Projet de Programmation : Partie OCaml

Université Paul Sabatier, Toulouse
Module: Projet de Programmation L3
Projet Java/OCaml 2012/2013
Sujet v1 - 29/11/2012 - ajoût détails partie III

INTRODUCTION[1]

Le calcul des propositions repose sur des règles de formation indiquant comment construire des propositions (formules) complexes à partir des propositions élémentaires, ou atomes, que sont les variables propositionnelles, et d'éventuelles constantes comme ⊥. Ces règles déterminent quels assemblages de signes, quels mots, sont bien formés et désignent des formules. La définition dépend d'un choix de connecteurs, et d'un choix d'atomes.
On se donne un ensemble $\mathcal{P}$ de variables propositionnelles. L'ensemble des formules du calcul des propositions (sur $\mathcal{P}$), ou simplement des formules, est défini par induction, c'est-à-dire que c'est le plus petit ensemble tel que :

- un atome *p* de $\mathcal{P}$ est une formule ;
- ⊥ est une formule ;
- ⊤ est une formule ;
- si *P* et *Q* sont des formules, alors (*P* ∧ *Q*), (*P* ∨ *Q*) et (¬*P)* sont des formules.

Exemples : Si *P*, *Q* et *R* sont des formules,

P ∧ (¬*P*) est une formule.

(*P* ∧ *Q*) ∨ *R* est une formule.

P ∧ *Q* ∨ n'est pas une formule.

L'interprétation d'une formule *A* est le fait d'attribuer une valeur de vérité (`false` ou `true`) aux atomes et aux constantes (⊥ vaut `false`, ⊤ vaut `true`). Il existe quatre cas d'interprétation:

- Quand la formule prend toujours la valeur `false` quelles que soient les valeurs des atomes, la formule est dite être une **antilogie** ou une **contradiction**. On dit également qu'elle est **insatisfaisable**.
- Lorsque la formule *A* prend toujours la valeur `true`, *A* est une **tautologie**. On dit aussi que *A* est **valide**.
- Si la formule prend au moins une fois la valeur `true`, on dit que l'on peut satisfaire *A*, ou que *A* est **satisfaisable** (ou encore "satisfiable" par mimétisme avec le terme anglais).
- Si la formule prend au moins une fois la valeur `true` et au moins une fois la valeur `false`, c'est une formule synthétique ou **contingente**.

[1]Extraits adaptés de “Calcul des propositions” du site Wikipédia

PRÉSENTATION DU SUJET

Vous devez réaliser un programme Ocaml qui devra pouvoir représenter l'ensemble des formules du calcul de propositions (sur $\mathcal{P}$) et dire si une formule est une antilogie, si elle est valide, si elle est satisfaisable et/ou contigente. $\mathcal{P}$ sera représenté par des caractères.
Vous devez également simplifier une formule. Cette partie sera en relation avec la partie Java.
On représente une interprétation d'une formule par une liste de valeurs de vérité (`true`/`false`) dont la longueur est le nombre d'atomes de la formule.

Exemple : Pour la formule F = (a ∨ b) ∧ (a ∨ c) , la liste ['a';'b';'c'] représente l'ensemble des atomes de F. La liste [`true`;`false`;`true`] représente une interprétation où les éléments de la liste sont les valeurs respectives des atomes 'a', 'b' et 'c'.

TRAVAIL DEMANDÉ[2]

PARTIE I : (Représenter une formule et tester sa propriété)

1. Type formule

- Définir le type **`formule`** qui représente une formule telle que définie dans l'introduction. Le domaine des atomes (sur $\mathcal{P}$) sera l'ensemble des caractères :pour simplifier, on utilisera le type `Char`.
- Donner une dizaine d'exemples de valeur du type `formule` (jeu de test sur lequel vous pourrez travailler par la suite)

2. Valeur d'une formule dans une interprétation

- Définir une fonction **`valeur`** qui, étant donnés l'ensemble des atomes d'une formule, l'un de ces atomes et une interprétation, retourne la valeur de vérité associée à l'atome dans l'interprétati

  Exemple : la valeur de vérité de 'a' parmi l'ensemble ['a';'b';'c'] dans l'interprétation [`true;false;true`] est `true`. Celle de 'b' est `false`.
- Définir les fonctions des tables de vérités des connecteurs : ∧, ∨ et ¬.

  Exemple : `et : bool -> bool -> bool`
- Définir une fonction **`applique`** qui à partir de la liste des atomes, d'une interprétation et d'une formule retourne la valeur de la formule dans cette interprétation.

  Exemple : A partir de l'ensemble ['a';'b';'c'] appliquer l'interprétation [`true;false;true`] à la formule F = (a ∨ b) ∧ (a ∨ c) renvoie `true`. Pour l'interprétation [`false;false;true`] cela renvoie `false`.

3. Construire les interprétations

---

[2] Merci à l'équipe pédagogique L3 Ocaml

- Définir les fonctions **`appartient`** et **`union`** sur les listes qui sont des ensembles d'atomes.
- Définir une fonction **`atomes`** qui construit l'ensemble des atomes qui composent une formule.

  Exemple : Pour la formule F = (a ∨ b) ∧ (a ∨ c) , la liste ['a';'b';'c'] représente l'ensemble des atomes de F.
- Définir une (simple) fonction **`interpretation_zero`** qui construit pour l'ensemble des atomes d'une formule donnée, une première interprétation constituée uniquement de la valeur de vérité `false`.
- Définir la fonction **`interpretation_suivante`** qui permet de passer d'une interprétation quelconque à une autre interprétation dite "suivante" (dans le sens "la ligne suivante dans la table de vérité").

  Exemple : l'interpretation suivante de
  - [`true;false;false;true`] vaut [`false;true;false;true`]
  - [`true;true;true;true`] vaut [`false;false;false;false`]

  Cette opération équivaut à une addition binaire, dans laquelle on placera (pour simplifer) les bits de plus faibles poids à gauche. L'utilisation répétée de cette fonction permettra de construire (si nécessaire) toutes les interprétations d'une formule.

4. Propriétés d'une formule

- Définir les fonctions **`satisfaisable`**, **`contingente`**, **`antilogie`** et **`tautologie`** qui détermine si une formule possède de telles propriétés.

  Exemple :
  - a ∨ b est satisfaisable et contingente
  - a ∨ ¬a est une tautologie
  - a ∧¬a est une antilogie

PARTIE II : simplification

Vous devez définir une fonction qui simplifie une formule à l'aide des théorèmes suivants :

○ P ∨ ⊤ = ⊤ ○ P ∧ ⊥ = ⊥ ○ ¬⊤ = ⊥ ○ ¬⊥ = ⊤ ○ P ∨ P = P ○ P ∧ P = P	○ P ∨ ¬P = ⊤ ○ P ∧ ¬P = ⊥ ○ P ∨ Q = Q ∨ P ○ P ∧ Q = Q ∧ P ○ ...

Attention, les 2 dernières simplifications risquent d'engendrer une récursion infinie. A vous de proposer des solutions pour simplifier au maximum en utilisant le plus de théorèmes possibles.

PARTIE III : lien avec Java

Afin de faire le lien avec la partie Java, vous ajouterez dans votre programme ocaml les fonctions suivantes :

- un ensemble de fonctions "constructrices" qui fabriquent des formules. Voici leurs signatures :

  fT : formule
  fF : formule
  fNon : formule -> formule
  fEt : formule -> formule -> formule
  fOu : formule -> formule -> formule
  fImp fEq, de même signature, devront être définies également même si vous ne l'avez pas prévu. Vous pouvez les exprimer à partir des autres opérateurs logiques (et, ou...).

- les fonctions pour réaliser les vérifications demandées. ftauto appelle votre fonction de vérification de tautologie. fsimpl appelle votre fonction de simplification.

  ftauto : formule -> bool
  fsimpl : formule -> formule

- et une fonction pour l'affichage "lisible" de la formule :

  faffiche : formule -> string

Nous vous fournissons un analyseur qu'il faudra recompiler avec votre programme. Pour que ça fonctionne, il faut aussi :

- renommer le nom de type de votre formule par “formule”
- renommer le nom de votre principal fichier par “projet.ml”
- recompiler à l'aide de la commande “make”

Vous pouvez tester ensuite le programme "analyse" avec une chaîne de caractères en paramètre TERMINEE par '$'.

Exemples d'appels :

./analyse "a or ((b and 0) or 1)$" doit vous donner :
Tautologie: true
Simplification : 1

./analyse "a or ((b and 0) or c)$" doit vous donner :
Tautologie: false
Simplification : (a or c)

Pour les “trinômes”

Ajouter les connecteurs ($P \rightarrow Q$) et ($P \leftrightarrow Q$). Le bonus de simplification s'applique aussi. Les théorèmes correspondants :

○ $\bot \rightarrow P = \top$ ○ $\top \rightarrow P = P$ ○ $P \rightarrow \neg P = \neg P$	○ $P \rightarrow \top = \top$ ○ $P \rightarrow \bot = \neg P$ ○ ...

ORGANISATION

Organisation du travail :

- Vous devrez déposer un ou plusieurs fichiers Ocaml (.ml) qui regroupent le travail demandé sous moodle à la date limite correspondant à votre groupe.
  Attention, vos fichiers doivent respecter les extensions demandées et commencer par vos noms de famille par ordre alphabétique (exemple : Nom1Nom2Rapport.pdf).
  Pensez à commenter votre code afin qu'il soit lisible par le correcteur. Le programme doit compiler sous peine d'une note nulle.
- Vous pouvez fournir l'ocamldoc
- Lors de la validation, vous devrez
  - présenter un programme qui fonctionne,
  - pouvoir modifier votre programme à la demande,
  - pouvoir justifier de vos choix et de l'efficacité de vos programmes,
  - pouvoir tester nos formules
- Vous êtes en binôme ou trinôme, mais notés aussi individuellement, pensez à vous répartir astucieusement le travail.

Organisation dans le temps

- Distribution du sujet le 16 novembre
- Séance de questions/réponses la semaine du 26 novembre en groupe de TP
- Séance de validation la semaine du 10 décembre en groupe de TP

Organisation des ressources

Vous devez utiliser les ressources qui sont à votre disposition pour vous permettre de mener à bien ce projet (par ordre de priorité) : supports de cours et site Ocaml, forum, messagerie du responsable "projet" de votre groupe, de vos TP, de cours.

ÉVALUATION

La note du module est composée : d'une note sur le travail Ocaml, d'une note sur le travail Java. La note de chaque partie prend en compte non seulement la réalisation des programmes

demandés, mais aussi la qualité de la programmation (commentaires, ocaml ou javadoc, efficacité des calculs...). Il vous sera demandé un rapport de synthèse sur le travail Java.

PLAGIAT/TRAVAIL COLLECTIF

Toute fraude, même partielle sera sanctionnée par des notes de groupe ET individuelle nulles (et éventuellement par un recourt au conseil de discipline). En cas de travail collectif entre différents groupes de projet, les points des portions collectives pourront être partagés ou attribués entièrement SI le travail est réellement collectif et uniquement si cela a été signalé aux encadrants du projet. Rappel : il est illégal d'utiliser des sources sans en citer les auteurs.