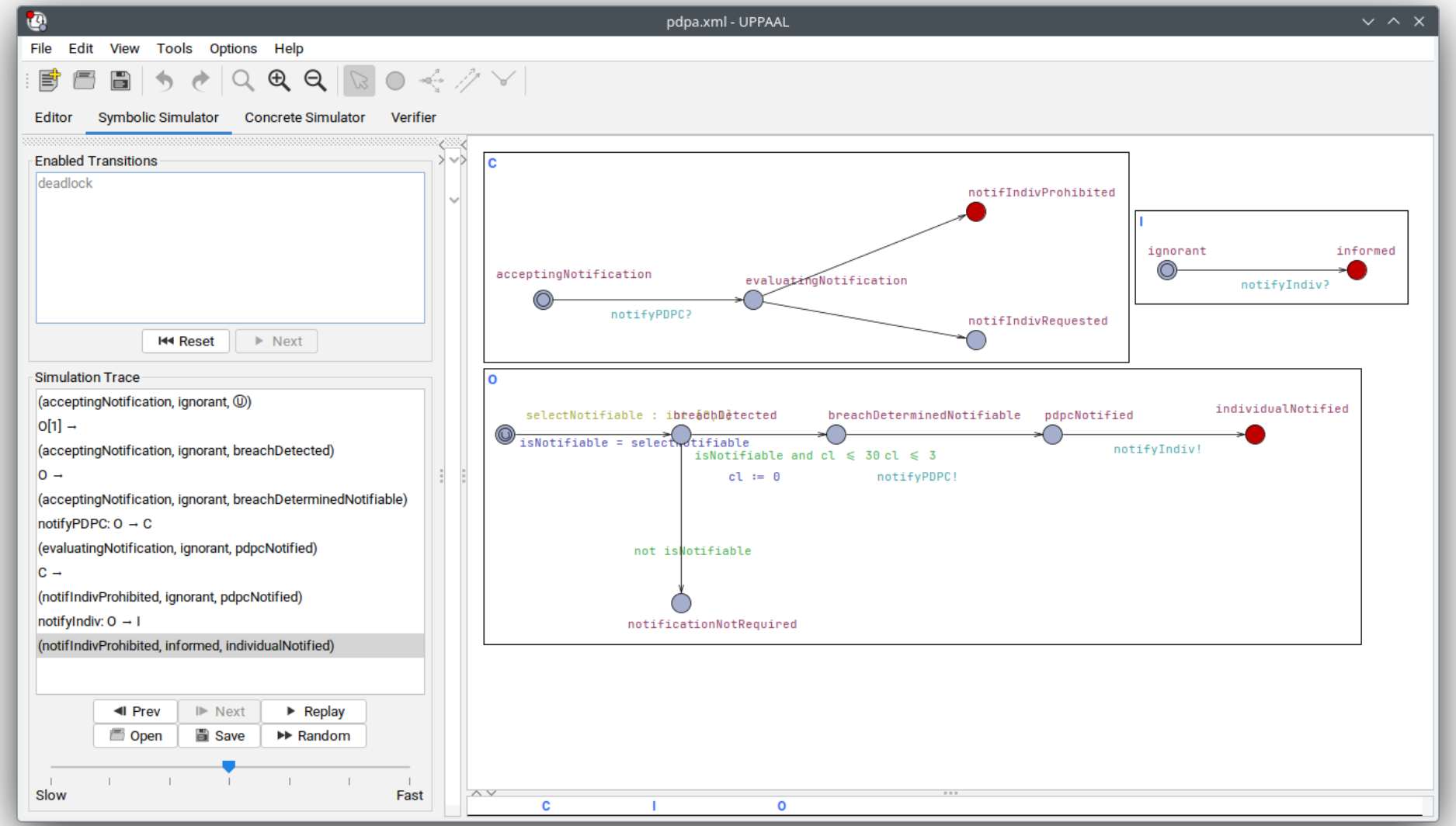
- (formule ϕ est satisfaite dans le modèle $\mathcal{M}$ / état initial de l'automate)
- Trace dans l'automate pour
  - preuve de propriété existentielle
  - réfutation de propriété universelle

Exemples :

- On arrive toujours finalement dans un état où une notification n'est pas requise ou l'Individu a été informé. (*Faux*)
- Il peut arriver que l'Individu a été informé et la Commission a interdit d'informer l'Individu. (*Vrai*)

Modélisation en Uppaal

Résultat de la requête : `E<> I.informed and C.notifIndivProhibited`



Plan

Évaluation

Observations

- Beaucoup d'imprécisions et erreurs détectées lors de la formalisation (suscite souvent l'indifférence des concepteurs)
- Souhaitable : Législation déjà en format formalisé

Objectifs partiellement atteints

- Raisonnement sur / simulation de scénarios concrets
- Exploration de la consistance d'un ensemble de règles

Un langage fragmentaire

Enjeux

- Concevoir un langage plus lisible
- Meilleure structuration des lois
- Intégration de règles statiques et dynamiques / processus
- Quelles structures de données informatiques (listes / enregistrements / autres types inductifs)
- . . . et langages de programmation ?

Des capacités de raisonnement et d'explication

Enjeux

- Traduction automatisée d'obligations (légales) en oblgations (de preuve)
- Découverte automatisée d'incohérences de règlements et lois.
- Sélection de la bonne méthode d'inférence
- Organisation de l'interaction entre différentes théories (arithmétique, temporelle, . . .)
- Meilleure présentation des contre-modèles
- Possibilité d'explication de décisions, peut-être dans dialogue interactif

Représentation et raisonnement sur des connaissances

Prendre en compte des connaissances générales sur le monde :

- Temporelles : jours de la semaine, jours ouvrables / fériés, ...
- Connaissances basiques sur des parts du corps, des sciences (anatomie, médecine, physique, géographie, ...)

Méta-raisonnement du style : une loi est applicable sous certaines conditions

Interfaçage avec langage naturel

Prévu à l'origine :

- Génération de langage naturel à partir du DSL
- Utilisation de Grammatical Framework
- Peut-être aussi écriture de lois en langage naturel

. . . mais pas réalisé

Désormais : prise en compte d'IA générative

- Traduction de textes en langage naturel vers DSL ?
- (Avantages d'un texte de loi formel)
- Meilleur traitement de connaissances générales du monde