ELEMENTS DE PLANIFICATION

Document émis le 27 septembre 1985.

PAR O. CROISSANT

FORWARD CHAINING, BACKWARD CHAINING, PLANNING, EVENT-DRIVEN, GOAL-DIRECTED ET BACKTRACKING

INTRODUCTION

D'habitude, les gens opposent OPS 5 à EMYCIN en disant que l'un est forward alors que l'autre est backward mais que par expérience, OPS 5 est plus puissant, plus universel que EMYCIN et donc plus adapté à des problèmes compliqués comme le planning qui n'est jamais très bien comparé aux autres problèmes.

Ce papier tente de préciser les choses.

COMPARAISON OPS-S.1

Une caractérisation possible de OPS 5 est :

Il s'agit d'un unificateur (logique du premier ordre) agissant sur une pile d'événements, chaque conclusion créant un événement nouveau : une instance d'un objet générique, la recherche de l'unificateur ayant lieu en remontant le temps.

Une caractérisation de la partie EMYCIN de S.1 (préférée pour l'étude car plus homogène et achevée que le EMYCIN original) est un ensemble de règles de détermination d'attributs se déclenchant en backchaining et utilisant l'unidétermination des attributs pour son contrôle : la détermination des attributs a donc lieu sur un ensemble de variables fixes.

Donc du point de vue contrôle, lorsque S.1 se pose la question de savoir si un attribut est déterminé ou pas, cette question est fondamentale. Elle suppose en effet que si l'attribut est déjà déterminé, S.1 ne créera pas un nouvel attribut avec le même nom pour avoir le plaisir de le redéterminer, car ce processus pourrait ne pas avoir de fin. Par conséquent, S.1 n'a pas les moyens de s'assurer que la détermination est encore valable, et la solution minimaliste la plus rigide a été adoptée : un attribut est déterminé une bonne fois pour toutes.

Au contraire, dans OPS 5, une instance de l'objet est recréée à chaque conclusion et l'accès à une valeur est simplement liée à la chronologie de sa détermination. Donc la différence fondamentale est cette création de variables dynamiques pour chaque règle appliquée. Donc si on rajoute le mode de déclenchement, on obtient la comparaison suivante :

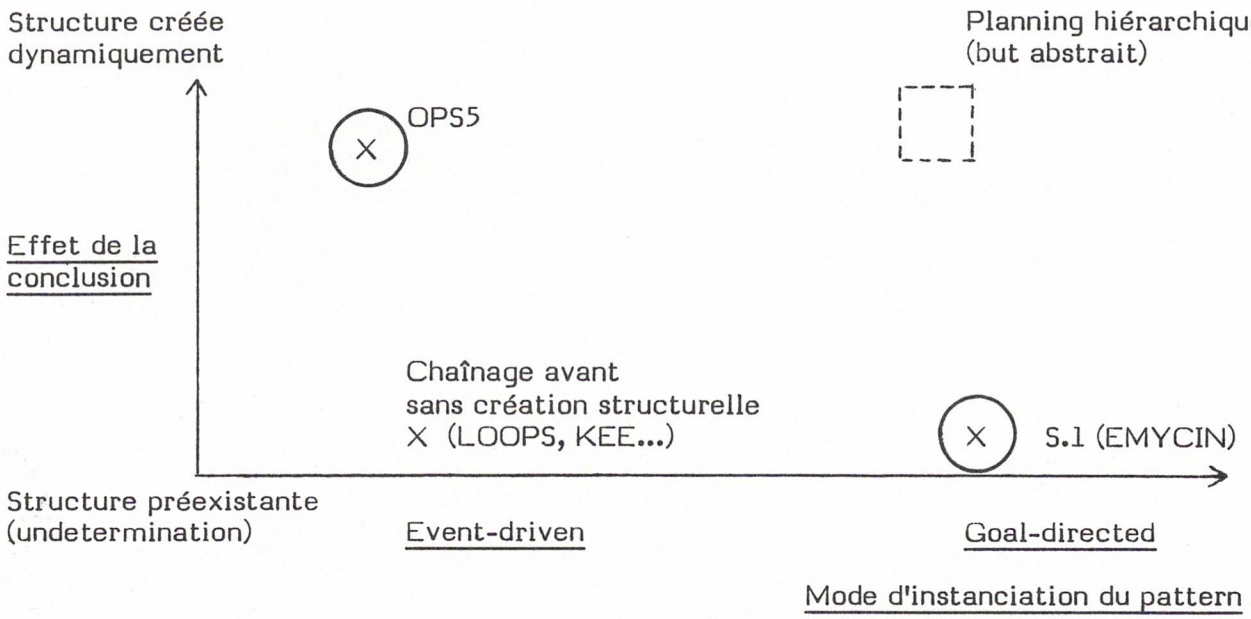


Figure 1

On voit donc émerger l'idée d'un outil goal-directed mais avec création structurelle : comment est-ce possible ?

Le manque dont on discutait au sujet de l'absence de moyens pour S.1 de savoir si la détermination était encore valable est en fait un manque de connaissances donc un espace de liberté dans lequel on peut créer l'objet adéquat. Cet objet doit donc être capable de vérifier si la détermination est correcte. Cet objet a un nom : c'est une contrainte. C'est ici que l'on commence à voir le rapport avec ce qu'il y a marqué dans le coin en haut à droite de la figure 1.

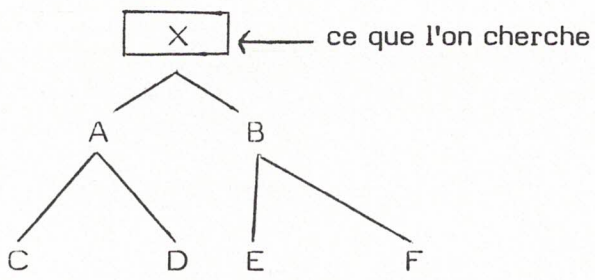
PLANNING = BACKCHAINING + BACKTRACKING

Qu'est-ce qu'un problème dévolué classiquement au backchaining : c'est un problème de sélection (structurée ou pas...) ou de calcul, en utilisant un "réservoir à formules". Le but est donc de déterminer un attribut X ; cela implique finalement l'utilisation de valeurs saisies par d'autres moyens que les règles : ce sont les feuilles :

Les règles: X ← A,B

A ← C,D

B ← E,F



Qu'est-ce qu'un problème de planning hiérarchique ? (Certains problèmes de planning sont plus compliqués, le mot "hiérarchique" se réfère donc à la structure que nous allons exposer.)

Il s'agit d'obtenir une liste de tâches élémentaires connaissant le but recherché. Le mot "hiérarchique" insiste sur le fait que l'on a pour cela des connaissances du type :

- [dans un certain contexte,] tâche X peut se décomposer en:

  tâche Y_1 ... tâche Y_n

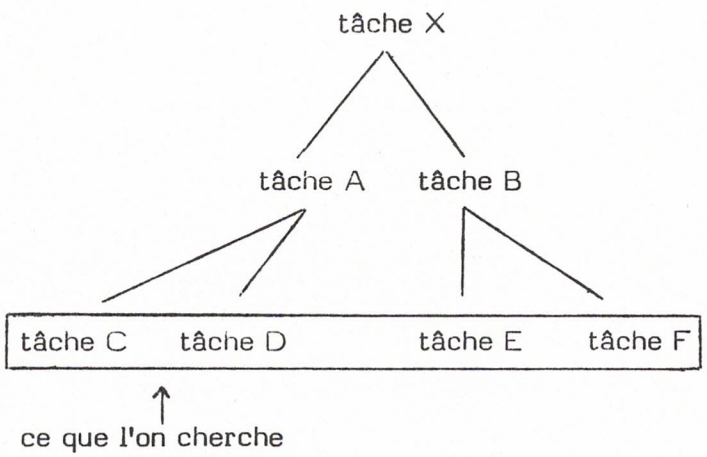
Ces connaissances s'enchaînent de la même manière que pour la détermination backchainante : l'appel par le nom de la tâche. Mais cette fois-ci, ce qui nous intéresse n'est pas le noeud initial mais les feuilles finales :

Les règles:

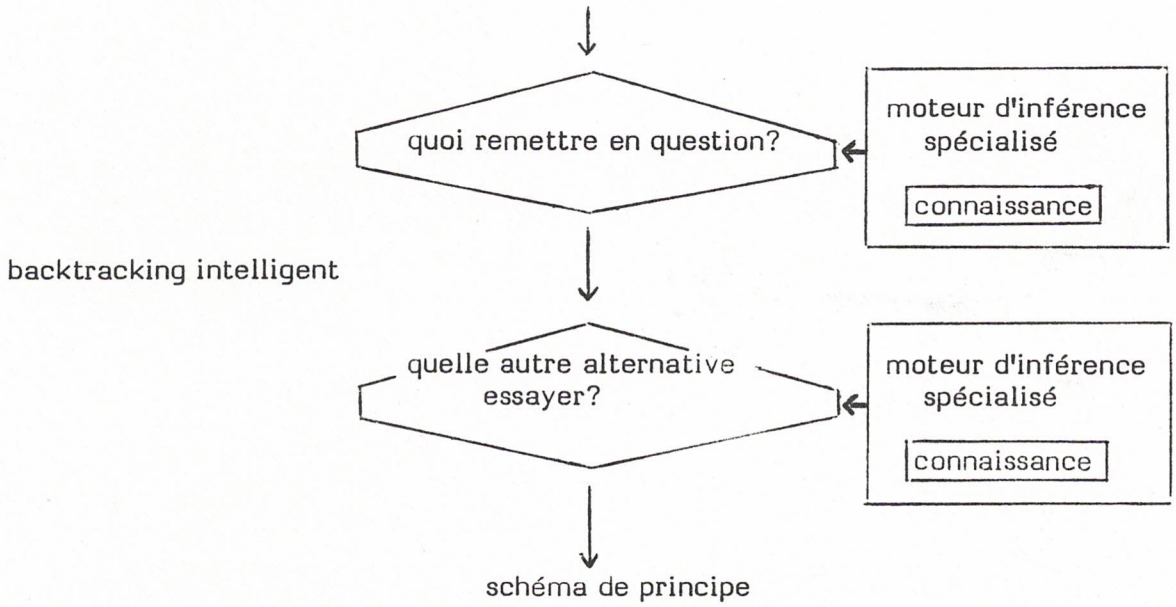
tâche X ⟵ tâche A,tâche B

tâche A ⟵ tâche C,tâche D

tâche B ⟵ tâche E,tâche F



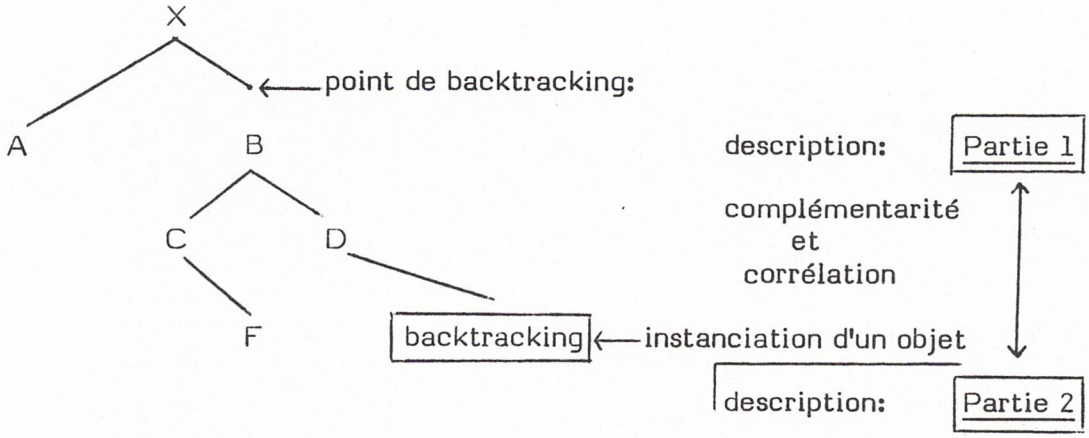
En fait, si les choses s'arrêtaient là, il n'y aurait pas de problème et le planning se résoudrait en S.1. La réalité est que toute la difficulté du planning réside dans le fait que la solution finale doit satisfaire un certain nombre de contraintes et donc que les règles de décomposition de la tâche X doivent pouvoir être multiples afin, non seulement de s'adapter au contexte, mais aussi et surtout de permettre l'étude d'alternatives lorsqu'une contrainte n'aura pas été satisfaite. Or le backchaining, pour être efficace, doit être dirigé par de la connaissance, c'est-à-dire qu'il faut préciser au moment opportun ce qu'il convient de remettre en question et quelle est l'alternative suivante, donc :



L'inférence spécialisée sera cachée dans la syntaxe du langage de la base de connaissances, la connaissance étant exactement ce qui est exprimé en ce langage, le backtracking est déclenché à un point précis dans le temps : c'est le lieu d'expression de la contrainte. Il apparaît donc naturel de situer une partie de la connaissance nécessaire dans un objet appelé contrainte :

Contrainte	nom :	contrainte X
	pattern :	une expression booléene
	backtracking :	en cas de non-succès, la description de QUOI remettre en question et COMMENT

En fait, pour faciliter la description, il peut être astucieux d'intercaler des points de backtracking dans l'arbre qui va du noeud initial aux feuilles. Lors de ces points, on peut trouver la description de la connaissance complémentaire donc :



Un exemple d'une telle implémentation :

point de backtracking = statement dans un bloc procédural :

Hypothèse.for.backtracking = { H1 : Panne = block 1
H2 : Panne = block 2
H3 : Panne = block 3

Backtracking.point.name = panne.intuition }

A partir de l'exécution d'un tel statement, toutes les valeurs d'attribut calculées ou conclues, le seront avec l'index H1, H2 ou H3, ce qui permettra de savoir quels attributs remettre en question. Le contrôle passant pour la première fois à ce satement, c'est l'hypothèse H1 qui est prise en considération ; si c'est une autre fois, c'est l'hypothèse qui n'a pas encore été essayée qui est mise à l'ordre du jour. Certaines règles déclenchant forward ou backward peuvent alors, au moment opportun, déclencher le backtracking :

REGLE 913

if coherence [panne, test] is not confirmed

then backtracks.to.the.point { panne.intuition }

La création d'un outil adapté au planning hiérarchique présuppose donc un outil backchainant et backtrackant et capable de créer des structures dynamiques lors du backtracking

(post.backtracking.blocks associé à un point de backtracking).

Ceci permet de coder le raisonnement suivant (donné à titre d'exemple) :

- domaine : celui de MYCIN
- raisonnement: on suppose d'abord qu'il n'y a qu'une seule pathologie que l'on tente de déterminer. Puis, si cela n'est pas cohérent, on décide qu'il y en a a peut-être deux et on répartit alors les informations déjà recueillies sur ces deux maladies et on continue le diagnostic.
- [pathologie: comprendre une instance de l'objet pathologie, ceci implique donc une création dynamique lors du backtracking].

REPRESENTATION DE LA CONNAISSANCE

La dernière remarque concerne la représentation de la connaissance.

Très souvent, la nature des objets à déterminer est un type de tâche à effectuer ensuite ou à décomposer. C'est toute la différence entre manipuler le symbole et utiliser sa signification (axe syntaxe - sémantique).

Dans les tâches de planning, cette distinction est plus importante que dans les autres types de problèmes. Il convient donc d'offrir le maximum de souplesse sur ce point particulier. L'autonomie symbole-tâche fait donc ressortir la nécessité de réaliser une association valeur-bloc actif qui soit l'objet de nombreuses utilisations dans le codage du savoir-faire de planning.

Un exemple d'une telle représentation est :

- un certain type d'attribut sera appelé attribut de type tâche. Ces attributs se verront adjoindre une notion de supertype abstrait qui dépend de la valeur de l'attribut et qui est naturellement associé à un bloc procédural :

exemple :

attribut :	tâche.à.accomplir
type :	task
supertype :	task 1, task 2, task 3
legal values :	tâche 12 : task 1, tâche 13 : task 1, tâche 24 : task 2

taskblocks :	tâche spécialisée 1
supertype :	task 1
body :	< procedure >

L'exécution des taskblocks est alors activée au cours de statement de blocs procéduraux,

exemple : execute tâche.à.accomplir [M]

ou implicitement dès qu'il a été déclaré dans le slot

task.to.execute.immediately de l'attribut de type tâche.

Une des deux spécifications : - All
- {liste de supertype executable immédiatement}

Supposons donc un outil disposant du backchaining et de la dualité attribut/tâche décrite précédemment, le schéma du raisonnement suivi par l'outil est donc à replacer sur un plan à deux dimensions : la dimension symbolique (détermination d'attributs) et la dimension sémantique (taskblocks).