Tp2 : Logique des Prédicats

**Introduction :**

Dans ce qui suit, nous présentons notre utilisation de la logique des prédicats avec la librairie Java Tweety pour modéliser des situations impliquant des personnes, des activités et des relations entre eux. Nous démontrons comment nous avons défini des prédicats, des constantes, des foncteurs, et des formules logiques pour représenter notre exemple, ainsi que la manière dont nous avons utilisé la librairie Tweety pour manipuler ces entités logiques.

**Méthodologie :**

**Choix d’un exemple concret :**

Nous explorerons l'utilisation de la librairie Tweety pour modéliser les relations entre différentes entités en utilisant des prédicats et des fonctions, avec comme scénario la représentation de relations entre personnes et activités. Voici une description détaillée de l'exemple :

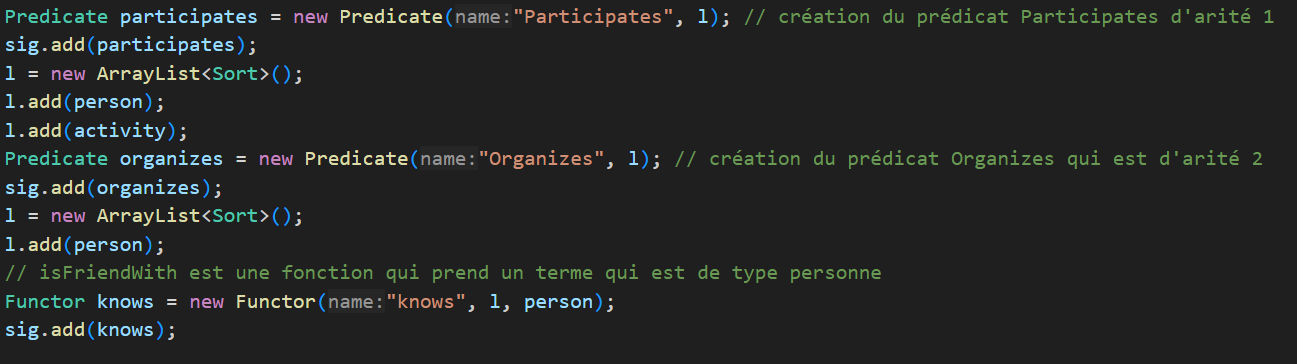
1. **Définition des entités logiques :** définir les sortes (ou types) pour les entités, telles que "personne" et "activité".



1. **Définitions des prédicats et des foncteurs :** pour représenter des relations entre les entités, tels que "Participates" (participation à une activité) et "Organizes" (organisation d'une activité).

Le prédicat "Organizes" est défini avec une arité de 2, indiquant qu'il s'agit d'un prédicat prenant deux termes en argument.

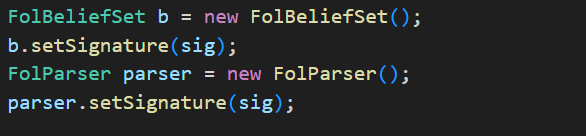
Une fonction "knows est définie avec une arité de 1, indiquant qu'elle prend un terme de type person en argument.



1. **Construction de la base de connaissances :**

En utilisant la classe **FolBeliefSet**, nous avons créé une base de connaissances logique pour stocker nos formules logiques.

Pour l’ajout des formules logiques à la base de connaissances le parser **FolParser est utilisé.**



1. **Ajout des axiomes :**

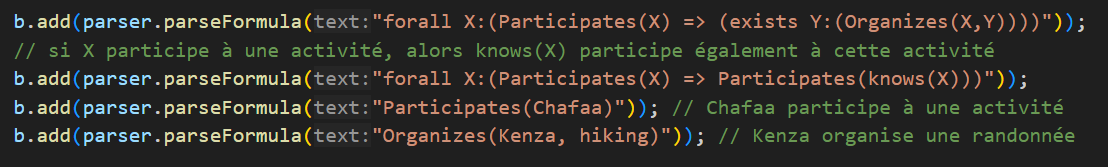
On ajoute plusieurs axiomes pour exprimer des règles logiques concernant les relations entre les personnes, les activités et les connaissances.

Pour tout individu X, s'il participe à une activité, alors il existe une activité Y qu'il organise.

Si une personne participe à une activité, alors toute personne qui la connaît participe également à cette activité

Kenza organise une randonnée (hiking).

Chafaa participe à une activité.



**Résultats :**

Nous avons affiché les formules logiques de notre base de connaissances pour vérifier qu'elles ont été correctement ajoutées.

Une image contenant texte, Police, capture d’écran

Le contenu généré par l’IA peut être incorrect.

Les résultats obtenus ont montré que nos formules logiques étaient cohérentes et conformes aux règles que nous avions définies.

**Conclusion :**

Ce travail démontre l'efficacité de l'utilisation de la logique des prédicats et de la librairie Java Tweety pour modéliser des problèmes impliquant des relations logiques entre différents types d'entités.

Tp3 : Logique Modale

Introduction :

Ce rapport présente une étude pratique de la logique modale à travers deux approches distinctes : la modélisation avec **Modal Logic Playground** et l'implémentation en Java avec **la librairie Tweety**. L'objectif principal de cette étude est de vérifier la valeur de vérité de quelques formules dans le cadre de la logique modale.  
  
**Méthodologie :**  
**Modal Logic Playground :**

**Fonctionnement :** Modal Logic Playground offre une interface conviviale permettant aux utilisateurs de saisir des formules de logique modale dans un langage formel et de les vérifier. Il permet de définir des mondes possibles, d'attribuer des valeurs de vérité à des propositions atomiques, et d'appliquer des opérateurs modaux tels que ◻ (nécessairement) et ◇ (possiblement).  
**Utilisation pour la modélisation :** Dans cette partie de l'expérience, nous avons utilisé Modal Logic Playground pour modéliser différentes situations et vérifier la validité de formules de logique modale. Nous avons saisi les formules pertinentes dans l'outil, défini les mondes possibles et interprété les résultats pour évaluer la vérité des propositions modales.  
1-Modelisation :

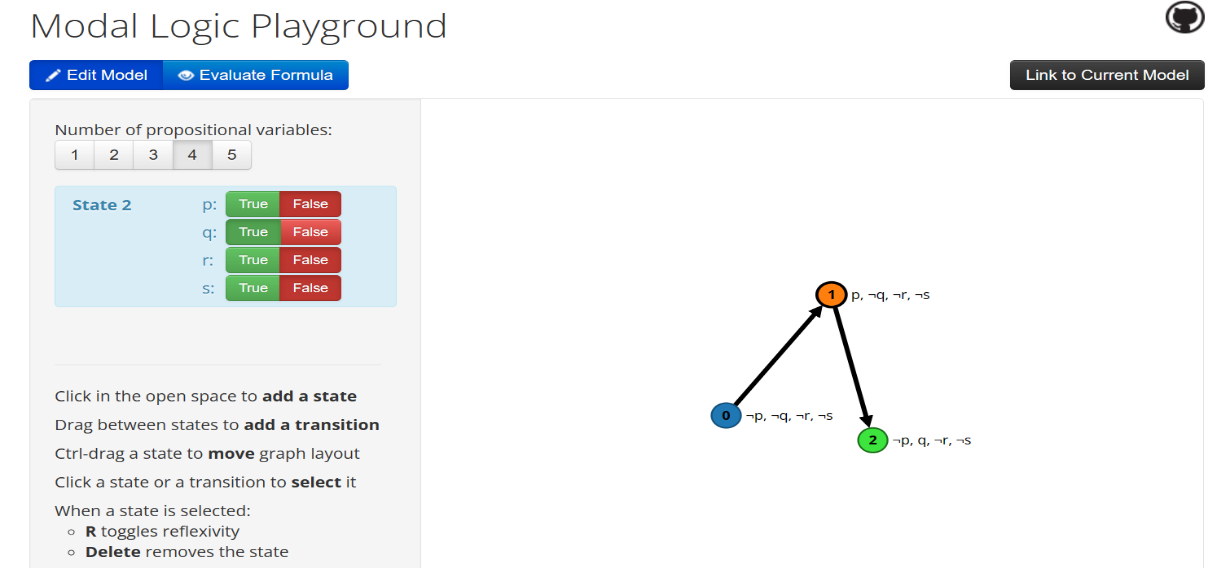
On choisit le nombre de variables propositionnelles : dans notre exemple = 4 (p, q, r, s) :  
initialement elle est ainsi :  
 

Figure .. : Modal Logic Playground

On prend un exemple de 7 mondes avec 4 variables puis on passe à l’évaluation des formules.

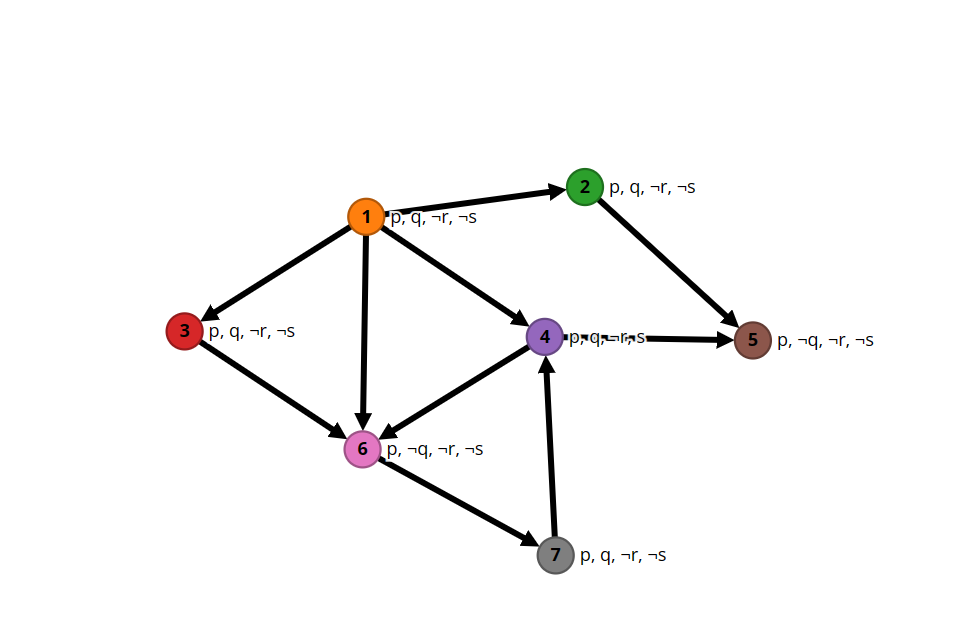
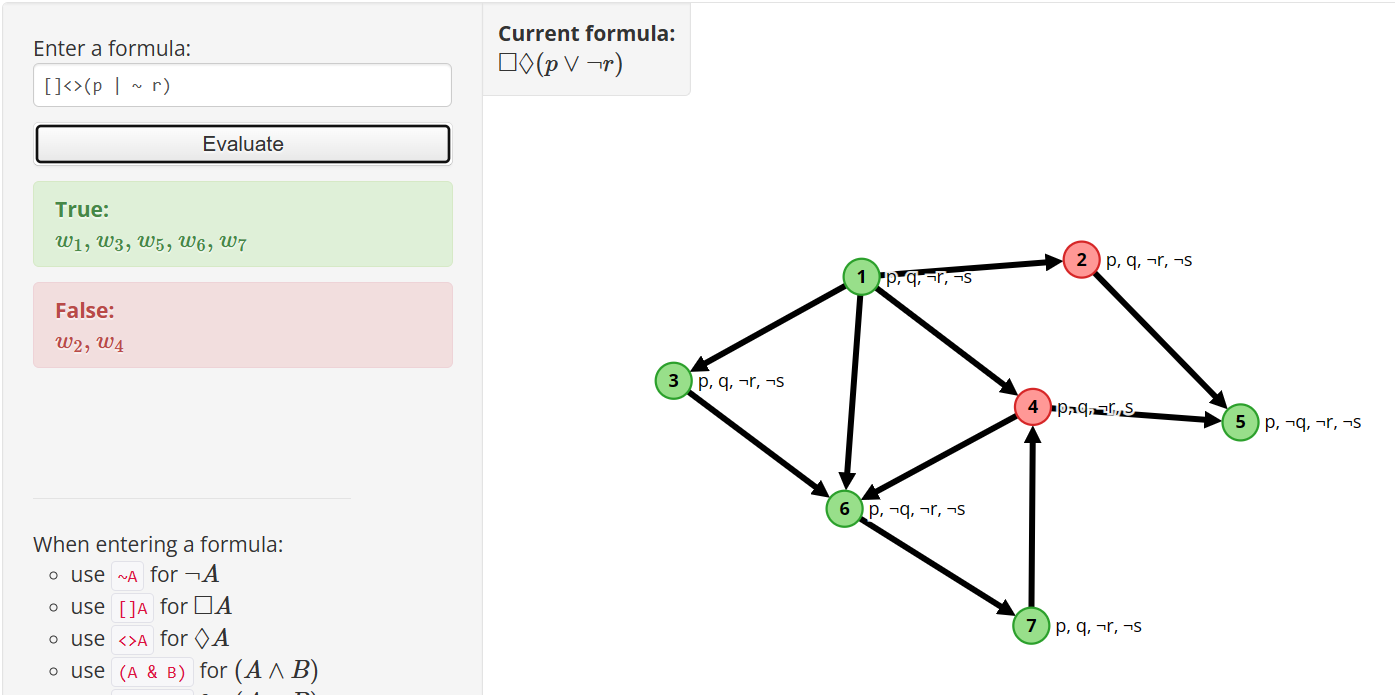
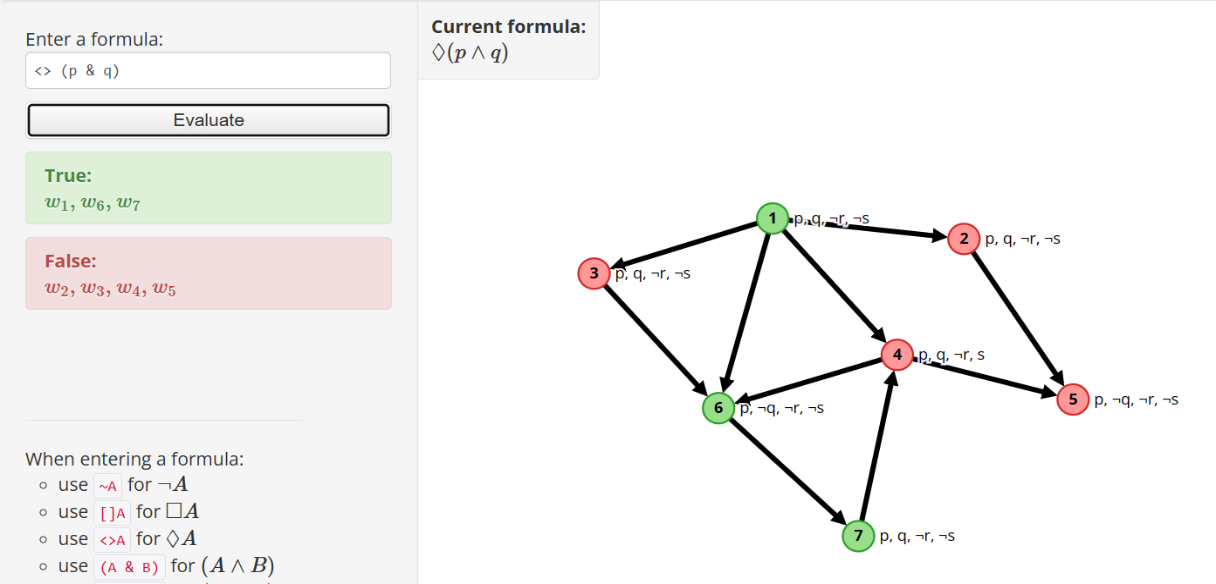


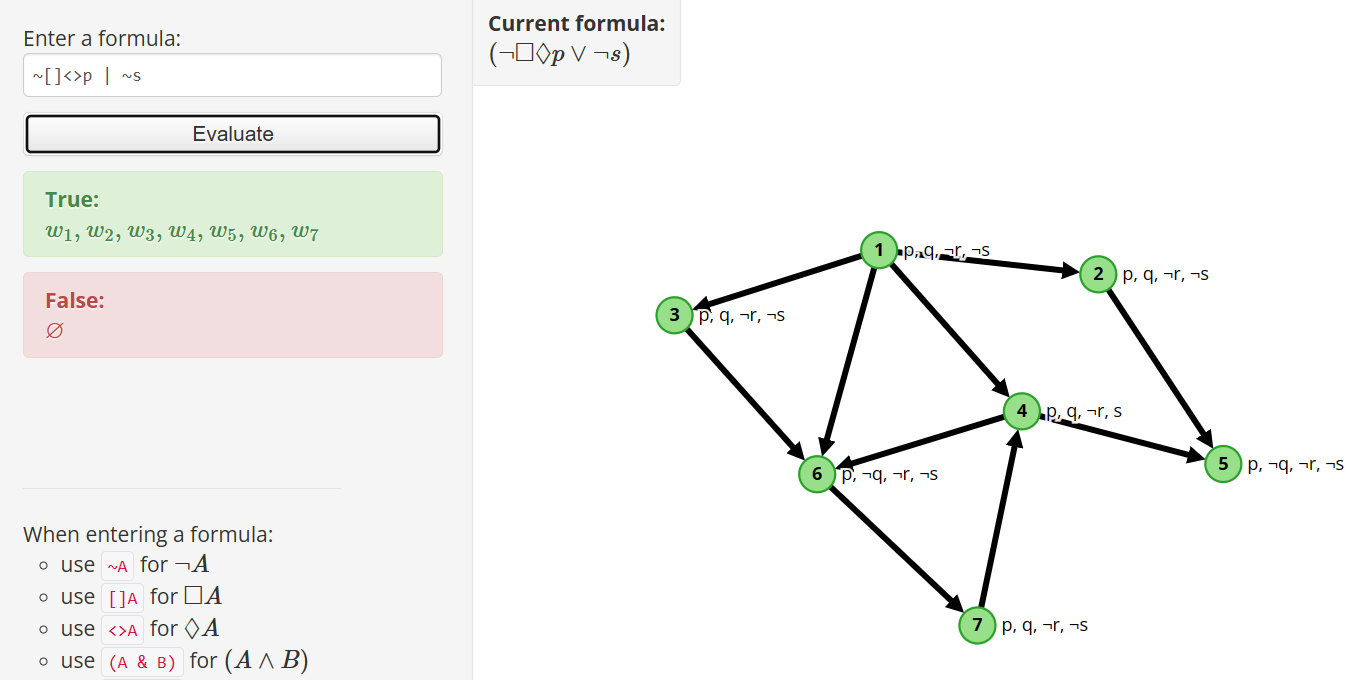
Figure : Exemple de modélisation d’un modèle modale

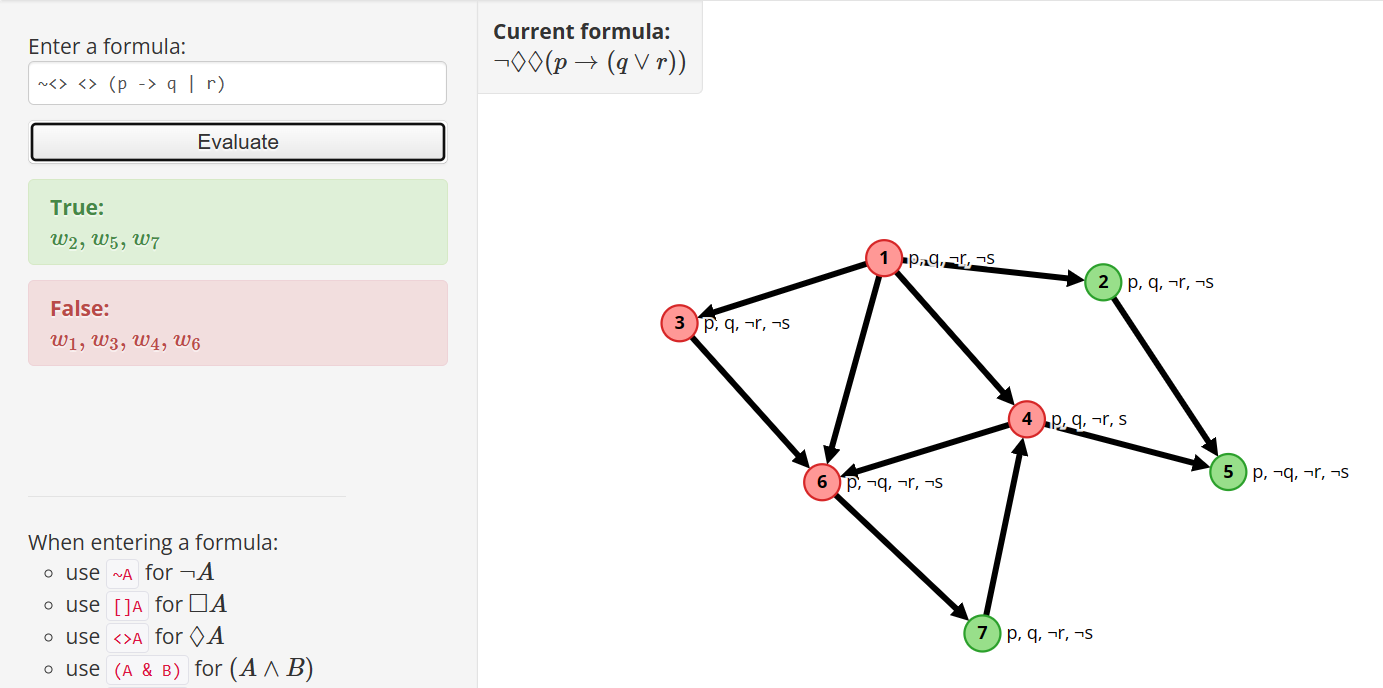
2-Evaluation :

[] <> (p | ~ r)   


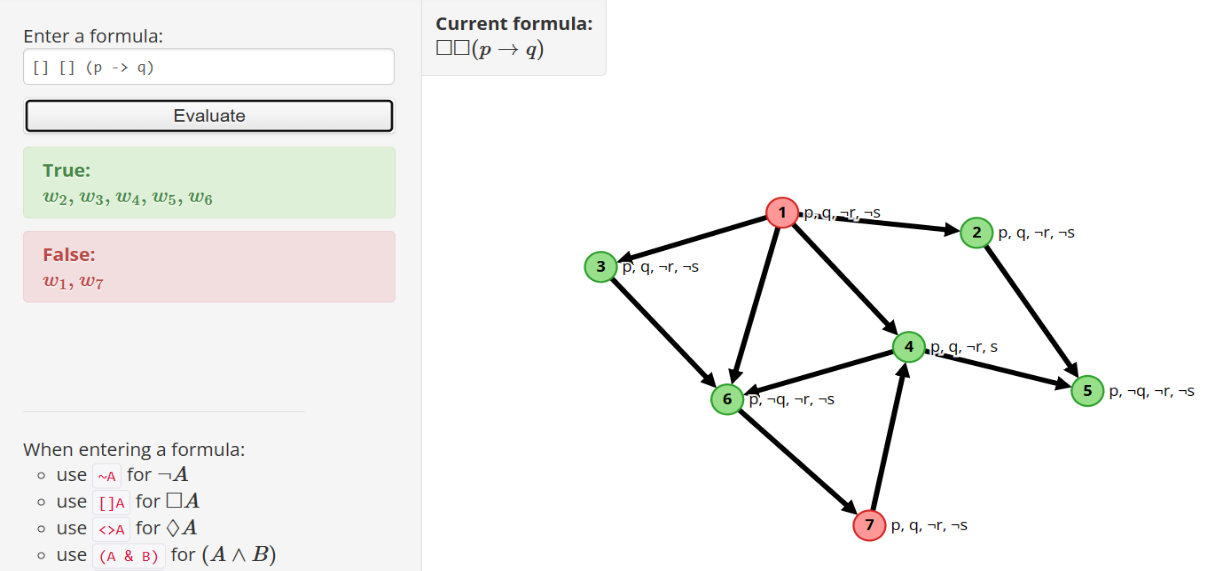
<> (p & q)  


~[]<>p | ~s

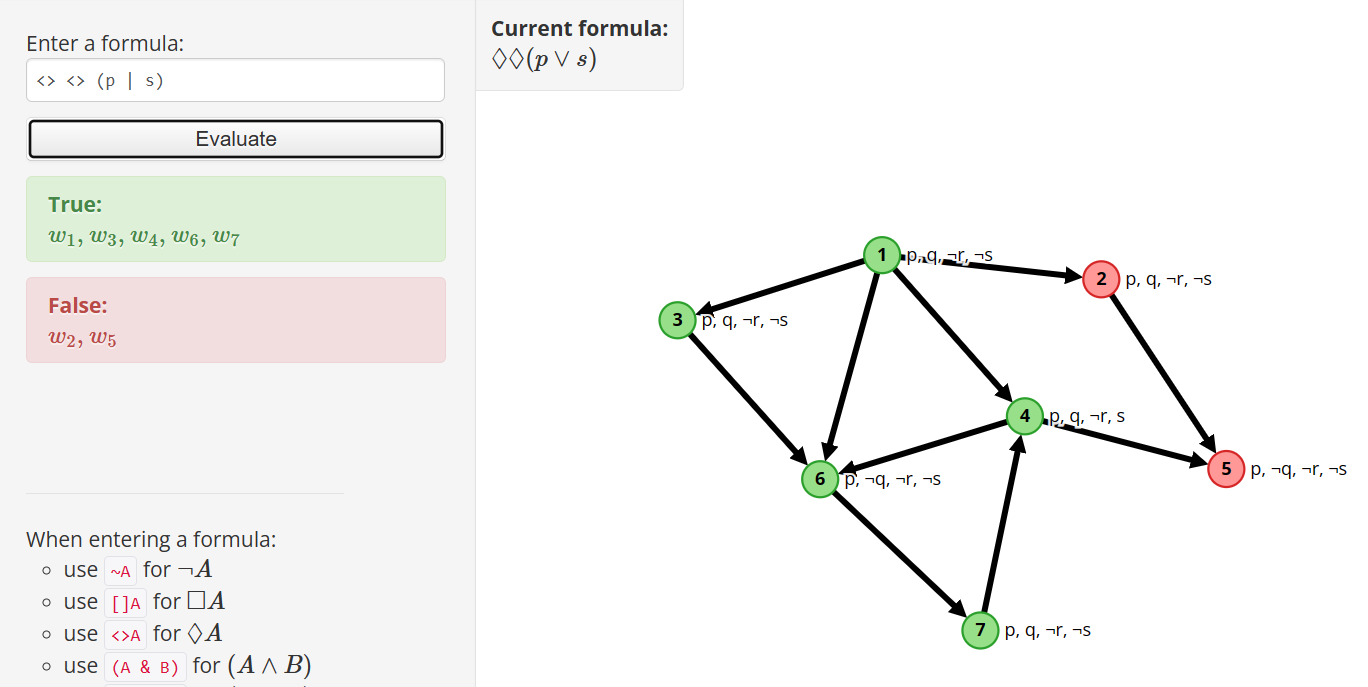


~<> <> (p -> q | r)

[] [] (p -> q)



<> <> (p | s)

  
  
**Librairie Java Tweety :**

**Fonctionnalités :** Tweety offre un ensemble de classes et de méthodes pour représenter des modèles logiques, définir des opérateurs modaux, et effectuer des opérations telles que la vérification de la satisfiabilité et la déduction.

**Utilisation pour l'implémentation en Java :**

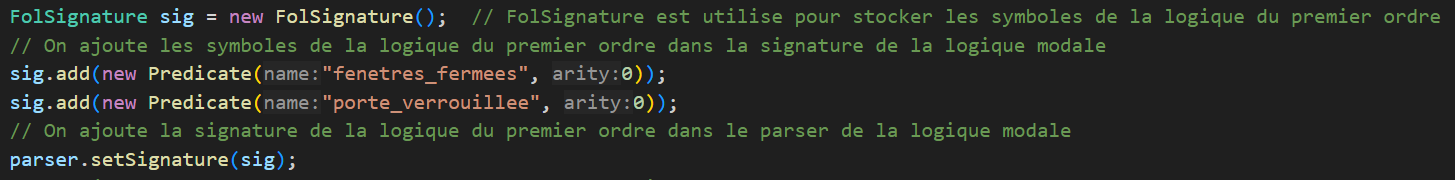
Voici un exemple démontrant l'utilisation de la librairie Java Tweety pour travailler avec des formules modales et des formules de logique du premier ordre. L'exemple montre comment créer une base de connaissances modale, ajouter des formules, utiliser un analyseur syntaxique (parseur) pour interpréter les formules, et utiliser un moteur d'inférence (raisonneur) pour répondre aux requêtes sur la base de connaissances modale.

1-Creation de la base de connaissance modale :  

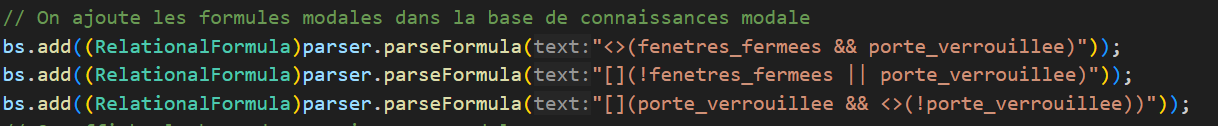

2- Initialisation du parseur :



3- Définition des symboles de la logique du premier ordre :



4-Ajout des formules modales a la base de connaissances :



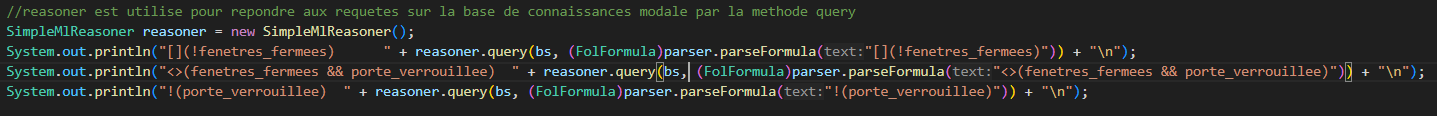
La base de connaissances modale que nous avons créée dans l'exemple comprend trois formules modales :

La première formule signifie "Il est possible que les fenêtres soient fermées et la porte soit verrouillée".

La deuxième formule signifie "Il est nécessaire que si les fenêtres ne sont pas fermées, alors la porte est verrouillée".

La troisième formule signifie "Il est nécessaire que si la porte est verrouillée, alors il existe un monde où la porte n'est pas verrouillée".

5-Utilisation du moteur d’inférence pour répondre aux requêtes :

Resultats d’exécution :

