Université Paris VII Année 1995-96
Licence d'Informatique
Enseignement de Logique 1^{er} semestre

Projet

Le but de ce projet est d'implanter la méthode des arbres de Beth qui est une méthode pour décider de la validité de formules. On ne s'intéresse ici qu'au calcul propositionnel.

Description de la méthode

Le principe de la méthode est le suivant : partant d'une formule F que l'on cherche à falsifier (ou à satisfaire), on se ramène, étape par étape, à devoir falsifier (ou à satisfaire) des sous-formules toujours plus petites de F. Lorsqu'on a ramené le problème à la falsifiabilité (ou la satisfiabilité) uniquement d'atomes, on s'arrête et on regarde s'il est possible d'attribuer à ces atomes des valeurs de vérités adéquates.

Les règles de falsification sont les suivantes :

- Pour falsifier $C \wedge D$, if faut soit falsifier C, soit falsifier D
- Pour falsifier $C \vee D$, if faut falsifier C *et* falsifier D.
- Pour falsifier $C \Rightarrow D$, if faut satisfaire C *et* falsifier D.

Les règles de satisfaction sont les suivantes :

- Pour satisfaire $C \wedge D$, if faut satisfaire C *et* satisfaire D.
- Pour satisfaire $C \vee D$, if faut soit satisfaire C, soit satisfaire D
- Pour satisfaire $C \Rightarrow D$, if faut soit falsifier C, soit satisfaire D.

A partir de ces règles de falsification et de satisfaction, et partant de F, à falsifier ou à satisfaire, on construit un arbre. Supposons par exemple que l'on cherche à falsifier F.

S'il y a deux manières de falsifier F (par exemple si F a la forme $C \wedge D$), alors on construit un embranchement de l'arbre, avec, notées sur chaque sous-arbre, les nouvelles formules à falsifier (ici, C sur un des sous-arbres, D sur l'autre). S'il n'y a qu'une manière de falsifier F (par exemple si F a la forme $C \vee D$), on construit un arbre avec juste un unique sous-arbre sur lequel on note les formules restant à falsifier (ici C et D). Et on recommence le procédé de falsification sur le (ou les) sous-arbre(s) que l'on vient de construire.

Lorsqu'une formule à falsifier a la forme $C \Rightarrow D$, on est amené à devoir, cette fois, satisfaire une formule (ici C). Ainsi, on se retrouve, à chaque étape, à devoir falsifier certaines formules et à devoir satisfaire d'autres. Dans la suite, on associera aux nœuds de l'arbre des notations de la forme $A_1, ..., A_p$ `vrai` | $B_1, ..., B_q$ `faux`. Cela signifiera qu'il faut falsifier B_1,...,B_q et satisfaire A_1,...,A_p.

Puisqu'à chaque étape de la méthode, on se retrouvera avec des nouveaux sous-arbres sur lesquels on aura indiqué zéro, une ou plusieurs formules à falsifier, et zéro, une ou plusieurs formules à satisfaire, il faudra faire un choix de la formule sur laquelle on va poursuivre l'algorithme. De même, lorsqu'il y aura des embranchements, il faudra faire un choix de quel sous-arbre on regarde d'abord. Mais, de toutes façons, puisqu'à chaque étape on décompose une nouvelle formule, au bout d'un temps fini, on n'aura plus que des atomes.

Que se passe-t-il si on débouche sur un sous-arbre n'ayant que des formules atomiques à satisfaire ou à falsifier. Deux cas peuvent se présenter. Le premier cas arrive lorsqu'aucun des atomes n'est à la fois à falsifier et à satisfaire. Par conséquent, on est capable d'exhiber une assignation des atomes telle que la formule F de départ soit falsifiable. Dans le deuxième cas, au contraire, il y a un même atome à rendre à la fois vrai et à la fois faux. C'est un échec : la branche en question ne peut fournir d'assignation des atomes qui permette de falsifier la formule F de départ. Il reste à explorer les autres branches, voir si, elles, permettent de trouver une assignation falsifiante de F. Et si, en fin de compte, aucune des branches ne permet de trouver une assignation falsifiante de F, c'est que F est valide.

Exemple 1 : $(A \wedge B) \Rightarrow (B \wedge A)$ est-elle valide ?

On construit de proche en proche un arbre de Beth :

a. Initialisation :

$$\mid (A \wedge B) \Rightarrow (B \wedge A) \ \texttt{faux}$$

qui signifie : il faut falsifier $(A \wedge B) \Rightarrow (B \wedge A)$.

b. Décomposition de $\Rightarrow$:

$$\frac{A \wedge B \ \texttt{vrai} \mid B \wedge A \ \texttt{faux}}{\mid (A \wedge B) \Rightarrow (B \wedge A) \ \texttt{faux}}$$

qui signifie : il faut satisfaire $A \wedge B$ et falsifier $B \wedge A$.

c. On choisit par exemple de décomposer $B \wedge A$

$$\frac{\dfrac{A \wedge B \ \texttt{vrai} \mid B \ \texttt{faux} \quad A \wedge B \ \texttt{vrai} \mid A \ \texttt{faux}}{A \wedge B \ \texttt{vrai} \mid B \wedge A \ \texttt{faux}}}{\mid (A \wedge B) \Rightarrow (B \wedge A) \ \texttt{faux}}$$

qui signifie : il faut soit continuer à satisfaire $A \wedge B$ tout en falsifiant B, soit continuer à satisfaire $A \wedge B$ tout en falsifiant A

d. On choisit par exemple de décomposer $A \wedge B$ à gauche :

$$\frac{\dfrac{\dfrac{A, B \ \texttt{vrai} \mid B \ \texttt{faux}}{A \wedge B \ \texttt{vrai} \mid B \ \texttt{faux}} \quad A \wedge B \ \texttt{vrai} \mid A \ \texttt{faux}}{A \wedge B \ \texttt{vrai} \mid B \wedge A \ \texttt{faux}}}{\mid (A \wedge B) \Rightarrow (B \wedge A) \ \texttt{faux}}$$

qui signifie : il faut, sur cette branche, satisfaire A et B tout en falsifiant B. C'est évidemment impossible : la branche est un échec.

e. Il reste à essayer la décomposition de $A \wedge B$ à droite :

$$\frac{\dfrac{\dfrac{A, B \ \texttt{vrai} \mid B \ \texttt{faux}}{A \wedge B \ \texttt{vrai} \mid B \ \texttt{faux}} \quad \dfrac{A, B \ \texttt{vrai} \mid A \ \texttt{faux}}{A \wedge B \ \texttt{vrai} \mid A \ \texttt{faux}}}{A \wedge B \ \texttt{vrai} \mid B \wedge A \ \texttt{faux}}}{\mid (A \wedge B) \Rightarrow (B \wedge A) \ \texttt{faux}}$$

là aussi, c'est un échec. Il n'y a alors plus de branches à explorer. Aucune branche ne permet donc de falsifier $(A \wedge B) \Rightarrow (B \wedge A)$. Cette dernière est donc une formule valide.

Exemple 2 : $(A \vee B) \Rightarrow (A \wedge B)$ est-elle valide ?

Après quelques étapes (et quelques choix) on tombe sur l'arbre

$$\dfrac{\dfrac{\dfrac{A\ \texttt{vrai} \mid B\ \texttt{faux} \quad B\ \texttt{vrai} \mid B\ \texttt{faux}}{A \vee B\ \texttt{vrai} \mid A\ \texttt{faux}} \qquad A \vee B\ \texttt{vrai} \mid B\ \texttt{faux}}{A \vee B\ \texttt{vrai} \mid A \wedge B\ \texttt{faux}}}{\mid (A \vee B) \Rightarrow (A \wedge B)\ \texttt{faux}}$$

Pour le sous-arbre le plus à gauche, celui annoté par A `vrai` $\mid B$ `faux`, on obtient une assignation invalidante en assignant `vrai` à A et `faux` à B. Par construction de l'arbre, on déduit que cette même assignation falsifie $(A \vee B) \Rightarrow (A \wedge B)$ qui n'est donc pas valide.

La détermination des règles de falsification et de satisfaction pour les formules 1 (formule toujours vraie) et 0 (formule toujours fausse) ainsi que celles pour les connecteurs $\neg$ et $\Leftrightarrow$ reste à la charge des étudiants (de même que l'application de la méthode pour décider si une formule est contradictoire).

Description du projet

Ecrire, par groupe de 2 ou 3, un programme offrant un menu permettant d'entrer des formules, soit interactivement, soit à partir d'un fichier, et qui déterminera, par la méthode des arbres de Beth, si les formules rentrées sont valides ou falsifiables, contradictoires ou satisfiables (dans les cas où une formule est simultanément falsifiable et satisfiable, une assignation pour chaque cas devra être exhibée).

Le programme devra être à même de comprendre des formules construites conformément à cette grammaire :

$$A \;::=\; \texttt{a} \mid \ldots \mid \texttt{z} \mid \texttt{0} \mid \texttt{1} \mid \texttt{\~{}}A \mid (A/\backslash A) \mid (A\backslash/A) \mid (A => A) \mid (A <=> A)$$

où les lettres `a`,...,`z` servent de noms pour les variables propositionnelles (et `~` sert pour la négation).

Le programme devra tourner, lors de la soutenance, sur une des machines locales mais le langage de programmation est laissé au choix : C ou Pascal conviendront mais l'utilisation d'un dialecte du Lisp (Scheme par exemple) ou d'un dialecte de ML (Caml-Light par exemple) s'avèrera sans doute être un choix fructueux pour ceux qui prendront la peine de s'y intéresser.

Le projet comportera de plus un dossier avec :

- Une *notice d'utilisation* du programme.
- Un *rapport* sur l'ensemble du projet avec, en particulier, toutes les définitions logiques requises.
- Un *listing des différentes parties du projet*, comportant suffisamment de commentaires.
- Un *listing d'exécution* sur des exemples **caractéristiques** illustrant les différentes réponses que le programme peut apporter. Ce listing d'exécution est indispensable et le fichier d'exemples correspondant doit être pris parmi ceux proposés dans les cours, travaux dirigés, polycopiés et examens des années précédentes.

On attachera la plus grande importance à la présentation du dossier et à la présentation orale du projet (qui devra être préparée !).