LOGIQUE COMPUTATIONNELLE & PROLOG – 7

RÉSOLUTION SLD

7.1

Présentation de la résolution SLD

La *résolution SLD*, est la technique d'inférence qui s'applique à des programmes logiques définis. Cette technique est issue du *principe de la résolution* introduit par J. A. Robinson en 1965 pour des programmes logiques plus généraux que les programmes logiques définis. La résolution SLD fut présentée pour la première fois par R. Kowalski en 1974 et signifie *résolution* **L***inéaire avec fonction de* **S***élection pour clauses* **D***éfinies* (Linear resolution for Definite clauses with Selection function).

Pour démontrer qu'un fait peut être déduit d'un programme logique, on suppose que le contraire de ce fait existe dans le programme et en utilisant, de façon répétitive, le modus–ponens et la règle de l'élimination du quantificateur universel, on établit un résultat absurde.

Considérons par exemple le fait $F = B_1 \wedge \ldots \wedge B_m$ et un programme défini E. Pour démontrer que F est une conséquence logique de E on place la négation de ce fait dans le programme E, c'est-à-dire on rajoute à E le but B représenté par $\leftarrow B_1 \ldots, B_m$ et, par la suite, on cherchera à montrer que le programme E, ainsi enrichi, contient la clause vide. Le principe de résolution – qui est un algorithme de raisonnement – peut nous aider à résoudre ce problème. Chaque étape de l'algorithme de raisonnement transforme le but – qui est un ensemble de formules atomiques – en un nouveau but, c'est-à-dire en un nouvel ensemble de formules atomiques. Pour ce faire, l'étape est décomposée comme suit :

– On sélectionne une formule atomique parmi celles du but, mettons la formule B_k.

– On choisie une clause de E, mettons la clause $C : A \leftarrow A_1, \ldots, A_n$, qui *filtre* B_k, ce qui signifie que B_k et A sont unifiables. Supposons que θ_{A,B_k} soit leur UPG.
– On obtient le nouveau but en remplaçant la formule atomique B_k du but par $(A_1 \circ \theta_{A,B_k}, \ldots, A_n \circ \theta_{A,B_k})$ qui est constituée par les conditions de la clause C sur lesquelles nous avons effectué la substitution θ_{A,B_k}.

Si, en répétant cette démarche, on aboutit à la clause vide, on obtient alors la négation du but.

Comme le but $\leftarrow B_1, \ldots, B_m$ peut aussi s'écrire sous la forme $\forall X_1 \ldots, X_l \neg (B_1 \wedge \ldots \wedge B_m)$, on en déduit que $\neg \exists X_1 \ldots, X_l \, (B_1 \wedge \ldots \wedge B_m)$. Ce résultat est obtenu car nous avons mis dans E la négation du fait F que nous voulions démontrer et nous concluons ainsi que cette négation est absurde eu égard au programme E.

Notons que cet algorithme est non déterministe, car le choix de la formule atomique B_k du but n'est pas déterminé d'avance mais par contre il est libre. De plus, B_k étant choisie, il est possible qu'il y ait plusieurs clauses dans E qui filtrent B_k et, par conséquent, le choix de la clause à utiliser pour l'unification est aussi libre. Nous pouvons ainsi construire plusieurs solutions et même on peut avoir une infinité de solutions. Il est aussi possible, ayant sélectionné B_k, de ne pas pouvoir trouver dans E une clause qui filtre cette formule atomique et, dans ce cas, de ne pas pouvoir construire une solution.

7.2

Formalisation de la résolution SLD

En formalisant on a la définition suivante :

DÉFINITION 7.0.35 (Principe de résolution SLD) *Soient un but B défini par la question $\leftarrow B_1, \ldots, B_k, \ldots, B_m$ et un programme défini E contenant la clause $C : A \leftarrow A_1, \ldots, A_n$. Alors nous pouvons dériver de B et C un nouveau but B' défini par*
$\leftarrow (B_1, \ldots, B_{k-1}, A_1, \ldots, A_n, B_{k+1}, \ldots, B_m) \circ \theta_{A,B_k}$ *si*

(i) *B_k est un atome, appelé l'*atome choisi *dans B,*

(ii) *A filtre B_k, et*

(iii) *θ_{A,B_k} est l'UPG de A et B_k.*

Alors B' est appelée la résolvante *de B et C.*

Cette définition introduit la règle d'inférence suivante :

$$\textbf{(R-SLD)} \qquad \frac{\forall \neg (B_1 \wedge \ldots \wedge B_{k-1} \wedge B_k \wedge B_{k+1} \wedge \ldots \wedge B_m) \quad \forall (A \leftarrow A_1, \ldots, A_n)}{\forall \neg (B_1 \wedge \ldots \wedge B_{k-1} \wedge A_1 \wedge \ldots \wedge A_n \wedge B_{k+1} \wedge \ldots \wedge B_m) \circ \theta_{A,B_k}}$$

où

– $B_1, \ldots, B_k, \ldots, B_m$ et $A, A_1, \ldots, A_n$ sont des formules atomiques.

– θ_{A,B_k} est l'UPG de A et B_k.

Remarquons que les deux ensembles des variables de l'antécédent de cette règle d'inférence doivent être distincts. Dans le cas contraire on peut utiliser une substitution de renommage afin de satisfaire à cette exigence.

Pour obtenir la clause vide on est amené à appliquer plusieurs fois itérativement cette règle d'inférence. Lors de ces itérations successives il faut que l'ensemble de variables de la clause C, utilisée pendant une itération, soit disjoint des ensembles de variables des clauses utilisées pendant les itérations précédentes et de l'ensemble de variables du but. Afin de satisfaire à cette exigence on doit utiliser, comme nous venons de voir, la substitution de renommage que nous pouvons mettre au point de manière très simple en portant des indices au but et aux différentes clauses utilisées. Ainsi le but initial B sera noté B_0 ce qui donne $\leftarrow B_{0_1},\ldots,B_{0_m}$. La clause C, qui sera utilisée pour évaluer la résolvante B' de $B=B_0$ et de C, sera notée C_1 ce qui donne $A_1 \leftarrow A_{1_1},\ldots,A_{1_n}$. L'UPG θ_{A,B_k} entre la formule atomique $B_k = B_{0_k}$ du but $B=B_0$ et la formule atomique $A=A_1$ de la clause $C=C_1$ sera noté θ_1. Enfin la résolvante B' de B_0 et C_1 sera notée B_1. En poursuivant cette démarche nous avons que chaque nouveau but B_{i+1} est la résolvante du but B_i et de la clause C_{i+1} par application de l'UPG θ_{i+1}. Cette démarche peut être finie ou infinie. Il y a en fait deux cas qui permettent l'arrêt du processus :

– soit $B_i = \emptyset$, c'est-à-dire la clause vide,

– soit le sous-but sélectionné B_{i_k} ne peut pas être unifié avec la tête d'une clause de E.

On formalise cette démarche à l'aide de la définition suivante :

DÉFINITION 7.0.36 *Soient un but B défini par $\leftarrow B_1,\ldots,B_m$ et un programme défini E. Une* dérivation SLD *de $E\cup\{B\}$ est une suite finie ou infinie des triplets $\{(B_i,C_{i+1},\theta_{i+1})\}_i$ avec $B_0=B$, C_{i+1} une clause de E et θ_{i+1} UPG entre la tête de C_{i+1} et un sous-but B_{i_k} du but B_i de sorte que B_{i+1} dérive de B_i et de C_{i+1} en utilisant θ_{i+1}.*

Une dérivation SLD peut être représentée par une arborescence – appelée arborescence *(ou, improprement,* arbre*)* de dérivation SLD *– telle que*

– *la racine est le but $B=B_0$.*

– *B_{i+1} est un successeur de B_i. L'arc entre B_i et B_{i+1} est étiqueté par C_{i+1} et θ_{i+1}.*

Si la dérivation est finie, le nombre d'éléments de la suite des triplets $\{(B_i,C_{i+1},\theta_{i+1})\}_i$ s'appelle longueur de la dérivation.

La définition suivante introduit la notion de la réfutation ou dérivation SLD, qui est à la base de la démarche de démonstration de Prolog.

DÉFINITION 7.0.37 *Soient un but B défini par $\leftarrow B_1,\ldots,B_m$ et un programme défini E. Une* réfutation SLD *ou* dérivation SLD réussie *de $E\cup\{B\}$ est une dérivation SLD finie dont le dernier élément est la clause vide* [1].

Si, de plus, les unificateurs θ_i de cette dérivation ne sont pas les unificateurs les plus généraux, alors on parle de réfutation SLD sans restriction.

Dans le cas où une dérivation finie n'aboutit pas à la clause vide on parle de dérivation échouée.

1. Il faut remarquer qu'on parle ici de réfutation SLD et on précise que la dérivation SLD est réussie si $E\cup\{B\}$ aboutit à la clause vide, c'est-à-dire s'il échoue. Dans les chapitres précédents, on a établi que E induisait B si $E\cup\{\neg B\}$ échouait. La contradiction n'est qu'apparente. En effet dans les chapitres précédents, B était une clause qu'on transformait en question, c'est-à-dire à $\neg B$, tandis que dans tout le chapitre actuel, B est déjà sous forme d'une question et donc il n'est point nécessaire de la transformer en question.

Dans le cas d'une dérivation réussie ou échouée, la suite $\{\theta_i\}_i$ *des UPG restreinte aux variables du but B est appelée la* réponse calculée de la dérivation.

7.3

Justesse et complétude de la résolution SLD

Considérons un programme défini E. En appliquant sur E la résolution SLD, on obtient des réponses calculées. La *justesse* de la résolution SLDC implique que ces réponses soient correctes, c'est-à-dire, d'après ce que nous avons vu au chapitre précédent, que les réponses correctes forment un sous-ensemble du plus petit modèle $\widetilde{I}_E$ de Herbrand. Le théotème suivant valide cette exigence.

THÉORÈME 7.0.19 (Justesse de la résolution SLD) *Soient un but B, sous forme de question, et un programme défini E. Toute réponse calculée pour* $E \cup \{B\}$ *est une réponse correcte pour* $E \cup \{B\}$.

Il va de soi qu'on doit aussi s'assurer de la *complétude* de la résolution SLD, à savoir que la résolution SLD retrouvent toutes les réponses correctes à un but donné. Le théorème suivant précise ce point.

THÉORÈME 7.0.20 (Complétude de la résolution SLD) *Soient un but B, sous forme de question, et un programme défini E. Pour toute réponse correcte* σ *de* $E \cup \{B\}$ *il existe une réponse calculée* θ *de* $E \cup \{B\}$ *et une substitution* ρ *telle que* $\sigma = \theta \circ \rho$.

De ces deux théorèmes précédents on en conclut que *l'ensemble des succès de E soit égal au plus petit modèle de Herbrand* $\widetilde{I}_E$.

Le théorème de complétude établit l'existence d'une réfutation qui construit une réponse plus générale que toute réponse correcte donnée. Mais, malheureusement il n'indique pas la manière d'obtenir cette réfutation c'est-à-dire la manière de choisir à chaque étape le sous-but et la clause à unifier. Il s'agit en réalité d'un problème ouvert dont quelques éléments de réponse seront donnés pendant les cours du langage de programmation Prolog et des systèmes experts.

7.4

Prolog et résolution SLD

Nous avons vu que la dérivation SLD est non déterministe à cause du fait qu'il y 'a sélection du sous-but $\neg B_k$ à valider et de la clause la clause C adont la tête A est unifiable avec B_k (dans le cas où il y a plusieurs clauses unifiables avec le sous-but).

Le choix de la clause C se fait habituellement selon une stratégie propre au langage utilisé. Prolog choisit systématiquement la première clause de la base de données qui est

unifiable avec le but. Il est évident que si l'ordre des clauses dans la base de données, c'est-à-dire le programme défini E, se trouve modifié, l'ordre des réponses de Prolog sera aussi modifié. Prolog choisit toujours le littéral le plus à gauche dans la liste des buts à vérifier. Il opère ainsi selon une stratégie qui s'appelle *en profondeur d'abord* et qui permet de parcourir d'abord la branche de l'arbre qui est la plus à gauche. Ensuite il y a retour en arrière et parcours des autres branches, non encore explorées, selon le même principe : parcours de la branche la plus à gauche qui n'est pas encore explorée.

Nous illustrons cette démarche à l'aide du programme défini E suivant :

Programme 1

```
(P1)    p(a) <--
(P2)    r(b) <--
(P3)    q(X,Y) <-- p(X).
(P4)    q(X,Y) <-- r(X), p(Y).
```

et le but `B0 : <- q(V,W)..`

L'arbre de dérivation SLD est donné par la figure 7.1 :

On peut vérifier que la trace de l'exécution par Prolog de ce programme reproduit l'arbre de cette figure.

En étudiant plus en détail l'algorithme en profondeur d'abord, on peut remarquer que Prolog n'est pas toujours capable de fournir toutes les solutions et parfois de fournir même pas une solution. En effet lors du parcours d'une branche de l'arborescence, il est possible de se trouver dans une situation de bouclage à l'infini. Dans ce cas, il ne peut fournir que les solutions qu'il a trouvé jusque là et, s'il n'a encore pas trouvé de solution, il ne fournira pas de solution. Ce cas de figure est illustrer opar le programme suivant :

Programme 2

```
(P1)    a(1) <--
(P2)    b(X) <-- a(X).
(P3)    a(2) <--
(P4)    a(X) <-- b(X).
```

et le but `B0 : <- a(X).`

La figure 7.2 présente l'arborescence de dérivation qui permet de voir les différentes situations, énumérées ci-dessus, du parcours en profondeur d'abord.

7.5 Exercices

EXERCICE 7.1 *Soit le programme défini :*

- $p(X,Y) \leftarrow q(X,Z), r(Z,Y)$
- $p(X,Y) \leftarrow r(Y,X)$
- $q(X,Y) \leftarrow r(Y,X)$
- $r(a,b) \leftarrow$

Déterminer une dérivation SLD du but ← `p(b,X)` *et donner la réponse calculée de la dérivation.*

EXERCICE 7.2 *Soit le programme défini :*

- `p(f(Y))` ←
- `q(a)` ←

et le but ← `p( X)`

Déterminer une réponse correcte ainsi que la réponse calculée de dérivation et vérifier que celle-ci est plus générale que celle-là.

EXERCICE 7.3 *Considérons le programme défini suivant :*

- `p` ← `q,r`
- `p` ← `s`
- `q` ←
- `q` ← `s`
- `r` ←
- `s` ← `t`
- `s` ←

(1) Donner l'arbre de dérivation SLD pour le but ← `p`.

(2) Même chose si on retire du programme la 3e clause.

EXERCICE 7.4 *Soit le programme défini :*

- `A` ←
- `B` ← `A`
- `C` ← `B`

Montrer que le but ← `B` *est une conséquence logique du programme.*

EXERCICE 7.5 *En utilisant l'arborescence de dérivation SLD, établir le comportement de quatre programmes Prolog ci-après :*

$$E_1 = \left\{ \begin{array}{l} \texttt{ancetre(X,Y) :- parent(X,Y).} \\ \texttt{ancetre(X,Y) :- parent(X,Z), ancetre(Z,Y).} \end{array} \right\}$$

$$E_2 = \left\{ \begin{array}{l} \texttt{ancetre(X,Y) :- parent(X,Z), ancetre(Z,Y).} \\ \texttt{ancetre(X,Y) :- parent(X,Y).} \end{array} \right\}$$

$$E_3 = \left\{ \begin{array}{l} \texttt{ancetre(X,Y) :- parent(X,Y).} \\ \texttt{ancetre(X,Y) :- ancetre(Z,Y), parent(X,Z).} \end{array} \right\}$$

$$E_4 = \left\{ \begin{array}{l} \texttt{ancetre(X,Y) :- ancetre(Z,Y), parent(X,Z).} \\ \texttt{ancetre(X,Y) :- parent(X,Y).} \end{array} \right\}$$

Vérifier vos conclusions en utilisant Prolog.

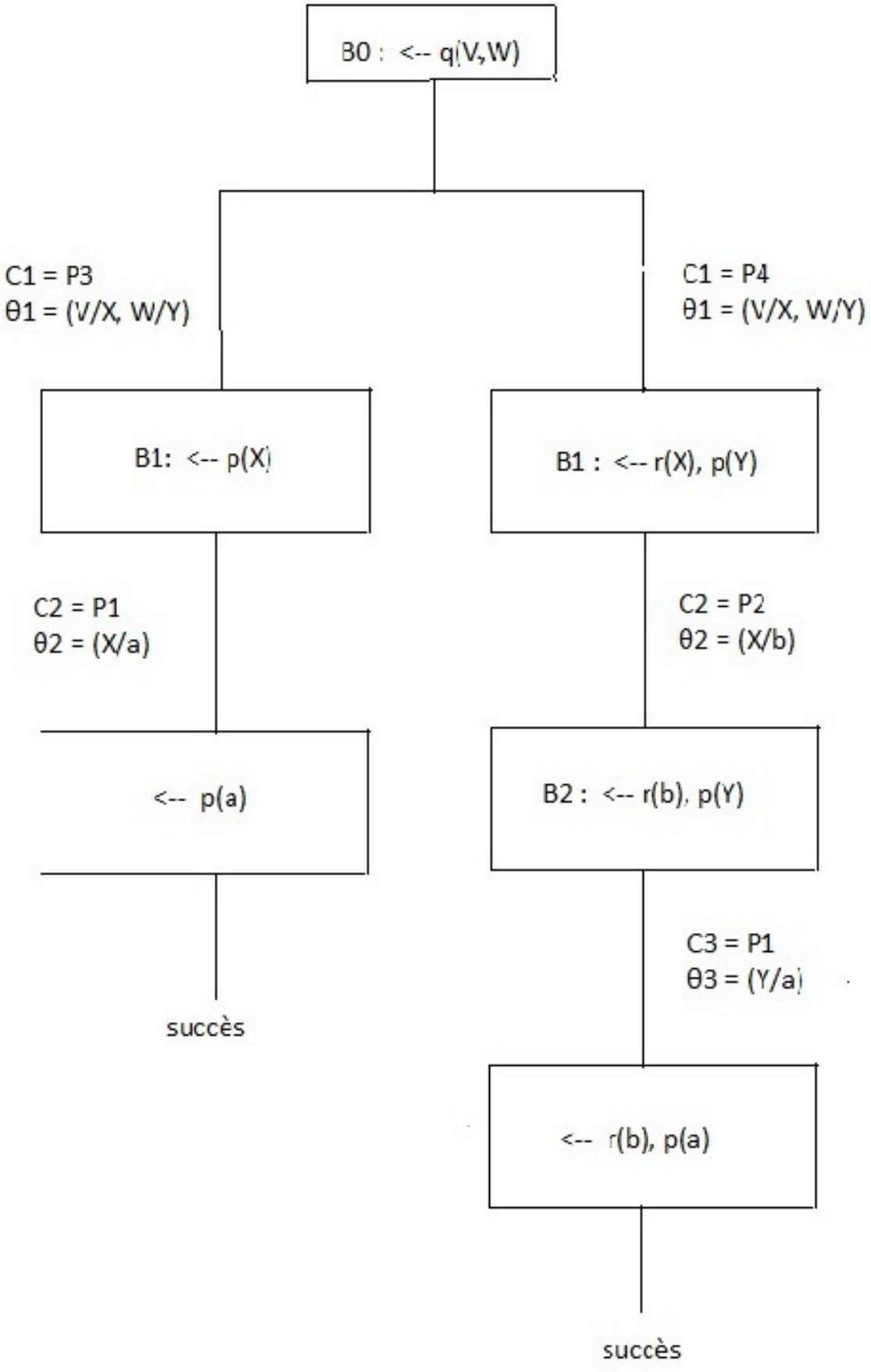

FIGURE 7.1 – Arborescence de dérivation du programme 1

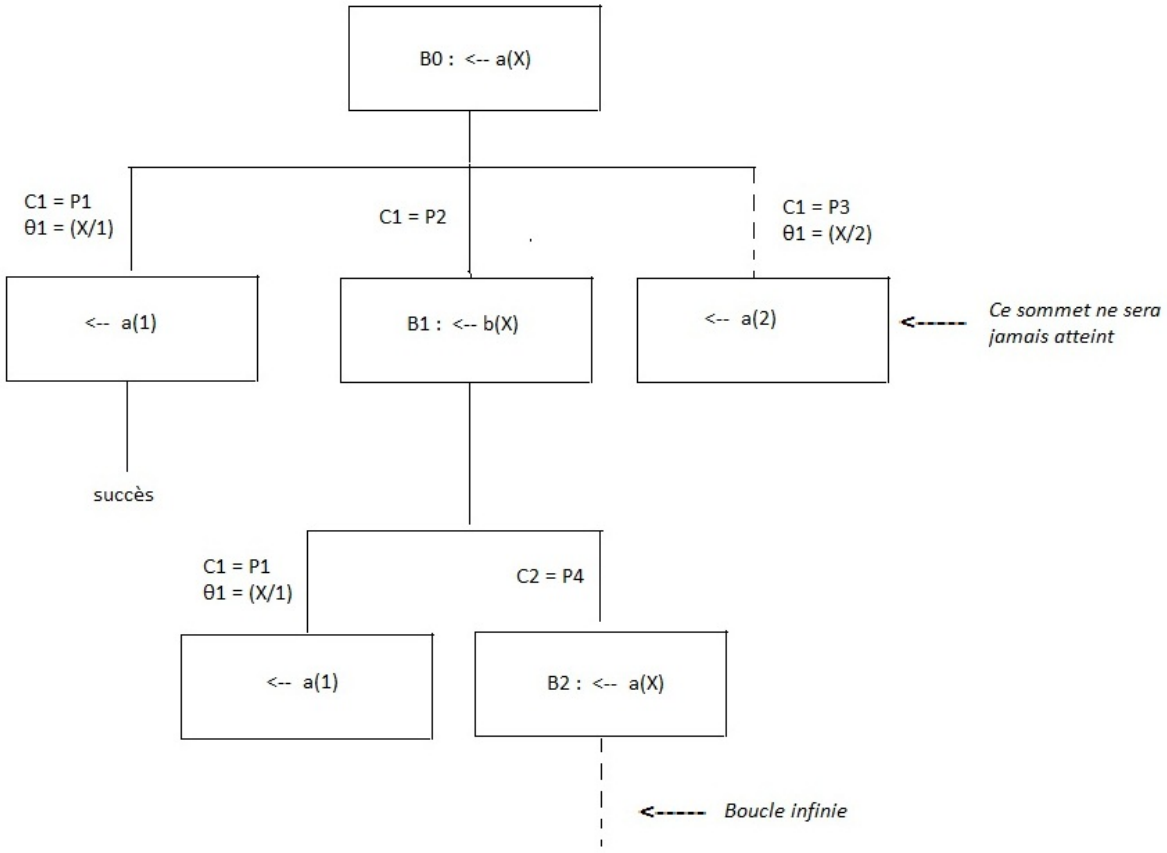

FIGURE 7.2 – Arborescence de dérivation du programme 2