Chapitre 5

Raisonnement en logique classique

On entend par logique classique, la logique propositionnelle ou la logique des prédicats, ces logiques ainsi que les notations qui s'y réfèrent sont pésentées, dans l'annexe A et dans l'annexe B, respectivement. La logique classique est traditionellement utilisée pour représenter les connaisances, mais également pour formaliser des raisonnement sur ces connaissances. Parmi les différents de types de raisonnement, nous nous intéressons plus particulièrement au raisonnement déductif. Nous présentons dans ce chapitre quatre approches de déduction automatique pour la logique classique, la résolution, les méthodes énumératives, les arbres de décision binaires et la programmation logique.

5.1. Rappels

Nous rappelons tout d'abord quelques définitions :

DÉFINITION 5.1.– *Soit F une formule d'un langage logique classique, noté $\mathcal{L}$, et I une interprétation, F est valide ou est une tautologie ssi pour toute interprétation I, $I(F) = 1$.*

Nous rappelons la définition d'une formule incohérente :

DÉFINITION 5.2.– *Soit F une formule d'un langage logique classique, noté $\mathcal{L}$, et I une interprétation, F est insatisfaisable ou incohérent ssi il n'existe pas d'interprétation I telle que, $I(F) = 1$.*

Cette définition s'étend à un ensemble de formules comme suit :

Chapitre rédigé par Odile PAPINI.

DÉFINITION 5.3.– *Soit $\mathcal{F}$ un ensemble de formules*[1] *de $\mathcal{L}$, et I une interprétation, $\mathcal{F}$ est insatisfaisable ou incohérent ssi il n'existe pas d'interprétation I telle que pour toute formule F, $F \in \mathcal{F}$, $I(F) = 1$.*

Les méthodes de raisonnement présentées sont des méthodes de réfutation, qui au lieu de prouver qu'une formule est valide (ou est une tautologie), prouvent que la négation d'une formule est incohérente et le raisonnement déductif s'appuie sur le résultat suivant :

THÉORÈME 5.1.– *Soit $F \in \mathcal{L}$, $\mathcal{F} \models F$ ssi $\mathcal{F} \cup \{\neg F\}$ est insatisfaisable ou incohérent.*

Cette proposition nous autorise à ramener le problème de de la conséquence logique à celui de la cohérence (ou de la satisfaisabilité). Nous allons examiner différentes méthodes, dans le cas de la logique propositionnelle, puis dans celui de la logique des prédicats [KOW 79], [DEL 86], [THA 90].

5.2. Résolution

5.2.1. *Résolution en logique propositionnelle*

La méthode de résolution [ROB 65] repose sur l'idée suivante. Il s'agit de tester l'incohérence d'un ensemble de clauses, si cet ensemble contient la clause vide[2] alors il est incohérent, s'il ne contient pas la clause vide il s'agit de tester si la clause vide peut être une conséquence logique de cet ensemble de clauses.

Informellement, soit A, B et X des formules propositionnelles, on suppose qu'il existe une interprétation qui satisfait $(A \vee X)$ et $(B \vee \neg X)$, si cette interprétation satisfait X alors on déduit qu'elle satisfait B, si cette interprétation ne satisfait pas X alors on déduit qu'elle satisfait A, dans les deux cas cette interprétation satisfait $A \vee B$, ce qui s'écrit plus formellement :

$$\{A \vee X,\ B \vee \neg X\} \models A \vee B.$$

Dans le cas particulier où A et B sont des clauses et X une proposition le résultat suivant donne le principe de résolution :

PROPOSITION 5.1.– *Soit S un ensemble de clauses, $c_1, c_2 \in S$ et soit l un littéral. Si l apparaît dans c_1 et $\neg l$ apparaît dans c_2, $c_1 = l_{i_1} \vee l_{i_2} \vee \cdots \vee l_{i_n} \vee l$ et $c_2 = l_{j_1} \vee l_{j_2} \vee \cdots \vee l_{j_m} \vee \neg l$ alors la clause $r = l_{i_1} \vee l_{i_2} \vee \cdots \vee l_{i_n} \vee l_{j_1} \vee l_{j_2} \vee \cdots \vee l_{j_m}$ est une conséquence logique de S, on l'appelle la résolvante.*

En conséquence, cette proposition fournit une règle d'inférence, appelé règle de résolution, qui, à partir d'un ensemble de clauses S, permet de produire de nouvelles clauses et qui préserve la cohérence.

---

1. On utilise souvent une conjonction de formules pour représenter un ensemble de formules.

2. disjonction vide, qui ne contient pas de littéraux.

Pour montrer qu'une formule ou un ensemble de formules propositionnelles est incohérent, on utilise la mise sous forme conjonctive normale qui, selon le théorème de normalisation, produit un ensemble de clauses équivalent. On applique ensuite la règle d'inférence jusqu'à ce que soit la clause vide soit produite, soit la règle d'inférence ne puisse plus s'appliquer. On note $\square$ la clause vide, et l'algorithme de résolution qui, à partir d'un ensemble initial de clauses S, produit en sortie un ensemble de clauses S', est le suivant :

algorithme résolution(S : un ensemble de clauses) : (possible, S')
données : S', $possible$,
début
$S' \leftarrow S$
$possible \leftarrow vrai$
tant que $\square \notin S'$ et $possible$ **faire**
  si il est possible de produire une nouvelle résolvante **alors**
    choisir l, c_1, c_2 tels que $l \in c_1$ et $\neg l \in c_2$
    calculer la résolvante r
    $S' \leftarrow S' \cup \{r\}$
  sinon
    $possible \leftarrow faux$
  fin si
fin tant que
fin

Figure 5.1. *Algorithme de résolution.*

EXEMPLE 5.1.– Soit S l'ensemble de clauses $S = \{p \vee q,\ p \vee r,\ \neg q \vee \neg r,\ \neg p\}$, les clauses sont numérotées (1) $p \vee q$, (2) $p \vee r$, (3) $\neg q \vee \neg r$, (4) $\neg p$. L'application de la résolution sur S produit les résolvantes suivantes : (5) $p \vee \neg r$ à partir de (1) et (3). (6) q à partir de (1) et (4), (7) $p \vee \neg q$ à partir de (2) et (3), (8) r à partir de (2) et (4), (9) p à partir de (2) et (5), (10) $\neg r$ à partir de (3) et (6), (11) $\neg q$ à partir de (3) et (8), (12) $\neg r$ à partir de (4) et (5), (13) $\neg q$ à partir de (4) et (7) et (14) $\square$ à partir de (4) et (9). Donc l'ensemble de clauses S est incohérent. Dans cet exemple, on a appliqué la règle de résolution en utilisant la stratégie qui sélectionne les paires de clauses ordonnées selon l'ordre lexicographique. Cette stratégie n'est pas optimale car certaines résolvantes sont utilisées plusieurs fois. On aurait pu par exemple, dès qu'on obtient la clause (7), produire (8) p à partir de (1) et (7), puis (9) $\square$ à partir de (4) et (8).

L'exemple précédent souligne la nécessité d'heuristiques pour le choix des clauses à utiliser en priorité pour produire des résolvantes. Plusieurs stratégies ont été proposées voir [].

5.2.1.1. *Propriétés de la résolution :*
PROPOSITION 5.2.– *Soit S un ensemble de clauses, soit r une résolvante obtenue aprés l'application de la règle résolution sur S, soit $S' = S \cup \{r\}$. Si S est satisfaisable alors S' est satisfaisable.*

PROPOSITION 5.3.– *Soit S un ensemble de clauses, S n'est pas satisfaisable (ou incohérent) si et seulement si la résolution produit la clause vide, notée $\square$.*

DÉFINITION 5.4.– *Soit S un ensemble de clauses, une déduction de C à partir de S est une séquence finie $c_1, c_2, \cdots, c_k$ de clauses telle que tout c_i est soit une clause de S soit est une résolvante obtenues à partir des clauses précédant c_i et $c_k = c$. Une déduction de $\square$ à partir de S est appelée une preuve par résolution ou réfutation de S.*

PROPOSITION 5.4.– *Soit F une formule propositionnelle, si $\neg F$ a une preuve par résolution alors F est une tautologie.*

THÉORÈME 5.2.– *théorème de complétude* Soit F une formule propositionnelle, si F est une tautologie alors $\neg F$ a une preuve par résolution.

THÉORÈME 5.3.– *Soit F une formule propositionnelle et $\mathcal{F}$ un ensemble de formules propositionnelles, $\mathcal{F} \models F$ si et seulement si $\mathcal{F} \cup \{\neg F\}$ a une preuve par résolution.*

5.2.2. *Résolution en logique des prédicats*

Dans la logique des prédicats, le théorème de Herbrand permet de ramener le problème de la satisfaisabilité d'une formule ou d'un ensemble de formules au problème de la satisfaisabilité d'un ensemble de clauses de la logique propositionnelle. Pour cela, on associe à la forme conjonctive normale équivalente à la formule ou à l'ensemble de formules de la logique des prédicats, l'ensemble C des clauses correspondantes et on construit un ensemble de clauses, appelé système de Herbrand en instanciant les variables avec des éléments puisés dans l'univers de Herbrand, ce qu'on appelle saturation , pour plus détails voir l'annexe B. On applique ensuite la résolution. J. -A. Robinson a montré [ROB 79] que la résolution et la saturation sont des opérations semi-commutatives, cela signifie que l'on peut intervertir les opérations de saturation et de résolution. L'intérêt de commencer par la résolution avant l'instanciation dans l'univers de Herbrand présente l'avantage de réduire la complexité algorithmique, en effet la résolution opère sur des paires de clauses qui sont en nombre fini et ont un nombre fini de littéraux.

Afin de pouvoir permettre l'application du principe de résolution, et faire apparaître plus rapidement des contradictions, une phase préliminaire est nécessaire, il s'agit de l'unification dont le but est de mettre en coïncidence des atomes par un bon choix des termes substitués aux variables [ROB 79].

DÉFINITION 5.5.– *Une substitution est une application, notée σ, de l'ensemble des variables dans l'ensemble des termes qui est presque partout égale à l'identité (elle ne diffère de l'identité qu'en un nombre fini de variables).*

DÉFINITION 5.6.– *Une substitution de renommage (ou permutation) est une substitution qui est une bijection sur l'ensemble des variables.*

DÉFINITION 5.7.–

– Soit t un terme (resp. un litteral l), on appelle instance de t, le terme noté $\sigma(t)$ (resp. instance de l, le litteral noté $\sigma(l)$), obtenu en remplaçant toutes les occurences de la variable x par $\sigma(x)$.

– soit t_1 et t_2 des termes, t_2 est une instance de t_1 s'il existe une substitution σ telle que $t_2 = \sigma(t_1)$.

DÉFINITION 5.8.– *Deux termes t et t' (resp. deux littéraux l et l') sont unifiables s'il existe une substitution σ telle que $\sigma(t) = \sigma(t')$ (resp. $\sigma(l) = \sigma(l')$). La substitution est appelée l'unificateur.*

DÉFINITION 5.9.– *Un unificateur σ' est plus général que σ, s'il existe une substitution σ'' telle que $\sigma = \sigma'' \circ \sigma'$*

Parmi tous les unificateurs de deux termes t et t' (resp. deux litteraux l et l'), on peut montrer qu'il en existe un qui est plus général que tous les autres on l'appelle l'unificateur principal. Intuitivement c'est une substitution, la plus générale possible qui fait coincider les deux termes (resp. les deux littéraux).

L'unification n'est possible qu'entre une variable et un terme, si ce dernier ne contient pas la variable. L'algorithme d'unification de deux termes t_1 et t_2 qui retourne un couple ($possible$, σ) où $possible$ est une variable booléenne qui est vraie si les deux termes sont unifiables et σ est l' unificateur principal de t_1 et t_2, est donné par la figure 5.2. L'algorithme d'unification se termine toujours et retourne l'unificateur principal.

La résolution pour la logique des prédicats repose sur le résultat suivant :

THÉORÈME 5.4.– *Soit S un ensemble de clauses, c_1 et c_2 des clauses de S, $l_1 \in c_1$ et $\neg l_2 \in c_2$. Soit θ une substitution de renommage telle que $\theta(c_1)$ et c_2 n'aient aucune variable libre en commun et soit σ l'unificateur principal de $\theta(c_1)$ et c_2 et soit la clause $r = \sigma(\theta(c_1 \backslash \{l_1\}) \vee (c_2 \backslash \{\neg l_2\}))$ appelée résolvante alors S et $S \cup \{r\}$ sont logiquement équivalents.*

Ce théorème fournit une règle d'inférence, règle de résolution, pour la logique des prédicats, et l'algorithme de résolution dans le cas de la logique des prédicats qui, à partir d'un ensemble initial de clauses S, produit en sortie un ensemble de clauses S' se généralise comme suit :

EXEMPLE 5.2.– Soit l'ensemble de formules de la logique des prédicats : $\mathcal{F} = \{\forall x, \forall y, \forall z\ ((F(x,y) \wedge F(y,z)) \rightarrow G(x,z)),\ \forall x, \exists y F(y,x),\ \neg\forall x, \exists y G(y,x)\}$.

algorithme unification(t_1, t_2 : des termes) : ($possible$, σ')
données : n, m, i, x, t, σ', $possible$
début
si l'un des termes (t_1 ou t_2) est une variable x, (l'autre est t) **alors**
  si $x = t$ **alors**
    $possible \leftarrow vrai$
    $\sigma \leftarrow \emptyset$
  sinon
    si x apparaît dans t **alors**
      $possible \leftarrow faux$
    sinon
      $possible \leftarrow vrai$
      $\sigma \leftarrow (\sigma(x) = t)$
    fin si
  fin si
sinon
  ($t_1 = f(x_1, \cdots, x_n)$ et $t_2 = g(y_1, \cdots, y_m)$)
  si $f \neq g$ ou $n \neq m$ **alors**
    $possible \leftarrow faux$
  sinon
    $i \leftarrow 0$
    $possible \leftarrow vrai$
    $\sigma \leftarrow \emptyset$
    tant que $i < n$ et $possible$ **faire**
      $i \leftarrow i + 1$
      $(possible, \sigma') \leftarrow unification(\sigma(x_i), \sigma(y_i))$
      si $possible$ **alors**
        $\sigma \leftarrow \sigma' \circ \sigma$
      fin si
    fin tant que
  fin si
fin si
fin

Figure 5.2. *Algorithme d'unification.*

Cet ensemble de formules produit le système de Herbrand suivant : $\{\neg F(x, y) \vee \neg F(y, z) \vee G(x, z), F(f(x'), x'), \neg G(y', a)\}$.

Pour unifier la première et la troisième clause on utilise la substitution σ_1, telle que $\sigma_1(x) = y'$ et $\sigma_1(z) = a$, on a $\neg F(y', y) \vee \neg F(y, a) \vee G(y', a)$ et $\neg G(y', a)$ qui produisent la résolvante $\neg F(y', y) \vee \neg F(y, a)$.

algorithme résolution(S : un ensemble de clauses) : (possible, S')
données : S', $possible$,
début
$S' \leftarrow S$
$possible \leftarrow vrai$
tant que $\square \notin S'$ et $possible$ **faire**
  si il est possible de produire une nouvelle résolvante **alors**
    choisir l_1, l_2, c_1, c_2 tels que $l_1 \in c_1$, $l_2 \in c_2$ et l_1, l_2 unifiables
    calculer la résolvante r à partir de l'unificateur principal
    $S' \leftarrow S' \cup \{r\}$
  sinon
    $possible \leftarrow faux$
  fin si
fin tant que
fin

Figure 5.3. *Algorithme de résolution.*

Pour unifier cette résolvante et la deuxième clause on utilise la substitution σ_2 telle que $\sigma_2(y) = f(a)$ et $\sigma_2(x') = a$, on obtient $\neg F(y', f(a)) \vee \neg F(f(a), a)$ et $F(f(a), a)$ qui produisent la résolvante $\neg F(y', f(a)$ que l'on cherche à unifier à $F(f(x'), x')$.

Pour cela, on utilise la substitution σ_3 telle que $\sigma_3(x') = f(a)$ et $\sigma_3(y') = f(f(a))$ et on obtient $F(f(f(a)), f(a))$ et $\neg F(f(f(a)), f(a))$ qui produisent comme résolvante la clause vide $\square$. Donc l'ensemble de formules $\mathcal{F}$ est incohérent.

5.2.2.1. *Propriétés de la résolution :*
PROPOSITION 5.5.– *Soit S un ensemble de clauses de la logique des prédicats, S n'est pas satisfaisable (ou incohérente) si et seulement si la résolution produit la clause vide.*

THÉORÈME 5.5.– *théorème de complétude* Soit F une formule de la logique des prédicats, si F est une tautologie alors $\neg F$ a une preuve par résolution.

5.3. Méthodes énumératives

Les méthodes énumératives se situent au niveau sémantique, c'est à dire, au niveau de l'interprétation des formules (voir annexe 1). On rappelle qu'une interprétation, notée I, est une application de l'ensemble des variables propositionnelles, noté $\mathcal{P}$, dans $\{0, 1\}$ telle que $I(0) = 0$ et $I(1) = 1$, on appelle modèle d'une formule F, une interprétation I telle que $I(F) = 1$, et contre-modèle d'une formule F, une interprétation I telle que $I(F) = 0$.

Pour montrer qu'une formule ou un ensemble de formules propositionnelles est incohérent, on utilise la mise sous forme conjonctive normale qui, selon le théorème

de normalisation, produit un ensemble de clauses équivalent. Les méthodes énumératives consistent énumérer les interprétations du langage et à montrer qu'elles sont des contre-modèles de cet ensemble de clauses.

DÉFINITION 5.10.– *Une interprétation partielle est une application d'un sous-ensemble de variables propositionnelles noté $\mathcal{P}'$, $\mathcal{P}' \subset \mathcal{P}$, dans* $\{0, 1\}$.

L'idée de base est la construction d'un arbre binaire de recherche où chaque noeud représente une interprétation partielle des variables propositionnelles, apparaissant dans l'ensemble des clauses. Nous allons successivement examiner les méthodes des arbres sémantiques, de Quine et de Davis et Putnam dans le cadre de la logique propositionnelle.

5.3.1. *Arbres sémantiques*

La méthode des arbres sémantiques repose sur la construction d'un arbre binaire [CHA 73].

Pour chaque ensemble de clauses S, contenant l'ensemble des variables propositionnelles $\mathcal{V} = \{p_1, p_2, \cdots, p_n\}$ on construit un arbre correspondant à S comme suit :

– chaque arc est étiqueté par un littéral p_i ou $\neg p_i \in \mathcal{V}$. La branche p_i correspond à l'affectation p_i à Vrai et la branche $\neg p_i$ correspond à l'affectation p_i à Faux ;

– les littéraux étiquetant les arcs issus d'un même noeud sont opposés ;

– aucune branche ne comporte plus d'une occurrence de chaque atome.

Chaque noeud de l'arbre correspond à une interprétation partielle de S.

DÉFINITION 5.11.– *arbre complet* Soit $\mathcal{V}$ l'ensemble des variables propositionnelles apparaissant dans un ensemble de clauses S. L'arbre sémantique correspondant à S est dit complet ssi chaque feuille de l'arbre correspond à une interprétation de l'ensemble des variables propositionnelles $\mathcal{V}$, apparaissant dans S.

DÉFINITION 5.12.– *branche fermée* Une branche d'un arbre sémantique correspondant à S est dite fermée ssi il existe un neoud n tel que l'interprétation partielle correspondante à celui-ci est un contre-modèle d'une des clauses de S et tel que les interprétations partielles de tout noeud ancêtre de n ne sont pas des contre-modèles d'une des clauses de S.

DÉFINITION 5.13.– *arbre fermé* Un arbre sémantique correspondant à S est dit fermé ssi toutes les branches de cet arbre sont fermées.

THÉORÈME 5.6.– *version* 1 *du théorème de Herbrand* S un ensemble de clauses est incohérent ssi pour tout arbre sémantique complet correspondant à S, il existe une arbre fini fermé correspondant à S.

THÉORÈME 5.7.– *version* 2 *du théorème de Herbrand* S un ensemble de clauses est incohérent ssi il existe un sous-ensemble de clauses S', $S' \subset S$ incohérent.

Pour prouver qu'un ensemble de clauses S est cohérent il suffit de trouver une feuille qui produit un modèle de S, c'est-à-dire, une interprétation qui satisfait l'ensemble des clauses. Pour prouver l'incohérence d'un ensemble de clauses il faut vérifier que toutes les feuilles de l'arbre sémantique correspondant produisent des contre-modèles, c'est à dire, des interprétations qui falsifient cet ensemble de clauses. Comme l'arbre sémantique comporte 2^n feuilles cette méthode est assez inefficace.

EXEMPLE.– Soit S l'ensemble de clauses $S = \{p \vee q,\ p \vee r,\ \neg q \vee \neg r,\ \neg p\}$, l'ensemble des propositions est $\mathcal{V} = \{p, q, r\}$ et l'arbre sémantique correspondant à S est illustré dans la figure 5.4.

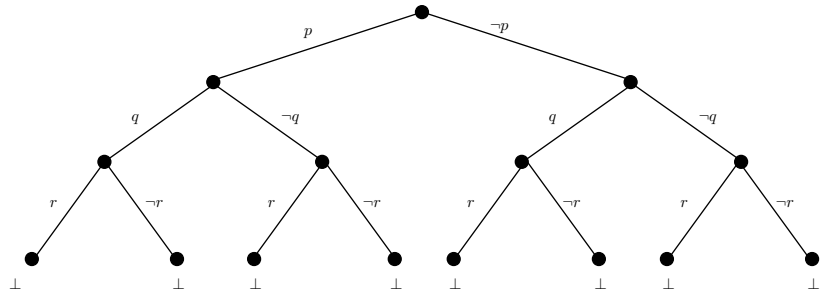

Figure 5.4. *Arbre sémantique correspondant à l'ensemble de clauses* $S = \{p \vee q, p \vee r, \neg q \vee \neg r, \neg p\}$.

Cette méthode s'étend naturellement à la logique des prédicats. Un ensemble de formules de la logique des prédicats est transformé en un système de Herbrand et la méthode des arbres sémantiques construit un arbre à partir des littéraux des clauses du système de Herbrand.

5.3.2. *Méthode de Quine*

L'algorithme de Quine [QUI 50] est une amélioration de la méthode des arbres sémantiques, elle repose sur le principe suivant. Si toutes les extensions possibles d'une interprétation partielle attribuent la même valeur de vérité à la formule testée il est inutile de construire le sous-arbre issu du noeud correspondant à cette interprétation partielle. Cette amélioration permet de ne pas construire la totalité de l'arbre sémantique.

Soit S une ensemble de clauses correspondant à la forme conjonctive normale d'un ensemble de formules propositionnelles. Soit p une proposition, S se partitionne en trois sous-ensembles de clauses, l'ensemble des clauses où p apparaît, noté S_p, l'ensemble des clauses où $\neg p$ apparaît, noté $S_{\neg p}$ et l'ensemble des clauses où ni p, ni $\neg p$ n'apparaît, noté $S" = S \backslash (S_p \cup S_{\neg p})$. On note S'_p l'ensemble des clauses de S_p privées de p et $S'_{\neg p}$ l'ensemble des clauses $S_{\neg p}$ privées de $\neg p$.

PROPOSITION 5.6.– *S est incohérent si et seulement si $S'_p \cup S$" et $S'_{\neg p} \cup S$" sont incohérents.*

L'algorithme de Quine est donné par la figure 5.5 :

algorithme quine(S : un ensemble de clauses) : (coherent)
données : S_p, $S_{\neg p}$, S", S'_p, $S'_{\neg p}$, $coherent$
début
si $S = \emptyset$ **alors**
    $coherent \leftarrow vrai$
sinon
    si $S = \{\Box\}$ **alors**
        $coherent \leftarrow faux$
    sinon
        choisir un atome $p \in S$
        calculer S_p, $S_{\neg p}$ et $S" = S \backslash (S_p \cup S_{\neg p})$
        calculer S'_p et $S'_{\neg p}$
        quine($S'_p \cup S$")
        quine($S'_{\neg p} \cup S$")
    fin si
fin si
fin

Figure 5.5. *Algorithme de Quine.*

La méthode s'étend naturellement à un ensemble de formules de la logique des prédicats transformé en un système de Herbrand.

5.3.3. *Méthode de Davis et Putnam*

La méthode de Davis et Putnam [DAV 60] est une amélioration de la méthode de Quine qui tire parti de la la forme conjonctive normale des formules propositionnelles, pour sélectionner au mieux les propositions afin d' éviter la construction de branches inutiles dans l'arbre. Une proposition p est sélectionnée en priorité si :

– S contient une clause monolittérale, c'est à dire une clause contenant seulement p, (respectivement $\neg p$). Dans ce cas, S'_p (respectivement $S'_{\neg p}$ contient la clause vide, donc $S'_p \cup S$" (respectivement $S'_{\neg p} \cup S$") est incohérent, la recherche d'incohérence se ramène seulement à celle de $S'_{\neg p} \cup S$" (respectivement $S'_p \cup S$").

– Si un seul des littéraux p ou $\neg p$ apparaît dans S. Dans ce cas, S_p ou $S_{\neg p}$ est vide, la recherche d'incohérence se ramène seulement à celle de $S'_{\neg p} \cup S$" ou à celle de $S'_p \cup S$".

EXEMPLE 5.3.– Soit S l'ensemble de clauses $S = \{p \vee q,\ p \vee r,\ \neg q \vee \neg r,\ \neg p\}$, comme $\neg p$ est une clause monolittérale on choisit en priorité la proposition p. $S_p = \{p \vee q,\ p \vee r\}$, $S_{\neg p} = \{\neg p\}$ et S" $= \{\neg q \vee \neg r\}$. On obtient $S'_p = \{q,\ r\}$ et $S'_{\neg p} = \{\square\}$, on est ramené à la recherche de l'incohérence de $S'_p \cup S$" $= \{\ q,\ r,\ \neg q \vee \neg r\}$ et de $S'_{\neg p} \cup S$" $= \{\square,\ \neg q \vee \neg r\}$. Or $S'_{\neg p} \cup S$" est incohérent car il contient la clause vide. On note $U = S'_p \cup S$", on choisit la proposition q. $U_q = \{q\}$, $U_{\neg q} = \{\neg q \vee \neg r\}$ et U" $= \{r\}$. On obtient $U'_q = \{\square\}$ et $U'_{\neg q} = \{\neg r\}$, on est ramené à la recherche de l'incohérence de $U'_q \cup U$" $= \{\square,\ r\}$ et de $U'_{\neg q} \cup U$" $= \{r, \neg r\}$. Or $U'_q \cup U$" est incohérent car il contient la clause vide et en choisissant r, on obtient que $U'_{\neg q} \cup U$" est également incohérent.

La méthode s'étend naturellement à un ensemble de formules de la logique des prédicats transformé en un système de Herbrand.

Des améliorations de la méthode de Davis et Putnam, connues sous le vocable *évaluation sémantique*, ont été proposées par Oxusoff et Rauzy [OXU 89]. Elles reposent sur des propriétés de coupure qui permettent d'élaguer l'arbre de décision binaire et sur des heuristiques définies pour guider le choix du prochain littéral à évaluer.

5.4. Diagrammes de décision binaires

Les méthodes énumératives construisent des arbres de décision binaires, dont la taille peut être importante. E. Bryant [BRY 86] propose une représentation plus compacte une formule propositionnelle par un graphe acyclique orienté, appelée diagramme de décision binaire ou plus simplement BDD.On se concentre sur une représentation compacte d'une formule de manière à rendre plus rapide les opérations qui sont généralement coûteuses comme le test de satisfaisabilité.

DÉFINITION 5.14.– *Un diagramme de décision binaire représentant une formule propositionnelle est un graphe acyclique orienté comprenant :*

– un seul sommet sans prédecesseur appelé source ;

– des sommets intermédiaires. Chaque sommet intermédiaire est étiqueté par une variable propositionnelle apparaissant dans la formule et possède deux successeurs, un fils gauche, , l'arc étiqueté par $else$ correspondant à l'affectation à 0 de la proposition et un fils droit, l'arc étiqueté par $then$ correspondant à l'affectation à 1 de la variable propositionnelle ;

– deux sommets sans successeur, appelés puits, étiquetés par les constantes 0 et 1 (correspondant respectivement aux valeurs de vérité, Faux et Vrai).

Le principe est identique à celui utilisé par les arbres de décision binaires, toute affectation des variables propositionnelles, correspond à un chemin dans le graphe. Un chemin de la source vers le puits étiqueté par 1 correspond à un modèle de la formule et un chemin de la source vers le puits 0 correspond à un contre-modèle de la formule.

Lorsqu'on dispose d'un ordre total sur les variables propositionnelles qui apparaissent dans la formule propositionnelle, on peut définir un diagramme de décision binaire ordonné, ou OBDD, qui est un arbre de décision binaire où tous les chemins de la source vers les sommets sans successeurs visitent les sommets en respectant l'ordre sur les variables propositionnelles.

Un diagramme de décision binaire ordonné et réduit ou ROBDD est un diagramme de décision binaire ordonné auquel on appliqué deux réductions visant à réduire le nombre de sommets. La première réduction supprime les sommets redondants, c'est à dire les sommets dont le sucesseur gauche est égal au successeur droit, la deuxième réduction fusionne les paires de sommets qui ont respectivement le même fils gauche et le même fils droit et on a le résultat suivant :

THÉORÈME 5.8.– *Pour toute formule propositionnelle il existe un diagramme de décision binaire ordonné et réduit unique (à un ordre des variable près) et tout autre diagramme de décision binaire contient un plus grand nombre d'arcs.*

EXEMPLE 5.4.– Soit F une formule propositionnelle $F = (p \wedge q) \vee r$, l'ensemble des variables propositionnelles est $\mathcal{V} = \{p, q, r\}$, on utilise l'ordre lexicographique sur les variables et le diagramme de décision binaire réduit et ordonné correspondant à F est illustré dans la figure 5.6.

p	q	r	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Le processus de réduction pour la construction du ROBDD est illustré par la figure cidessus. La première réduction permet de supprimer le sommet le plus à droite étiqueté par q ainsi que le sommet le plus à gauche étiqueté par r, le successeur gauche est égal au successeur droit. La deuxième réduction permet de fusionner les sommets les plus à droite étiquetés par r, ils ont même successeur gauche et même successeur droit.

Lorsqu'on dispose un ordre total sur les variables propositionnelles, on obtient un ROBDD unique et canonique, c'est à dire que deux formules propositionnelles équivalentes possèdent le même ROBDD. Les ROBDD satisfont la propriété suivante :

PROPOSITION 5.7.– *Soit un ROBDD représentant F, une formule propositionnelle, dont on dispose d'un ordre total sur les variables propositionnelles.*

– Tout chemin dans le ROBDD de la source vers le puits 1 *définit une interprétation partielle telle que toutes les interprétations qui la contiennent sont des modèles de F ;*

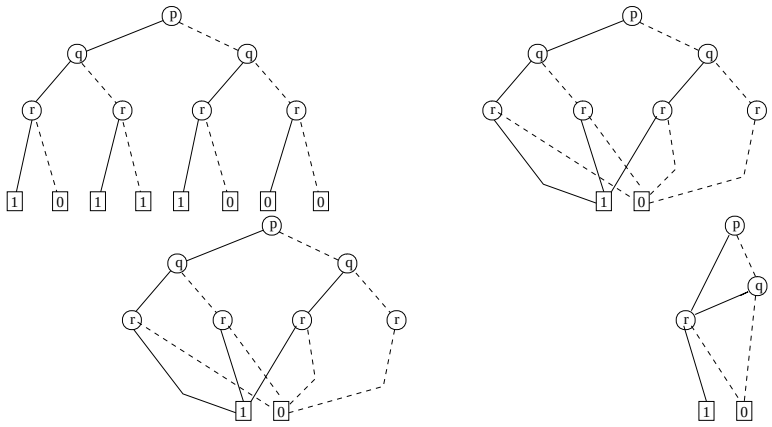

Figure 5.6. *Construction du ROBDD correspondant à la formule* $F = (p \wedge q) \vee r$.

– Pour tout modèle de F*, il existe un chemin et un seul dans le ROBDD de la source vers le puits* 1 *tel que l'interprétation partielle correspondante est contenue dans le modèle de* F.

EXEMPLE 5.5.– Soit S l'ensemble de clauses $S = \{p \vee q,\ p \vee r,\ \neg q \vee \neg r\}$, l'ensemble des propositions est $\mathcal{V} = \{p, q, r\}$ et le diagramme de décision binaire correspondant à S est illustré dans la figure 5.7.

p	q	r	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Dans cet exemple, la première réduction permet de supprimer le sommet le plus à droite étiqueté par q ainsi que le deuxième sommet à partir de la gauche étiqueté par r, dans chacun des cas le successeur gauche est égal au successeur droit. Par ailleurs, il y a deux chemins de la source vers le puits $1 : p = 1,\ q = 1,\ r = 0$ et $p = 1,\ q = 0$. Le premier chemin définit un modèle de S, alors que le second chemin définit implicitement deux modèles de S selon l'affectation de la variable r.

Soit maintenant S l'ensemble de clauses $S = \{p \vee q,\ p \vee r,\ \neg q \vee \neg r,\ \neg p\}$, l'ensemble des propositions est $\mathcal{V} = \{p, q, r\}$ et le diagramme de décision binaire

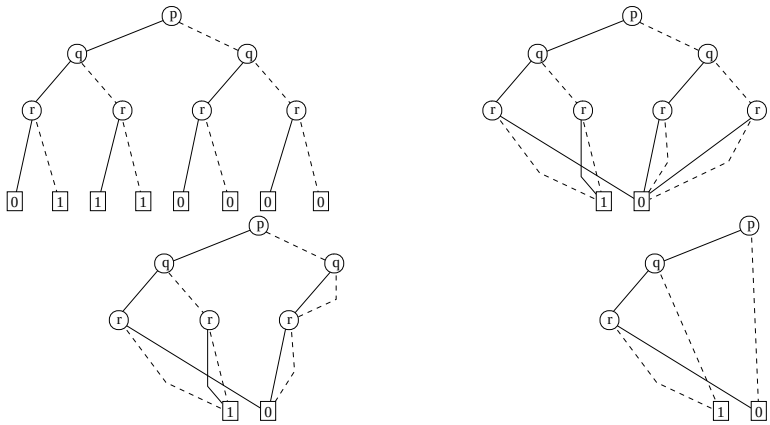

Figure 5.7. *Construction du ROBDD correspondant à l'ensemble de clauses* $S = \{p \vee q, p \vee r, \neg q \vee \neg r, \}$.

correspondant à S est illustré dans la figure 5.8.

p	q	r	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Le ROBDD correspondant à S se réduit au puits 0. En effet, la deuxième réduction fusionne les sommets étiquetés par q, puis les sommets étiquetés par r. La première réduction supprime le sommet r, puis le sommet q, enfin le sommet p et il ne reste plus que le puits 0.

L'avantage des ROBDD réside dans une représentation plus compacte des formules propositionnelles. La recherche de cohérence revient à chercher dans le ROBDD un chemin de la source vers le puits 1, cette recherche s'effectue en temps constant[3] ce qui constitue l'un des avantages majeurs des ROBDD. La taille occupée par un ROBDD, lors de sa construction, dépend de l'ordre total sur les variables et l'inconvénient est que, dans le pire des cas, la taille du ROBDD reste exponentielle en la taille des formules représentées. Cependant, dans la pratique, au lieu de réduire un arbre de décision binaire de gande taille on utilise une construction incrémentale

3. Si la formule est incohérente le ROBDD se réduit au puits 0.

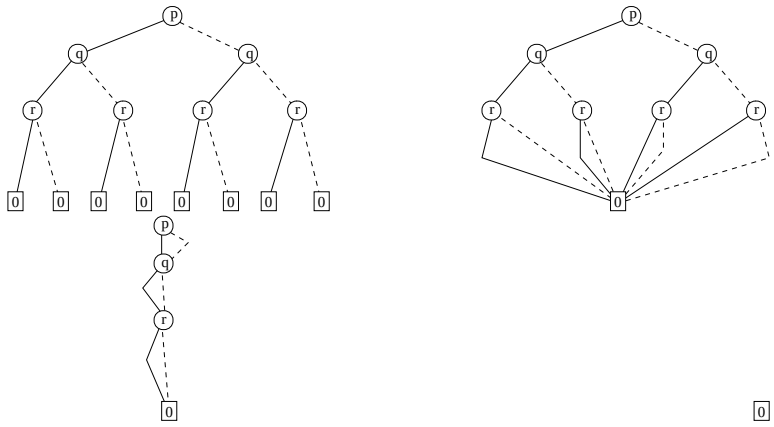

Figure 5.8. *Construction du ROBDD correspondant à l'ensemble de clauses* $S = \{p \vee q, p \vee r, \neg q \vee \neg r, \neg p\}$.

du ROBDD. Soit R_F et R_G les ROBDD correspondant respectivement aux formules propositionnelles F et G, le ROBDD correspondant à la formule $F\ op\ G$, où op représente un connecteur binaire booléen, est construit à partir de R_F et de R_G avec une complexité en temps de l'ordre de $O(S_F \times S_G)$ où S_F est le nombre de sommets du ROBDD R_F et S_G est le nombre de sommets du ROBDD R_G.

5.5. Programmation logique

La programmation logique s'inscrit dans la programmation déclarative qui consiste à représenter dans un formalisme adéquat des connaissances sur un domaine donné à partir desquelles peuvent être déduites les réponses à des questions posées par un utilisateur. Un programme est constitué de faits et de règles qui sont exploitées par un mécanisme de déduction pour répondre à toute question dont la réponse peut être logiquement déductible des connaissances fournies préalablement.

D'un point de vue théorique, la programmation logique repose sur la logique. Les faits et les règles d'un programme logique peuvent être traduits en un ensemble de clauses de Horn[4] et le mécanisme de déduction repose sur résolution [ROB 65] présentée en 5.2 qui permet de décider de la satisfaisabilité d'un ensemble de clauses de Horn.

Le premier langage de la programmation logique est le langage PROLOG [GIA 85, COL 96] mis au point au début des années 1970, à Marseille. La sémantique de ce langage est donnée en termes de point fixe et la complétude de la stratégie de résolution est prouvée dans [KOW 79].

4. Clauses comportant un seul littéral positif.

A la fin des années 1980, la programmation logique a été étendue afin de traiter des contraintes. Parmi les principaux langages développés, on peut citer CLP(R) [JAF 92], principalement dédié aux contraintes numériques, PROLOG III [COL 90] qui traite les contraintes linéaires sur les rationnels et les booléens, CHIP [DIN 88] orienté sur les contraintes dans les domaines finis. Sur le plan théorique, la programmation logique avec contraintes emprunte à la logique le mécanisme de résolution et au domaine de la satisfaction de contraintes (CSP) présenté en 7 les algorithmes de satisfaction de contraintes.

Les sections qui suivent présentent comment la programmation logique a été utilisée pour la représentation et le raisonnement sur le temps et l'espace.

5.5.1. *Programmation logique pour le temps*

D'un point de vue logique, le temps est essentiellement représenté selon deux familles d'approches, les formalismes basés sur la logique modale 3 et ceux basés sur la logique classique du premier ordre. La logique modale formalise la position temporelle d'une formule relativement à un instant courant implicite et utilise des opérateurs modaux pour représenter le passé, le présent et le futur. La logique classique du premier ordre modélise la position temporelle absolue d'une formule sur l'axe du temps défini explicitement.

Sans chercher l'exhaustivité, cette section présente brièvement un certain nombre de formalismes basés sur la logique classique dont la mise en oeuvre utilise la programmation logique. Parmi ceux-ci, des extensions du langage Prolog comme le langage Datalog$_{1s}$ [CHO 93] qui est une extension du langage Datalog [GAL 78, MIN 87] pour les bases de données déductives temporelles et le langage Temporal Prolog de Hrycej [HER 88] [5] qui s'appuie sur le modèle des contraintes temporelles de Allen [ALL 84] et qui permet de raisonner sur les intervalles temporels. Parmi les langages de programmation logique avec contraintes, la programmation logique temporelle annotée par contraintes (TACLP) [FRü 96] repose sur la logique temporelle annotée qui se définit comme un compromis entre la logique modale temporelle et la logique classique. Enfin deux formalismes de raisonnement de sens commun qui permettent de représenter l'initialisation et la persitence de propriétés avec le temps. Le calcul des situations[MCC 69] traite des transitions entre situations globalement alors que le calcul des évènements [KOW 86] traite des effets des actions localement.

5.5.1.1. *Les extensions du langage Datalog*

Le langage Datalog [GAL 78, MIN 87, BID 91] est une extension de Prolog aux bases de données déductives. Contrairement à Prolog, Datalog ne possède pas de fonction, ce qui conduit à un univers de Herbrand fini. Plusieurs extensions de Datalog ont été proposées pour le traitement du temps. Les langages Datalog$^{<Z}$

5. à distinguer du langage également appelé Temporal Prolog de Gabbay [GAB 87] défini pour la logique modale temporelle.

[REV 90, REV 93] et Datalog$^{<Q}$ [KAN 90, KAN 94] relèvent plutôt de la programmation logique avec contraintes, ils permettent de manipuler respectivement des contraintes sur les entiers et des contraintes sur les rationnels. Le langage Datalog$_{1s}$ [CHO 88, CHO 93] est le langage Datalog étendu avec une fonction unaire, la fonction successeur. Afin de d'accroître leur expressivité, ces langages ont été ensuite étendus par l'ajout de la négation stratifiée[APT 88] dans le corps des règles [CHO 94].

5.5.1.2. *Le langage Temporal Prolog de Hrycej*

Le langage Temporal Prolog de Hrycej [HER 88] est une extension du langage Prolog capable de traiter les assertions logiques référencées temporellement et les contraintes temporelles. Ce langage s'appuie sur le modèle des contraintes temporelles de Allen afin de raisonner sur les intervalles temporels et leurs relations. Le langage Temporal Prolog étend le langage Prolog par l'addition de deux types de clauses Â :

– les *références temporelles* sont utilisés pour spécifier qu'une assertion est vraie pendant un intervalle de temps, par exemple $P\ in\ T$ où P est un fait ou une règle et T un identificateur d'intervalle. Les références temporelles peuvent être retirées en utilisant les prédicats $P\ dur\ T$ ou $P\ mkdur\ T$;

– les *contraintes temporelles* axiomatisent les relations entre intervalles de temps. Elles sont formalisées par le prédicat $constrain - rel(T, S, R)$ où T et S sont des intervalles, et R est une relation entre intervalle du type $<$ (avant), $>$ (après), m (rencontre), mi (rencontré), o (intersecte), di (contient) etc. Les contraintes d'intervalles peuvent être retirées en utilisant le prédicats $get - rel(T, S, R)$.

Un programme Temporal Prolog contient des clauses ordinaires et des références temporelles. Des contraintes temporelles peuvent éventuellement rajoutées au programme. Temporal Prolog nécessite un solveur de contraintes temporelles qui est une version légèrement modifiée d'un solveur de contraintes de Allen. Les liens entre le solveur de contraintes et Temporal Prolog sont établis au travers du prédicat $constrain - rel$.

5.5.1.3. *La programmation logique temporelle annotée par contraintes*

La logique temporelle annotée [FRü 96] est une logique intermédiaire entre la logique modale temporelle et la logique du premier ordre qui tire parti des avantages de ces deux approches en évitant la prolifération de variables temporelles et de quantificateurs. Cette logique est défini pour le raisonnement temporel qualitatif mais aussi quantitatif, elle permet de traiter l'information temporelle définie et indéfinie, les instants et les intervalles de temps, elle est adaptée pour différents modèles du temps (linéaire ou circulaire, discret ou continu, borné ou non borné).

La programmation logique temporelle annotée par contraintes est un instance d'un langage de programmation logique plus général la programmation logique annotée par contraintes [KIF 92]. C'est un formalisme qui repose sur deux concepts de base, les annotations simplicité conceptuelle et les contraintes qui permettent une mise en oeuvre efficace du raisonnement.

5.5.1.4. *La logique temporelle annotée*

Le principe de base de la logique temporelle annotée est de séparer les aspects temporels des aspects non temporels. La logique temporelle annotée est une logique du premier ordre dont les formules sont annotées des labels temporels, les annotations , qui indiquent la période temps pour laquelle la formule est valide. Les annotations permettent de représenter des instants, des intervalles de temps, des durées.

Trois sortes d'annotations sont définies sur des ensemble d'instants :

– $A\ at\ t$ la formule A est valide à un instant t,

– $A\ th\ I$ la formule A est valide pour l'ensemble d'instants I,

– $A\ in\ I$ la formule A est valide à un instant dans l'ensemble d'instants I.

La logique temporelle annotée comprend les axiomes de la logique du premier ordre et de nouveaux théorèmes sont définis sur les relations entre les annotations, en particulier, négation, conjonction et disjonction []. Les différents théorèmes capturent les axiomes des logiques temporelles définies dans la littérature.

Une instance particulière de la logique temporelle annotée peut être définie en considérant une relation d'ordre partiel entre les instants, notée $\leq$. Soit t et s deux instants, $r \leq r$ signifie que r précde ou est ègal à t. Soit lb et ub les bornes inférieure et supérieure de l'axe du temps, toute variable temporelle t est bornée, i.e. $lb \leq ub$. Selon l'ordre partiel choisi il est possible de représenter le temps comme linéaire, branching, circulaire, discret ou continu, borné ou non borné. Les durées et périodes temporelles peuvent être représentées par des intervalles. Les différents opérateurs temporels des logiques temporelles standard peuvent être représentés en logique temporelle annotée comme l'illustre le tableau suivant :

opérateur temporel	*logique modale*	*logique temporelle annotée*
quelquefois dans le passé	$P\ A$	$A\ in[-\infty, 0)$
toujours dans le passé	$H\ A$	$A\ th[-\infty, 0)$
quelquefois dans le futur	$F\ A$	$A\ in(0, \infty]$
toujours dans le futur	$G\ A$	$A\ th(0, \infty]$
avant	$\bullet\ A$	$A\ at - 1$
après	$\circ\ A$	$A\ at + 1$
depuis	$A\ S\ B$	$\exists r(A\ in[r, 0) \wedge B\ th(r, 0))$
jusqu′à	$A\ U\ B$	$\exists r(A\ in(0, r] \wedge B\ th(0, r))$

5.5.1.5. *Logique annotée avec contraintes*

La logique annotée avec contraintes est une logique du premier ordre étendue avec un ensemble de termes que sont les annotations ainsi qu'avec un ensemble de prédicats particuliers, appelés contraintes relationnelles et un ensemble de contraintes fonctionnelles. Les formules de cette logique sont des formules étiquetées par des annotations.

Cette logique est la plus petite théorie des contraintes qui permet d'axiomatiser des opérations de treillis pour les annotations. Les annotations sont ordonnées selon un ordre partiel, qui est contrainte relationnelle notée $\sqsubseteq$, et forment un semi-treillis supérieur, i.e. tout sous-enemble fini non vide possède une plus petite borne supérieure qui est une contrainte fonctionnelle notée $\sqcup$.

La mise en oeuvre du raisonnement en logique annotée avec contraintes utilise les règles d'inférence usuelles de le la logique classique comme la règle de modus ponens, mais également des règles d'inférence supplémentaires qui utilisent la structure de treillis des annotations :

$$(\sqsubseteq)\quad \frac{A\ \alpha,\quad \beta \sqsubseteq \alpha}{A\ \beta} \qquad\qquad (\sqcup)\quad \frac{A\ \alpha,\quad A\ \beta}{A\ (\alpha \sqcup \beta)}$$

La règle d'inférence $(\sqsubseteq)$ spécifie que si une formule est valide pour une annotation elle est également valide pour toutes les annotations qui sont inférieures selon l'ordre partiel du treillis des annotations.

La règle d'inférence $(\sqcup)$ spécifie que si une formule est valide pour une annotation et que cette même formule est valide pour une autre annotation alors elle est valide pour la plus petite borne supérieure des deux annotations.

Ces deux règles d'inférence conduisent à la règle d'inférence :

$$(\sqsubseteq\ \sqcup) \quad \frac{A\ \alpha, \quad A\ \beta, \quad \gamma \sqsubseteq \alpha \sqcup \beta}{A\ \gamma}$$

A partir de ces trois règles d'inférences, la résolution est définie pour les formules annotées comme suit :

$$(A - résolution) \quad \frac{A\ \alpha, \quad B, \quad (B \rightarrow A\ \alpha), \quad \gamma \sqsubseteq \alpha \sqcup \beta}{A\ \gamma}$$

5.5.1.6. *Logique temporelle annotée avec contraintes*

Une instance particulière de la logique annotée avec contraintes est la logique temporelle annotée avec contraintes qui reflète la structure du temps. Les annotations sont des annotations temporelles qui constituent un treillis.

Soit $t, s_1, s_2, s_1, s2$ des instants du treillis où lb et ub sont respectivement les bornes inférieure et supérieure, $[s_1, s_2]$ et $[r_1, r_2]$ sont des intervalles de temps. Les axiomes suivants définissent un ordre partiel pour le treillis des annotations temporelles :

$$\begin{array}{ll}
(th\ \bot) & th\ [ub, lb] = \bot \\
(in\ \top) & in\ [ub, lb] = \top \\
(at\ th) & at\ t = th[t, t] \\
(at\ in) & at\ t = in[t, t] \\
(th\ \sqsubseteq) & th[s_1, s_2] \sqsubseteq th[r_1, r_2] \text{ ssi } r_1 \leq s_1,\ s_1 \leq s_2,\ s_2 \leq r_2 \\
(in\ \sqsubseteq) & in[r_1, r_2] \sqsubseteq th[s_1, s_2] \text{ ssi } r_1 \leq s_1,\ s_1 \leq s_2,\ s_2 \leq r_2
\end{array}$$

Les axiomes $(at\ th)$ et $(at\ in)$ spécifient que $th\ I$ et $in\ I$ sont équivalents à $at\ t$ lorsque l'intervalle I est réduit à un seul instant. L'axiome $(th\ \sqsubseteq)$ sécifie que si une formule est valide pour chaque instant d'un intervalle I alors elle est valide pour chaque instant des sous-intervalles de I. L'axiome $(in\ \sqsubseteq)$ spécifie que si une formule est valide à un instant donnée d'un intervalle I alors elle est valide pour tout elle est valide pour un instant des intervalles qui contiennent I. Une conséquence de ces axiomes est :

$$in\ (th\ \sqsubseteq) \quad in[s_1, s_2] \sqsubseteq th[r_1, r_2] \text{ ssi } s_1 \leq r_2,\ r_1 \leq s_2,\ s_1 \leq s_2,\ r_1 \leq r_2$$

Une formule annotée par in est valide pour tout intervalle qui intersecte un intervalle pour lequel la formule est valide en tout instant.

L'axiomatisation de la plus petite borne supérieure pour des annotations temporelles est donnée par :

$$(th \sqcup) \quad th[s_1, s_2] \sqcup th[r_1, r_2] = th[s_1, r_2] \text{ ssi } s_1 < r_1,\ r_1 \leq s_2,\ s_2 < r_2.$$

Les contraintes du treillis sur les intervalles de temps peuvent se réduire à des conjonctions de contraintes sur des instants. Dans ce cas, la complexité de traitement de telles contraintes par un algorithme de de consistance de chemin[MEI 91] est de l'ordre de $O(n^3)$ où n est le nombre de variables qui repr ésentent les instants.

5.5.1.7. *La programmation logique temporelle annotée par contraintes*

Un fragment clausal de la programmation logique temporelle annotée par contraintes, notée TACLP, peut être défini. Il constitue un langage de programmation logique temporel efficace.

Un programme logique annoté par contraintes est un ensemble fini de clauses de la forme :

$$A\ \alpha \leftarrow C_1 \wedge \cdots \wedge C_n \wedge B_1\ \alpha_1 \wedge \cdots \wedge B_m\ \alpha_m$$

où A est un atome, les C_i sont des contraintes relationnelles, les B_i sont formules propositionnelles atomiques, α et $alpha_i$ sont des annotations temporelles avec $n \geq 0$ et $m \geq 0$. Le mécanisme de déduction pour TACLP repose sur une $A - résolution$ spécialisée pour les annotations temporelles [FRü 96].

L'implantation de la programmation logique temporelle par contraintes annotée utilise une traduction en programmation logique par contraintes (CLP). Une fonction de traduction, notée $comp$, permet de traduire les règles d'un programme TACLP en un programme CLP comme suit :

$comp(A(x_1, \cdots, x_n)\ \alpha)$ pour une formule annotée $A(x_1, \cdots, x_n)\ \alpha$;

$comp(A(x_1, \cdots, x_n))$ pour une formule non annotée $A(x_1, \cdots, x_n)$;

$comp(C)$ pour une contrainte C.

Pour tout symbole de prèdicat annoté p de arité n où les x_i sont des variables distinctes, la clause $comp(p(x_1, \cdots, x_n)\ th[ub, lb])$ est introduite.

La règle de Modus Ponens est traduite par $comp(A) \leftarrow comp(B)$ où A est une formule non annotée et la A- résolution est traduite par les trois règles suivantes :

$comp(A)\ th[t_1, t_2]) \leftarrow (s_1 \leq t_1 \wedge t_2 \leq r_2) \wedge (s_1 \leq r_1 \wedge r_1 \leq s_2 \wedge s_2 \leq r_2) \wedge comp(B) \wedge (r_1 = s_1 \wedge r_2 = s_2 \vee comp(A\ th[r_1, r_2])))$ pour une formule annotée $A\ th[s_1, s_2] \leftarrow B$;

$comp(A)\ in[t_1, t_2]) \leftarrow (t_1 \leq s_1 \wedge s_2 \leq t_2) \wedge comp(B)$ pour une formule annotée $A\ in[s_1, s_2] \leftarrow B$;

$comp(A)\ in[t_1, t_2]) \leftarrow (t_1 \leq s_2 \wedge s_1 \leq t_2) \wedge comp(B)$ pour une formule annotée $A\ th[s_1, s_2] \leftarrow B$;

la programmation logique temporelle par contraintes annotée a été étendue au traitement de l'espace par les formalismes STACPL [RAF 01] et MuTACLP [MAN 00].

5.5.1.8. *Le calcul des évènements*

Le calcul des évènements[6] est un formalisme de raisonnement temporel dédié à la formalisation de scénarios caractérisés par un ensemble dévènements [KOW 86]. L'occurrence d'un évènement a pour effet d'initialiser ou de mettre fin à la validitè de propriétés données.

Etant donné une description des évènements[7], de leur occurrence, et des propriétés qu'ils affectent, le calcul des évènements détermine l'intervalle de temps maximum pour lequel une propriété se déroule sans interruption.

Les concepts de base du calcul des évènements sont les évènements, les actions, les instants et les fluents[8].

La mise en oeuvre du calcul des évènements a été initialement proposée dans le cadre de la programmation logique avec négation par échec. Les connaissances sont représentées par des faits et des règles. Les faits de base du calcul des évènements sont les suivants :

$holds - at(F, T)$: le fluent F est VRAI à l'instant T,

$happens(E, T)$: l'évènement E se produit à l'instant T,

$initiates(E, F, T)$: l'évènement E rend le fluent VRAI à l'instant T,

$terminates(E, F, T)$: l'évènement E rend le fluent FAUX à l 'instant T,

$T_1 < T_2$: l'instant T_1 est avant l'instant T_2,

$initially(F)$: le fluent F est vrai à partir de l'instant initial,

$clipped(T_1, F, T_2)$: le fluent F est terminé entre l'instant T_1 et T_2.

Les axiomes du calcul des évènements sont exprimés comme des règles d'un programme logique :

6. Event Calculus en anglais.

7. La description des évènements peut être incomplète.

8. Les fluents sont des propriétés dont la validité peut varier au cours du temps.

EC1) $holds - at(F, T) \leftarrow initially(F), not\ clipped(0, F, T)$

EC2) $holds - at(F, T_2) \leftarrow happens(E, T_1), T_1 < T_2, initiates(E, F, T_1), not\ clipped(T_1, F, T_2)$

EC3) $clipped(T_1, F, T_2) \leftarrow happens(E, T), terminates(E, F, T), T_1 \leq T, T < T_2$

Le premier axiome $EC1)$ exprime qu'un fluent est valide à l'instant T si il a été valide à l'instant 0 et ne s'est pas terminé entre l'instant 0 et l'instant T.

Le second axiome $EC2)$ exprime qu'un fluent est valide à l'instant T_2 si il a été initialis 'e à un instant T_1 précédant T_2 et ne s'est pas terminé entre les instants T_1 et T_2.

Le troisième axiome $EC3)$ exprime qu'un fluent s'est terminé entre les instants T_1 et T_2 s'il s'était valide à l'instant T où il s'est terminé.

La description de scénarios par des programmes logiques permet de dériver la validité de propriétés sur des intervalles de temps. Le mécanisme de déduction est celui de PROLOG basé sur la résolution[DEN 92].

Le calcul des évènements a été ensuite reformulé comme un formalisme logique du premier ordre[SHA 99], pour modéliser les actions et les effets des actions et traiter en particulier du probème du cadre[9]. Couplé avec un mécanisme de circonscription [LIF 94], il permet le raisonnement non-monotone. Le raisonnement dans cette formulation du calcul des évènements se traduit par la recherche de modèles de formules du premier ordre [MUE 04]. Afin de traiter des actions parallèles le calcul des évènements a ́té étendu avec des modalités[CER 96].

5.5.1.9. *Le calcul des situations*

Le calcul des situations[10] est un formalisme logique basé sur la logique des prédicats qui a été introduit pour représenter et raisonner sur les actions et le changement [MCC 69].

Les concepts de base du calcul des situations[LEV 98] sont les actions qui permettent de changer l'état du monde et les situations qui peuvent être considérées comme létat du monde après l'exécution d'une séquence d'actions. Une situation particulière est la situation initiale qui est représentée par une constante, notée S_0. Une fonction particulière, notée $do(s, a)$ où s est une situation et a est une action, permet de calculer la situation résultant de l'exécution de l'action a dans la situation s.

Tout comme le calcul des évènements le calcul des situations utilise les instants et les fluents.

9. Frame problem en anglais.

10. Situation Calculus en anglais.

Le comportement des actions est formalisé par deux types d'actions. Le premier type d'axiome concerne les pré-conditions nécessaires qui doivent ête satisfaites pour que l'action puisse être exécuté dans la situation courante. Ces axiomes utilisent le prédicat $Poss(a, s)$ qui indique s'il est possible d'exécuter l'action a dans la situation s. Le deuxième type d'axiomes représentent les effets des actions sur les fluents.

Le calcul des situations est utilisé en intelligence artificielle pour prédire les effets des actions, en particulier, dans les domaines de la planification et du diagnostic et plusieurs implantations ont été réalisées en PROLOG [PIN 93, LEV 97, LIN 97].

5.5.2. *programmation logique pour l'espace*

Au début des années 1990, dans le domaine de l'EIAO (Enseignement Intelligemment Assisté par Ordinateur), sont apparus, les premiers logiciels d'aide à la démonstration automatique en géométrie. Ces outils ont été le fruit de projets de recherche initiés au milieu des années 1980 [AND 85, KOE 90, HOL 96, BAR 89]. Ils sont issus d'une collaboration entre chercheurs en informatique, le plus souvent en intelligence artificielle, chercheurs en didactique des mathématiques et enseignants de collège. De nombreux travaux se sont déroulés en France suite à l'essort du langage PROLOG [GIA 85, COL 96]. Parmi ces démonstrateurs on peut citer, par exemple, MENTONIEZH [PY 96], CABRI-EUCLIDE [LAB 95, LAB 94, LUE 97], GEOMUS [BAZ 93], ARCHIMEDE [CHO 87], CHYPRE [BER 94], DEFI [Ag 92], TALC [DES 94], GEOSPECIF [BOU 97b]. La caractéristique commune de ces démonstrateurs est que leur implantation utilise la programmation logique, à l'exception de CABRI-EUCLIDE qui est implanté en langage C.

Ces tuteurs intelligents ont pour objectif la résolution de problèmes en géométrie élémentaire. L'activité géométrique revêt divers aspects, tracé et construction de figures, recherche de propriétés, apprentissage de démonstrations, vérification de cohérence de preuves.

Les démonstrateurs développés peuvent être classés selon P. Balbiani [BAL 94] en trois catégories. Ceux qui adoptent une approche logique. Les problèmes sont traduits en formules du calcul des prédicats du premier ordre et les preuves des énoncés géométriques sont établies selon des techniques de déduction automatique, comme dans TALC et MENTONIEZH.

Ceux qui suivent une approche algébrique (ou analytique). Les problèmes géométriques sont traduits en problèmes algébriques équivalents par des techniques de la géométrie analytique. La résolution de problèmes revient à la résolution de systèmes d'équations comme dans le cas de GEOSPECIF.

Enfin ceux qui adoptent une approche heuristique qui cherche à simuler des règles expertes utilisées par une personne lors de la résolution d'un problème géométrique comme dans ARCHIMEDE et GEOMUS.

5.5.2.1. *La géométrie élémentaire*

La géométrie élémentaire est définie par un ensemble d'objets "géométriques" : points, droites, segments, cercles, angles, triangles, quadrilatères, etc $\cdots$, un ensemble de relations entre ces objets : incidence, parallélisme, perpendicularité etc $\cdots$, et un ensemble de théorèmes et d'axiomes. Différentes axiomatisations de la géométrie ont été proposées, la plus ancienne est celle d'Euclide avec un système de 5 axiomes :

1) Pour tout point P et tout point Q non égal à P il existe une unique droite L telle que P et Q sont sur L.

2) Pour tout segment AB et tout segment CD il existe un point unique E tel que B est entre A et E et le segment CD est égal au segment BE.

3) Pour tout point O et tout point A non égal à O il existe un cercle de centre O et rayon OA.

4) Tous les angles droits sont égaux entre eux.

5) Pour toute droite L et pour tout point P qui n'est pas sur L il existe une unique droite M telle que P est sur M et M est parallèle à L.

Les axiomes d'Euclide ne sont pas adaptés pour une utilisation par un système formel, car ils utilisent beaucoup de connaissances implicites qui devraient être représentées par une axiomatisation formelle. Hilbert reprend l'axiomatisation d'Euclide en précisant certains termes et en ajoutant des axiomes qui étaient implicites la formulation d'Euclide. Suite à une réflexion sur la compatibilité et l'indépendance des axiomes et grâce à l'utilisation de la théorie des ensembles, il propose une axiomatisation en 20 axiomes qui peuvent être classés en 5 groupes : les axiomes d'appartenance, axiomes d'ordre, axiomes de congruence, axiomes des parallèles et axiomes de continuité [DES 94]. Tarski propose ensuite une axiomatisation sans l'utilisation de la théorie des ensembles, il s'agit d'un ensemble de 13 axiomes formulés en logique des prédicats en n'utilisant que les prédicats *égal*, *entre*, et *congruent*.

5.5.2.2. *Représentation des connaissances géométriques élémentaires*

Tarski propose une construction de la géométrie élémentaire dans un langage logique du premier ordre, il montre que son modèle est décidable, mais n'est pas finiment axiomatisable [CAR 98] [BAL 94, BAL 96]. Ce langage est trop complexe pour être directement transposé dans un démonstrateur automatique simple et efficace. Les connaissances géométriques élémentaires : objets géométriques, relations entre ces objets et théorèmes relèvent, à priori, de la logique classique cependant la géométrie élémentaire ne relève pas de la logique des prédicats du premier ordre car elle manipule des propriétés d'existence, des quantifications sur des relations des ensembles infinis. Aussi, plusieurs travaux ont porté sur la représentation de la connaissance géométrique élémentaire, les représentations proposées ne sont que des approximations de la géométrique élémentaire et ont une puissance d'expression inférieure à celle-ci.

Les formalismes de représentation, issus de la logique des prédicats du premier ordre sont définis de telle sorte que l'univers de Herbrand soit fini afin d'assurer la propriété de décidabilité.

Le langage LDL (Logical Description Langage) [DES 94], utilisé dans TALC, dispose des types point, droite, demi-droite, segment, cercle, distance. Deux catégories de prédicats sont définis, des prédicats de typage qui expriment le type et les composants d'un objet géométrique et des prédicats de propriétés qui expriment les relations géométriques entre les objets. Les relations représentées sont l'appartenance, le parallélisme, la perpendicularité, le sens d'orientation, la distance, l'égalité. Une formule bien formée du langage LDL est une conjonction de formules de bases. Une formule de base est soit un atome soit la négation d'un atome de propriété. Toutes les formules sont closes. Le type de tout identificateur est défini par un seul atome de typage. Les formules sont restreintes aux formules ne contenant ni terme fonctionnel, ni quantification existentielle. Les axiomes et les théorèmes sont traduites en clauses de Horn.

Le langage HDL (Hypotheses Description Langage) [PY 96] est le formalisme de représentation utilisé dans le projet MENTONIEZH. Il est défini à partir d'un ensemble fini de variables, de constantes, des types point, droite et cercle qui représentent les objets de base, d'une fonction qui associe à un couple de points une droite, d'un ensemble de treize prédicats représentant les relations entres les objets, un prédicat d'arité 2 : différent, trois prédicats d'arité 3 : aligné, milieu, triangle rectangle, neuf prédicats d'arité 4 : égale, losange, médiatrice, demi-distance, noncroisé, parallèles, parallélogramme, perpendiculaires, rectangle. Les théorèmes et axiomes sont traduits en clauses de Horn.

L'approche algébrique utilise la géométrie analytique formalisée par Descartes (1596-1650) pour démontrer certains résultats de la géométrie. Les objets géométriques sont les points, les droites et les cercles. Les objets de base sont les points représentés par des couples de coordonnées dans un repère donné. Les coordonnées sont des nombres constructibles[11]. Les points fixés sont représentés par des couples de constantes et les points construits par des couples de variables. Les droites et les cercles sont définis comme des ensembles de points vérifiant les équations qui les caractérisent. Les relations entres les objets géométriques sont l'appatenance d'un point à une droite, à un cercle, le parallélisme, la perpendicularité, le milieu. Ces relations s'expriment sous forme de systèmes d'équations.

5.5.2.3. *Raisonnement en géométrie élémentaire*

Disposant d'une formalisation de la géométrie élémentaire, le rôle du démonstrateur est de produire l'ensemble des faits déductibles à partir des hypothéses.

Lorsque la géométrie élémentaire est formalisée avec les langages LDL ou HDL, les énoncés sont exprimés sous forme de clauses de Horn et la déduction prend la forme de la résolution. Dans le cas de MENTONIEZH [PY 96], la stratégie de résolution 5.2 correspond au fonctionnement d'un moteur d'inférence travaillant par saturation sur une base de faits, en chaînage avant et selon un parcours en largeur d'abord.

11. L'ensemble des nombres constructibles est le plus petit sous-corps des réels stable par la racine carrée.

La mise en oeuvre du démonstrateur est réalisée en PROLOG. Dans TALC [DES 94], la stratégie de résolution 5.2 correspond au fonctionnement du moteur d'inférence de PROLOG, c'est à dire en chaînage arriere et selon un parcours en profondeur d'abord. La mise en oeuvre du démonstrateur est réalisée en PROLOGII+. Dans les deux cas la stratégie de résolution, plus précisémment la SDL-résolution est complète. Dans CABRI-EUCLIDE [LUE 97], la géométrie élémentaire est formalisée avec le langage LDL, cependant la mise en oeuvre du démonstrateur est réalisée en langage C pour des raisons d'efficacité, de portabilité et de compatibilité avec CABRI-GEOMETRE développé en C.

Le démonstrateur automatique pour la géométrie GEOMUS [BAZ 93] est réalisé à partir du démonstrateur automatique de théorèmes MUSCADET [PAS 02]. Celui-ci est composé d'une base de connaissances et d'un moteur d'inférence. L'architecture de MUSCADET est indépendante de la connaissance qui est représentée. Les connaissances sont représentées par des règles et des méta-règles exprimées en logique du premier ordre. La méthode de déduction est la méthode de déduction naturelle. La mise en oeuvre est réalisée en PROLOG.

GEOSPECIF adopte une approche de programmation logique avec contraintes dédiée à la géométrie. Les objets géométriques sont décrits de façon déclarative. Le démonstrateur effectue une traduction algébrique du problème à résoudre puis effectue la résolution du système déquations algébriques selon un algorithme de satisfaction de contraintes quadratiques. Cet algorithme est basé sur l'algorithme de résolution de contraintes non-linéaires de PROLOG III.