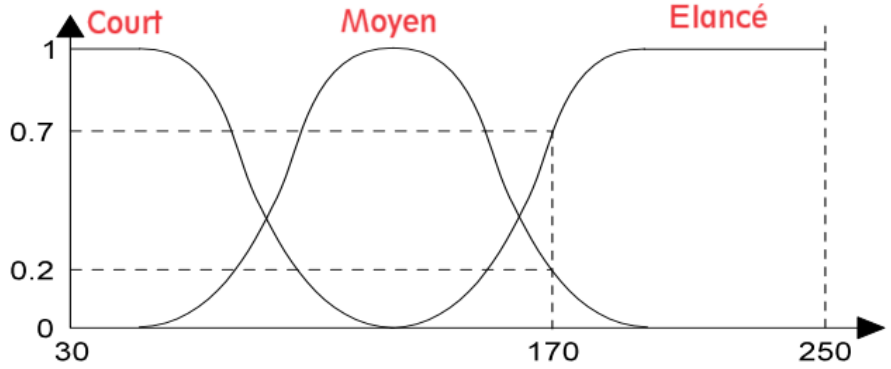
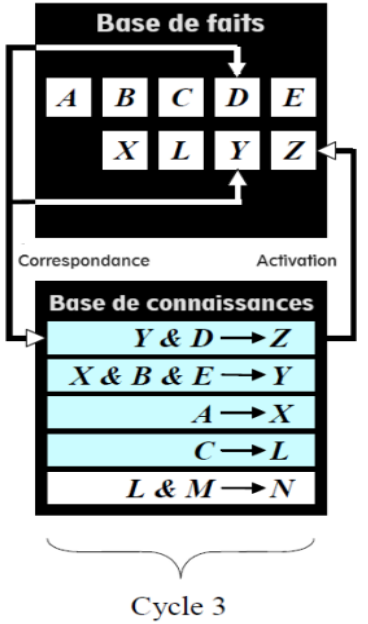
Construire un système expert avec Experta

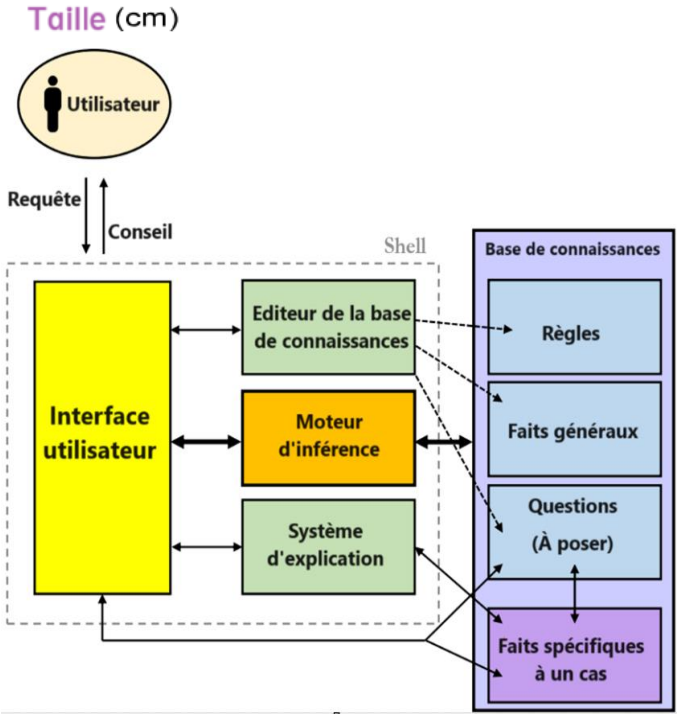
Dr. NSENGE MPIA HERITIER, Ph.D

Précédemment

- Base de connaissances
  - Règles, faits, questions, faits spécifiques à un cas
- Moteur d'inférence
  - Cycle Correspondance-Résolution-Action
  - Chaînage avant
  - Chaînage arrière
- Autres composants
  - Interface utilisateur
  - Éditeur de la base de connaissances
  - Système d'explication
- Autres types de systèmes experts
  - Flou, basé sur des cadres, neuronal, neuro-flou
- Avantages et inconvénients
  - Nombreux
- Shell
  - Mise en œuvre plus facile et plus rapide des systèmes experts







Plan de la leçon

- Rôles dans le développement des systèmes experts
- Acquisition de connaissances et ingénierie
- Introduction à Experta
  - Exemple d’un système expert de Météorologie
  - Exemple de diagnostic des types des diabètes

Revisiter : Pourquoi développer/utiliser des systèmes experts ?

- L'expertise humaine est:
  - Peu disponible (peu d'experts sur la tâche)
  - Coûteuse à développer ou à embaucher
  - Difficile à obtenir rapidement
    - Expertise difficile à appliquer rapidement (tâches complexes)
    - Il n'est pas pratique d'avoir un être humain constamment disponible
  - Peut être perdue :
    - Retraite/quitter son emploi
    - Décès
- Le système expert remplit les fonctions suivantes en ce qui concerne l'expertise :
  - Documenter/préserver
  - Reproduire/mettre à disposition
  - Enseigner/diffuser
- Les ordinateurs ne s'ennuient pas/ne se fatiguent pas/ne se frustrent pas/ne s'effraient pas

Choisir un problème

- Lorsque nous choisissons un problème qui pourrait bénéficier de la construction d'un système expert, il est important d'être réaliste quant aux coûts impliqués et de s'assurer que les dépenses sont justifiées par rapport aux bénéfices attendus.
  - Les systèmes experts nécessitent un développement itératif, ce qui représente un investissement important en termes de temps.
  - Toutefois, les coûts de construction d'un système expert peuvent être justifiés, en particulier lorsque l'expertise est rare
- Nous voulons nous assurer que le choix d'un système expert en tant que technique est approprié.
  - Ils ne seraient probablement pas très appropriés:
    - pour les problèmes nécessitant de la dextérité manuelle ou des aptitudes physiques,
    - pour les problèmes faisant appel à des connaissances de bon sens,
    - pour les problèmes pouvant être résolus avec des techniques plus simples comme:
      - Organigramme codé sous forme de programme simple
      - Feuille de calcul
      - c'est-à-dire que si des méthodes simples fonctionnent, il n'est pas nécessaire de chercher une solution plus complexe
  - Probablement approprié pour:
    - Le problème qui nécessite une expertise très spécialisée
    - Mais sa résolution ne prendrait que peu de temps à un expert humain (une heure au maximum)
- Il faut identifier les experts disponibles et coopératifs
  - Une équipe de développement et des exemples d'utilisateurs peuvent également être nécessaires

Construire un système expert : Un investissement qui en vaut la peine ?

- Pour savoir si la mise en place d'un système expert est un investissement rentable, il convient de commencer par une analyse coût-bénéfice.
- Et inclure des considérations telles que :
  - Le problème peut-il être résolu efficacement par la programmation conventionnelle ?
  - Existe-t-il un besoin/désir pour un système expert ?
  - Existe-t-il au moins un expert humain disposé à coopérer ?
  - L'expert peut-il expliquer ses connaissances à un ingénieur de la connaissance ?
  - L'ingénieur des connaissances peut-il comprendre l'expert ?
  - Les connaissances relatives à la résolution des problèmes sont-elles principalement heuristiques et incertaines ?
  - Les coûts en temps et en argent seront-ils rentables ?

Coup d’œil sur la construction d'un système expert

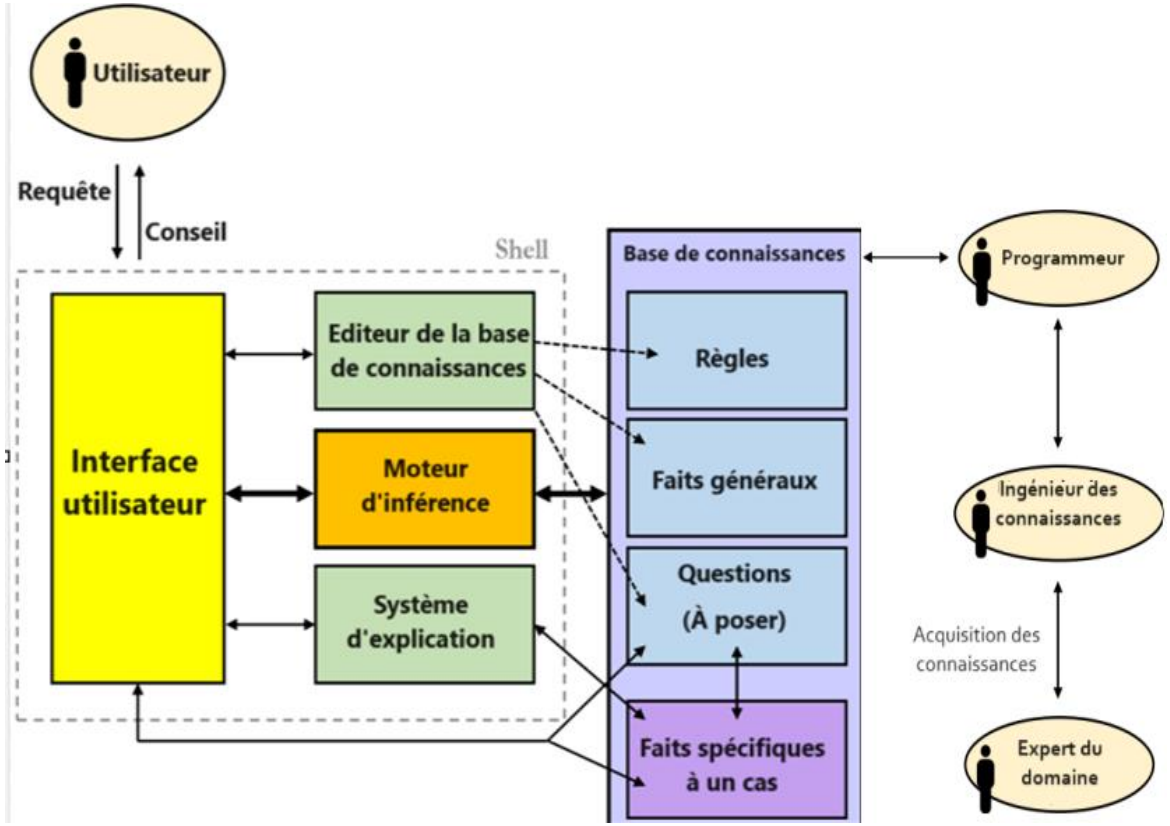
- Lorsque vous essayez de construire un système expert, votre première tentative a peu de chances d'être couronnée de succès
  - L'expert a généralement du mal à exprimer exactement les connaissances et les règles qu'il utilise pour résoudre un problème
  - Sens commun/connaissance subconsciente
  - Semble si évident qu'il ne prend pas la peine de le mentionner.
- Acquisition de connaissances :
  - Il faut d’abord assembler/extraire les connaissances pertinentes
- Ingénierie des connaissances :
  - Construire la base de connaissances du système expert
- Développement du prototype initial et présentation à l'expert
  - Vérifier les performances du système et donner un retour d'information
  - Raffinement de la base de connaissances
- Développement itératif à partir du prototype
  - Feedback de la part de l'expert et des utilisateurs finaux du système

Rôles dans le développement des systèmes experts

Schéma général d'un système expert avec les rôles

- L'un des rôles dans le système expert est celui de l'expert du domaine.
  - Un expert du domaine travaille avec un ingénieur de la connaissance pour faciliter le processus d'acquisition des connaissances.
  - À partir de là, l'ingénieur de la connaissance peut travailler avec un développeur de logiciels qui prendra les connaissances acquises et les transformera en un système expert.

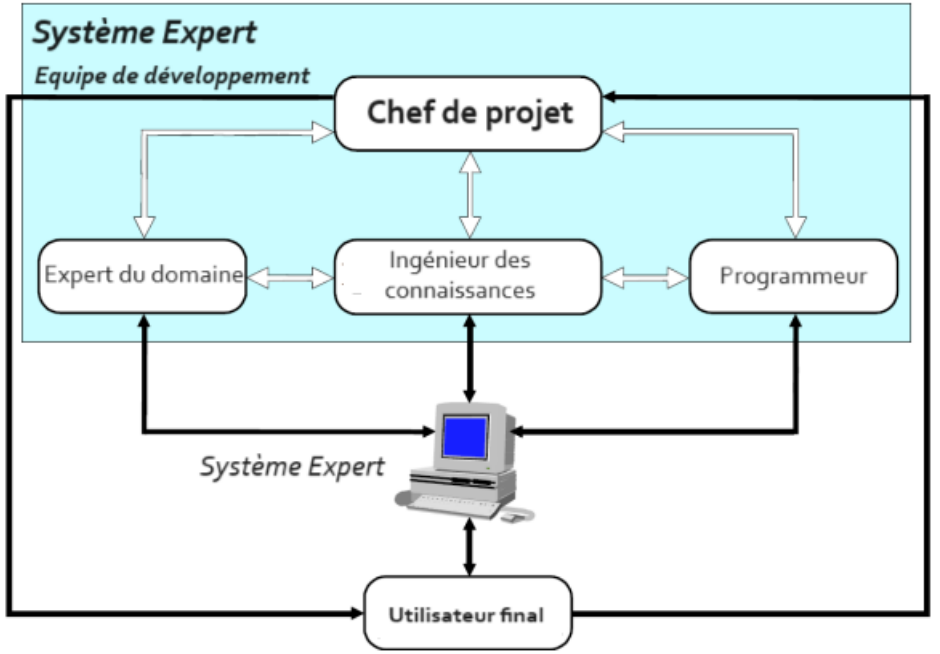
  - En des compétences, ces rôles peuvent être assumés par la même personne ou, si un expert en la matière possède également une expérience de la programmation informatique, il peut également remplir tous ces rôles.



Équipe de développement

- Outre l'*expert du domaine*, *l'ingénieur de la connaissance* et le *développeur*, l'équipe de développement comprend souvent le Chef de projet et l'utilisateur final.

- Le succès du développement d'un système expert dépend bien sûr de la qualité de la collaboration entre ces membres.



Expert du domaine

- Toute personne peut être considérée comme un expert de domaine si elle possède une connaissance approfondie (des faits et des règles) et une forte expérience pratique dans un domaine particulier
- En général, un expert est une personne compétente qui peut faire des choses que les autres ne peuvent pas faire
- Il doit être:
  - capable de communiquer ses connaissances
  - capable de réfléchir à ses propres connaissances et raisonnements internes
  - capable de participer et de consacrer beaucoup de temps au projet

- Membre le plus important de l'équipe de développement du système expert

Ingénieur des connaissances

- Quelqu'un capable de concevoir, de construire et de tester un système expert.
- Elle interroge l'expert du domaine pour savoir comment un problème particulier est résolu.
- Détermine les méthodes de raisonnement utilisées par l'expert pour traiter les faits et les règles et décide de la manière de les représenter dans le système expert
- Ils choisissent un logiciel de développement ou un shell de système expert.
- Il est responsable du test, de la révision et de l'intégration du système expert sur le lieu de travail.
- Travaille en étroite collaboration avec l'expert du domaine et l'utilisateur final.

Programmeur

- Personne responsable de la programmation proprement dite,
  - décrivant la connaissance du domaine en termes compréhensibles par un ordinateur.
- Nécessite des compétences en programmation symbolique :
  - Exemple: LISP, Prolog...
- Nécessite une expérience des systèmes experts en shell :
  - Exemple: CLIPS, Experta...
- Expérience en informatique classique dans les langages de langages de programmation :
  - Exemple: C, Pascal, FORTRAN, C+, Java, Python...

Chef de projet

- Chef de l'équipe de développement d'un système expert
- Responsable du maintien du projet sur la bonne voie
- Veille à ce que tous les produits livrables et toutes les étapes soient respectés
- Interagit avec tous les autres membres de l'équipe

Utilisateur final

- Personne qui utilise le système expert lorsqu'il est développé

- L'utilisateur doit en fin de compte être :
  - Avoir confiance dans les performances du système expert
  - Se sentir à l'aise en l'utilisant

- La conception de l'interface utilisateur est essentielle à la réussite du projet
  - La contribution de l'utilisateur peut être cruciale

Acquisition et ingénierie des connaissances

Acquisition de connaissances

- L'extraction et la formulation de connaissances provenant de diverses sources, en particulier d'experts humains
  - Prend environ 60 à 70 % du temps de développement d'un système expert.
- Experts humains :
  - Médecins, avocats, ingénieurs, analystes en investissement
- Autres sources :
  - Documents multimédias, manuels, bases de données, rapports de recherche et Internet.
- Les informations à recueillir sont typiques :
  - Vocabulaire ou jargon
  - Concepts et faits généraux
  - Les problèmes qui se posent couramment, les solutions aux problèmes qui se posent
  - Compétences pour résoudre des problèmes particuliers

Acquisition de connaissances auprès d'un expert

- Le processus d'acquisition des connaissances comprend généralement trois étapes principales
  1. L'élicitation (Action d'aider un expert à formaliser ses connaissances pour permettre de les sauvegarder ou de les partager) des connaissances est l'interaction entre l'expert et l'ingénieur des connaissances/le programme afin d'obtenir les connaissances de l'expert d'une manière systématique.
  2. Les connaissances ainsi obtenues sont généralement stockées dans une forme de représentation intermédiaire conviviale pour l'homme.
  3. Ingénierie des connaissances : La représentation intermédiaire des connaissances est ensuite compilée sous une forme exécutable (par exemple, des règles de production) que le moteur d'inférence peut traiter.

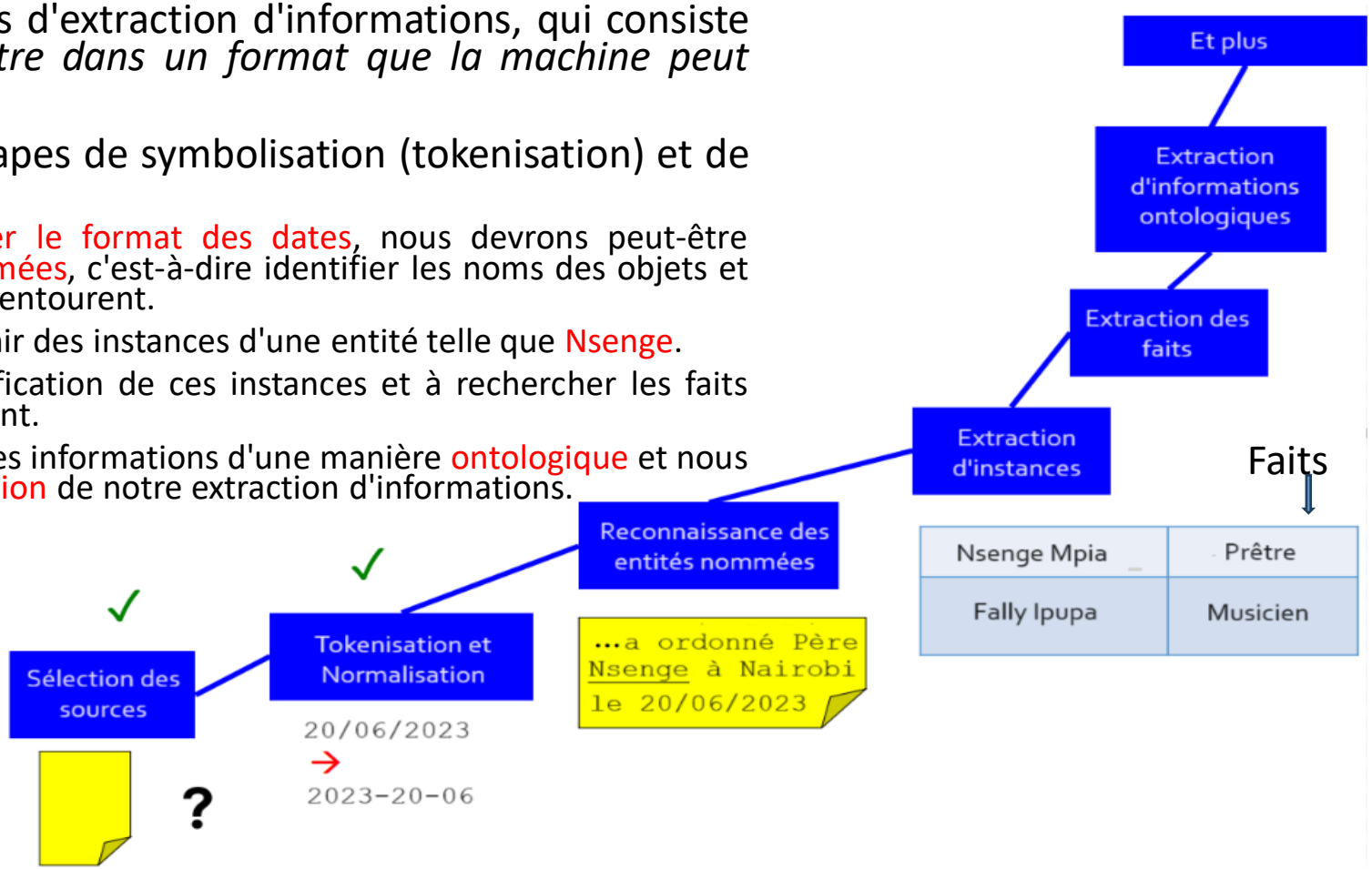
- De nombreuses itérations de ces étapes sont possibles
- Il s'agit d'un processus qui prend beaucoup de temps, et les techniques automatisées d'élicitation des connaissances et d'apprentissage automatique sont des alternatives modernes de plus en plus courantes.

Saisir les connaissances tacites/implicites

- L'un des problèmes que les ingénieurs de la connaissance rencontrent souvent est que les experts humains utilisent des connaissances tacites/implicites (par exemple, des connaissances procédurales) qui sont difficiles à saisir.
- Il existe plusieurs techniques utiles pour acquérir ces connaissances :
  1. **Analyse du protocole** :
     - Enregistrez l'expert en train de réfléchir à haute voix pendant qu'il joue son rôle et analysez-le par la suite.
     - Décomposez leur protocole/compte-rendu en unités atomiques de pensée les plus petites et laissez-les s'exprimer.
  2. **Observation des participants** :
     - L'ingénieur des connaissances acquiert des connaissances tacites par le biais d'une expérience pratique du domaine avec l'expert.
  3. **L'induction automatique** :
     - Cette méthode est utile lorsque les experts sont en mesure de fournir des exemples des résultats de leur prise de décision, même s'ils ne sont pas en mesure d'articuler les connaissances ou le processus de raisonnement sous-jacents.
- La ou les meilleures méthodes à utiliser dépendent généralement du domaine du problème et de l'expert.

Extraction d'informations

- Si les connaissances doivent être acquises à partir d'autres sources telles que les bases de données, les datasets ou le web, il peut être nécessaire de procéder à **l'extraction d'informations**.
  - Il s'agit du processus d'extraction d'informations structurées à partir de documents non structurés lisibles par une machine.
- L'organigramme illustré montre le processus d'extraction d'informations, qui consiste à *partir d'un matériau source pour le mettre dans un format que la machine peut traiter*.
- Il peut être nécessaire de passer par des étapes de symbolisation (tokenisation) et de normalisation.
  - Par exemple, si nous essayons de normaliser le format des dates, nous devrons peut-être procéder à la reconnaissance des entités nommées, c'est-à-dire identifier les noms des objets et rechercher les informations pertinentes qui les entourent.
  - L'étape d'extraction d'instances permet d'obtenir des instances d'une entité telle que Nsenge.
  - L'extraction de faits consiste à utiliser l'identification de ces instances et à rechercher les faits associés à ces instances dans le texte environnant.
  - Nous pouvons également essayer d'organiser ces informations d'une manière ontologique et nous pouvons aller plus loin en termes de sophistication de notre extraction d'informations.



Reconnaissance des entités nommées

- Il s'agit de trouver des entités (personnes, villes, organisations, dates, ...) dans un texte.
- Imaginons que nous ayons la phrase et le document :

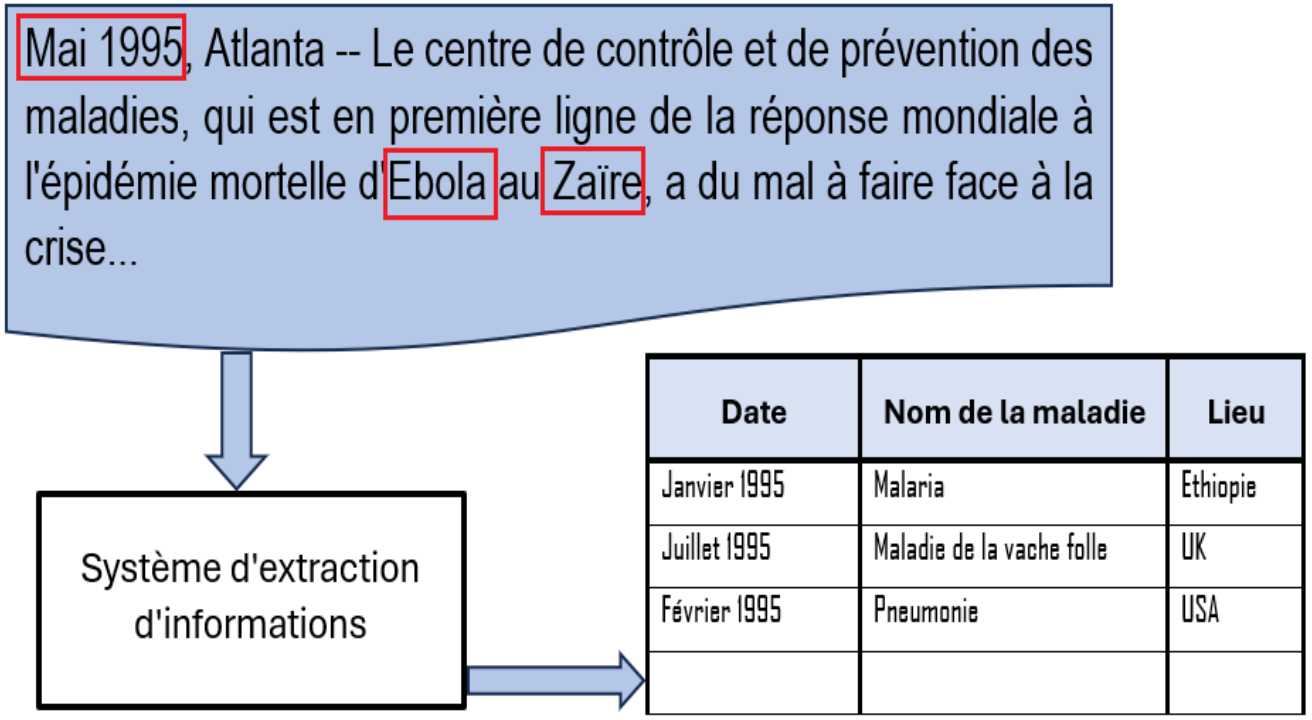
Nsenge Mpia a été ordonné prêtre en 2023 à Nairobi, Kenya

- Le processus de reconnaissance des entités nommées recherche des entités d'identification dans le texte.
- Dans ce cas, nous avons un nom, une année, une ville et un Pays.

Nsenge Mpia a été ordonné prêtre en 2023 à Nairobi, Kenya

Extraction de relations : Epidémies de maladies

- Une fois les entités nommées identifiées, nous pouvons également procéder à *l'extraction des relations*.
- Imaginons que nous voulions extraire d'un document des informations sur des épidémies.
- Nous pourrions d'abord rechercher les *entités nommées* telles que les dates, les lieux et les noms de maladies, puis convertir ce texte brut en un ensemble formaté de variables.
  - Cette opération peut être automatisée par un système d'extraction d'informations qui nous permet d'extraire les informations pertinentes du texte et de les introduire dans notre nouvelle base de données.

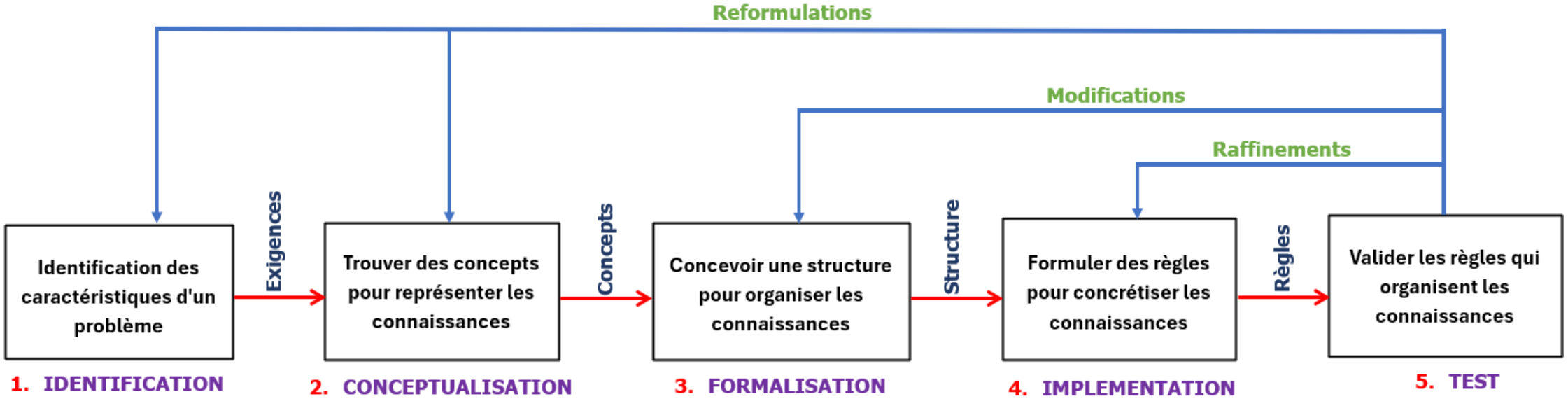


Analyse des niveaux de connaissance

- Identification des connaissances :
  - Utilisation des entretiens approfondis au cours desquels l'ingénieur des connaissances encourage l'expert à parler de la manière dont il fait ce qu'il fait
  - L'ingénieur des connaissances doit comprendre le domaine suffisamment bien pour savoir quels sont les objets et les faits dont il faut parler.
- Conceptualisation des connaissances :
  - Trouver les concepts primitifs et les relations conceptuelles du domaine du problème
- Analyse épistémologique :
  - Découvrir les propriétés structurelles de la connaissance conceptuelle, telles que les relations taxonomiques (classifications)
- Analyse logique :
  - Décider comment effectuer un raisonnement dans le domaine du problème.
  - Ce type de connaissances peut être particulièrement difficile à acquérir.
- L'analyse de la mise en œuvre/Implémentation :
  - Élaborer des procédures systématiques pour mettre en œuvre et tester le système

Les étapes de l'acquisition des connaissances

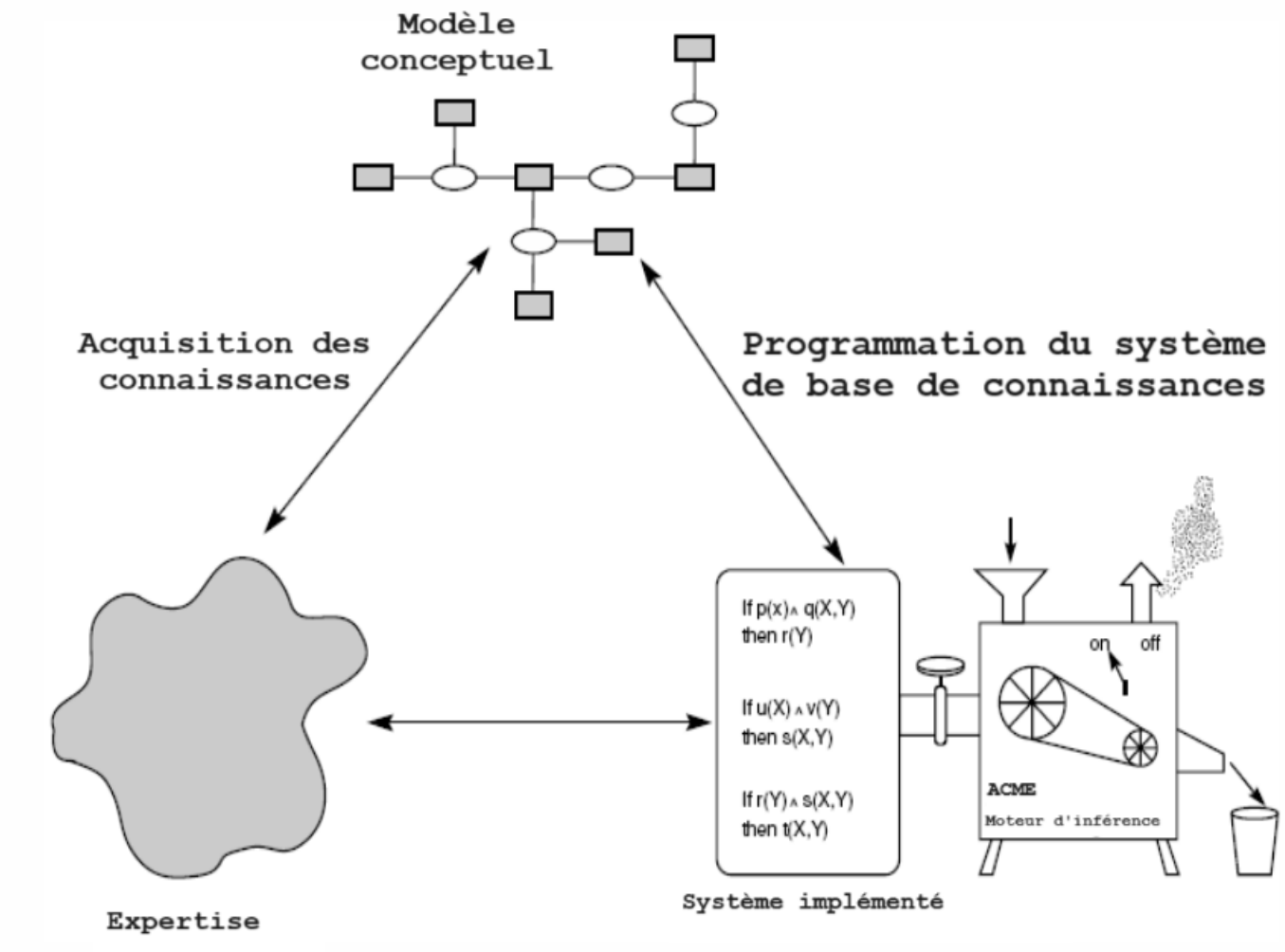
- Les choses commencent par l'identification **des caractéristiques d'un problème**, puis par la conceptualisation, suivie de la formalisation, puis de la mise en œuvre/implémentation et enfin de la phase de test.
- Le processus est en quelque sorte cyclique puisque nous affinons les connaissances dans le cadre du développement de notre système expert.



Acquisition de connaissances et modélisation

- Auparavant, la construction d'un système expert *relevait principalement d'une approche par essais et erreurs confiée à un ingénieur en connaissances*.
  - Une fois que ce dernier disposait des connaissances des experts, il remplissait sa base de connaissances et se contentait de la tester et de jouer avec elle jusqu'à ce qu'il obtienne un résultat satisfaisant.
- À la fin des années 80, on a découvert que la création d'un modèle de domaine réel était la voie à suivre, en d'autres termes, construire un modèle de connaissances avant de mettre en œuvre quoi que ce soit.
- Ce modèle pourrait être:
  - un graph de dépendance de ce qui peut causer ce qui se produit.
  - un modèle d'association qui est une collection de dysfonctionnements et les manifestations que nous attendons de ces dysfonctionnements.
  - Il peut également s'agir d'un modèle fonctionnel dans lequel les composants sont énumérés et décrits par des fonctions et des comportements.
- Ces dernières années, l'accent mis sur la meilleure façon de construire un système expert s'est déplacé vers l'utilisation d'outils d'acquisition de connaissances.
  - Un outil d'acquisition de connaissances est un système qui permet à un expert du domaine de saisir ses connaissances sous la forme d'un modèle graphique qui contient les composants de l'élément diagnostiqué ou conçu, ainsi que leurs fonctions et les règles permettant de décider comment diagnostiquer ou concevoir chacun d'entre eux.

Cycle d'ingénierie de base de connaissances avec KADS



- Nous illustrons ici un cycle d'ingénierie de base de connaissances utilisant l'acquisition de connaissances et la structuration de la documentation (KADS).
  - Ici, l'expert fournit une sorte de modèle conceptuel. L'expert fournit le modèle par le biais d'une approche KADS.
  - L'ingénieur de la connaissance construit une base de connaissances basée sur le modèle de domaine fourni.
  - Ici, le moteur d'inférence peut être prêt à l'emploi ou fabriqué sur mesure, ou encore une combinaison des deux.
- Cette stratégie permet d'automatiser la construction d'un système expert.

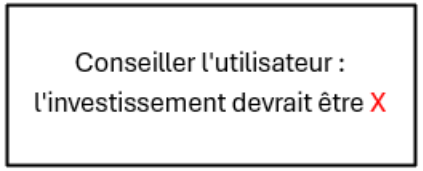
Note: un modèle conceptuel n'est pas exécutable, il s'agit plutôt d'une construction conceptuelle

Représentation intermédiaire

- Lorsque nous passons de l'acquisition des connaissances à l'ingénierie des connaissances du système expert lui-même, nous pouvons utiliser une représentation intermédiaire.
  - La représentation intermédiaire est une représentation structurée de la connaissance, qui n'est pas encore sous la forme d'un code pouvant être placé dans la base de connaissances d'un système expert.
  - Il peut s'agir, par exemple, de la décomposition d'un problème en un graphe AND-OR, comme nous l'avons vu dans les leçons précédentes.
    - Il s'agit d'une technique pratique pour réduire un problème en éléments nécessaires à la construction d'un système de production.
    - Tout d'abord, l'objectif principal est identifié ; il est divisé en deux ou plusieurs sous-objectifs ; ceux-ci sont également divisés.
    - Un graphe simple de but ou sous-buts est créé.
    - Chaque but est inscrit dans une case, appelée nœud, avec ses sous-buts en dessous, reliés par des liens.
    - Les nœuds feuilles au bas de l'arbre sont les cases au bas du graphique qui n'ont pas de liens en dessous d'eux.
      - Les données nécessaires pour résoudre le problème.

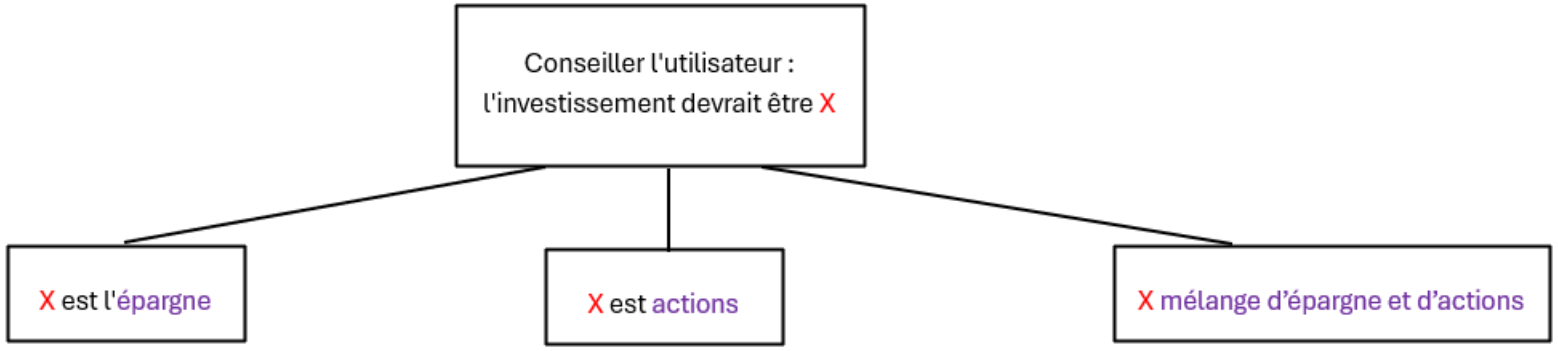
Exemple : Décomposition du problème

- Nous essayons de construire un graphe AND-OR. Nous allons commencer par placer une boîte en haut représentant le but.
- Dans ce cas, pour conseiller l'utilisateur sur le type d'investissement; notre investissement devrait être X



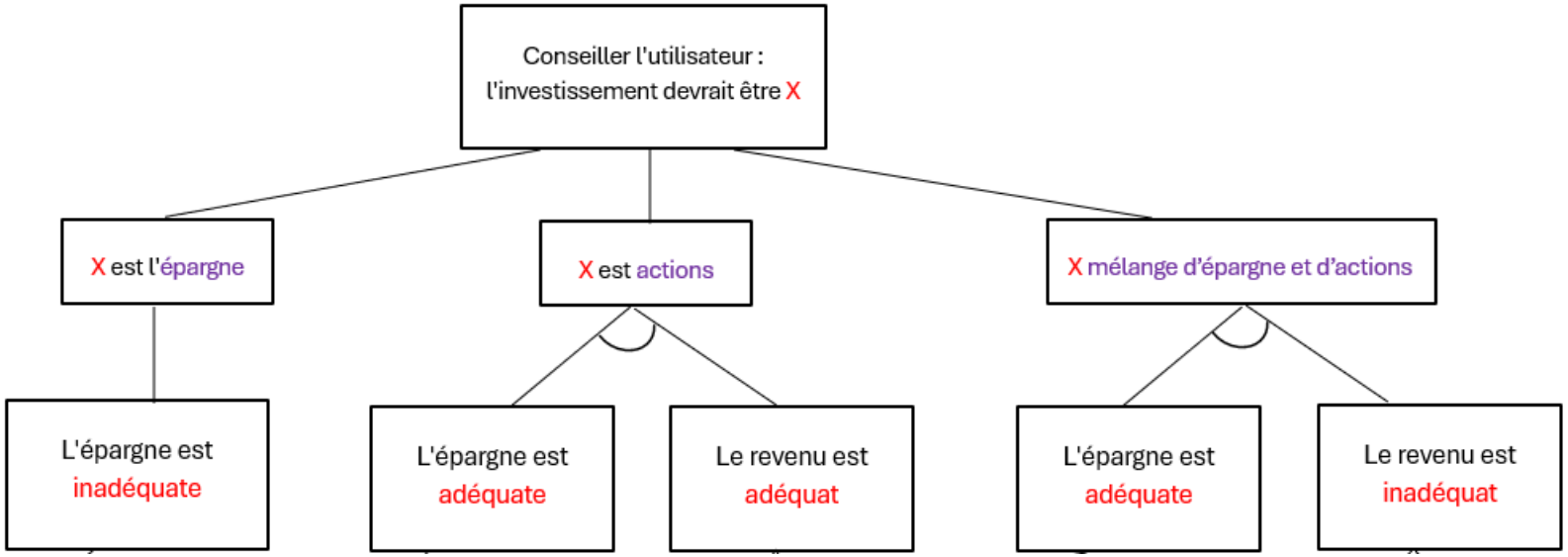
Exemple : Décomposition du problème (Cont.)

- Disons que nous avons ici trois options possibles pour les recommandations de type d'investissement à faire.
  - L'investissement peut se faire sous forme d'épargne, d'actions ou d'un mélange des deux.



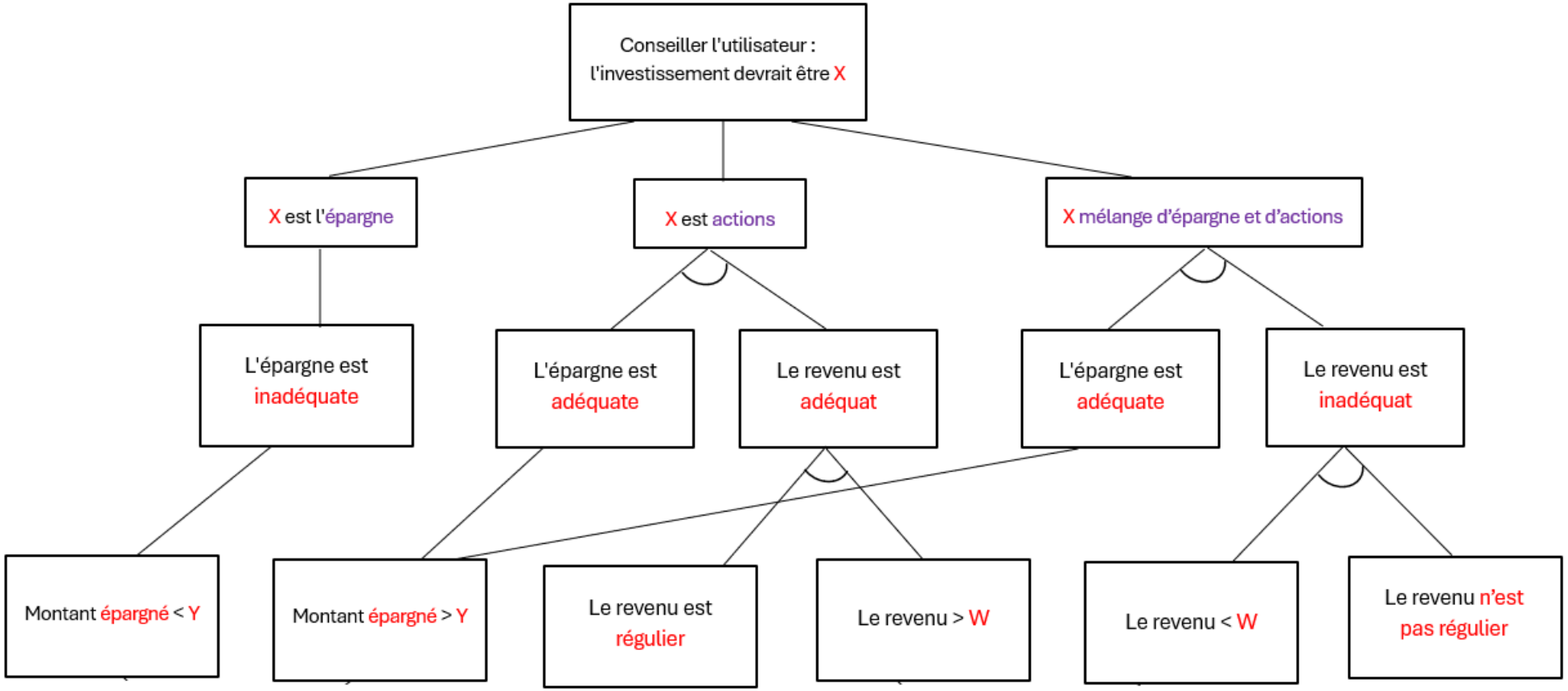
Exemple : Décomposition du problème (Cont.)

- Peut-être que l'information dont nous avons besoin pour recommander la mise en place d'une épargne est de vérifier si l'épargne est inadéquate.
  - En d'autres termes, si elle est inadéquate, nous devrions l'augmenter.
- En ce qui concerne les actions, nous devrons vérifier deux choses, car nous avons ici une branche ET(AND) indiquée par l'arc. Dans ce cas, nous recommanderons donc les actions si l'épargne est adéquate et si les revenus sont adéquats.
- Pour ce qui est de recommander un mélange des deux, nous devons également vérifier deux choses...



Exemple : Décomposition du problème (Cont.)

- Nous pouvons continuer à construire l'arbre pour décider si ces nœuds sont vrais ou faux.
- Ainsi, par exemple, nous pourrions vérifier si le revenu est inadéquat en regardant si le revenu est inférieur à une certaine valeur W et si le revenu n'est pas régulier.

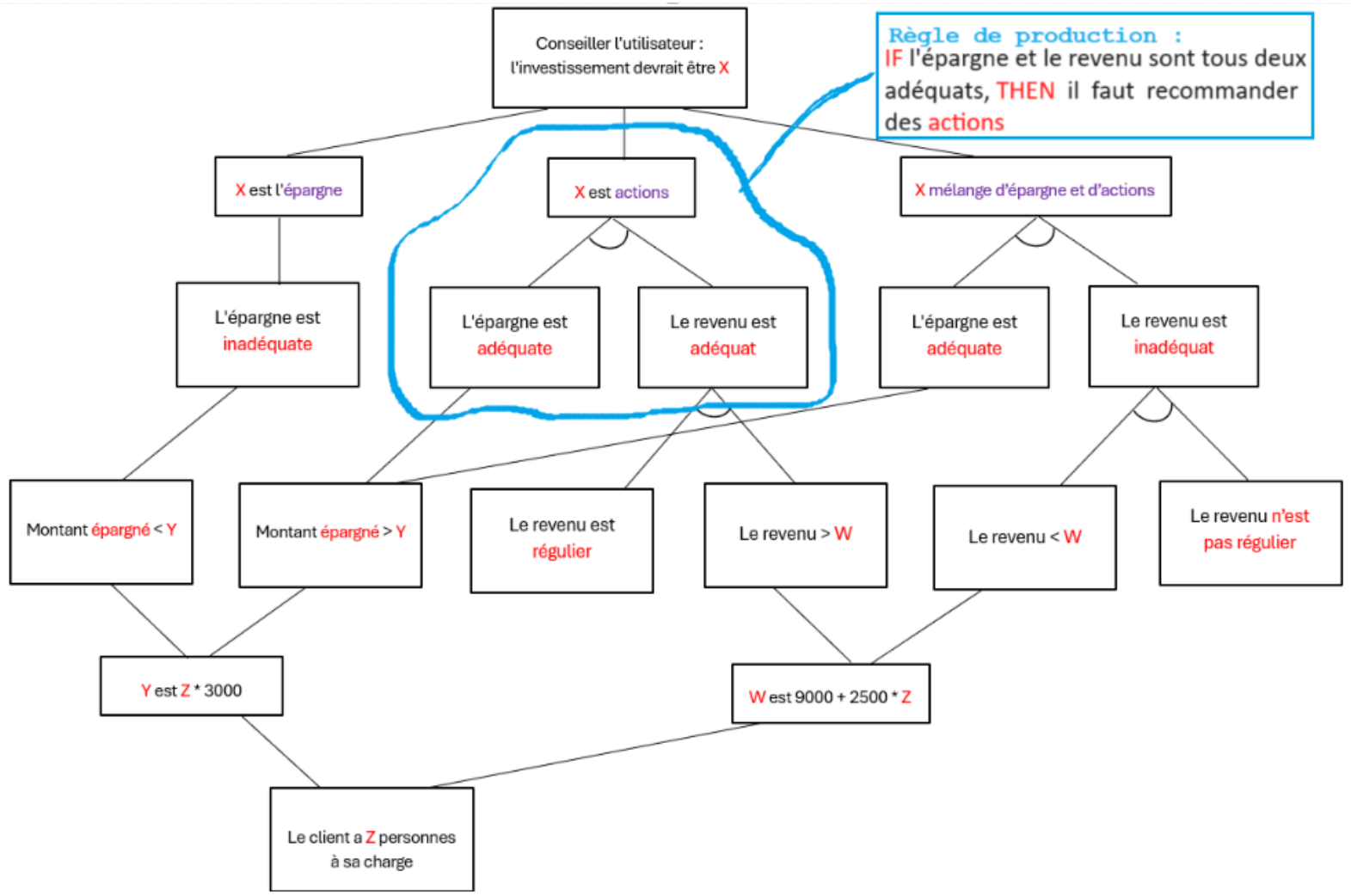


Exemple : Décomposition du problème (Cont.)

- À partir de cet arbre, nous pouvons maintenant identifier ce qui deviendra les règles de production.

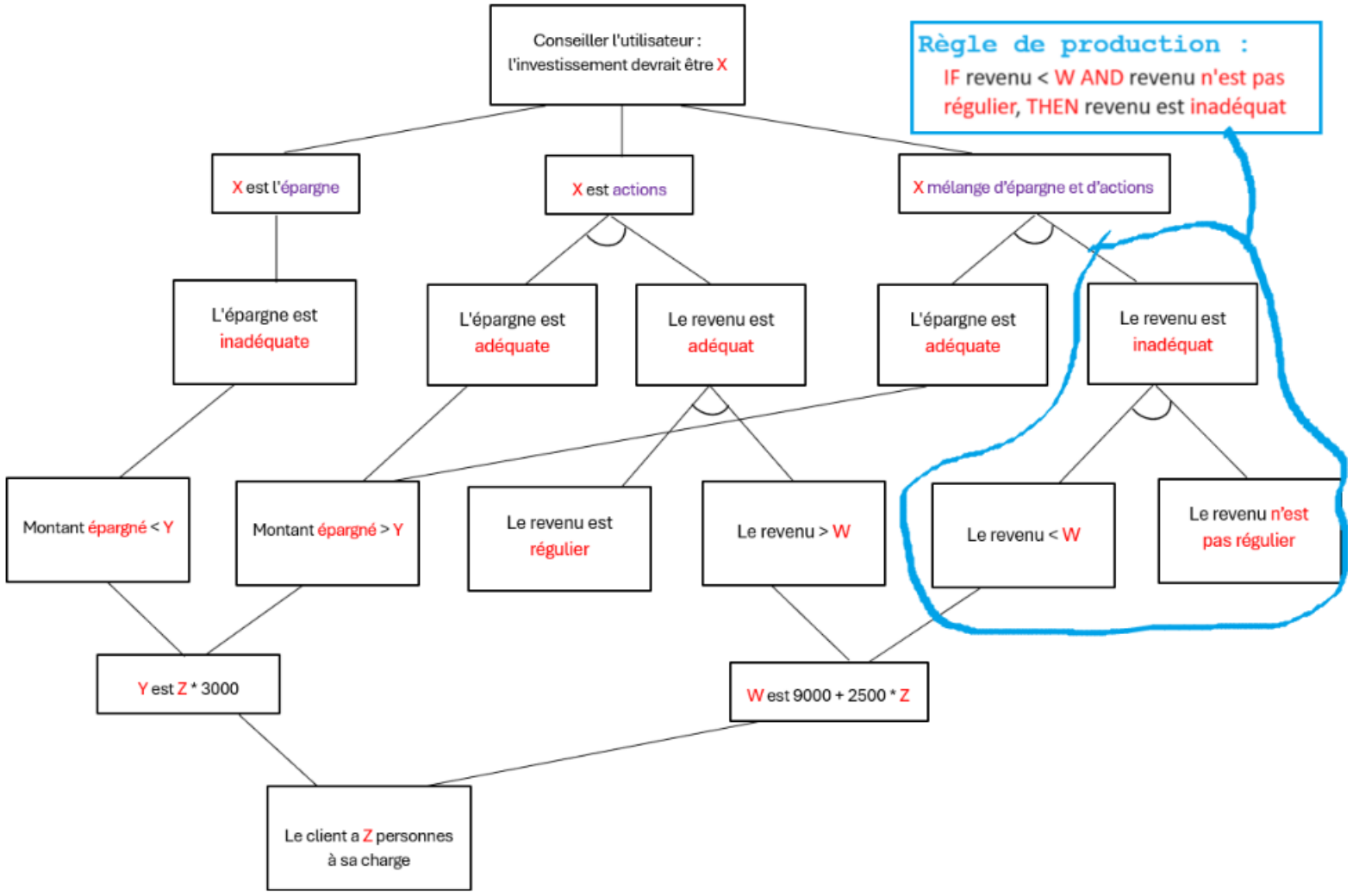
- Par exemple, dans la partie encerclée de l'arbre où la recommandation d'actions est basée sur l'adéquation de l'épargne et du revenu, nous pouvons maintenant dire que:

  - « IF l'épargne et le revenu sont tous deux adéquats, THEN il faut recommander des actions ».



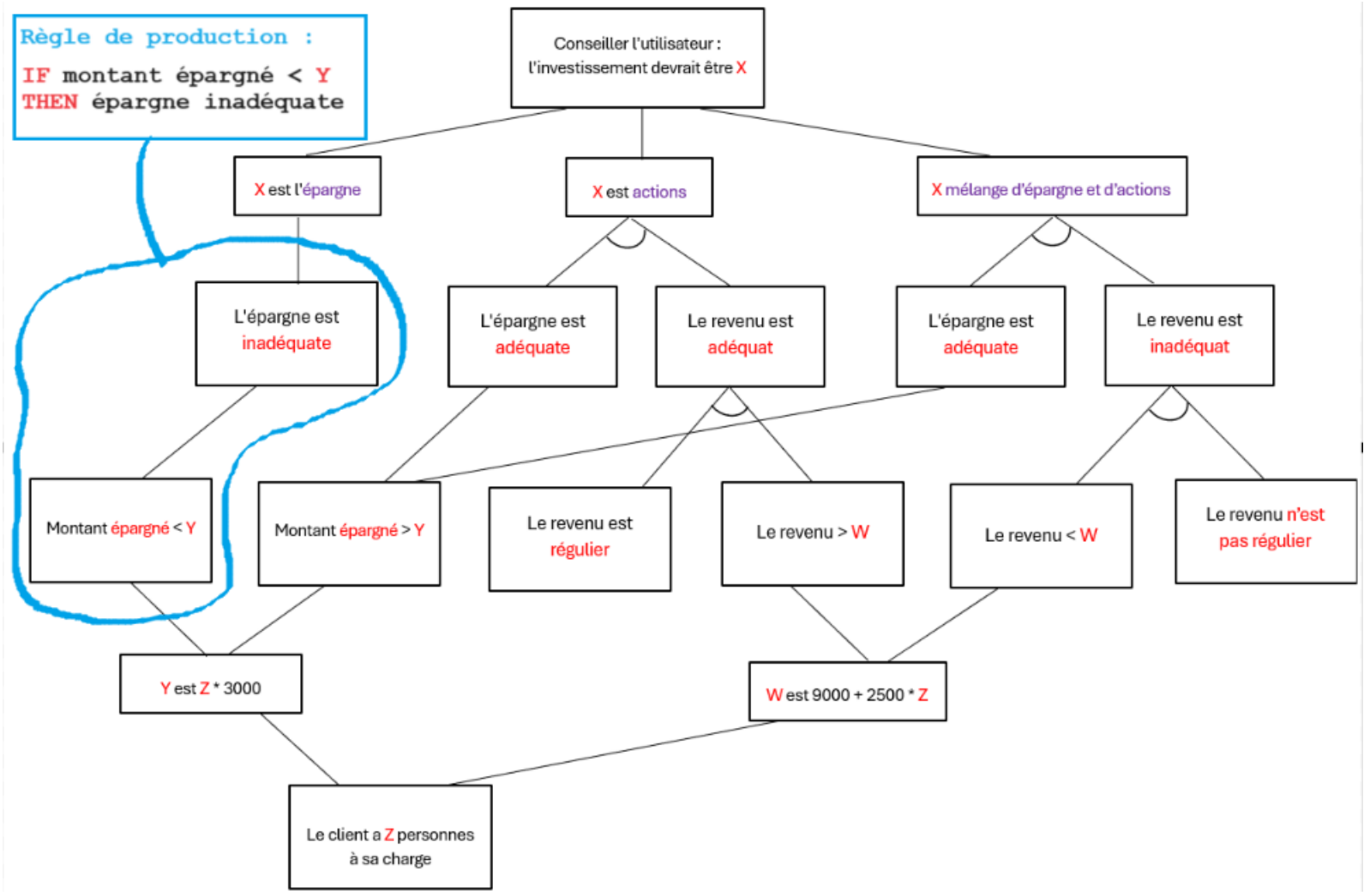
Exemple : Décomposition du problème (Cont.)

- Nous pouvons également décider que le revenu est inadéquat s'il est inférieur à W et n'est pas régulier



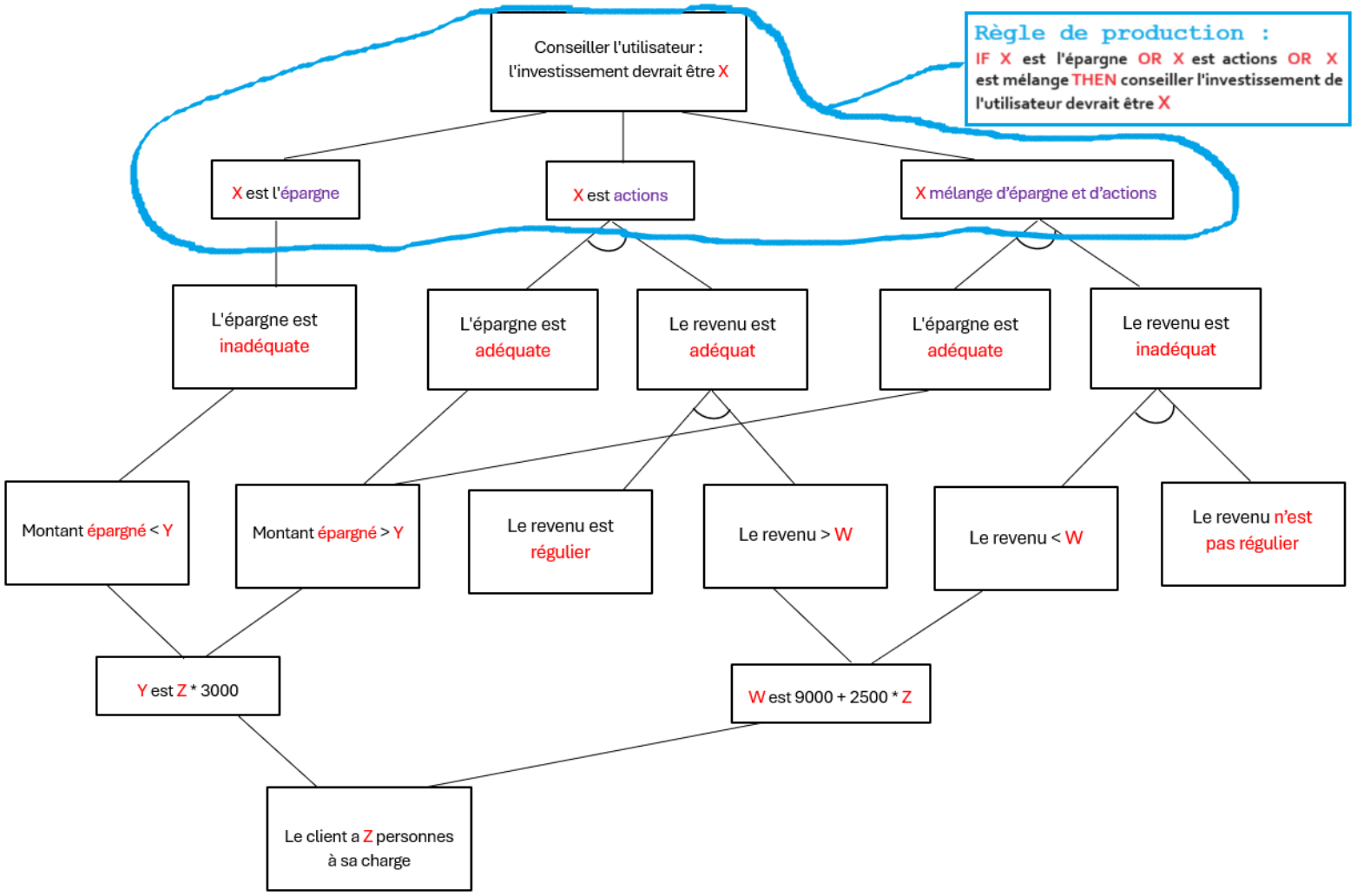
Exemple : Décomposition du problème (Cont.)

- En outre, nous pouvons conclure que l'épargne est inadéquat si le montant épargné est inférieur à Y.



Exemple : Décomposition du problème (Cont.)

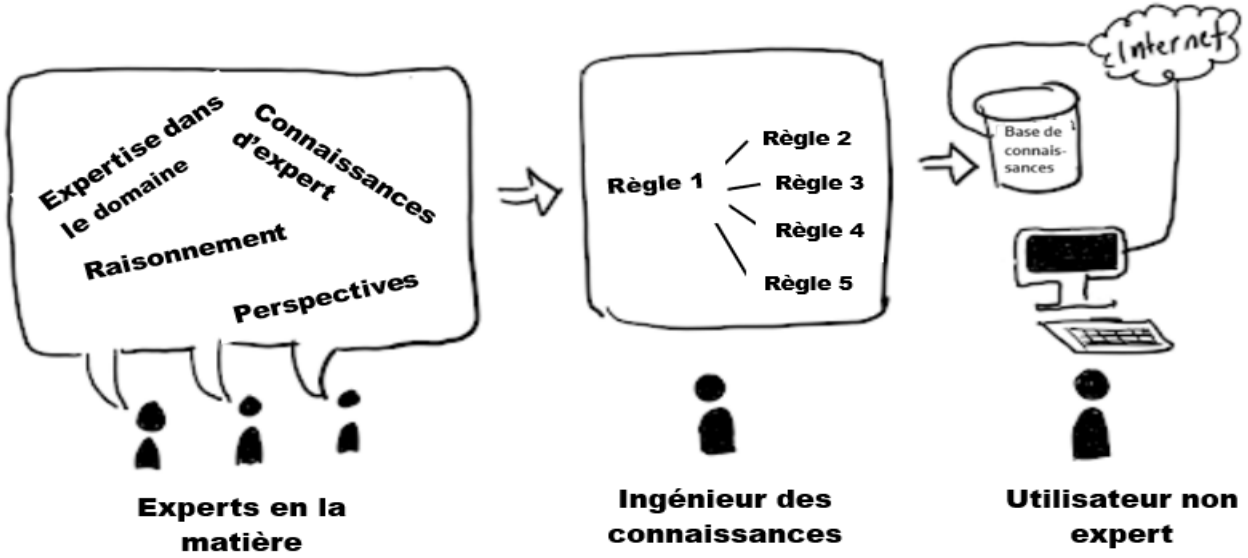
- En fin de compte, nous pouvons utiliser cet arbre pour identifier toutes les règles de production dont nous aurons besoin pour prendre toutes les décisions nécessaires pour recommander, dans ce cas, le type d'investissement qu'un utilisateur devrait faire dans une banque.



Ingénierie de la connaissance

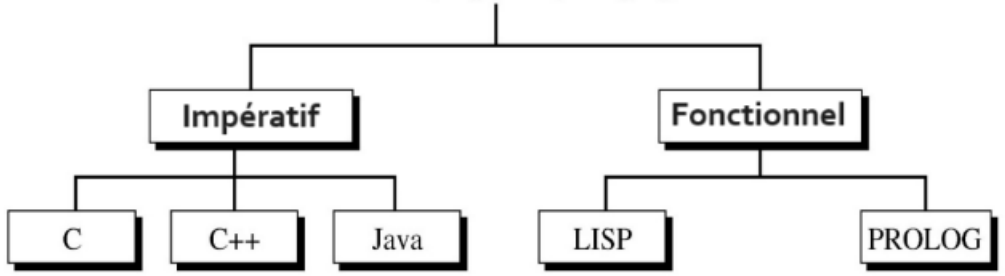
- Maintenant que nous avons parlé de l'acquisition des connaissances et un peu de la décomposition des problèmes, passons à l'ingénierie des connaissances proprement dite.

- C'est ici que nous codons la base de connaissances qui sera utilisée par le système expert.



Implémentation

- Notre première réflexion porte sur la manière dont nous voulons implémenter la base de connaissances.
- Il se peut que nous voulions utiliser des langages de programmation fonctionnelle déclaratifs tels que Lisp, Prolog et Miranda, qui sont généralement le premier choix pour la mise en œuvre de systèmes d'intelligence artificielle basés sur des règles, tels que les systèmes experts, dès le départ.
  - Parce que la structure syntaxique des règles écrites dans ces langages peut être étroitement liée à une forme choisie de représentation des connaissances, et les moyens par lesquels une requête est résolue à un modèle de raisonnement connexe.
  - Un programme déclaratif est une séquence de faits et de règles, un ensemble de conditions qui décrivent un espace de solution.
  - Il existe une certaine dépendance à l'égard de l'ordre dans lequel ces éléments sont écrits, mais pas autant que dans les programmes procéduraux.
- À la différence, un programme procédural/impératif consiste en une séquence de commandes.
  - Il est nécessaire que le programmeur réfléchisse soigneusement, pour chaque nouveau problème, aux étapes à suivre pour le résoudre et à l'ordre dans lequel elles doivent être exécutées.
  - Les réseaux neuronaux, qui sont un exemple d'application de "calcul de nombres", sont généralement mis en œuvre dans des langages procéduraux tels que Java, C/C++, etc.



Implémentation (Cont.)

- Choisir un outil de développement de système expert
  - Tenir compte des avantages en termes de coûts
  - Tenir compte de la fonctionnalité technique et de la flexibilité de l'outil
  - Tenir compte de la compatibilité de l'outil avec l'infrastructure d'information existante
  - Tenir compte de la fiabilité et de l'assistance du fournisseur
- Codage du système
  - La principale préoccupation à ce stade est de savoir si le processus de codage est efficace et correctement géré pour éviter les erreurs.
- Évaluation du système
  - Deux types d'évaluation :
    - Vérification : pas d'erreur dans le code et obtention de résultats identiques à ceux obtenus par l'expert.
    - Validation : résoudre le problème correctement

Autres considérations

- Lorsque nous construisons des systèmes experts, nous devons également nous demander si nous devons:
  - nous préoccuper du calcul des valeurs de certitude
  - ou de la détermination des priorités des règles lorsqu'elles se déclenchent,
  - ainsi que du type de mécanisme de résolution des conflits que nous pourrions souhaiter incorporer.
- Et aussi si nous avons un système qui va nécessiter une sorte de maintenance de la vérité
  - Un système de maintien de la vérité est utile si l'on souhaite raisonner avec des valeurs par défaut et des croyances.
  - Les systèmes de maintien de la vérité sont également appelés systèmes de maintien de la raison ou systèmes de révision des croyances.
    - Ils permettent essentiellement de maintenir la cohérence entre les anciennes connaissances et les connaissances actuelles dans la base de connaissances par le biais de la révision.
  - Si les croyances actuelles contredisent les connaissances contenues dans la base de connaissances, cette dernière est mise à jour avec les nouvelles connaissances.

    - Note: Dans certains cours d'intelligence artificielle, une leçon entière est consacrée à ce sujet, mais je l'ai simplement mentionné ici pour que vous sachiez que ces choses existent.

Interactions des règles

- Une mise en garde s'impose lorsque l'on réfléchit aux possibilités d'interactions entre les règles.
  - Lorsque vous tentez d'étendre un système expert en ajoutant simplement de nouvelles règles, cela peut s'avérer dangereux pour sa fonctionnalité.
    - Plus précisément, des interactions inattendues entre les règles sont susceptibles de se produire, vous pourriez ajouter de nouveaux conflits qui doivent être résolus ou quelque chose de ce genre.
    - La nécessité de prendre en compte toutes ces interactions possibles entre les règles rend les grands systèmes basés sur des règles peu maniables et difficiles à mettre à jour.
  - Ainsi, s'il peut être facile de saisir la signification des règles individuelles, il peut être beaucoup plus difficile d'appréhender les questions liées aux interactions entre les règles.

Maintenance des systèmes experts

- Une fois qu'un système expert a été développé et mis en œuvre, il est important de garder à l'esprit qu'il peut nécessiter une certaine maintenance au fil des ans.

- Ainsi, même une fois développée, la base de connaissances d'un système expert doit être continuellement mise à jour par l'ajout, la suppression ou la modification de règles afin de tenir compte des nouvelles connaissances et des changements de circonstances.

- La maintenance n'est peut-être pas vraiment importante pour tous les problèmes liés aux systèmes experts, mais pour un grand nombre d'applications dans le monde réel, il s'agit d'une chose sérieuse à laquelle il faut penser.

Introduction à Experta

Qu'est-ce que Experta?

- Experta est un système expert développé en Python qui implémente un moteur d'inférence à base de règles.

- Il est conçu pour permettre la création et l'exécution de systèmes experts basés sur des règles de manière simple et efficace.

- Experta offre des fonctionnalités telles que la création de faits, de règles et l'exécution de l'inférence à base de règles pour résoudre des problèmes complexes en utilisant des connaissances humaines…

- https://readthedocs.org/projects/experta/downloads/pdf/stable/

Pourquoi Experta ?

- Experta présente plusieurs avantages pour le développement de systèmes experts :

1. **Facile à utiliser**: Experta offre une syntaxe simple et intuitive pour définir des faits et des règles, ce qui facilite la création de systèmes experts même pour les débutants.
2. **Moteur d'inférence puissant**: Experta utilise un moteur d'inférence avancé qui permet d'effectuer des inférences efficaces sur la base de règles définies, ce qui permet de résoudre des problèmes complexes.
3. **Personnalisable**: Experta est hautement personnalisable, ce qui permet aux développeurs de définir leurs propres types de faits et de règles pour répondre à des besoins spécifiques.
4. **Documentation complète**: Experta est bien documenté, avec des exemples et des tutoriels détaillés, ce qui facilite l'apprentissage et l'utilisation du système.
5. **Open-source**: Experta est un logiciel open-source, ce qui signifie qu'il est gratuit à utiliser et que sa communauté de développeurs est active, offrant un support et des mises à jour régulières.

- Experta est une bibliothèque Python pour construire des systèmes experts fortement inspirés de CLIPS.

- Le système expert le plus couramment utilisé aujourd'hui s'appelle CLIPS(C Language Integrated Production System), mais il s'agit d'un système interprété en langage de codage C
  - Si vous souhaitez (éventuellement) découvrir CLIPS, cliquez sur le lien suivant ci-dessous
  - Lien : Site web de CLIPS http://clipsrules.sourceforge.net/

Composants de Experta

- **DefFacts** est un décorateur en **Experta** qui permet de définir des faits initiaux (ou des faits de départ) pour le système expert.
  - Ces faits seront automatiquement ajoutés à la base de faits de l'expert système au démarrage. Cela est utile pour définir des faits de base ou des faits de contexte qui sont nécessaires pour le fonctionnement initial du système expert .
  - Un décorateur en Python est une fonction qui prend une autre fonction en argument et retourne une nouvelle fonction modifiée ou étendue. Les décorateurs sont utilisés pour modifier ou étendre le comportement des fonctions ou des classes sans modifier leur code source.
  - Voici un exemple d'utilisation de **DefFacts** pour définir des faits initiaux dans **Experta** :

```
1  # Définition des faits initiaux
2  @DefFacts()
3  def faits_initiaux():
4      yield Symptome("Fièvre")
5      yield Symptome("Toux")
```

  - Dans l'exemple ci-haut, **faits_initiaux()** est utilisé avec **DefFacts** pour définir deux faits initiaux (Symptome("Fièvre") et Symptome("Toux")) qui seront ajoutés à la base de faits du système expert au démarrage. Ces faits peuvent être utilisés dans les règles du système expert pour déclencher des actions ou des déductions.

Composants de Experta (Cont.)

- **DefFacts** est un décorateur en **Experta** qui permet de définir des faits initiaux (ou des faits de départ) pour le système expert.
  - Ces faits seront automatiquement ajoutés à la base de faits de l'expert système au démarrage. Cela est utile pour définir des faits de base ou des faits de contexte qui sont nécessaires pour le fonctionnement initial du système expert .
  - Un décorateur en Python est une fonction qui prend une autre fonction en argument et retourne une nouvelle fonction modifiée ou étendue. Les décorateurs sont utilisés pour modifier ou étendre le comportement des fonctions ou des classes sans modifier leur code source.
  - Voici un exemple d'utilisation de **DefFacts** pour définir des faits initiaux dans **Experta** :

Composants de Experta (Cont.)

- **Moteur d'inférence (Inference Engine):** Le moteur d'inférence est le composant qui applique les règles aux faits pour déduire de nouvelles informations.
  - Il effectue le processus de raisonnement du système expert en utilisant les règles définies pour arriver à des conclusions.

- Dans l'exemple ci-dessous, le moteur d'inférence est représenté par la classe **MonMoteurInference** qui hérite de **KnowledgeEngine**.

```
1  # Définition des règles
2  class MonMoteurInference(KnowledgeEngine):
3      @Rule(AND(Symptom(name='fièvre', value=True),
4                Symptom(name='toux', value=True)))
5      def rule_flu(self):
6          self.declare(Diagnosis('grippe'))
7
8      @Rule(AND(Symptom(name='fièvre', value=True),
9                Symptom(name='gorge_irrite', value=True)))
```

- Les règles sont définies à l'intérieur de cette classe à l'aide des annotations **@Rule**.

Composants de Experta (Cont.)

- **Règles(Rules):** Les règles sont des instructions qui indiquent au système expert comment traiter les faits pour parvenir à un résultat.
- Chaque règle est composée d'une condition (ou ensemble de conditions) qui, si elle est remplie, déclenche l'exécution de la règle.
- Les règles sont utilisées pour modéliser la logique du domaine d'expertise.

Composants de Experta (Cont.)

- **Le cycle de vie d'un système expert**: Le cycle de vie d'un système expert, comme celui implémenté avec la librairie Experta en Python, comprend généralement les étapes suivantes :

1. **Initialisation** : Lors de l'initialisation, les faits de départ sont définis et le moteur d'inférence est prêt à être utilisé.
2. **Récupération des faits** : Le système peut récupérer des faits à partir de différentes sources, telles que des entrées utilisateur, des bases de données ou des **capteurs**.
3. **Injection des faits** : Les faits récupérés sont injectés dans le moteur d'inférence à l'aide de la méthode *declare()* pour être utilisés dans le raisonnement.
4. **Raisonnement** : Le moteur d'inférence applique les règles définies pour déduire de nouveaux faits à partir des faits existants.
5. **Exécution des règles** : Les règles sont exécutées en fonction des faits présents dans la base de faits.
6. **Modification des faits** : Les faits peuvent être modifiés ou **rétractés** en fonction des nouvelles informations déduites pendant le raisonnement.
7. **Affichage des résultats** : Les résultats du raisonnement, tels que les diagnostics ou les recommandations, sont affichés à l'utilisateur.
8. **Réinitialisation** : Après avoir terminé une série de raisonnements, le système peut être réinitialisé pour être prêt à traiter de nouvelles entrées.

```
# Réinitialisation du système pour une nouvelle utilisation
expert_system.reset()
```

- Ces étapes sont gérées par le moteur d'inférence pour assurer le bon fonctionnement du système.

Installation d’Experta

- Pour installer experta, on va saisir: *pip install experta*
- Dans experta, le chaînage arrière n'est pas directement supporté comme dans d'autres systèmes experts.
- Cependant, vous pouvez émuler le chaînage arrière en demandant manuellement à l'utilisateur des données basées sur les règles et les faits du système.

TD 1: Prédiction météo/Chaînage avant

- Dans cet exemple, le système expert prédit les conditions météorologiques sur la base des données de température et d'humidité saisies par l'utilisateur.
- L'interface utilisateur invite l'utilisateur à saisir les niveaux de température et d'humidité, puis le système expert utilise le chaînage avant pour prédire si le temps sera ensoleillé, nuageux ou pluvieux.

TD 1: Prédiction météo/Chaînage avant

- Cet exercice est écrit en Jupyter notebook
- Il comprend trois règles sur les conditions de météo

Entrée [1]:
```
#Cette instruction importe toutes les classes et fonctions nécessaires de la bibliothèque Experta pour créer un système expert.
from experta import *
```

Entrée [2]:
```
# Cette classe définit un fait (Fact) nommé "Climat" qui est utilisé pour représenter les informations météorologiques
#telles que la température et l'humidité.
class Climat(Fact):
    """Informations du climat"""
    pass
```

Entrée [3]:

```python
# Cette classe définit le moteur d'inférence pour prédire la météo.
#Elle hérite de la classe KnowledgeEngine de la bibliothèque Experta.
class PredictionClimat(KnowledgeEngine):
    #__init__(self) est la méthode d'initialisation de la classe PredictionClimat.
    #Elle définit une variable self.prediction pour stocker la prédiction météorologique.
    #En Python, self est un paramètre spécial utilisé dans la définition de méthodes d'une classe. C'est comme this en Java
    def __init__(self):
        super().__init__()
        #On initialise l'attribut prediction de l'instance de la classe PredictionClimat à None.
        #Cela est utile car, dans la méthode predict_ensoleille, predict_nuageux, et predict_pluvieux, on modifie la valeur de
        #self.prediction en fonction des règles métier définies. Cependant, si aucune de ces règles n'est déclenchée lors de
        #l'exécution du moteur de règles, la valeur de self.prediction restera None.
        self.prediction = None
        #Début des règles
        #@Rule sont des décorateurs qui définissent les règles du système expert. Chaque règle est une méthode qui spécifie
        #les conditions (représentées par des faits) pour déclencher la règle et la prédiction associée.
    #La règle ci-dessous peut être interprétée comme suit :
        #Si la température est "chaude" et l'humidité est "élevée", OU
        #Si la température est "chaude" et l'humidité est "normale", OU
        #Si la température est "douce" et l'humidité est "élevée", alors cette règle est déclenchée.
    #En d'autres termes, la règle prédit un temps ensoleillé en fonction de ces conditions météorologiques.
    #Si l'une des combinaisons de conditions est vraie, la méthode predict_ensoleille de la classe PredictionClimat
    #sera appelée, ce qui entraînera la prédiction de "Ensoleillé" dans l'attribut self.prediction.
    @Rule(OR(
        AND(Climat(temperature='chaude'), Climat(humidite='élevée')),
        AND(Climat(temperature='chaude'), Climat(humidite='normale')),
        AND(Climat(temperature='douce'), Climat(humidite='élevée'))
    ))
    #On définit une méthode appelée predict_ensoleille prenant self comme premier paramètre, ce qui signifie qu'elle agit sur
    #l'instance prediction de la classe PredictionClimat.
    def predict_ensoleille(self):
        #On affecte la chaîne de caractères 'Ensoleillé' à l'attribut prediction de l'instance actuelle de PredictionClimat.
        #Cela signifie que selon les règles définies dans le système expert, si les conditions pour un temps ensoleillé sont
        #remplies, cette méthode mettra à jour la prédiction de la météo de l'instance actuelle à 'Ensoleillé'.
        self.prediction = 'Ensoleillé'
    #Cette règle prédit un temps nuageux en fonction de la condition météorologique: temperature est douce et humidité normale.
    @Rule(AND(Climat(temperature='douce'), Climat(humidite='normale')))
    #On définit la méthode appelée predict_nuageux prenant self comme premier paramètre, ce qui signifie qu'elle agit sur
    #l'instance prediction de la classe PredictionClimat au cas où la condition nuageux est respectée.
    def predict_nuageux(self):
        #On affecte Nuageux dans prediction au cas où la condition ci-dessus est remplie
        self.prediction = 'Nuageux'
    #Règle au cas où temperature est bonne et humidité est élevée ou la temperature est bonne et humidité est normale
    @Rule(OR(
        AND(Climat(temperature='bonne'), Climat(humidite='élevée')),
        AND(Climat(temperature='bonne'), Climat(humidite='normale'))
    ))
    #On modifie la valeur de prediction en Pluvieux
    def predict_pluvieux(self):
        self.prediction = 'Pluvieux'
```

Entrée [4]:
```
# Cette instruction crée une instance du moteur d'inférence PredictionClimat pour notre système expert.
engine = PredictionClimat()
```

Entrée [5]:
```
#Cette ligne demande à l'utilisateur d'entrer la température actuelle.
temperature = input("Entrez la température (chaude, douce, bonne): ")
#Cette ligne demande à l'utilisateur d'entrer le niveau d'humidité actuel.
humidite = input("Entrez l'humidité (élevée, normale): ")
```

Entrée [6]:
```
#Cette méthode réinitialise le moteur de règles pour supprimer les faits précédemment déclarés.
engine.reset()
#L'instruction ci-dessous crée un fait de type Climat avec les valeurs de temperature et humidite fournies en arguments,
#puis déclare ce fait dans le moteur d'inférence "engine". Cela signifie que le moteur d'inférence prendra en compte ce fait
#lors de l'exécution des règles pour déterminer la prédiction météorologique.
#Cela signifie que le moteur d'inférence "engine" utilisera les valeurs de température et d'humidité que l'utilisateur
#a entrées (temperature et humidite) pour déterminer la prédiction météorologique en fonction des règles
#que n'avons définies dans la classe PredictionClimat.
engine.declare(Climat(temperature=temperature, humidite=humidite))
```

Entrée [7]:
```
#Cette méthode exécute le moteur d'inférence pour déterminer la prédiction météorologique en fonction des faits déclarés.
engine.run()
```

Entrée [8]:
```
#Cette ligne affiche la prédiction météorologique obtenue par le système expert.
print("Prédiction météo:", engine.prediction)
```

TD 2: Prédiction météo/Chaînage avant avec UI

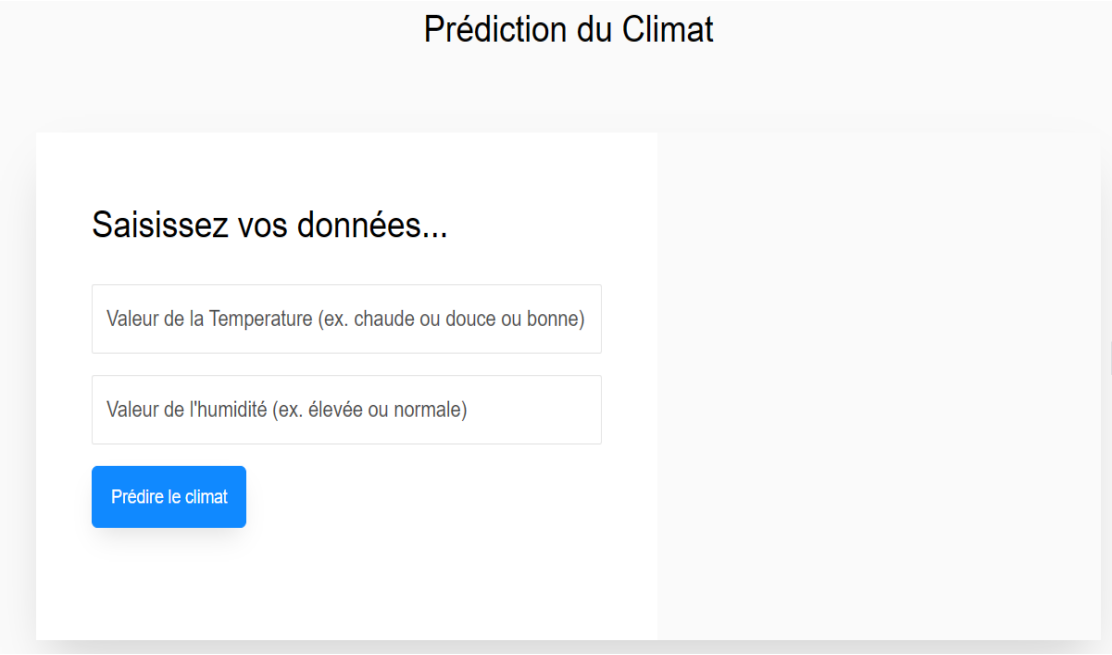
- Cet exemple montre comment vous pouvez créer une interface web pour un système expert experta en utilisant Flask.
- L'utilisateur peut saisir la température et l'humidité, et le système expert prédit les conditions météorologiques en fonction des données saisies.

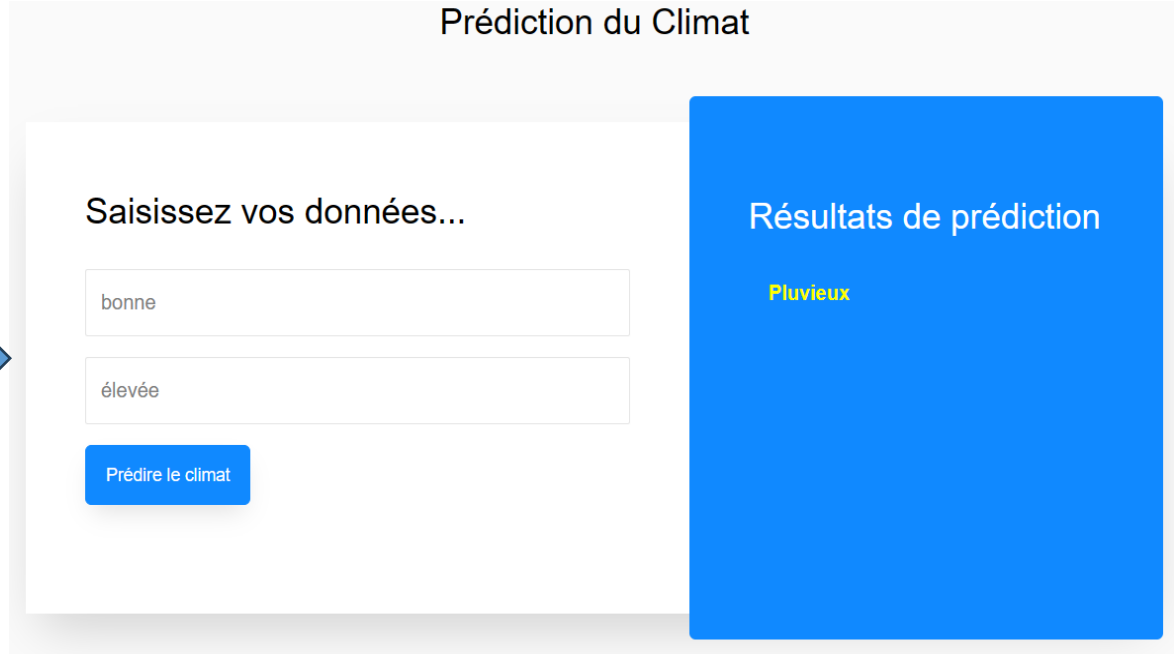
TD 2: Prédiction météo/Chaînage avant avec UI

- On va adapter l'exemple ci-haut pour créer une application avec l'interface utilisateur en utilisant Flask

- On doit avoir un dossier contenant ces fichiers:
  - Un dossier nommé static dans lequel on doit mettre les fichiers css et js
  - Un dossier templates pour mettre nos fichiers HTML
  - Un fichier nommé app.py qui contient notre code Experta et Flask

Ce qu’on veut construire comme système





Contenu du fichier app.py

```python
#On importe les librairies utile pour notre code
from flask import Flask, render_template, request
#On importe toutes les classes de Experta
from experta import *
#On crée une instance de l'application Flask.
app = Flask(__name__)
#Cette classe définit un fait (Fact) nommé "Climat" qui est utilisé pour représenter les informations
#météorologiques telles que la température et l'humidité.
class Climat(Fact):
    """Informations du Climat"""
    pass
#Cette classe définit le moteur d'inférence pour prédire la météo.
#Elle hérite de la classe KnowledgeEngine de la bibliothèque Experta.
class PredictionClimat(KnowledgeEngine):
    #__init__(self) est la méthode d'initialisation de la classe PredictionClimat.
    #Elle définit une variable self.prediction pour stocker la prédiction météorologique.
    #En Python, self est un paramètre spécial utilisé dans la définition de méthodes d'une classe.
    #C'est comme this en Java
    def __init__(self):
        super().__init__()
        #On initialise l'attribut prediction de l'instance de la classe PredictionClimat à None.
        #Cela est utile car, dans la méthode predict_ensoleille, predict_nuageux, et predict_pluvieux, on modifie la valeur de
        #self.prediction en fonction des règles métier définies. Cependant, si aucune de ces règles n'est déclenchée lors de
        #l'exécution du moteur de règles, la valeur de self.prediction restera None.
        self.prediction = None
    #Début des règles
        #@Rule sont des décorateurs qui définissent les règles du système expert. Chaque règle est une méthode qui spécifie
        #les conditions (représentées par des faits) pour déclencher la règle et la prédiction associée.
    #La règle ci-dessous peut être interprétée comme suit :
        #Si la température est "chaude" et l'humidité est "élevée", OU
        #Si la température est "chaude" et l'humidité est "normale", OU
        #Si la température est "douce" et l'humidité est "élevée", alors cette règle est déclenchée.
    #En d'autres termes, la règle prédit un temps ensoleillé en fonction de ces conditions météorologiques.
    #Si l'une des combinaisons de conditions est vraie, la méthode predict_ensoleille de la classe PredictionClimat
    #sera appelée, ce qui entraînera la prédiction de "Ensoleillé" dans l'attribut self.prediction.
```

Contenu du fichier app.py (Cont.)

```
36      @Rule(OR(
37          AND(Climat(temperature='chaude'), Climat(humidite='élevée')),
38          AND(Climat(temperature='chaude'), Climat(humidite='normale')),
39          AND(Climat(temperature='douce'), Climat(humidite='élevée'))
40      ))
41      #On définit la méthode au cas où c'est ensoleillé
42      def predict_ensoleille(self):
43          self.prediction = 'Ensoleillé'
44      #Règle pour le cas où c'est nuageux
45      @Rule(AND(Climat(temperature='douce'), Climat(humidite='normale')))
46      def predict_nuageux(self):
47          self.prediction = 'Nuageux'
48      #Règle au cas où c'est pluvieux
49      @Rule(OR(
50          AND(Climat(temperature='bonne'), Climat(humidite='élevée')),
51          AND(Climat(temperature='bonne'), Climat(humidite='normale'))
52      ))
53      def predict_pluvieux(self):
54          self.prediction = 'Pluvieux'
55  #Cette instruction crée une instance du moteur d'inférence PredictionClimat pour notre système expert.
56  engine = PredictionClimat()
57  #On définit une route pour l'URL racine de l'application, qui peut accepter les méthodes GET et POST.
58  @app.route('/', methods=['GET', 'POST'])
59  # la méthode index() est la vue associée à la route. Elle récupère les données de température et d'humidité soumises via un formulaire,
60  #les transmet à l'expert système pour la prédiction, puis affiche la prédiction sur une page HTML.
61  def index():
62      #On déclare ceci au cas où aucun critère de prédiction est trouvé
63      prediction_vide = False
64      #On vérifie si on a cliqué sur un bouton en HTML appartenant à la balise form utilisant la méthode POST
65      if request.method == 'POST':
66          #On récupère la temperature saisie dans le formulaire HTML
67          temperature = request.form['temperature']
68          #On récupère l'humidité' saisie dans le formulaire HTML
69          humidite = request.form['humidite']
70          #Cette méthode réinitialise le moteur de règles pour supprimer les faits précédemment déclarés.
```

Contenu du fichier app.py (Cont.)

```
71          engine.reset()
72          #L'instruction ci-dessous crée un fait de type Climat avec les valeurs de temperature et humidite fournies en arguments,
73          #puis déclare ce fait dans le moteur d'inférence "engine". Cela signifie que le moteur d'inférence prendra en compte ce fait
74          #lors de l'exécution des règles pour déterminer la prédiction météorologique.
75          engine.declare(Climat(temperature=temperature, humidite=humidite))
76          #Cette méthode exécute le moteur d'inférence pour déterminer la prédiction météorologique en fonction des faits déclarés.
77          engine.run()
78          #On obtient la valeur prédite du climat à partir de engine qui est l'instance de notre système expert
79          prediction = engine.prediction
80          #prediction_vide = True est utilisé pour déterminer si aucune prédiction météorologique n'a été trouvée pour les conditions
81          #météorologiques fournies par l'utilisateur. Lorsque l'application reçoit des données de température et d'humidité via un
82          #formulaire et exécute le système expert pour prédire le climat, elle vérifie si une prédiction a été faite.
83          #Si aucune prédiction n'a été faite (c'est-à-dire que engine.prediction est None), prediction_vide est définie sur True.
84          #Cela est utilisé dans le template HTML (index.html) pour afficher un message spécial ou une indication à l'utilisateur
85          #indiquant que la prédiction n'a pas pu être effectuée en raison de données insuffisantes ou d'autres raisons.
86          if prediction:
87              prediction_vide= True
88          #On renvoie la réponse à la requête HTTP. Lorsque l'utilisateur soumet un formulaire avec des données de température et d'humidité,
89          #l'application exécute le système expert pour prédire le climat, puis elle doit afficher le résultat de la prédiction sur une
90          #nouvelle page web. La fonction render_template prend en charge la création de la page HTML à partir d'un modèle
91          #(index.html dans ce cas) et remplace les variables de modèle (comme prediction et prediction_vide) par les valeurs fournies.
92          #Ces variables sont ensuite accessibles dans le template HTML pour affichage.
93          #En résumé, return render_template('index.html', prediction=prediction, prediction_vide=prediction_vide) permet à l'application de
94          #renvoyer une réponse HTML contenant le résultat de la prédiction météorologique pour que l'utilisateur puisse le voir sur son
95          #navigateur.
96          return render_template('index.html', prediction=prediction, prediction_vide=prediction_vide)
97      #Cette instruction est utilisée pour renvoyer la page d'accueil (index.html) lorsqu'aucune prédiction météorologique n'a été effectuée.
98      return render_template('index.html', prediction=None,prediction_vide=prediction_vide)
99  #On démarre l'application Flask en mode débogage si le script est exécuté directement.
100 if __name__ == '__main__':
101     app.run(debug=True)
```

Contenu du fichier index.html de templates

```
<!doctype html>
<html lang="en">
  <head>
    <title>Prédiction de Climat</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <link href='https://fonts.googleapis.com/css?family=Roboto:400,100,300,700' rel='stylesheet' type='text/css'>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css">
    <!-- Pour intégrer css avec Flask, il faut utiliser la méthode url_for() -->
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='css/style.css') }}">
    </head>
    <body>
    <section class="ftco-section">
        <div class="container">
            <div class="row justify-content-center">
                <div class="col-md-6 text-center mb-5">
                    <h2 class="heading-section">Prédiction du Climat</h2>
                </div>
            </div>
            <div class="row justify-content-center">
                <div class="col-lg-10 col-md-12">
                    <div class="wrapper">
                        <div class="row no-gutters">
                            <div class="col-md-7 d-flex align-items-stretch">
                                <div class="contact-wrap w-100 p-md-5 p-4">
                                    <h3 class="mb-4">Saisissez vos données...</h3>
                                    <div id="form-message-warning" class="mb-4"></div>
                                    <form method="POST" id="contactForm" name="contactForm" action="/">
                                        <div class="row">
                                            <div class="col-md-12">
                                                <div class="form-group">
                                                    <input type="text" class="form-control" name="temperature" id="subject" placeholder="
                                                    Valeur de la Temperature (ex. chaude ou douce ou bonne)">
                                                </div>
                                            </div>
```

Contenu du fichier index.html de templates (Cont.)

```
35                                  <div class="col-md-12">
36                                      <div class="form-group">
37                                          <input type="text" class="form-control" name="humidite" id="subject" placeholder="Valeur
                                            de l'humidité (ex. élevée ou normale)">
38                                      </div>
39                                  </div>
40                                  <div class="col-md-12">
41                                      <div class="form-group">
42                                          <input type="submit" value="Prédire le climat" class="btn btn-primary">
43                                      </div>
44                                  </div>
45                              </div>
46                          </form>
47                      </div>
48                  </div>
49                  <!-- Si prediction qui vient de flask contient quelque chose, on affiche ce code -->
50                  {% if prediction %}
51                      <div class="col-md-5 d-flex align-items-stretch">
52                          <div class="info-wrap bg-primary w-100 p-lg-5 p-4">
53                              <h3 class="mb-4 mt-md-4">Résultats de prédiction</h3>
54                              <div class="dbox w-100 d-flex align-items-start">
55                                  <div class="">
56                                      <span class=""></span>
57                                  </div>
58                                  <div class="text pl-3">
59                                      <p style="color: yellow">
60                                      <!-- On affiche les résultats de prédiction se trouvant dans la variable prediction -->
61                                      <b>{{ prediction }}</b>
62                  {% else %}
63                  <!-- Au cas où la variable prediction_vide est True -->
64                      {% if prediction_vide %}
65                           <p style="color: yellow">
66                              <b>Aucune prédiction disponible pour les données fournies.</b>
67                          </p>
```

Contenu du fichier index.html de templates (Cont.)

```
68                                  <!-- Fin de la condition qui vérifie si la variable prediction_vide est None -->
69                                  {% endif %}
70                                  </p>
71                                </div>
72                              </div>
73
74                            </div>
75                                  </div>
76                                  <!-- Fin de la condition qui vérifie si la variable prediction contient quelque chose -->
77                                  {% endif %}
78                                </div>
79                              </div>
80                            </div>
81                          </div>
82                        </div>
83                    </section>
84                    <!-- Pour intégrer js avec Flask, il faut utiliser la méthode url_for() -->
85                    <script src="src="{{ url_for('static', filename='js/jquery.min.js') }}></script>
86                    <script src="src="{{ url_for('static', filename='js/popper.js') }}></script>
87                <script src="src="{{ url_for('static', filename='js/bootstrap.min.js') }}></script>
88                <script src="src="{{ url_for('static', filename='js/jquery.validate.min.js') }}></script>
89                <script src="src="{{ url_for('static', filename='js/main.js') }}></script>
90                    </body>
91            </html>
```

TD 3: Diagnostic grippe/Chaînage avant

- Dans cet exemple, le système expert définit des règles pour diagnostiquer la grippe

- Le système expert s'exécute ensuite et détermine le diagnostic sur la base du processus d'inférence par chaînage avant.

- Cet exercice ne prévoit pas d'afficher un message spécifique lorsque l'utilisateur répond 'non' à la question sur les symptômes.
- On va exécuter ce code dans le cmd de anaconda en faisant:

  python diagnostic.py

TD 3: Diagnostic grippe/Chaînage avant (Cont.)

```python
#Ce code lorsque l'utilisateur ne présente qucun signe, le système ne fait rien
from experta import *
#On définit deux classes de faits : Symptome et Diagnostic.
class Symptome(Fact):
    pass
class Diagnostic(Fact):
    pass
# Définition des règles
class ExpertGrippe(KnowledgeEngine):
    @DefFacts()
    #On initialise les faits avec un diagnostic inconnu, et deux symptômes (fièvre et douleurs_corporelles) avec des valeurs None.
    def initial_facts(self):
        #Le mot-clé yield est utilisé dans Python pour créer un générateur, une fonction spéciale qui peut être interrompue et reprise
        #ultérieurement. Dans le contexte de notre code, yield est utilisé dans la méthode initial_facts de la classe ExpertGrippe
        #pour définir une fonction génératrice qui renvoie successivement des faits. Voici ce que fait chaque yield dans votre code :
        #1. yield Diagnostic('inconnu'): Renvoie un fait de type Diagnostic avec la valeur 'inconnu'.
            #Ce fait sera le premier fait initialisé dans le système expert.
        #2. yield Symptome(name='fièvre', value=None): Renvoie un fait de type Symptome avec le nom 'fièvre' et une valeur initiale None.
            #Ce fait représente le symptôme de fièvre avec une valeur non définie.
        #3. yield Symptome(name='douleurs_corporelles', value=None): Renvoie un fait de type Symptome avec le nom 'douleurs_corporelles'
        #et une valeur initiale None. Ce fait représente le symptôme de douleurs corporelles avec une valeur non définie.
        yield Diagnostic('inconnu')
        yield Symptome(name='fièvre', value=None)
        yield Symptome(name='douleurs_corporelles', value=None)
    @Rule()
    #On demande à l'utilisateur s'il a de la fièvre et des douleurs corporelles, puis on déclare ces symptômes avec les valeurs
    #correspondantes (True si l'utilisateur répond 'oui', False sinon).
    def debut_diagnostic(self):
        self.declare(Symptome(name='fièvre', value=input("Avez-vous la fièvre? ") == 'oui'))
        self.declare(Symptome(name='douleurs_corporelles', value=input("Avez-vous des douleurs corporelles? ") == 'oui'))
    @Rule(AND(Symptome(name='fièvre', value=True),
              Symptome(name='douleurs_corporelles', value=True)))
    def regle_grippe(self):
        self.declare(Diagnostic('grippe'))
```

TD 3: Diagnostic grippe/Chaînage avant (Cont.)

```
35      @Rule(Diagnostic('grippe'))
36      def affiche_grippe(self):
37          #on affiche un message si le diagnostic est la grippe.
38          print("Vous avez la grippe.")
39      @Rule(OR(Symptome(name='fièvre', value=False),
40               Symptome(name='douleurs_corporelles', value=False)))
41      #On déclare un diagnostic inconnu si au moins un des symptômes est False.
42      def regle_inconnue(self):
43          self.declare(Diagnostic('inconnu'))
44      @Rule(Diagnostic('inconnu'))
45      def affiche_maladie_inconnue(self):
46          print("Maladie inconnue")
47  # Création de l'instance de l'expert
48  expert_system = ExpertGrippe()
49  # Reinitialisation des faits
50  expert_system.reset()
51  # Lancement du système
52  expert_system.run()
```

TD 3.1: Diagnostic grippe/Chaînage avant (Cont.)

- On modifie ce code pour qu'il prédise aussi au cas où l'utilisateur saisit non.
- On a ajouté ce code:

```
25      #On ajoute ce code pour modifier le code précedent
26      @Rule(AND(Symptome(name='fièvre', value=False),
27                Symptome(name='douleurs_corporelles', value=False)))
28      def regle_pas_grippe(self):
29          print("Vous n'avez pas de grippe.")
30      #Fin du code de modification
```

- Dans le slide qui suit, on y a mis le code complet modifié

TD 3.1: Code complet modifié

```python
from experta import *
# Définition des faits
class Symptome(Fact):
    pass
class Diagnostic(Fact):
    pass
# Définition des règles
class ExpertGrippe(KnowledgeEngine):
    @DefFacts()
    def initial_facts(self):
        yield Diagnostic('inconnu')
        yield Symptome(name='fièvre', value=None)
        yield Symptome(name='douleurs_corporelles', value=None)
    @Rule()
    def debut_diagnostic(self):
        self.declare(Symptome(name='fièvre', value=input("Avez-vous la fièvre? ") == 'oui'))
        self.declare(Symptome(name='douleurs_corporelles', value=input("Avez-vous des douleurs corporelles? ") == 'oui'))
    @Rule(AND(Symptome(name='fièvre', value=True),
              Symptome(name='douleurs_corporelles', value=True)))
    def regle_grippe(self):
        self.declare(Diagnostic('grippe'))
    @Rule(Diagnostic('grippe'))
    def affiche_grippe(self):
        print("Vous avez la grippe.")
    #On ajoute ce code pour modifier le code précedent
    @Rule(AND(Symptome(name='fièvre', value=False),
              Symptome(name='douleurs_corporelles', value=False)))
    def regle_pas_grippe(self):
        print("Vous n'avez pas de grippe.")
    #Fin du code de modification
    @Rule(OR(Symptome(name='fièvre', value=False),
             Symptome(name='douleurs_corporelles', value=False)))
    def regle_inconnue(self):
        self.declare(Diagnostic('inconnu'))
    @Rule(Diagnostic('inconnu'))
    def affiche_maladie_inconnue(self):
        print("Maladie inconnue")
# Création de l'instance de l'expert
expert_system = ExpertGrippe()
# Reinitialisation des faits
expert_system.reset()
# Lancement du système
expert_system.run()
```

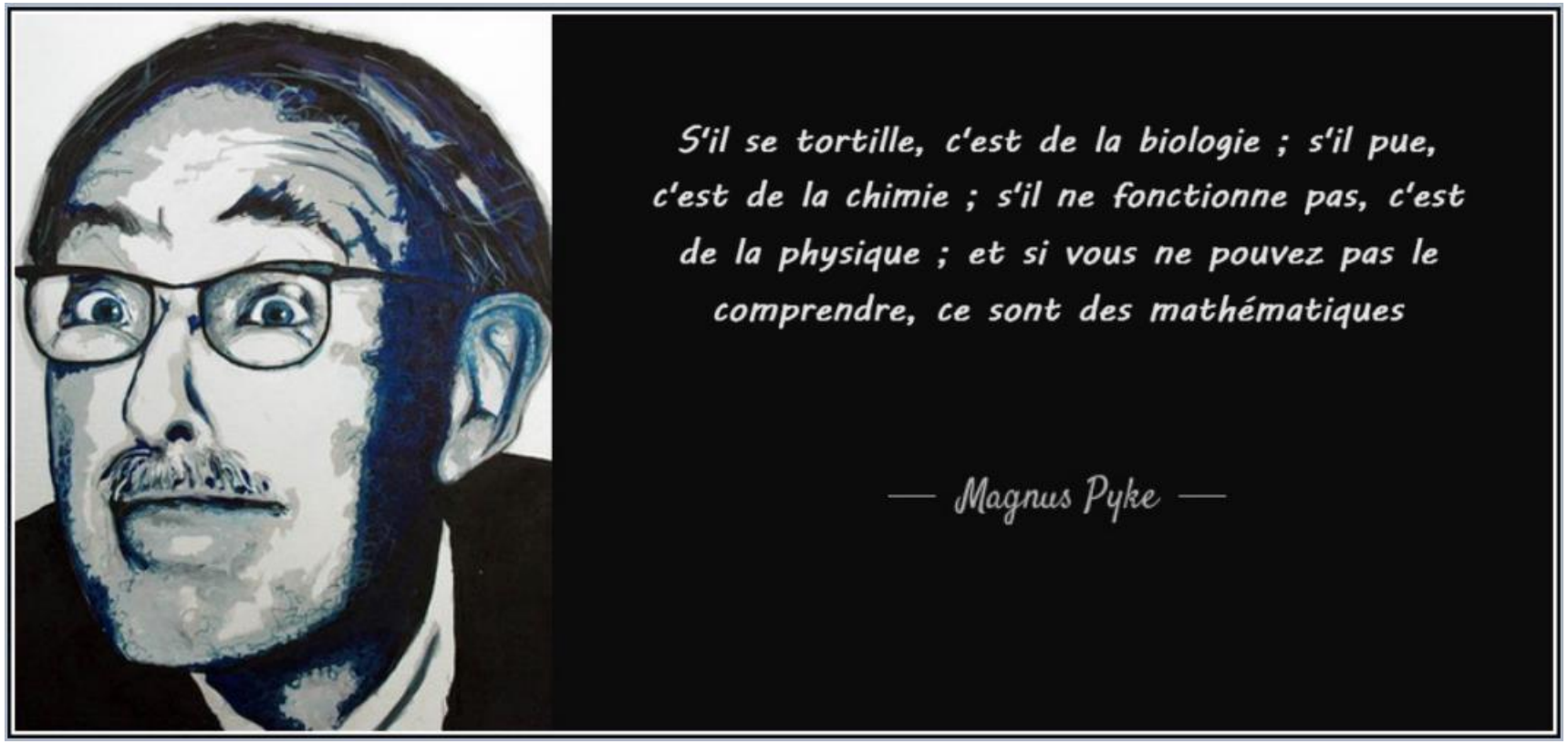
TD 4: Code précédent écrit avec le chaînage arrière

- Comme chaînage arrière, on commence d'abord par récupérer le diagnostic à faire (grippe) puis on vérifie les faits correspondants à cette maladie.

```
from experta import *
# Définition des faits
class Symptome(Fact):
    pass
class Diagnostic(Fact):
    pass
# Définition des règles
class ExpertGrippe(KnowledgeEngine):
    @DefFacts()
    def initial_facts(self):
        yield Diagnostic(input("Saisir le nom de la maladie à diagnostiquer (ex. grippe): "))
    @Rule(Diagnostic('grippe'))
    def demande_symptomes(self):
        if input("Avez-vous la fièvre? ") == 'oui':
            self.declare(Symptome(name='fièvre', value=True))
        else:
            self.declare(Symptome(name='fièvre', value=False))
        if input("Avez-vous les douleurs corporelles? ") == 'oui':
            self.declare(Symptome(name='douleurs_corporelles', value=True))
        else:
            self.declare(Symptome(name='douleurs_corporelles', value=False))
    @Rule(Symptome(name='fièvre', value=True),
          Symptome(name='douleurs_corporelles', value=True))
    def regle_grippe(self):
        print("Vous avez la grippe.")
    @Rule(Symptome(name='fièvre', value=False),
          Symptome(name='douleurs_corporelles', value=False))
    def regle_pas_grippe(self):
        print("Vous n'avez pas de grippe.")
    @Rule(OR(Symptome(name='fièvre', value=False),
             Symptome(name='douleurs_corporelles', value=False)))
    def regle_inconnu(self):
        print("Maladie inconnue.")
# Création de l'instance de l'expert
expert_system = ExpertGrippe()
# Reinitialisation des faits
expert_system.reset()
# Lancement du système
expert_system.run()
```

Synthèse de la leçon

- Rôles dans le développement des systèmes experts
- Acquisition et ingénierie des connaissances
- Introduction à Experta
  - Exemple météorologique
  - Exemple de diagnostic de maladie


S'il se tortille, c'est de la biologie ; s'il pue, c'est de la chimie ; s'il ne fonctionne pas, c'est de la physique ; et si vous ne pouvez pas le comprendre, ce sont des mathématiques
— Magnus Pyke —

Travail pratique 3: Rappel

- **Rappel**
  - Présentation de mi-parcours
  - Travail pratique 3

- **Objectifs :**
  - Représentation des connaissances, systèmes experts et Experta
  - Pratiquer la représentation des cadres
  - Apprendre à installer et à utiliser Experta
  - Pratiquer la représentation des règles
  - Construire un système expert très simple pour le diagnostic
  - Apprendre à programmer des questions
  - Comprendre les chaînages avant et arrière dans les systèmes experts basés sur des règles