SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Année 2025-2026

M1 AI2D
M1 MIND

Logique et représentation des connaissances
LRC
UM4IN800

Au programme (≠ ordre des séances)

1. Représentation de connaissances, cas général
    - logique des prédicats du premier ordre
    - logiques de description
2. Représentation de connaissances non factuelles
    - logiques modales, en particulier épistémique
    - connaissances communes, partagées, ...
3. Représentation de connaissances temporelles
    - intervalle d'Allen
    - réseaux de Petri
    - logiques temporelles

LRC - TD 1

1 Logique des propositions

Exercice 1 – Sémantique, d'après *Lassaigne & de Rougemont*

1. Soit $F = (p \wedge q \wedge \neg r) \vee (p \wedge \neg q \wedge r) \vee (\neg p \wedge q \wedge r)$.

   F et $\neg F$ sont-elles satisfiables ? Sont-elles des tautologies ? Justifier.

2. Trouver une formule G telle que $(F \wedge G) \vee (\neg F \wedge \neg G)$ soit une tautologie.

3. Soit F' obtenue en remplaçant p par $\neg p$ (et réciproquement). F' est-elle conséquence de F ? F est-elle conséquence de F' ? Justifier.

Exercice 2 – Sémantique

Quel est le nombre maximum de formules non équivalentes que l'on peut former avec n variables propositionnelles ? Quelles sont-elles pour $n = 1$?

Exercice 3 – Méthode des tableaux sémantiques

Que peut-on dire des formules suivantes en utilisant la méthode des tableaux sémantiques ?

- $F_1 = a \wedge \neg(b \to a)$
- $F_2 = ((a \vee c) \wedge (b \vee c)) \to (\neg b \to ((a \wedge b) \vee c))$
- $F_3 = \neg((a \to b) \to (\neg b \to \neg a))$
- $F_4 = ((a \to b) \wedge (b \to c)) \vee ((c \to b) \wedge (b \to a))$
- $F_5 = (a \to b) \to ((b \to c) \leftrightarrow (a \to c))$
- $F_6 = ((a \to b) \wedge (b \to c)) \to (a \to c)$

Exercice 4 – Preuves de Hilbert

1. Justifier chaque étape de la démonstration ci-dessous dans le système formel de Hilbert et identifier le résultat démontré :

F_1	p	[ ]
F_2	$\neg q \to r$	[ ]
F_3	$\neg\neg p \to \neg r$	[ ]
F_4	$(\neg\neg p \to \neg r) \to (r \to \neg p)$	[ ]
F_5	$r \to \neg p$	[ ]
F_6	$(r \to \neg p) \to (\neg q \to (r \to \neg p))$	[ ]
F_7	$\neg q \to (r \to \neg p)$	[ ]
F_8	$(\neg q \to (r \to \neg p)) \to ((\neg q \to r) \to (\neg q \to \neg p))$	[ ]
F_9	$(\neg q \to r) \to (\neg q \to \neg p)$	[ ]
F_{10}	$\neg q \to \neg p$	[ ]
F_{11}	$(\neg q \to \neg p) \to (p \to q)$	[ ]
F_{12}	$p \to q$	[ ]
F_{13}	q	[ ]

2. En utilisant le théorème de la déduction, rappelé ci-dessous :

$$\text{Si } A_1, A_2, \ldots, A_n \vdash B \text{ alors } A_1, A_2, \ldots, A_{n-1} \vdash (A_n \to B)$$

établir que $\vdash (A \to B) \to ((B \to C) \to (A \to C))$

Exercice 5 – Preuve de Hilbert

Démontrer les théorèmes suivants dans le système formel de Hilbert :

1. $\vdash \neg B \to (B \to C)$
2. $\vdash \neg\neg B \to B$
3. $\vdash B \to \neg\neg B$
4. $\vdash (A \to B) \to (\neg B \to \neg A)$

2 Représentation en LPPO

Exercice 6

En utilisant les symboles de prédicats et fonctions suivants

$A(x)$	x est anglais	$e(x)$	dénote le pire ennemi de x
$D(x, y)$	x déteste y	n	dénote Napoléon
$C(x, y)$	x connaît y		

représenter les phrases suivantes par des formules de la LPPO :

1. tout Anglais déteste quelqu'un
2. le pire ennemi de Napoléon est anglais
3. tout Anglais déteste son pire ennemi
4. Napoléon déteste les Anglais
5. au moins un Anglais déteste Napoléon
6. tout le monde connaît quelqu'un qu'il déteste et quelqu'un qu'il ne déteste pas
7. celui qui connaît son pire ennemi ne le déteste pas

Exercice 7

On considère le domaine des œuvres littéraires, le domaine des auteurs et le domaine des êtres humains. Les symboles de constantes `a, m, s` représentent respectivement Alice, "Les mots" et Jean-Paul Sartre. Les prédicats unaires D et R sont tels que $D(x)$ représente "x est un membre du département de littérature" et $R(x)$ "x est un roman", les prédicats binaires E et L tels que $E(x, y)$ représente "x a écrit y" et $L(x, y)$ "x a lu y".

1. Représenter les phrases suivantes par des formules de la LPPO
   (a) Un des membres du département de littérature a lu Les mots.
   (b) Tous les membres du département de littérature ont lu Les mots.
   (c) Alice a lu un roman de Sartre.
   (d) Un des membres du département de littérature n'a lu que des romans de Sartre.
   (e) Aucun des membres du département de littérature n'a lu tous les romans de Sartre.
   (f) Tous les membres du département de littérature qui ont lu Les mots ont lu tous les romans de Sartre.
2. Exprimer en langage naturel la signification des formules suivantes
   (a) $\exists x(D(x) \wedge \neg L(x, m))$
   (b) $\forall x((R(x) \wedge E(s, x)) \to L(a, x))$
   (c) $\forall x(D(x) \to \exists y(R(y) \wedge E(s, y) \wedge L(x, y)))$
   (d) $\neg\exists x(D(x) \wedge \forall y((R(y) \wedge L(x, y)) \to E(s, y))$

Système de Hilbert et tableaux sémantiques

1 Système de Hilbert

1.1 Composantes

Le système de Hilbert est caractérisé par trois schémas d'axiome et une règle d'inférence :

- Schémas d'axiome :
  - **SA1 :** $A \to (B \to A)$
  - **SA2 :** $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$
  - **SA3 :** $(\neg B \to \neg A) \to (A \to B)$
- Règle d'inférence
  - **Modus Ponens (MP) :** $A, A \to B \vdash B$

1.2 Déduction

La déduction d'une formule A dans une théorie Δ est une suite finie $A_0, \ldots, A_n$ telle que $A_n = A$ et pour tout i,

- A_i est l'instanciation de l'un des axiomes,
- A_i est l'une des hypothèses, c'est-à-dire $A_i \in \Delta$
- A_i est obtenue par modus ponens appliqué à A_j et A_k avec $j < i$ et $k < i$

On peut aussi appliquer toutes les substitutions nécessaires, à condition de les effectuer dans l'ensemble de la formule.

Si on trouve une telle suite, on peut noter $\Delta \vdash A$.

2 Méthode des tableaux sémantiques

La méthode des tableaux sémantiques permet d'établir si un ensemble de fomules logiques est valide, satisfiable ou insatisfiable.

2.1 Composantes

La méthode des tableaux est basée sur des règles syntaxiques de décomposition, qui distinguent deux types de formules, nommés α et β.

Formule α	α_1	α_2
$\neg\neg\varphi$	φ	φ
$\varphi_1 \wedge \varphi_2$	φ_1	φ_2
$\neg(\varphi_1 \vee \varphi_2)$	$\neg\varphi_1$	$\neg\varphi_2$
$\neg(\varphi_1 \rightarrow \varphi_2)$	φ_1	$\neg\varphi_2$
$\varphi_1 \leftrightarrow \varphi_2$	$\varphi_1 \rightarrow \varphi_2$	$\varphi_2 \rightarrow \varphi_1$

Formule β	β_1	β_2
$\varphi_1 \vee \varphi_2$	φ_1	φ_2
$\neg(\varphi_1 \wedge \varphi_2)$	$\neg\varphi_1$	$\neg\varphi_2$
$\varphi_1 \rightarrow \varphi_2$	$\neg\varphi_1$	φ_2
$\neg(\varphi_1 \leftrightarrow \varphi_2)$	$\neg(\varphi_1 \rightarrow \varphi_2)$	$\neg(\varphi_2 \rightarrow \varphi_1)$

2.2 Satisfiabilité

La recherche d'un modèle pour un ensemble de formules $\mathcal{F}$ par la méthode des tableaux peut être représentée de différentes façons, nous utilisons ici une forme arborescente.

- Initialisation : créer un nœud racine, étiqueté par l'ensemble $\mathcal{F}$ et marqué comme non traité
- Décomposition itérative : choisir un nœud non traité et le marquer comme traité
  - si l'étiquette du nœud contient deux littéraux complémentaires, marquer le nœud comme fermé
  - sinon, si toutes les formules associées au nœud sont des variables propositionnelles, marquer le nœud comme ouvert
  - sinon, choisir une formule F de l'étiquette du nœud
    - si elle est de type α
      - créer un sous-nœud marqué comme non traité
      - lui associer l'étiquette $\mathcal{F} \setminus \{F\} \cup \{\alpha_1, \alpha_2\}$ où α_1 et α_2 sont les formules obtenues par réécriture de F
    - sinon (si elle est de type β)
      - créer deux sous-nœuds marqués comme non traités
      - leur associer respectivement les étiquettes $\mathcal{F} \setminus \{F\} \cup \{\beta_1\}$ et $\mathcal{F} \setminus \{F\} \cup \{\beta_2\}$ où β_1 et β_2 sont les formules obtenues par réécriture de F

Si l'arbre contient une feuille ouverte, alors $\mathcal{F}$ est satisfiable.
Si toutes les feuilles de l'arbre sont fermées, alors $\mathcal{F}$ est insatisfiable.

Méthode de résolution par tableaux

Compte-rendu de TME

A la fin de chaque TME, il vous est demandé de rédiger un court compte-rendu (taille indicative, une dizaine de lignes), réalisant un bilan argumenté de la séance : quelles sont les notions principales que vous avez acquises ? quels sont les éléments à retenir du TME ?

Outre ce bilan, le compte-rendu doit contenir les réponses aux "questions étoilées".

Le compte-rendu doit commencer par les noms des 2 étudiants et être soumis sur le moodle (https://moodle-sciences-25.sorbonne-universite.fr).

Selon les semaines, le type du fichier demandé sera différent. Aujourd'hui, un fichier texte est attendu.

1 Présentation rapide du logiciel

Nous allons utiliser pour ce TME (et d'autres par la suite) le logiciel LoTREC, développé par des chercheurs de l'université de Toulouse, et disponible à l'adresse suivante:
`http://www.irit.fr/Lotrec/`

- téléchargez la version 2.0 du logiciel. Après après avoir extrait l'archive dans votre répertoire de travail (en respectant l'arborescence des fichiers), vous avez accès à un fichier `lotrec.jar` qu'il suffit d'exécuter par la commande
  `java -jar lotrec.jar`

A l'ouverture, le logiciel demande avec quelle logique on souhaitez travailler. Pour cette séance consacrée à la logique propositionnelle, choisir `Classical-Propositional-Logic`.

L'interface se compose de 3 parties:

- dans la partie supérieure gauche se trouve le détail de la logique utilisée;
- dans la partie inférieure gauche se trouve une fenêtre permettant d'entrer des formules dans la logique spécifiée;
- dans la partie de droite (`Premodels view`) se trouve une représentation sous forme de graphe correspondant aux constructions effectuées par LoTREC pour chercher les modèles dans cette logique.

2 Prise en main

Dans cet exercice, il vous faut entrer des formules dans le cadre prévu à cet effet, en notation préfixe et en écrivant les variables propositionnelles en majuscule. Le bouton `Build Premodels` permet de construire directement l'arbre obtenu par la méthode des tableaux, le bouton `Step by step` permet de suivre la construction pas à pas et l'ordre d'exécution des règles. Pour ce dernier mode, dans la fenêtre qui s'ouvre, il faut sélectionner toutes les règles puis cliquer sur `Start`. Dans le panneau `Controls`, il faut ensuite cliquer sur `Resume` pour commencer, LoTREC affiche alors la prochaine règle qu'il va utiliser, qu'on déclenche en cliquant sur `Next`.

Par rapport à la représentation vue en TD, vous constaterez que les règles α ne créent pas à proprement parler de nouveaux nœuds, mais elles augmentent le nœud courant. Il s'agit là d'une simple différence de représentation. Notez également que, par défaut, LoTREC n'affiche qu'une des feuilles de l'arbre. Il est possible d'accéder à l'intégralité de l'arbre en cliquant sur `Tableaux Tree` dans `Premodels List`. Pour les règles β, LoTREC n'affiche pas à quelle feuille de l'arbre elles sont appliquées.

1. Utiliser le mode pas à pas avec la formule prédéfinie par LoTREC pour en comprendre le fonctionnement.

2* Reprendre les formules de l'exercice 2 du TD1, en utilisant le mode pas à pas. Dans chaque cas, indiquer si la formule est valide, satisfiable ou insatisfiable en justifiant l'exploitation de l'arbre.

3 Diagnostic médical (simpliste)

On dispose des connaissances suivantes sur la grippe :

(a) La fièvre est définie comme une température supérieure à 38°.

(b) Les patients qui ont la grippe doivent prendre du tamiflu.

(c) Les patients qui ont de la fièvre et qui toussent ont la grippe.

(d) Le patient tousse et a une température supérieure à 38°.

1. Formaliser ces connaissances en utilisant les variables propositionnelles `grippe`, `tamiflu`, `fièvre`, `toux`, `sup38`.

2. Exprimer une formule permettant de déterminer s'il faut prendre du tamiflu et utiliser LoTREC pour conclure dans le cas du patient considéré.

3* On introduit une incertitude sur le fait que toux et fièvre implique grippe: une nouvelle règle indique que cela peut être une grippe *ou* une bronchite. Vérifier si la prise de tamiflu est toujours indiquée dans ce cas, en étudiant les modèles obtenus.

4 Modèles et pré-modèles

Dans l'arbre construit, une branche ouverte correspond à un *pré-modèle*, c'est-à-dire un modèle partiellement spécifié, auquel peuvent correspondre plusieurs modèles. On note $M(P)$ le(s) modèle(s) correspondant au pré-modèle P.

1. Construire sur papier la table de vérité et énumérer les modèles obtenus qui satisfont la formule
$$((a \to b) \wedge b \wedge c) \vee ((c \to b) \wedge (b \to a))$$

2. Construire avec LoTREC les pré-modèles obtenus par application de la méthode des tableaux. Représenter les pré-modèles sur l'hypercube des interprétations possibles.

3* Peut-on construire une formule donnant 3 pré-modèles P_1, P_2, P_3, tels que $M(P_1) \subset M(P_2) \subset M(P_3)$?

4* Selon-vous, existe-t-il un lien entre le nombre de pré-modèles obtenus et le nombre de modèles d'une formule?

5 Exploitation de l'arbre

On revient ici sur l'analyse globale de l'arbre construit.

1* Définir une formule non valide qui conduit à un arbre dont toutes les feuilles sont ouvertes.

2* Définir une formule valide qui conduit à un arbre contenant une feuille fermée.

LRC - TD 2

Logique de description $\mathcal{ALC}$

1 Sémantique

Exercice 1
Pour chacun des énoncés suivants en $\mathcal{ALC}$

1. indiquer son type (axiome ou assertion)
2. donner la traduction en LPPO
3. exprimer le sens de cet énoncé en langage naturel

(a) $\texttt{Chat} \sqsubseteq \texttt{Felin}$

(b) $\texttt{Chien} \sqsubseteq \neg\texttt{Chat}$

(c) $\texttt{pierre} : \exists\texttt{aParent}.\texttt{Medecin}$

(d) $\texttt{lesMots}, \texttt{sartre} : \texttt{ecritPar}$

(e) $\texttt{hotd} : \texttt{Serie} \sqcap \forall\texttt{aActeurPrincipal}.\texttt{Britannique}$
(Note : hotd représente "House of the Dragon")

(f) $\texttt{Animal} \sqcap \forall\texttt{mange}.\bot \sqsubseteq \bot$

(g) $\texttt{Professeur} \sqcap \exists\texttt{appartient}.\texttt{DeptLitterature} \sqsubseteq \exists\texttt{aLu}.(\texttt{Roman} \sqcap \exists\texttt{ecritPar}.\texttt{Philosophe})$

(h) $\texttt{Princesse} \equiv \exists\texttt{aParent}.(\texttt{Roi} \sqcup \texttt{Reine}) \sqcup \exists\texttt{aEpoux}.\exists\texttt{aParent}.(\texttt{Roi} \sqcup \texttt{Reine})$

2 Raisonnement par algorithme tableau

Exercice 2 Vérification de cohérence
On considère la Abox $\mathcal{A}_1$ suivante :

$\mathcal{A}_1$:

$\texttt{aristote} : \texttt{PhilosopheAncien} \sqcap \forall\texttt{aEcrit}.\texttt{Serieux}$
$\texttt{laPoétique\_livre2} : \neg\texttt{Serieux} \sqcup \texttt{Subversif}$
$\texttt{aristote}, \texttt{laPoétique\_livre2} : \texttt{aEcrit}$

1. Exprimer en langage naturel les connaissances de cette Abox.
2. En utilisant la méthode des tableaux pour $\mathcal{ALC}$, prouver que $\mathcal{A}_1$ est cohérente selon la Tbox vide ($\emptyset$), c'est-à-dire, prouver que $\langle \emptyset, \mathcal{A}_1 \rangle \not\models \bot$.

Exercice 3 Reconnaissance d'instance
On reprend le problème précédent, mais reformulé avec une Tbox :

$\mathcal{T}$:

$\texttt{Autorité} \equiv \texttt{PhilosopheAncien} \sqcap \forall\texttt{aEcrit}.\texttt{Serieux}$
$\texttt{Frivole} \equiv \neg\texttt{Serieux}$
$\texttt{Subversif} \equiv \texttt{Comédie} \sqcap \neg\texttt{Frivole}$

$\mathcal{A}$:

$\texttt{aristote} : \texttt{Autorité}$
$\texttt{laPoétique\_livre2} : \texttt{Comédie}$
$\texttt{aristote}, \texttt{laPoétique\_livre2} : \texttt{aEcrit}$

Appliquer la méthode des tableaux $\mathcal{ALC}$ pour prouver que LaPoétique_Livre2 est subversif en suivant les étapes suivantes :

1. Formaliser la question en un problème de reconnaissance d'instance et expliquer comment se ramener à un problème de vérification de cohérence.

2. Indiquer si $\mathcal{T}_2$ est une Tbox définitoire, simple ou générale et si elle est acyclique ou non.

3. Déployer $\mathcal{T}_2$ et appliquer les transformations nécessaires pour obtenir la Abox racine de l'algorithme tableau.

4. Résoudre ce tableau et conclure.

Exercice 4 Satisfiabilité de concept

On considère la Tbox $\mathcal{T}_3$ suivante :

$$\mathcal{T}_3: \boxed{\begin{array}{l} \texttt{Herbivore} \sqsubseteq \exists \texttt{mange.Plante} \\ \texttt{Carnivore} \sqsubseteq \exists \texttt{mange.Animal} \\ \texttt{Sauvage} \sqsubseteq \texttt{Animal} \sqcup \texttt{Plante} \end{array}}$$

Déterminer avec la méthode des tableaux en $\mathcal{ALC}$ si le concept $\exists \texttt{mange.Sauvage} \sqcap \neg(\texttt{Herbivore} \sqcup \texttt{Carnivore})$ est satisfiable selon $\mathcal{T}_3$ en détaillant toutes les étapes.

Exercice 5 Test de subsomption

On considère la Tbox $\mathcal{T}_4$ suivante :

$$\mathcal{T}_4: \boxed{\begin{array}{l} \forall \texttt{mange.Animal} \sqsubseteq \texttt{CarnivoreStrict} \\ \neg \texttt{Plante} \sqcap \neg \texttt{Animal} \sqsubseteq \top \end{array}}$$

Montrer avec la méthode des tableaux en $\mathcal{ALC}$, en détaillant toutes les étapes, que d'après ces axiomes, tout individu qui ne mange aucune plante est un carnivore strict, c'est-à-dire que

$$\mathcal{T}_4 \models \neg \exists \texttt{mange.Plante} \sqsubseteq \texttt{CarnivoreStrict}$$

.

Logiques de description

Ce document récapitule la syntaxe de $\mathcal{ALC}$ et $\mathcal{SHOIN}$, puis donne l'expression de l'algorithme tableau pour $\mathcal{ALC}$.

Pour approfondir le sujet, il est possible de se référer à `http://dl.kr.org/` qui centralise un certain nombre de ressources sur le sujet, ainsi qu'à `http://www.cs.man.ac.uk/~ezolin/dl/` qui donne les différentes extensions de $\mathcal{ALC}$ et les résultats de complexité qui s'y rapportent.

Toutes les logiques de description sont construites sur une signature $(\mathbb{C}, \mathbb{R}, \mathbb{I})$ qui définit le vocabulaire (symboles non prédéfinis) :

- $\mathbb{C}$ donne l'ensemble des concepts atomiques
- $\mathbb{R}$ donne l'ensemble des rôles atomiques
- $\mathbb{I}$ donne l'ensemble des constantes

1 Logique de description $\mathcal{ALC}$

1.1 Expressions de concept

A désigne un concept atomique (élément de $\mathbb{C}$) et R désigne un rôle atomique (élément de $\mathbb{R}$). Un concept C de $\mathcal{ALC}$ est défini comme:

$$C ::= A|\top|\bot|\neg C|C \sqcap C|C \sqcup C|\exists R.C|\forall R.C$$

1.2 Base de connaissances

- Tbox - Axiomes ontologiques (C_1, C_2 sont des concepts):

  Axiome d'équivalence : $C_1 \equiv C_2$
  Axiome d'inclusion : $C_1 \sqsubseteq C_2$

- ABox - Assertions (a est une constante, C un concept et R un rôle)

  Assertions de concepts : $a : C$
  Assertions de rôles : $\langle a, b\rangle : R$

1.3 Sémantique

Etant donné une structure $M = (\Delta^M, .^M)$,

- $(\top)^M = \Delta^M$
- $(\bot)^M = \emptyset$
- $(\neg C)^M = \Delta^M \setminus C^M$
- $(C_1 \sqcap C_2)^M = (C_1)^M \cap (C_2)^M$
- $(C_1 \sqcup C_2)^M = (C_1)^M \cup (C_2)^M$
- $(\exists R.C)^M = \{x \in \Delta^M | \exists y \in \Delta^M, (x, y) \in R^M \wedge y \in C^M\}$
  Exemple 1 : avoir au moins un enfant qui est humain : $\exists$a_enfant.Humain
  Exemple 2 : ne pas avoir d'enfants qui sont humains (on peut toutefois en avoir qui ne soient pas humains) : $\neg\exists$a_enfant.Humain
- $(\forall R.C)^M = \{x \in \Delta^M | \forall y \in \Delta^M, (x, y) \in R^M \rightarrow y \in C^M\}$
  Exemple 3 : avoir uniquement des enfants humains (mais ne pas nécessairement en avoir) : $\forall$a_enfant.Humain

2 Logique de description $\mathcal{SHOIN}$

Par rapport à $\mathcal{ALC}$, on ajoute les notions suivantes:

- Nouveaux axiomes dans la Tbox:
  - Axiomes de transitivité de rôle (R^+, $\mathcal{S} = \mathcal{ALC} + R^+$) : Trans R
  - Axiomes d'inclusion de rôles ($\mathcal{H}$) : $R_1 \sqsubseteq R_2$
- Nouveaux constructeurs de concepts
  - Concepts nominaux ($\mathcal{O}$) : $\{i\}$
  - Restrictions de cardinalité ($\mathcal{N}$) : $\geq nR$
- Constructeur de rôle inverses ($\mathcal{I}$) : R^{-1}

2.1 Expressions

A désigne un concept atomique (élément de $\mathbb{C}$), P désigne un rôle atomique (élément de $\mathbb{R}$), i désigne une constante (élément de $\mathbb{I}$) et n un entier naturel strictement positif.

Un *rôle* R et un *concept* C sont définis comme :

$$R ::= P|P^{-1} \qquad C ::= A|\top|\bot|\neg C|C \sqcap C|C \sqcup C|\exists R.C|\forall R.C|\{i\}|\geq nR|\leq nR$$

2.2 Base de connaissances

- Tbox - Axiomes ontologiques (C_1, C_2 sont des concepts, P un rôle atomique et R un rôle quelconque):

  Axiome d'équivalence de concepts: $C_1 \equiv C_2$

  Axiome d'inclusion de concepts : $C_1 \sqsubset C_2$

  Axiome de transitivité de rôle : Trans P

  Axiome d'inclusion de rôles: $R \sqsubseteq P$
- ABox : inchangée par rapport à $\mathcal{ALC}$

2.3 Sémantique

Etant donné une structure $M = (\Delta^M, .^M)$, la sémantique des expressions existantes en $\mathcal{ALC}$ est inchangée.

Pour les expressions de rôles :

- $(P^{-1})^M = \{(x, y) \in \Delta^M \times \Delta^M, (y, x) \in P^M\}$
  Exemple 4 : avoir uniquement des parents humains (mais ne pas nécessairement en avoir) : $\forall$a_enfant^{-1}.Humain

Pour les expressions de concepts:

- $(\geq n\ R)^M = \{x \in \Delta^M | Card(\{y \in \Delta^M, (x, y) \in R^M\}) \geq n\}$
- $(\leq n\ R)^M = \{x \in \Delta^M | Card(\{y \in \Delta^M, (x, y) \in R^M\}) \leq n\}$
  Exemple 5 : avoir moins de 3 enfants (ie 3 ou -) : ≤ 3 a_enfant
  Exemple 6 : avoir plus de 2 enfants (ie 2 ou +) : ≥ 2 a_enfant
- $(\{a\})^M = \{a^M\}$
- Par extension $\{a_1, a_2, \ldots a_n\}$ est utilisé comme raccourci de $\{a_1\} \sqcup \{a_2\} \ldots \sqcup \{a_n\}$ et on a $(\{a_1, a_2, \ldots a_n\})^M = \{a_1^M, a_2^M, \ldots a_n^M\}$
  Exemple 7 : les parents de Robert (avoir Robert comme enfant) : a_enfant.{Robert} Exemple 8 : les parents de Robert ou d'Alex (avoir Robert ou Alex comme enfant) : a_enfant.{Robert, Alex}
  Exemple 9 : les parents de Robert et d'Alex (avoir Robert et Alex comme enfant) : a_enfant.{Robert}$\sqcap$a_enfant.{Alex}

Pour les nouveaux axiomes ontologiques ;

- M est un modèle de Trans R ssi R^M est une relation binaire transitive (ie pour tout x, y, z de Δ^M tels que $(x, y) \in R^M$ et $(y, z) \in R^M$, on doit aussi avoir $(x, z) \in R^M$).
  Exemple 10 : la relation de contenance doit être transitive / si une chose en contient une autre, elle en contient aussi le contenu : Trans contient

- M est un modèle de $R \sqsubseteq P$ ssi $R^M \subseteq P^M$.
  Exemple 11 : Le responsable d'une UE enseigne toujours dans cette UE :
  responsableUE $\sqsubseteq$ enseigne

3 Algorithme tableau en $\mathcal{ALC}$

3.1 Principe général

1. Déploiement dynamique de la Tbox : voir descriptions ci-dessous dans la section 3.2.

2. Mise sous forme normale négative de la ABox obtenue : toutes les négations sont devant des concepts atomiques et non devant des expressions de concepts, en utilisant itérativement les règles de réécriture suivantes

$$\begin{array}{rcl} \neg\top & \longrightarrow & \bot \\ \neg(\neg C) & \longrightarrow & C \\ \neg(C \sqcap D) & \longrightarrow & \neg C \sqcup \neg D \\ \neg(C \sqcup D) & \longrightarrow & \neg C \sqcap \neg D \end{array} \qquad \begin{array}{rcl} \neg(\forall R.C) & \longrightarrow & \exists R.\neg C \\ \neg(\exists R.C) & \longrightarrow & \forall R.\neg C \end{array}$$

3. Application itérative des règles de tableau

   Règles conjonctives

Règle	**condition**	**action**
$R_\sqcap$	$a : C \sqcap D$	ajout de $a : C$ et de $a : D$
$R_\forall$	$a : \forall R.C$ et $a, e : R$	ajout de $e : C$
$R_\exists$	$a : \exists R.C$ et aucun e	ajout de $a, b : R$ et de $b : C$
	tel que $e : C$ et $a, e : R$	où b nouvelle constante

   Règle disjonctive

Règle	**condition**	**action**
$R_\sqcup$	$a : C \sqcup D$	Duplication en $\mathcal{A}_g$, $\mathcal{A}_d$
		Ajout de $a : C$ à $\mathcal{A}_g$ et de $a : D$ à $\mathcal{A}_d$

4. Identification des feuilles fermées, qui contiennent des CLASH = des contradictions : feuille contenant soit $a : \bot$, soit $a : C$ et $a : \neg C$

5. Analyse : la ABox est cohérente ssi le tableau contient au moins une feuille complète ouverte.

   Elle est incohérente ssi toutes les feuilles sont fermées

3.2 Déploiement de Tbox acyclique

1. Prétraitement

   (a) réécriture des axiomes définitoires simples, c'est-à-dire de la forme $A \equiv C$ avec A concept atomique

   $\longrightarrow$ remplacer toutes les occurrences de A par C dans la Abox

   (b) réécriture des axiomes d'inclusion généraux, c'est-à-dire de la forme $C \sqsubseteq D$ où C et D sont non atomiques

   $\longrightarrow$ remplacer l'axiome par $\top \sqsubseteq \neg C \sqcup D$ dans la Tbox

(c) mise sous forme normale négative de la Tbox

2. Déploiement dynamique : à mettre en œuvre initialement
puis à chaque ajout d'assertion de concept

(a) inclusion simple : pour chaque occurrence $a : A$ dans la Abox avec $A \sqsubseteq C$ dans la Tbox
$\longrightarrow$ ajouter $a : C$ dans la Abox

(b) inclusion générale : pour chaque constante a apparaissant dans la Abox et chaque inclusion $\top \sqsubseteq C$
$\longrightarrow$ ajouter $a : C$ dans la Abox

Remarque : selon le type de la Tbox, notamment si elle est définitoire ou simple, le principe général du déploiement précédent peut être optimisé : certaines étapes disparaissent. Par exemple, il n'est pas nécessaire de faire de déploiement dynamique pour une Tbox définitoire.

Méthode des tableaux pour $\mathcal{ALC}$ avec Lotrec

Le but de ce TME est de programmer une nouvelle logique dans Lotrec pour pouvoir faire de la vérification de cohérence de Abox selon une Tbox en $\mathcal{ALC}$, dans le cas où la Abox est sous forme normale négative et où la Tbox est vide.

La façon dont nous proposons de réaliser cette implémentation est inspirée des logiques prédéfinies `Classical-Propositional-Logic` et `S5-explicit-edges`.

Pensez à sauvegarder régulièrement. Vous pouvez utiliser l'interface de Lotrec, ou éditer le fichier xml de votre logique.

Exercice 1 Définition des connecteurs

Au lancement de Lotrec, choisir dans la section `Create your own` l'option `New Logic File` : les onglets `Connectors`, `Rules` et `Strategy` sont donc vides et l'objectif du TME est de les remplir progressivement.

Définir d'abord les connecteurs, qui seront communs à tous les exercices, en les ajoutant par `Add` à l'onglet `Connectors` et en utilisant les noms suivants

Expressions de concepts	
$\neg C$	`not C`
$C \sqcap D$	`inter C D`
$C \sqcup D$	`union C D`
$\forall R.C$	`forall R C`
$\exists R.C$	`exists R C`

Assertions	
$a : C$	`inst A C`
$a, b : R$	`instR A B R`

Axiomes ontologiques	
$C \sqsubseteq D$	`incl C D`

Remarque : pour la partie `Display`, il est possible de faire un copier-coller des symboles depuis le pdf, en utilisant _ pour les variables, par exemple _ ⊓ _ pour la conjonction.

Exercice 2 Initialisations multiples : définition de Abox

On appelle règle d'initialisation l'équivalent de la règle `ExampleOfModelAndFormula` vue dans les TME précédents. Cette règle doit ajouter toutes les assertions de la ABox considérée, par des instructions du type `add w inst aristote PhilospheAncien` où `w` désigne le nœud courant et où on écrit ensuite les formules considérées selon les connecteurs définis dans l'exercice précédent.

De plus, cette règle doit marquer le nœud comme ABox, par l'instruction `mark w ABox`.

Ecrire deux règles de ce type, pour représenter les deux ABox suivantes : la première tirée du TD précédent permet de tester les règles simples, tandis que la deuxième est un nouvel exemple abstrait destiné à tester plus précisément les difficultés liées à la règle $R_\exists$.

$\mathcal{A}_1$:

aristote : PhilosopheAncien ⊓ ∀aEcrit.Serieux
laPoétique_livre2 : ¬Serieux ⊔ Subversif
aristote, laPoétique_livre2 : aEcrit

$\mathcal{A}_2$:

a : C ⊓ ∀R1.(C ⊓ D) ⊓ ∃R1.D
b : ¬C ⊔ F
a, b : R1
c : ∃R2.∃R1.∃R2.(C ⊓ ¬C)

NB Afin de ne pas avoir besoin de modifier la stratégie quand on veut changer l'exemple, et donc la règle d'initialisation choisie, on peut mettre une condition à chaque règle d'initialisation, du type `hasElement w EX1` (respectivement `EX2`) et mettre toutes les règles d'initialisation dans la stratégie. L'utilisateur doit alors dans la partie `Compose your own formula` écrire `EX1` ou `EX2` pour que la stratégie déclenche la règle correspondante.

Exercice 3 Règles du tableau autres que $R_\exists$

Définir les règles du tableau autres que $R_\exists$ (rappelées en annexe) en s'inspirant des règles de `Classical-Propositional-Logic`.

Définir également les règles pour identifier les feuilles fermées.

Exercice 4 Stratégies

Ecrire une stratégie pour la vérification de cohérence et la tester sur la Abox $\mathcal{A}_1$ définie dans l'exercice 1.

Il peut être pertinent de créer des sous-stratégies pour obtenir l'arbre le plus simple possible. En particulier, il est bon de vérifier dès que possible les conditions d'arrêt de ne déclencher la règle $R_\sqcup$ qu'une fois toutes les autres règles déclenchées.

Exercice 5 Cas de la règle $R_\exists$

Pour gérer le connecteur $\exists$, il y a deux difficultés :

- il faut vérifier qu'il n'existe pas déjà une instance qui convienne : c'est l'objectif de la deuxième question.
- il faut pouvoir générer de nouveaux identifiants pour l'éventuelle nouvelle instance : c'est l'objectif de la première question.

1. Génération de nouvelles instances Chaque application de la règle $R_\exists$ demande la création d'une nouvelle instance. On ne peut pas, en Lotrec, utiliser un compteur entier, pour créer successivement `I1`, `I2`, `I3` par exemple. On choisit à la place la notation `sI`, `s(sI)`, `s(s(I))` etc. Il faut de plus savoir quelles sont les instances déjà créées et quelle est la suivante qu'on peut créer.

Pour cela, on crée d'abord un nouvel opérateur unaire `nv` qui, appliqué à une variable `I`, s'affiche sous la forme `sI` (suivant de `I`).

On crée ensuite un nœud, qu'on nommera `v`, lié au nœud `w` qui contient les assertions étudiées, dont l'objectif est de permettre de voir les instances déjà créées. La règle d'initialisation `varBox` de ce "nœud des variables" `v` a pour condition que le nœud `w` soit marqué comme étant une Abox, et pour action de créér un nouveau nœud `v`, de le relier à `w` par une relation `Var` et d'ajouter dans ce nouveau nœud la formule `nv I`.

Quand on a besoin d'une nouvelle instance (typiquement la règle $R_\exists$), on peut récupérer la dernière instance libre `nv` I en vérifiant si la boîte de variable contient `nv` I sans contenir `nv nv` I. On peut alors utiliser `nv` I comme nouvelle variable, et générer la variable suivante (`nv nv` I) dans la boîte de variables.

2. Déroulement du tableau La condition "aucun e tel que $e : C$ et $a, e : R$" n'est pas directement expressible en Lotrec. Elle sert à éviter de générer un individu s'il y en a déjà un qui convient. C'est le même principe que pour la règle `Pos` de la logique `S5-explicit-edges` et on va donc utiliser la même technique : avoir une règle qui marque l'assertion $a : \exists R.C$ comme `[Done]` si c'est le cas, et qui est toujours déclenchée avant la règle $R_\exists$.

Note, on se contentera de vérifier s'il y a une instance e telle que $e : C$ et $a, e : R$ sont explicitement dans la Abox, sans chercher à vérifier si $e : C$ serait entrainé logiquement par la Abox (si par exemple on avait $C = C1 \sqcap C2$ avec $e : C1$ et $e : C2$).

3. Ajout dans la stratégie Il ne faut évidemment pas oublier d'ajouter cette nouvelle règle à la stratégie.

NB Pour des objectifs d'efficacité à discuter, il est pertinent d'appliquer autant que possible les règles $R_\cap$ et $R_\forall$ avant de gérer la règle $R_\exists$.

Tester la stratégie obtenue notamment sur la Abox $\mathcal{A}_2$ définie dans l'exercice 1 et sur d'autres cas.

Pour aller plus loin

Les exercices précédents permettent de définir les connecteurs et de traiter le cas d'une Abox sous forme normale négative (NNF) sans Tbox.

Pour traiter des cas plus généraux, il faut réaliser la mise sous forme NNF dynamique [Exercice 6], avant de permettre la prise en compte d'une Tbox simple [Exercice 7], puis d'une Tbox générale ne contenant que des axiomes d'inclusion [Exercice 8]. D'autres extensions pourraient considérer des Tbox définitoires ou acycliques.

Exercice 6 Mise sous NNF dynamique

Etendre la logique définie précédemment avec un nouveau lot de règles pour gérer des Abox non mises sous forme normale.

On définit pour cela une nouvelle stratégie incluant un nouveau jeu de règles traduisant les principes de réécriture, que l'on rappelle ici:

$$\begin{array}{rcl} \neg(\neg C) & \leftrightarrow & C \\ \neg(C \sqcap D) & \leftrightarrow & \neg C \sqcup \neg D \\ \neg(C \sqcup D) & \leftrightarrow & \neg C \sqcap \neg D \end{array} \qquad \begin{array}{rcl} \neg(\forall R.C) & \leftrightarrow & \exists R.\neg C \\ \neg(\exists R.C) & \leftrightarrow & \forall R.\neg C \end{array}$$

On note qu'il est difficile en Lotrec de mettre la Abox complètement sous forme normale dès le départ car chaque règle s'applique sur la Abox complète, dans laquelle les formules sont imbriquées dans des assertions. La solution est de faire cela dynamiquement en ne traitant les négations que lorsqu'elles apparaissent tête d'une assertion (c'est-à-dire quand la boîte comprend une assertion $a : \neg(E)$ où E est un concept complexe).

Pour tester ces nouvelles règles, on crée une nouvelle règle d'initialisation conditionnée par `EX2` :

$\mathcal{A}_3$:

$\texttt{a} : \texttt{C} \sqcap \neg(\exists \texttt{R1}.(\neg\texttt{C} \sqcup \neg\texttt{D}) \sqcup \forall \texttt{R1}.\neg\texttt{D})$
$\texttt{b} : \neg\texttt{C} \sqcup \texttt{F}$
$\texttt{a},\texttt{b} : \texttt{R1 c} : \neg\forall \texttt{R2}.\forall \texttt{R1}.\forall \texttt{R2}.\top$

Exercice 7 Prise en compte d'une Tbox simple

Etendre la logique précédente de façon à pouvoir prendre en compte une Tbox simple, c'est-à-dire une Tbox ne contenant que des inclusions de la forme $C \sqsubseteq E$ où C est un concept atomique.

On rappelle que pour prendre en compte une telle Tbox, il faut *déployer les axiomes d'inclusion simples* comme suit : Pour chaque triplet (a, C, D) (avec a constante, C concept atomique et D expression de concept) tel que la Abox contienne l'assertion $a : C$ et que la Tbox contienne l'inclusion $C \sqsubseteq D$, on **ajoute** (si non présent) l'assertion $a : D$. On itère tant que cela génère des ajouts.

Ici, il est nécessaire, après le déploiement initial, de refaire un déploiement *chaque fois qu'on ajoute une assertion $a : C$ où C concept est un concept atomique apparaissant à gauche d'une inclusion.*

Pour représenter la Tbox, la méthode est de créer un nouveau nœud lors de l'initialisation, de le marquer `[Tbox]` et de le relier à la Abox par la relation `TboxDe`.

Pour tester l'algorithme on utilisera l'exemple de l'exercice 4 de la feuille de TD, rappelée ci-dessous (utilisant la condition `EX3` pour sa règle d'initialisation):

$\mathcal{T}_4$:

$\texttt{Herbivore} \sqsubseteq \exists \texttt{mange}.\texttt{Plante}$
$\texttt{Carnivore} \sqsubseteq \exists \texttt{mange}.\texttt{Animal}$
$\texttt{Sauvage} \sqsubseteq \texttt{Animal} \sqcup \texttt{Plante}$

$\mathcal{A}_4$: $e : \exists \texttt{mange}.\texttt{Sauvage} \sqcap \neg(\texttt{Herbivore} \sqcup \texttt{Carnivore})$

Pour récapituler, afin d'étendre la logique, il faut définir des règles pour l'initialisation et le déploiement et une stratégie qui les incluent correctement.

Exercice 8 Prise en compte d'une Tbox générale

Proposer une dernière extension de la logique précédente de façon à pouvoir prendre en compte une Tbox générale, c'est-à-dire une Tbox pouvant contenir des inclusions de la forme $E_1 \sqsubseteq E_2$ où E_1 est un concept non atomique. Cette logique devra contenir une règle d'initialisation pour l'exemple de l'exercice 5, rappelé ci-dessous:

$\mathcal{T}_5$: $\forall \texttt{mange.Animal} \sqsubseteq \texttt{CarnivoreStrict}$, $\neg\texttt{Plante} \sqcap \neg\texttt{Animal} \sqsubseteq \top$ $\quad$ $\mathcal{A}_5$: $e : \neg\exists\texttt{mange.Plante} \sqcap \neg\texttt{CarnivoreStrict}$

Prise en compte d'une Tbox générale:

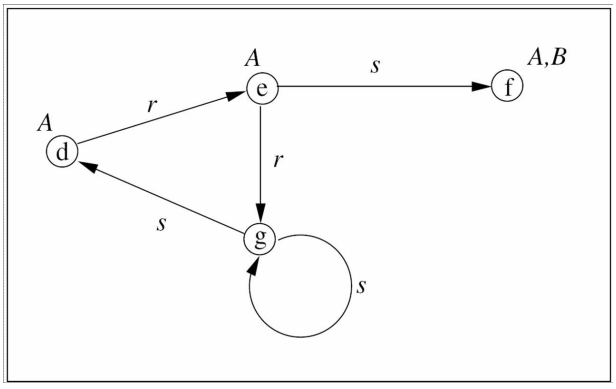
- *Réécriture des axiomes généraux.* Chaque axiome d'inclusion générale $C \sqsubseteq D$ est remplacé par sa réécriture $\top \sqsubseteq \neg C \sqcup D$. On met ensuite la Tbox sous NNF.
- *Déploiement des axiomes généraux*:
  Pour chaque constante a apparaissant dans la Abox, on **ajoute** (si non présent) l'assertion $a : \top$, puis on effectue un déploiement des axiomes $\top \subseteq E$ comme dans le cas d'une Tbox simple.
- Ici, il est nécessaire, après le déploiement initial, de refaire un déploiement *chaque fois qu'on ajoute une nouvelle constante (règle $R_\exists$).*

LRC - TD 3

Exercice 1 Interprétations

Soit une logique sur un ensemble de concepts $\mathbb{C} = \{A, B\}$ et un ensemble de rôles $\mathbb{R} = \{r, s, t\}$.

On considère la structure M avec $|M| = \{d, e, f, g\}$ représentée graphiquement par la figure ci-dessus.



1. Expliciter la fonction d'interprétation $.^M$ (en donnant les interprétations de chaque concept atomique, rôle atomique et constante)
2. Pour chacune des expressions de concepts C suivantes en $\mathcal{ALC}$, lister tous les éléments x de $|M|$ tel que $x \in C^M$:
   (a) $A \sqcap B$
   (b) $\exists s.\top$
   (c) $\forall s.A$
   (d) $\exists s.\neg A$
   (e) $\forall t.A \sqcap \forall t.\neg A$
   (f) $\neg\exists r.(\neg A \sqcap \neg B)$
3. Faire de même pour ces expressions de concepts C en $\mathcal{SHOIN}$:
   (a) $\geq 2s$
   (b) $\forall s.\neg\{\mathtt{d}, \mathtt{e}\}$
   (c) $\exists r^{-1}.\{\mathtt{d}\}$
4. Cette structure vérifie-t-elle l'axiome $\mathcal{SHOIN}$ Trans s ? quid de Trans r ?

Exercice 2 Représentations

On considère les concepts et les rôles suivants

- Concepts atomiques : `Humain`, `Chien`, `Masculin`, `Feminin`, `NonBinaire`, `Homme`, `Femme`, `Mere`, `Frere`
- Rôles atomiques :
  - `aSoeur` : aSoeur(X,Y) signifie que X a pour sœur Y
  - `aEnfant` : aEnfant(X,Y) signifie que X a un enfant Y
- Constantes : `Pierre`, `Jean`, `Alex`

1. Représenter en $\mathcal{ALC}$ les connaissances suivantes :

(a) Pierre est un humain de genre masculin.

(b) Pierre a une sœur.

(c) Les hommes sont les humains de genre masculin.

(d) Les mères ont des enfants.

(e) Tous les enfants d'un humain sont humains.

(f) On ne peut pas être à la fois un humain et un chien.

(g) On ne peut pas être à la fois masculin et féminin, sauf à être non binaire.

(h) Tout homme qui a une sœur est un frère.

2. Représenter en $\mathcal{SHOIN}$ les connaissances suivantes en considérant en plus les rôles `aSupHierarch` (x a pour supérieur hiérarchique y) et `collègue` (x a pour collègue y)

   (a) La mère d'un frère a au moins 2 enfants.

   (b) Pierre et Jean ont les mêmes parents.

   (c) Pierre et Alex ont les mêmes enfants.

   (d) Les supérieurs hiérarchiques de nos supérieurs hiérarchiques sont aussi nos supérieurs hiérarchiques.

   (e) Les supérieurs hiérarchiques sont des collègues.

3. Représenter les concepts `MereDeFilles` (mère qui n'a que des enfants filles), `GrandMere` et `FilleUnique` à partir des concepts déjà existants en indiquant dans quelle logique vous vous placez ($\mathcal{ALC}$ ou $\mathcal{SHOIN}$).

1 Le langage *Prolog*

1.1 Syntaxe

Syntaxe des éléments de base

*Prolog*est un langage de programmation fondé sur la logique des prédicats du premier ordre [1].

- Les individus, qui correspondent aux constantes, sont représentés par des mots qui commencent par une minuscule.
- Les variables sont représentées par des mots qui commencent par une Majuscule.
- Les prédicats sont représentés par des mots qui commencent par une minuscule.

Ainsi, dans la proposition `frere(X,charlemagne)`, le prédicat est `frere`, d'arité 2 : il est appliqué à la variable `X` et à la constante `charlemagne`.

Syntaxe de la base de connaissances

Les règles sont de la forme `prop :- prop1, prop2, prop3.` qui correspond à la formule de LPPO $(prop1 \wedge prop2 \wedge prop3) \rightarrow p$.

Le prédicat à gauche de `:-` est la *tête* de la règle et les propositions à droite définissent le *corps* de la règle. Les propositions du corps sont séparées par des virgules.

Prolog exploite la représentation sous forme normale conjonctive des formules : ainsi, la formule $((prop_1 \wedge prop_2 \wedge prop_3) \vee (prop_4 \wedge prop_5)) \rightarrow prop$, dont la forme normale conjonctive est $(\neg prop_1 \vee \neg prop_2 \vee \neg prop_3 \vee prop) \wedge (\neg prop_4 \vee \neg prop_5 \vee prop)$, est représentée par les 2 règles

```
prop :- prop1, prop2, prop3.
prop :- prop4, prop5.
```

Chaque règle correspond à une clause de Horn, c'est-à-dire une clause comportant au plus un littéral positif.

Les faits correspondent aux clauses de Horn positives, qui contiennent un littéral positif et aucun littéral négatif. Ils sont représentés par des règles dont le corps est vide, c'est-à-dire de la forme : `prop.`

Stockage La base de connaissances, qui contient les faits et les règles, est écrite dans un fichier texte, dont le nom a pour extension `pl`. Il est préférable que le nom du fichier commence par une minuscule (cf ci-dessous la procédure de chargement du fichier).

Tout éditeur de texte peut être utilisé (l'éditeur `emacs` propose une coloration syntaxique pour prolog, si on utilise le mode prolog, en tapant alt+x prolog-mode).

Attention: il faut regrouper dans le fichier tous les faits et règles ayant la même tête, sinon l'interprète prolog provoque une erreur lors du chargement du fichier.

Les symboles `/* */` permettent de mettre des commentaires dans le fichier.

1.2 L'interprète `swiprolog`

L'interprète prolog sous Linux s'appelle `swipl`. Lorsqu'on lance le programme, on entre en mode "requête" indiqué par l'invite `?-`

Toutes les commandes se terminent par un point.

[1] Les versions actuelles de prolog permettent de définir des formules d'ordre supérieur, mais nous n'aborderons pas ce point dans le cadre de LRC. Il existe aussi une variante modale de prolog.

- `[nom].` ou `consult(nom).` ou `['nom.pl'].` : charger le fichier `nom.pl`.

  La commande `[nom].` ne convient pas si le nom du fichier commence par une majuscule : prolog considère que l'on fait référence à une variable et provoque une erreur. Elle ne convient pas non plus si le nom du fichier contient des caractères spécifiques (tiret, point, etc).

  A chaque nouveau chargement d'un fichier existant, les faits et les règles précédents sont écrasés par les nouvelles définitions.

- `listing.` : visualiser l'état courant de la base de connaissances, c'est-à-dire savoir quels sont les faits et les règles que prolog a en mémoire.

- `help.` : afficher la fenêtre d'aide et la liste des prédicats prédéfinis dans SWI-Prolog.

- `help(pred).` : afficher l'aide concernant le prédicat `pred`.

- `trace.` : suivre pas à pas l'évaluation des requêtes.

- `notrace.` : quitter le mode trace.

- `halt.` : quitter Prolog.

1.3 Requêtes

Une fois le programme chargé, on peut entrer une requête, c'est-à-dire une proposition de la LPPO contenant des variables, se terminant par ".".

Considérons par exemple le cas où un fichier `toto.pl` contient les faits

```
toto(a,b).
toto(c,b).
```

Chargement du fichier

```
?- [toto].
//1 compiled 0.01 sec, 624 bytes
true.
```

Requête sans solution il n'existe pas de valeur du second argument de `toto` telle que le prédicat soit vérifié avec en premier argument `b`. Prolog répond `false`.

```
?- toto(b,X).
false.
```

Requête avec une ou plusieurs solutions prolog indique l'instanciation effectuée pour que le prédicat soit vérifié sous les conditions exprimées par la requête :

```
?- toto(a,X).
X = b.
```

S'il n'y a qu'une instanciation possible, l'interprète rend la main. Sinon, il attend une saisie de l'utilisateur :

```
?- toto(X,b).
X = a
```

La touche "Entrée" termine, la touche ";" donne successivement les autres solutions.

Tester les exemples précédents en mode trace, pour observer les étapes d'unification effectuées (après renommage des variables, de la forme $\_G2673$ ou un autre entier).

1.4 Tests d'égalité

Il existe en prolog plusieurs types d'égalité, faisant référence à des concepts différents. Ici, deux seulement sont rappelés, ceux qui portent sur des valeurs numériques sont omis.

- unifiabilité `=` et non-unifiabilité `\=` : la comparaison entre termes réussit s'il existe une instanciation des variables présentes qui rende égaux les deux termes. Ainsi

```
?- p(a,q(X)) = p(Y,q(b)).
X = b,
Y = a.
?- p(a,q(X)) = p(Y,r(b)).
false.
```

- identité `==` et non-identité `\==` : la comparaison entre termes réussit s'ils sont identiques ou s'ils sont liés à la même valeur :

```
?- X == a.
  false.
?- X = a.
   X = a.
?- X = 3, Y = 3, X == Y.
   X = Y, Y = 3.
?- X = 3, Y = 3, X \== Y.
   false.
```

2 Exercices

Exercice 1

On considère le programme formé des trois clauses suivantes :

```
r(a,b).
r(f(X),Y) :- p(X,Y).
p(f(X),Y) :- r(X,Y).
```

1. Calculer, à la main, les réponses aux requêtes `r(f(f(a)),b)` et `p(f(a),b)` en indiquant à chaque étape les inférences par résolution et les unifications réalisées.
2. Tester avec prolog, examiner et commenter la trace.

Exercice 2

On considère le programme formé des trois clauses suivantes :

```
r(a,b).
q(X,X).
q(X,Z) :- r(X,Y),q(Y,Z).
```

1. Calculer, à la main, les réponses à la requête `q(X,b)` puis à la requête `q(b,X)`.
2. Tester avec prolog, examiner et commenter la trace.

Exercice 3 Raisonnement simple

1. Donner la base de connaissances prolog représentant les assertions suivantes, en utilisant des règles et des faits

   > Les étudiants sérieux révisent leurs examens. Un étudiant consciencieux fait toujours ses devoirs pour le lendemain. Les étudiants qui révisent leurs examens réussissent. Les étudiants qui font leurs devoirs pour le lendemain sont sérieux. Pascal et Zoé sont des étudiants consciencieux.

2. Donner la requête prolog qui permet de répondre à la question “qui va réussir?”.

3. Calculer la réponse attendue à la main et vérifier que prolog donne le résultat attendu.

Exercice 4 Relations familiales

Il est demandé de joindre au compte-rendu de TME le fichier `.pl` contenant les réponses aux questions suivantes, ainsi que des commentaires indiquant les tests réalisés et les résultats obtenus.

1. Définir quelques faits de type `pere` et `mere`, comme par exemple `pere(pepin, charlemagne)` (Pépin est le père de Charlemagne) ou `mere(berthe,charlemagne)`.

   N.B. Pour faciliter les vérifications des prédicats suivants, il est conseillé d'utiliser une famille que vous connaissez bien (typiquement la vôtre...).

2. Définir le prédicat `parent`/2 à partir des prédicats `pere`/2 et `mere`/2.

   Dans ces notations, l'entier après le slash indique l'arité des prédicats concernés : ici, les trois prédicats considérés ont deux arguments.

   `parent(X, Y)` doit être satisfait si `X` est le père ou la mère de `Y`.

   Ainsi, la requête `parent(X, charlemagne)` doit être satisfaite pour les deux instanciations `X = pepin` et `X = berthe`.

3. Tester (avec une base de faits plus conséquente) différents types de requêtes, comme par exemple `parent(charlemagne, X)`, `parent(pepin, Y)` ou `parent(A, B)`.

4. Définir et tester le prédicat `parents`/3, tel que `parents(X,Y,Z)` est satisfait si `X` est le père de `Z` et `Y` est la mère de `Z`.

5. Définir et tester les prédicats `grandPere`/2 et `frereOuSoeur`/2.

6. Définir et tester le prédicat `ancetre`/2 tel que `ancetre(X,Y)` est satisfait si `X` est un ancêtre de `Y`.

   Pour ce prédicat, faire varier la position des différentes propositions utilisées et commenter les résultats obtenus.

Exercice 5 Statuts entrée et sortie des variables

Il est demandé de joindre au compte-rendu de TME le fichier `.pl` contenant les réponses aux questions suivantes, ainsi que des commentaires indiquant les tests réalisés et les résultats obtenus.

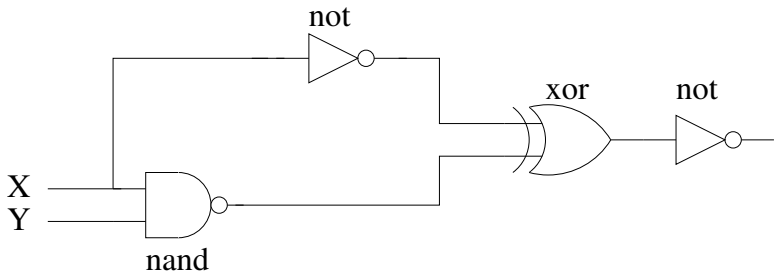
1. Définir les prédicats `et`/3, `ou`/3 et `non`/2 correspondant aux connecteurs logiques par un ensemble de **faits**, en utilisant les constantes 0 et 1 pour les entrée et sortie de chaque connecteur.

   Ainsi `et(0,1,0).` définit que le prédicat `et` est vérifié pour le triplet (0,1,0), ce qui représente le fait que le et logique de 0 et 1 vaut 0. 4 faits de ce type définissent donc le prédicat complètement.

   Particularité importante de prolog Il faut noter que prolog ne définit pas des fonctions qui prennent des arguments et *renvoient* des résultats, contrairement aux langages de programmation classique : il énonce le fait qu'*il est vrai qu'il existe une relation*, définie par le nom de la fonction, entre les paramètres d'entrée et de sortie.

2. Tester différents types de requêtes de la forme `et(X,Y,1)`, `et(0,0,R)` ou `et(X,Y,R)`.

3. Définir le prédicat `circuit`/3 tel que `circuit(X, Y, Z)` est satisfait si et seulement si `X` et `Y` correspondent à des valeurs en entrée pour lesquelles `Z` est la valeur de sortie du circuit indiqué sur la figure ci-dessous.



4. Donner la requête permettant de construire la table de vérité du circuit.

   NB la table de vérité attendue est celle de l'implication.

5. Tester différents types de requêtes en faisant varier le statut variable/valeur fixée de chacun des arguments.

LRC - TD 4

Résolution

1 Cas propositionnel

Exercice 1 – Diagnostic médical simpliste, encore

On dispose des connaissances suivantes sur la grippe :

(a) La fièvre est définie comme une température supérieure à 38°.

(b) Les patients qui ont la grippe doivent prendre du tamiflu.

(c) Les patients qui ont de la fièvre et qui toussent ont la grippe.

(d) Le patient tousse et a une température supérieure à 38°.

1. Formaliser ces connaissances en utilisant les variables propositionnelles `grippe`, `tamiflu`, `fièvre`, `toux`, `sup38`.
2. En utilisant la méthode de résolution, montrer que le patient doit prendre du tamiflu.

2 Cas LPPO

Exercice 2 – Unification

On considère un langage termes où x, y, z, u, v, w sont des symboles de variable, a un symbole de constante, f, g des symboles de fonction d'arités respectives 1 et 2, et p un symbole de prédicat d'arité 3.

Pour chacun des couples (F_1, F_2) suivants, indiquer si F_1 et F_2 sont unifiables.

1. $F_1 : p(x, f(y), g(f(u), w))$ et $F_2 : p(z, f(f(a)), g(f(g(a, z)), v))$
2. $F_1 : p(x, f(x), g(f(y), x))$ et $F_2 : p(z, f(f(a)), g(f(g(a, z)), v))$
3. $F_1 : p(x, f(x), g(f(x), x))$ et $F_2 : p(z, f(f(a)), g(f(g(a, z)), v))$

Exercice 3

On considère un langage comportant les constantes c, d, les prédicats unaires E, I, S et le prédicat binaire R. Montrer par la méthode de résolution que $C_1, C_2, C_3, C_4 \models E(c)$.

$C_1 = I(d)$
$C_2 = S(d)$
$C_3 = \neg I(y) \vee R(c, y)$
$C_4 = E(x) \vee \neg S(y) \vee \neg R(x, y)$

Exercice 4

Soit un langage où a est un symbole de constante, f un symbole de fonction unaire, P et Q deux symboles de prédicat unaire et R et S deux symboles de prédicat binaire. Montrer, par la méthode de résolution, que les ensembles suivants ne sont pas satisfiables.

1. $\{\neg P(x) \vee P(f(x)), P(f(a)), \neg P(f(f(a)))\}$
2. $\{\neg R(x, y) \vee \neg R(y, z) \vee S(x, z), R(f(x), x), \neg S(y, a)\}$

Exercice 5 – Résolution de règles

On considère la base de règles suivantes :

```
q(f(X)) :- p(X).
p(f(a)).
p(X):-p(f(X)).
```

Développer l'arbre SLD pour la requête `?-q(f(Y)).`

3 Petit problème (annales)

Exercice 6 – Représentation et mise sous forme clausale

1. En faisant appel aux prédicats unaires $peintre(x)$ et $tableau(x)$ ainsi qu'aux prédicats binaires $a\_peint(x, y)$, $inspiré\_par(x, y)$, $dans\_série(x, y)$ et $eq(x, y)$, traduire les propositions suivantes

   (a) "Les Ménines" est un tableau peint par Velázquez.

   (b) Tous les peintres ont peint au moins un tableau.

   (c) Les tableaux sont peints par un peintre et un seul.

   (d) "Les Ménines" est une série de tableaux peints par Picasso.

   (e) Picasso s'est inspiré de Velázquez, mais Picasso n'est pas Velázquez.

   (f) Velázquez n'a peint aucun tableau de la série "Les Ménines"

2. Mettre la théorie sous forme clausale

Exercice 7 – Résolution

Soit $S = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{12}\}$ l'ensemble des 12 clauses suivantes où x, y et z sont des variables universellement quantifiées et a, b, $"LesMénines"$, $Picasso$ et $Vélazquez$ des constantes:

C_1: $\neg tableau(x) \vee \neg a\_peint(y, x) \vee \neg a\_peint(z, x) \vee eq(y, z)$
C_2: $tableau("LesMénines")$
C_3: $\neg tableau(x) \vee a\_peint(b, x)$
C_4: $inspiré\_par(Picasso, Vélazquez)$
C_5: $a\_peint(Velázquez, "LesMénines")$
C_6: $\neg dans\_série(x, "LesMénines") \vee tableau(x)$
C_7: $\neg peintre(x) \vee a\_peint(x, a)$
C_8: $\neg dans\_série(x, "LesMénines") \vee a\_peint(Picasso, x)$
C_9: $\neg eq(Picasso, Velázquez)$
C_{10}: $\neg peintre(x) \vee tableau(a)$
C_{11}: $\neg tableau(x) \vee peintre(b)$
C_{12}: $\neg tableau(x) \vee \neg dans\_série(x, "LesMénines") \vee \neg a\_peint(Vélazquez, x)$

Montrer par réfutation en utilisant la règle de résolution que

$$C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11} \vdash C_{12}$$

Algorithme d'unification [Martelli et al 1982]

Unification de $P(t_1, \ldots, t_k)$ et $P(t'_1, \ldots, t'_k)$

1. Initialisation. $G \leftarrow \{t_1 \doteq t'_1, \ldots, t_k \doteq t'_k\}$ et $\sigma = \emptyset$
2. Tant que G non vide, choisir $C = (\alpha \doteq \beta)$ dans G et:
   (a) (swap) Si α est une fonction et β une variable, les inverser
   NB: Les constantes sont des fonctions d'arité 0
   (b) (del) Si α et β sont égaux, retirer C de G
   (c) Si $\alpha = f(t_1, \ldots, t_i)$ est une fonction
   - (conflict) Si $\beta = g(s_1, \ldots, s_j)$ est une fonction différente ($g \neq f$ ou $i \neq j$), renvoyer $\perp$ (échec)
   - (decomp) Sinon ($\beta = f(s_1, \ldots, s_i)$), retirer C de G et y ajouter $\{t_1 \doteq s_1, \ldots, t_i \doteq s_i\}$

   (d) Si $\alpha = X$ est une variable :
   - (check) Si X apparaît dans β : renvoyer $\perp$ (échec)
   - (eliminate) Sinon, retirer C de G, remplacer X par β dans toutes les expressions de G et σ, puis ajouter $\{X \mapsto \beta\}$ à σ.
3. Renvoyer σ

1 Rappels de quelques éléments de prolog (suite)

Listes Une liste est une suite finie d'objets, de taille quelconque. Elle est représentée entre crochets, les éléments étant séparés par des virgules : `[1, 2, 3, 4]` pour une liste d'entiers, `[a, b, c]` pour une liste de caractères, `[]` pour la liste vide. Cette structure peut être définie récursivement par

- la liste vide, représentée par `[]`.
- si `T` est un terme et `L` une liste, `[T | L]` représente la liste de premier élément `T` (tête de liste), suivi de `L` (queue de liste). L'opérateur `|`, appelé **cons** permet de construire les listes.

Ainsi, les symboles `[]` et `|` peuvent être vus comme des symboles de fonction permettant de construire des termes.

Remarque : un élément d'une liste peut être un terme quelconque, par exemple une liste (voir les exemples ci-dessous).

Unification Comme rappelé dans le TME précédent, l'opérateur `=` ne correspond pas à une instruction d'affectation comme c'est le cas classiquement, mais à un *test d'unification*. Le test est satisfait s'il existe une instanciation des variables présentes qui rend égales les deux expressions, il échoue sinon.

Par exemple, la requête `toto(X, a) = toto(b,c)` échoue, car aucune valeur de `X` ne permet de rendre les deux expressions égales. Par contre, `toto(X,c) = toto(b,c)` réussit et rend `X=b`.

2 Exercices

Exercice 1 - Listes et unification

Calculer sur papier le résultat que prolog renvoie pour les requêtes suivantes, selon les principes de construction de liste et d'unification résumés ci-dessus, puis vérifier, en utilisant prolog, que vos réponses sont exactes.

1. `?- [a, [b, c], d] = [X].`
2. `?- [a, [b, c], d] = [X, Y, Z].`
3. `?- [a, [b, c], d] = [a | L].`
4. `?- [a, [b, c], d] = [X, Y].`
5. `?- [a, [b, c], d] = [X | Y].`
6. `?- [a, [b, c], d] = [a, b | L].`
7. `?- [a, b, [c, d]] = [a, b | L].`
8. `?- [a, b, c, d | L1] = [a, b | L2].`

Exercice 2 - Prédicats de manipulation classique

Il existe en prolog les prédicats `append/3`, `delete/3`, et `reverse/3`. Il est demandé ici de les reprogrammer (et donc de ne pas les utiliser).

Dans chaque cas, tester les prédicats en faisant varier les statuts entrée/sortie des arguments.

1. Ecrire le prédicat `concatene`/3 tel que `concatene(X,Y,Z)` est satisfait si `Z` est la concaténation des deux listes `X` et `Y`. Ainsi, `concatene([a,b,c],[d],L2).` doit conduire à l'unification `L2 = [a,b,c,d]`. Remarquer que ce prédicat permet de décomposer une liste donnée en sous-listes quand on fait varier le statut entrée/sortie des paramètres.

2. Ecrire le prédicat `inverse`/2 tel que `inverse(L1,L2)` est satisfait si la liste `L2` est l'inverse de la liste `L1`.

   Ainsi, `inverse([a,b,c,d],L2).` doit conduire à l'unification `L2 = [d, c, b, a]`.

3. Ecrire le prédicat `supprime`/3 tel que `supprime(X,Y,Z)` est satisfait si `Z` est la liste `X` de laquelle toutes les occurrences de `Y` ont été supprimées.

   Ainsi `supprime([a,b,a,c], a, L).` doit unifier `L` avec la liste `[b,c]`.

4. Ecrire le prédicat `filtre`/2 tel que `filtre(L1,L2,L3)` est satisfait si `L3` est la liste `L1` obtenue après suppression de tous les éléments qui apparaissent dans la liste `L3`.

   Ainsi `filtre([1,2,3,4,3,4,2,1],[2,4], L)` doit unifier `L` avec la liste `[1, 3, 3, 1]`.

Exercice 3 – Palindromes

Un palindrome est une mot (ou une portion de phrase) qui se lit de la même façon de gauche à droite et de droite à gauche, comme par exemple "non", "Laval", "Esope reste ici et se repose" (on ne compte pas les espaces, ni les majuscules ni les accents). On souhaite écrire ici des prédicats permettant de tester si des listes de lettres correspondent à des palindromes ou non.

1. Définir un prédicat `palindrome/1` qui est satisfait si un mot, représenté par la liste de ses lettres (minuscules, sans espaces ni accents), est un palindrome.

   Ainsi `palindrome([l,a,v,a,l])` doit réussir, `palindrome([n,a,v,a,l])` doit échouer.

2. Définir un prédicat `palindrome2/1` identique à `palindrome` mais qui n'utilise pas la fonction `inverse` : il n'est en fait pas nécessaire de parcourir toute la liste pour l'inverser, on peut s'arrêter dès qu'on rencontre un problème.

Intervalles d'Allen

1 Problème (annale ER2 2021)

On considère la situation suivante :

Alors qu'il est chez lui, Jean reçoit une notification d'envoi d'un colis. Il reste alors chez lui jusqu'à finalement recevoir une notification de non-livraison pour cause d'absence. Peut-on en déduire que le livreur n'a pas vérifié qu'il était présent lors de sa tournée ?

Exercice 1 – Graphe temporel

On considère trois intervalles d'Allen t_E, t_H et t_A correspondant respectivement à la notification d'envoi de colis, une période où Jean est chez lui et la notification d'abandon de livraison.

On donne les relations suivantes entre ces intervalles

- C_1 : $t_E\{d\}t_H$. La notification d'envoi s'est déroulée entièrement pendant la période où Jean était à domicile.
- C_2 : $t_H\{e^t\}t_A$. La période considérée où Jean est chez lui se termine par la réception de la notification d'abandon de livraison.

1. Construire le graphe temporel complet entre t_E, t_H et t_A.

On considère maintenant un intervalle d'Allen supplémentaire t_L correspondant au passage du livreur tel que

- C_3 : $t_E\{<\}t_L$.
- C_4 : $t_L\{<\}t_A$.

La propagation de C_3 est donnée (il n'est pas demandé de la refaire) : elle a pour seul effet de mettre à jour la contrainte entre t_H et t_L en $t_H\{<, m, o, e^t, d^t\}t_L$.

2. Propager la contrainte C_4 entièrement en détaillant toutes les étapes pour les arcs étiquetés par plusieurs relations (le graphe n'est pas contradictoire aussi les arcs comportant une seule relation ne sont pas modifiés lors de ces mises à jour).

   Cette propagation doit notamment aboutir à dériver la contrainte $t_L\{d\}t_H$ que l'on nomme C_5.

3. Donner le graphe temporel final complet et proposer une représentation graphique sur un axe temporel de ces 4 intervalles.

Exercice 2 – Logique réifiée d'Allen

On formalise la situation de non-livraison de l'exercice précédent en logique réifiée d'Allen avec les éléments suivants :

- Les constantes sont j représentant Jean, c qui représente le colis ainsi que les constantes d'intervalles de temps t_E, t_H et t_A de l'exercice précédent (et d'autres qui seront obtenues par skolemisation).
- Les fluents sont de la forme $atHome(P)$ où P est une personne
- Les événements possibles sont :
  - $nEnv(C, D)$: notification d'envoi du colis C au destinataire D

– $nAb(C, D)$: notification d'abandon de livraison du colis C pour cause d'absence de son destinataire D.

On lie les constantes d'intervalles de temps à la situation par les trois faits suivants et on donne aussi factuellement les deux premières contraintes temporelles de l'exercice précédent :

- F_1 : $OCCURS(nEnv(c, j), t_E)$
- F_2 : $OCCURS(nAb(c, j), t_A)$
- F_3 : $HOLDS(atHome(j), t_H)$
- F_4 : $t_E\{d\}t_H$
- F_5 : $t_H\{e^t\}t_A$

La société de livraison est censée garantir la règle R exprimée par la formule suivante

$$R : \forall C.\forall D.\forall T.\forall T'.((OCCURS(nEnv(C, D), T) \wedge OCCURS(nAb(C, D), T')) \\ \rightarrow \exists T''.((T\{<\}T'') \wedge (T''\{<\}T') \wedge \neg HOLDS(atHome(D), T''))$$

1. Expliquer (en langage naturel) le sens de la règle R.
2. Il s'agit ici de transformer (R) pour faciliter le raisonnement :
   (a) Instancier la règle (R) pour la substitution $\sigma = \{C \leftarrow c, D \leftarrow j, T \leftarrow t_E, T' \leftarrow t_A\}$
   (b) Skolemiser la formule obtenue (en utilisant comme constante de skolem pour T'' la constante t_L - non encore utilisée dans cet exercice)
   (c) Mettre sous forme clausale la règle obtenue (qui ne comporte plus aucune variable). On doit obtenir trois clauses que l'on nommera F_6, F_7 et F_8.

   Note : On peut considérer que l'on a prouvé $R \vdash F_6, F_7, F_8$.
3. On considère de plus les formules
   - $F_9 : \neg HOLDS(atHome(j), t_L)$
   - $F_{10} : t_E\{<\}t_L$
   - $F_{11} : t_L\{<\}t_A$

   Montrer par résolution que $F_1, F_2, F_6, F_7, F_8 \vdash F_9$.

   On admet que l'on peut prouver de façon similaire (preuve non demandée) $F_1, F_2, F_6, F_7, F_8 \vdash F_{10}$ et $F_1, F_2, F_6, F_7, F_8 \vdash F_{11}$.

L'exercice précédent a permis de prouver que lorsqu'on avait les contraintes C_1, C_2, C_3 et C_4, on obtenait la contrainte C_5. Comme ces contraintes correspondent à F_4, F_5, F_{10}, F_{11}, en posant $F_{12} : t_L\{d\}t_H$, on considère donc avoir établi : $F_4, F_5, F_{10}, F_{11} \vdash F_{12}$.

On donne maintenant une forme clausale simplifiée de l'axiome d'incidence temporel (H1) de la logique réifiée d'Allen :

$$Ax_1 : \neg HOLDS(F, T) \vee \neg IN(T', T) \vee HOLDS(F, T')$$

$$\text{avec : } Ax_2 : \neg(T_1\{d\}T_2) \vee IN(T_1, T_2)$$

4. Montrer par résolution que $F_3, F_{12}, Ax_1, Ax_2 \vdash HOLDS(atHome(j), t_L)$.
5. Indiquer comment les questions précédentes permettent de prouver que la règle (R) n'est pas respectée dans la situation décrite (représentée par F_1, F_2, F_3, F_4, F_5, sachant que Ax_1, Ax_2 sont toujours supposé vrais).

2 Exercices

Exercice 3 – Représentation et composition simple

On considère les 3 intervalles d'Allen suivants, correspondant à différents phénomènes temporels :

- C : intervalle où le conducteur enclenche la clef
- T : intervalle durant lequel le moteur tourne
- R : intervalle durant lequel la voiture roule

1. Représenter les contraintes entre intervalles d'Allen qui traduisent
   (a) Le conducteur enclenche la clé jusqu'à ce que le moteur tourne
   (b) On ne peut rouler que pendant que le moteur tourne
2. Incidence temporelle. En utilisant l'évènement `EnclencherClef` et les fluents `roule(Voiture)` et `tourne(moteur)`, exprimer en logique réifiée d'Allen la situation.
3. Comment exprime-t-on les phrases (a) et (b) en logique réifiée d'Allen de façon générale (i.e. sans utiliser les constantes C,T,R). Ces formules suffisent-elles à retrouver les faits énoncés précédemment ?
4. A l'aide de la composition de relations, en déduire les contraintes entre C et R.

Exercice 4 – Graphe temporel

On souhaite représenter l'énoncé suivant avec un graphe temporel :

"Alfred lisait son journal tout en prenant son petit-déjeuner. Il acheva sa tasse de café et posa son journal. Après le petit-déjeuner, il partit se promener"

1. On considère les 4 intervalles de temps I_e associés aux 4 événements suivants:
   - $OCCURS(Lit(Alfred, journal), I_J)$: Alfred lit son journal
   - $OCCURS(PetitDejeuner(Alfred), I_D)$: Alfred prend son petit déjeuner
   - $OCCURS(Boit(Alfred, cafe), I_C)$: Alfred boit son café
   - $OCCURS(SePromene(Alfre), I_P)$: Alfred se promène

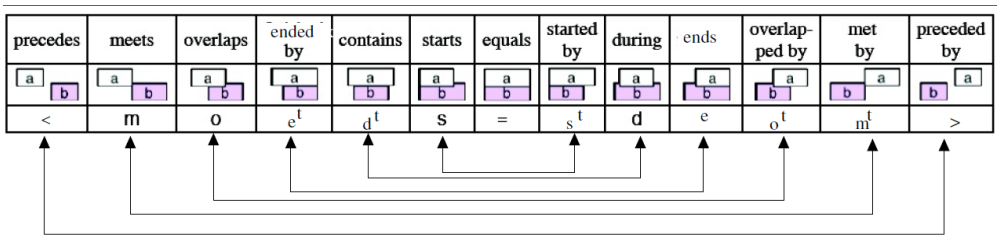
   Exprimer les contraintes entre I_J et I_D exprimées dans la première phrase, entre I_J et I_C exprimées dans la deuxième et entre I_D et I_P exprimées dans la troisième.
2. En utilisant de plus le fait que boire le café est une étape du petit-déjeuner, représenter toutes les relations dans un graphe temporel et montrer comment les contraintes se propagent.

Invervalles d'Allen

1 Relations d'Allen

Classées par le "degré" avec lequel a commence avant b puis par le "degré" avec lequel a finit après b.

Les flèches indiquent les relations transposées : $a\ R\ b \iff b\ R^t\ a$



2 Composition des relations

	$<$	m	o	e^t	d^t	s	$=$	s^t	d	e	o^t	m^t	$>$
$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$	$< mosd$	$< mosd$	$< mosd$	$< mosd$	tout
m	$<$	$<$	$<$	$<$	$<$	m	m	m	osd	osd	osd	$e^t = e$	$d^t s^t o^t m^t >$
o	$<$	$<$	$< mo$	$< mo$	$< moe^t d^t$	o	o	$oe^t d^t$	osd	osd	Concur	$d^t s^t o^t$	$d^t s^t o^t m^t >$
e^t	$<$	m	o	e^t	d^t	o	e^t	d^t	osd	$e^t = e$	$d^t s^t o^t$	$d^t s^t o^t$	$d^t s^t o^t m^t >$
d^t	$< m$ $oe^t d^t$	$oe^t d^t$	$oe^t d^t$	d^t	d^t	$oe^t d^t$	d^t	d^t	Concur	$d^t s^t o^t$	$d^t s^t o^t$	$d^t s^t o^t$	$d^t s^t o^t$ $m^t >$
s	$<$	$<$	$< mo$	$< mo$	$< moe^t d^t$	s	s	$s = s^t$	d	d	deo^t	m^t	$>$
$=$	$<$	m	o	e^t	d^t	s	$=$	s^t	d	e	o^t	m^t	$>$
s^t	$< mo$ $e^t d^t$	$oe^t d^t$	$oe^t d^t$	d^t	d^t	$s = s^t$	s^t	s^t	deo^t	o^t	o^t	m^t	$>$
d	$<$	$<$	$< mosd$	$< mosd$	tout	d	d	$deo^t m^t >$	d	d	$deo^t m^t >$	$>$	$>$
e	$<$	m	osd	$e = e^t$	$d^t s^t o^t m^t >$	d	e	$o^t m^t >$	d	e	$o^t m^t >$	$>$	$>$
o^t	$< mo$ $e^t d^t$	$oe^t d^t$	Concur	$d^t s^t o^t$	$d^t s^t o^t m^t >$	deo^t	o^t	$o^t m^t >$	deo^t	o^t	$o^t m^t >$	$>$	$>$
m^t	$< mo$ $e^t d^t$	$s = s^t$	deo^t	m^t	$>$	deo^t	m^t	$>$	deo^t	m^t	$>$	$>$	$>$
$>$	tout	deo^t $m^t >$	deo^t $m^t >$	$>$	$>$	deo^t $m^t >$	$>$	$>$	deo^t $m^t >$	$>$	$>$	$>$	$>$

où Concur=$oe^t d^t s = s^t deo^t$
et tout=$< moe^t d^t s = s^t deo^t m^t >$

3 Algorithme de propagation des contraintes

Propager(R_{ab})
    EmpilerR_{ab}
    Tant que la pile est non vide
        Dépiler R_{ij}
        Pour tout k dans $[1, n]$, $k \neq i$ et $k \neq j$
            $newR_{ik} \leftarrow$ conjonction(R_{ik}, composition(R_{ij}, R_{jk}))
            $newR_{kj} \leftarrow$ conjonction(R_{kj}, composition(R_{ki}, R_{ij}))
            Si $newR_{ik} = \emptyset$ ou $newR_{kj} = \emptyset$
                contradiction temporelle : arrêt
            Si $newR_{ik} \neq R_{ik}$
                $R_{ik} \leftarrow newR_{ik}$
                Empiler R_{ik}
            Si $newR_{kj} \neq R_{kj}$
                $R_{kj} \leftarrow newR_{kj}$
                Empiler R_{kj}

Intervalles d'Allen

L'objectif du TME est d'implémenter en python le calcul des relations obtenues par composition de relations, ainsi que la création d'un graphe temporel grâce à l'algorithme de propagation d'Allen. Les fichiers Python, dûment commentés, sont à envoyer pour le compte rendu.

Dans le formalisme choisi, les relations sont représentées par des chaînes de caractères.

Exercice 1 Implémentation de la composition

1. Télécharger depuis le moodle le fichier `LRC_definitions_Allen.py` qui définit :
   - un dictionnaire `transpose` de type `dict[str: str]`, tel que `transpose[r]` fournit la transposition de la relation `r`.
   - un dictionnaire `symetrie` de type `dict[str: str]`, tel que `symetrie[r]` fournit la symétrie de la relation `r`.
   - un dictionnaire `compositionBase` de type `dict[tuple[str,str]: set[str]]`, dont les clés sont des couples de relations et tel que `compositionBase[(r1,r2)]` est l'ensemble des relations obtenues lorsqu'on compose la relation `r1` avec la relation `r2`. Ce dictionnaire représente la table de composition réduite donnée en cours (diapositive 35 du cours 8).

Rappel des commandes python de base pour manipuler ensembles et dictionnaires :

`set()`	création d'un ensemble vide
`{e1, e1}`	création d'un ensemble à 2 éléments
`len(S)`	cardinal de l'ensemble
`e in S`	test d'appartenance $e \in S$
`for e in S`	parcours de l'ensemble
`S1 & S2`	intersection $S1 \cap S2$
`S1 | S2`	union $S1 \cup S2$
`S1 <= S2`	test d'inclusion $S1 \subseteq S2$
`S.add(x)`	ajoute x à S

`dict()`	création d'un dictionnaire vide
`{k1:v1, k2:v2}`	dictionnaire à 2 entrées
`len(D)`	nombre de clés
`k in D`	test d'appartenance des clés
`for k in D`	parcours des clés
`D[k]`	valeur associée à la clé `k`
`D[k] = v`	ajoute l'association `k:v` à `D` (écrase l'ancienne association si `k` déjà dans `D`)

2. Définir les fonctions `transposeSet` et `symetrieSet` telles que `transposeSet(S)` (resp. `symetrieSet(S)`) renvoie l'ensemble des relations transposées (resp. symétriques) des relations qui apparaissent dans `S`.

3. Définir la fonction `compose` telle que `compose(r1, r2)` renvoie l'ensemble des relations que l'on peut obtenir par composition des relations `r1` et `r2`.

   Il s'agit donc d'utiliser :
   - la table de composition directement
   - la transposition : $r_1 \circ r_2 = (r_2^t \circ r_1^t)^t$
   - la symétrie : $r_1 \circ r_2 = (r_1^s \circ r_2^s)^s$
   - les deux transformations simultanément : $r_1 \circ r_2 = (r_2^{st} \circ r_1^{st})^{ts}$

   Pour tester l'implémentation, vérifier par exemple que
   - $= \circ \mathrm{d} = \{\mathrm{d}\}$
   - $\mathrm{m} \circ \mathrm{d} = \{\mathrm{d}, \mathrm{o}, \mathrm{s}\}$
   - $\mathrm{ot} \circ > = \{>\}$
   - $> \circ \mathrm{e} = \{>\}$

- ot $\circ$ m = {dt,et,o}

4. Définir la fonction `compositionSet` telle que `compositionSet(S1, S2)` renvoie l'ensemble des relations que l'on peut obtenir par composition des relations qui apparaissent dans l'ensemble `S1` avec les relations qui apparaissent dans l'ensemble `S2`.

   Il s'agit de généraliser la fonction précédente à des ensembles.

Exercice 2 Gestion du graphe temporel

L'implémentation de l'algorithme d'Allen peut ne pas suivre les indications de cet exercice si une autre solution vous paraît plus facile, justifiez alors votre choix. Il est conseillé de représenter un graphe par :

- l'ensemble de ses nœuds
- l'ensemble des relations entre ses nœuds, par exemple à l'aide d'un dictionnaire ayant pour clés des couples de nœuds, et pour valeurs les ensembles des relations correspondantes.

Un graphe avec deux nœuds A et B et A{o,e}B sera donc représenté par :

- ses nœuds {'A','B'}
- ses relations {('A','B'):{'o','e'} }

1. Créer une classe `Graphe` ayant pour attributs `noeuds` et `relations`, ainsi qu'une fonction `getRelations` telle que `g.getRelations(i,j)` renvoie l'ensemble des relations entre les nœuds `i` et `j` du graphe `g`.

   Attention au traitement des cas où c'est la relation transposée qui est stockée dans le graphe ou s'il n'existe pas de relation stockée entre les nœuds `i` et `j`.

2. Créer une fonction `propagation` qui prend en entrée un graphe et deux nœuds de ce graphe, et utilise l'algorithme d'Allen pour propager la relation entre ces deux nœuds dans le reste du graphe. On suppose ici que les deux nœuds existent déjà, on ne fait que propager la relation en question.

3. Créer une fonction `ajouter` qui prend en entrée un graphe et une relation entre deux nœuds, et ajoute cette relation au graphe. On peut avoir les deux nœuds qui appartiennent déjà au graphe, l'un d'eux uniquement, ou aucun.

   Pour construire un graphe cohérent, il suffit maintenant de créer un graphe vide puis d'ajouter les relations une par une.

4. Tester la création de graphe avec des cas simples de graphes à trois nœuds A, B, C :

   - un graphe avec les relations A{<}B et A{>}C
   - un graphe avec les relations A{<}B et A{<}C
   - dans les deux cas, propager la relation B{=}C

5. Créer une fonction `retirer` qui prend en entrée un graphe et un nœud et retire du graphe ce nœud et toutes les relations qui le concernent (sans autre mise à jour).

6. Faire des versions 'verbose' de vos fonctions d'ajout et de propagation permettant d'afficher proprement toutes les étapes de la mise à jour.

7. Reprendre l'exercice 1 de la feuille de TD sur les intervalles d'Allen et faire automatiquement toutes les propagations impliquées dans le raisonnement : partir d'un graphe temporel vide et ajouter les relations au fur et à mesure qu'elle sont déduites.

   On peut ainsi obtenir, en utilisant les versions 'verbose' de l'algorithme, une correction de la propagation dans le graphe temporel.

8. Traiter de même les exercices 3 et 4 de la feuille de TD sur les intervalles d'Allen.

LRC - TD 6

Réseaux de Petri

1 Syntaxe

Exercice 1 Représentation

On donne le réseau de Petri $R = \{P, T, Pre, Post\}$ où :

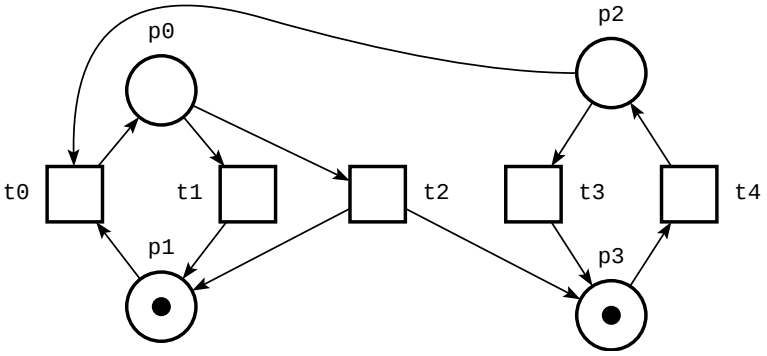
- $P = \{p_0, p_1, p_2, p_3\}$
- $T = \{t_0, t_1, t_2, t_3\}$
- $Pre : P \times T \to \mathbb{N}$ est défini par $Pre(p,t) = 1$ si $(p,t) \in \{(p_0,t_0), (p_1,t_0), (p_1,t_1), (p_2,t_3), (p_3,t_2)\}$ et $Pre(p,t) = 0$ sinon.
- $Post : P \times T \to \mathbb{N}$ est défini par $Post(p_1,t_2) = 2$, $Post(p,t) = 1$ si $(p,t) \in \{(p_0,t_3), (p_2,t_0), (p_2,t_1), (p_3,t_3)\}$ et $Post(p,t) = 0$ sinon.

On considère le marquage initial M_0 tel que $M_0(p) = 1$ si $p \in \{p_0, p_1, p_3\}$ et 0 sinon. On représente de façon équivalente ce marquage par le vecteur $(1,1,0,1)$.

1. Donner une représentation graphique du réseau R et y reporter le marquage initial M_0.
2. Indiquer les transitions franchissables dans le marquage initial.
3. Donner les matrices d'incidence avant et arrière de R.
4. Donner le marquage M_1 résultant du franchissement de la transition t_0 à partir de M_0, et le marquage M_2 résultant du franchissement de la transition t_2 à partir de M_1.
5. Le marquage $(1,1,1,1)$ est-il accessible à partir de M_0 ? Justifier.

Exercice 2 Graphe des marquages accessibles et vivacité

On considère le réseau de Petri suivant :



1. Calculer les matrices d'entrées et de sortie (ie les matrices d'incidence respectivement avant et arrière). En déduire la matrice d'incidence.
2. Calculer, à l'aide de la matrice d'incidence, le marquage résultant du tir de la séquence des transitions $(t_4\ t_3\ t_4\ t_3\ t_4\ t_0\ t_2\ t_4\ t_0\ t_2\ t_4\ t_3\ t_4\ t_0\ t_1)$.
3. Générer l'arbre des marquages accessibles.
4. En déduire le graphe des marquages accessibles.
5. Ce réseau de Petri est-il borné ? vivant ? quasi-vivant ? sans blocage ? Justifier.
6. Donner une transition vivante ou un marquage puits. Peut-on construire un graphe qui ait à la fois une transition vivante et un marquage puits accessible ?

2 Modélisation

Exercice 3 Processus de montage

Pour fabriquer une voiture en bois, les elfes du père Noël doivent fabriquer 4 roues et la carrosserie, puis assembler le tout. Pour fabriquer la carrosserie, il s'agit d'abord de la sculpter, puis de la peindre. La peinture doit être terminée avant que l'on assemble le jouet.

1. Modéliser le travail d'un elfe pour construire une voiture en bois avec un réseau de Petri. On utilisera les transitions `fabriquerRoue`, `sculpter`, `peindre` et `assembler` pour représenter les étapes du processus.

2. On considère maintenant qu'il y a N elfes, et on compte le nombre de jouets produits. Modéliser le processus.

Il est demandé, pour chaque place, d'indiquer ce qu'elle représente et ce que peut signifier son marquage.

Exercice 4 Salon de barbier

Modéliser le fonctionnement d'une boutique de barbier dans les 2 cas suivants :

1. La capacité de la salle n'est pas limitée ; le barbier se repose après chaque rasage et on peut représenter le fait qu'un rasage est en cours.

2. La capacité de la salle est limitée à N ; le barbier ne se repose pas entre chaque client, mais se met au repos quand la salle est vide. Si le barbier est au repos, l'entrée d'un client le réveille. Contrairement au cas 1, on ne représente pas un rasage en cours (c'est à dire qu'on considère une transition 'rasage d'un client' sans en distinguer le début et la fin).

Dans les deux cas, calculer le graphe des marquages accessibles.