

# Examen Programmation I (2016-17)

Support de cours papier autorisé.

## 1 Un peu de cambouis pour commencer

Un programmeur, JO, a écrit un programme en C, dont voici un extrait :

```
char *config_file = NULL;

int main(int argc, char *argv[])
{
    parse_cmdline(argc, argv);
    launch ();
    exit (0);
}

void parse_cmdline(int argc, char **argv)
{
    char opt;

    while ((opt = getopt(argc, argv, "hc:o:b:f:r:d:Dv:m")) != -1) {
        switch (opt) {
            case 'h':
                usage(argv[0]);
                exit(1);
                break;
            case 'c':
                config_file = optarg;
                break;
            /* autres cas (o, b, f, r, d, etc.) omis */
            default:
                usage(argv[0]);
                exit(1);
                break;
        }
    }
}

void usage (char *cmd)
```

```

{
  fprintf (stderr, "%s -h produces this help\n"
          "%s -c <file> launches %s with configuration file <file>.\n",
          cmd, cmd, cmd);
}

```

Je ne donne pas le code de la fonction `launch` : c'est celle qui accomplit la tâche du programme. La fonction `parse_cmdline` a pour but de lire les options éventuellement fournies au programme : `-h` demande à afficher une aide sommaire (ce que fait `usage`), `-c` définit le nom d'un fichier de configuration, etc.

Voici un extrait de la page man de `getopt` :

```

int getopt(int argc, char * const argv[], const char *optstring);
DESCRIPTION
  The getopt() function incrementally parses a command line
  argument list argv and returns the next known option character.
  An option character is known if it has been specified in the
  string of accepted option characters, optstring.

  The option string optstring may contain the following elements:
  individual characters, and characters followed by a colon to
  indicate an option argument is to follow. For example, an
  option string "x" recognizes an option '-x', and an option
  string "x:" recognizes an option and argument '-x argument'.
  It does not matter to getopt() if a following argument
  has leading white space.

  The getopt() function returns -1 when the argument list is
  exhausted. The interpretation of options in the argument list
  may be cancelled by the option '--' (double dash) which causes
  getopt() to signal the end of argument processing and return -1.
  When all options have been processed (i.e., up to the first
  non-option argument), getopt() returns -1.

```

1. Le programme de JO, `joe`, fonctionne pendant des années sans problème. Il tourne sur différentes variantes de Linux (Debian, Ubuntu, Fedora, etc.), sous Mac OS X, et sous Windows. Fin 2016, un industriel demande à évaluer le programme, il le compile sous ELinOS, un Linux embarqué. Il le lance en tapant `joe` sous un shell. Le programme s'arrête immédiatement, sans rien faire, après avoir affiché :

```

joe -h produces this help
joe -c <file> launches joe with configuration file <file>.

```

Expliquer le bug. En particulier, selon vous : qu'est-ce qui a fait que le programme bugge sous ELinOS ? pourquoi fonctionnait-il sous les autres systèmes d'exploitation ?

## 2 Le dcpo $\mathbb{P}(A)$ , dcpos algébriques

Dans cette partie, on fixe un ensemble  $A$  quelconque.

2. Montrer que  $\mathbb{P}(A)$  est un treillis complet. Que vaut la borne supérieure  $\sup E_i$  d'une famille  $(E_i)_{i \in I}$  arbitraire de  $\mathbb{P}(A)$  ?

3. Pourquoi  $\mathbb{P}(A)$  est-il un dcpo ?
4. Un élément  $E$  d'un dcpo  $X$  est dit *compact* si et seulement si, pour toute famille dirigée  $(E_i)_{i \in I}$  dans  $X$  telle que  $E \leq \sup_{i \in I} E_i$ , il existe un  $i \in I$  tel que  $E \leq E_i$ . On note  $\mathbb{F}(A)$  l'ensemble des parties finies de  $A$ . Montrer que tout élément de  $\mathbb{F}(A)$  est compact dans le dcpo  $\mathbb{P}(A)$ .
5. Un dcpo  $X$  est dit *algébrique* si et seulement si tout élément  $E$  de  $X$  est la borne supérieure d'une famille dirigée  $(E_i)_{i \in I}$  d'éléments  $E_i$  compacts inférieurs ou égaux à  $E$ . Montrer que  $\mathbb{P}(A)$  est algébrique.
6. Montrer que les éléments compacts de  $\mathbb{P}(A)$  sont exactement les éléments de  $\mathbb{F}(A)$ .
7. On rappelle le dcpo  $\mathbb{IR}$  des intervalles  $[a, b]$  de réels,  $a \leq b$ , ordonné par inclusion inverse ( $[a, b] \leq [c, d]$  ssi  $[a, b] \supseteq [c, d]$  ssi  $a \leq c$  et  $b \geq d$ ). On rappelle aussi que le sup d'une famille dirigée  $([a_i, b_i])_{i \in I}$  dans ce dcpo est  $\bigcap_{i \in I} [a_i, b_i] = [\sup_{i \in I} a_i, \inf_{i \in I} b_i]$ . Quels sont les éléments compacts de  $\mathbb{IR}$  ?
8.  $\mathbb{IR}$  est-il algébrique ? Justifier.

### 3 Prolog

On considère le langage suivant, qui est un sous-ensemble de Prolog. On rappelle la notion de terme, vue en cours dans le cadre du système de typage de ML. L'ensemble  $\mathcal{T}(\Sigma, X)$  des termes  $s, t, u, v, \dots$  est défini inductivement par : les variables  $x, y, z, \dots$  de  $X$  sont des termes de  $\mathcal{T}(\Sigma, X)$ , et les *applications*  $f(t_1, \dots, t_n)$  de  $f$  (avec  $f/n$  dans la signature  $\Sigma$ ) à  $n$  termes  $t_1, \dots, t_n$  de  $\mathcal{T}(\Sigma, X)$  sont dans  $\mathcal{T}(\Sigma, X)$ .

On supposera dans la suite que  $\Sigma$  contient au moins une constante  $a/0$ . L'ensemble  $\mathcal{T}(\Sigma)$  des *termes clos*, défini comme égal à  $\mathcal{T}(\Sigma, \emptyset)$ , est alors non vide.

Un *atome*  $A$  est juste une application  $p(t_1, \dots, t_n)$ , et  $p$  s'appelle le *prédicat* de l'atome. On notera  $\Pi$  l'ensemble des prédicats. Une *clause définie* est une expression de la forme  $A :- A_1, \dots, A_m$ . où  $A, A_1, \dots, A_m$  sont des atomes. (L'entier  $m$  peut être nul, auquel cas on écrira juste  $A$ . : ce genre de clause définie s'appelle un *fait*.)

Un *programme Prolog* est juste une liste finie de clauses définies. On peut lire un programme Prolog de deux façons : soit comme une axiomatisation logique, où une clause définie comme ci-dessus se lira «  $A_1$  et  $\dots$  et  $A_m$  impliquent logiquement  $A$  » ; soit comme un programme exécutable.

Par exemple, le programme Prolog suivant :

```
append(nil,L,L).
append(cons(X,L),L',cons(X,L')) :- append(L,L',L').
```

définit logiquement `append` de sorte que `append( $\ell_1, \ell_2, \ell_3$ )` soit vrai si et seulement si  $\ell_3$  est la concaténation des listes  $\ell_1$  et  $\ell_2$ . (Par convention les fonctions de  $\Sigma$  sont les noms commençant par une minuscule, et les variables commencent par une majuscule.) On peut l'utiliser pour *calculer* la concaténation de listes : si l'on entre le programme Prolog ci-dessus dans, par exemple, SWI-Prolog, et que l'on tape :

```
? append(cons(1,nil), cons(2,cons(3,nil)), Resultat).
```

SWI-Prolog va répondre :

yes

Resultat = cons(1,cons(2,cons(3,nil)))

Dans cette partie, on se fixe un programme Prolog  $P$ .

On appellera *environnement*  $\rho$  toute fonction des variables vers les termes clos. On définit l'application  $t\rho$  d'un environnement  $\rho$  à un terme  $t$  exactement pour pour les substitutions. En général, on manipulera les environnements exactement comme les substitutions.

On notera que  $t\rho$  est toujours un terme clos. On étendra la notion d'application de substitutions et d'environnements à tout atome, toute clause, de la façon évidente.

Pour tout  $E \in \mathbb{P}(\mathcal{T}(\Sigma))$ , définissons un ensemble  $T_P(E)$  comme suit. Voici d'abord l'idée intuitive.  $E$  est typiquement l'ensemble des atomes clos dont on sait qu'ils sont vrais, parce qu'ils ont été déduits en, disons,  $k$  étapes à partir des faits de  $P$ .  $T_P(E)$  est l'ensemble des atomes clos déduits de ceux de  $E$  à l'aide d'une clause définie de  $P$ , en une étape; si  $E$  est comme ci-dessus,  $T_P(E)$  sera donc l'ensemble des atomes clos déduits en  $k+1$  étapes à partir des faits de  $P$ . Formellement (et seule cette définition compte pour le raisonnement!) :

$$T_P(E) = \{A\rho \mid (A :- A_1, \dots, A_n) \in P, \\ \rho \text{ environnement tel que } A_1\rho, \dots, A_n\rho \in E\}.$$

9. Montrer que  $T_P$  est Scott-continue.

10. Pourquoi le plus petit point fixe de  $T_P$  existe-t-il?

On le notera  $\text{lfp}(T_P)$ , et c'est, intuitivement, l'ensemble des atomes clos *vrais* dans l'interprétation logique de  $P$ . C'est aussi, de façon plus banale, la sémantique dénotationnelle du programme Prolog  $P$ .

Etant donné un *but*, c'est-à-dire un atome  $B$ , pas nécessairement clos, précédé du symbole ? (comme ? `append(cons(1,nil), cons(2,cons(3,nil)), Resultat)`), avec une variable `Resultat`), Prolog va chercher des valeurs des variables dans  $B$  rendant  $B$  vrai. Formellement, Prolog va chercher s'il existe un environnement  $\rho$  tel que  $B\rho \in \text{lfp}(T_P)$  (et en retourner un, restreint aux variables de  $B$ , s'il en existe un).

On note  $\text{mgu}(A \doteq B)$  l'unificateur le plus général (le mgu) de l'équation  $A \doteq B$ . On a vu la notion en cours pour des équations entre termes. Ici il s'agit d'équations entre atomes : l'extension aux atomes est triviale.

11. Supposons qu'aucune variable n'apparaisse à la fois dans  $B$  et dans  $P$ .

Montrer que si  $B\rho \in \text{lfp}(T_P)$ , alors : (i) il existe une clause définie  $A :- A_1, A_2, \dots, A_n$  dans  $P$  telle que  $B \doteq A$  soit unifiable; (ii) en posant  $\theta = \text{mgu}(B \doteq A)$ ,  $B\rho$  s'écrit  $B\theta\rho'$  pour un certain environnement  $\rho'$ ; (iii)  $A_1\theta\rho', \dots, A_n\theta\rho'$  sont tous dans  $\text{lfp}(T_P)$ .

Pour une clause définie  $C$  de  $P$  et un atome  $B$ , on appellera *renommage de  $C$  loin de  $B$*  une clause de la forme  $C\rho$ , où  $\rho$  est un renommage (une substitution qui envoie les variables vers des variables, et qui est injective), de sorte qu'il n'y ait aucune variable en commun entre  $C\rho$  et  $B$ .

On définit une sémantique opérationnelle de Prolog comme suit. Une configuration  $C$  de cette sémantique est un multi-ensemble fini de buts. L'unique règle, dite de *résolution*, est :

$$C \cup \{?B\} \xrightarrow{\theta} C\theta \cup \{?A_1\theta, \dots, ?A_n\theta\}$$

où  $A :- A_1, A_2, \dots, A_n$  est un renommage d'une clause définie dans  $P$  loin de  $B$ , et  $\theta = \text{mgu}(B \doteq A)$ . (Oui, la sémantique est non déterministe : on ne précise pas comment choisir

le but  $?B$  dans la configuration.) Noter qu'on a pris soin d'étiqueter la transition par le mgu  $\theta$  : ce sera parfois pratique dans la suite. On notera  $C \rightarrow C'$  pour dire que  $C \xrightarrow{\theta} C'$  pour une certaine substitution  $\theta$ , si elle n'est pas importante. Ceci permet de définir la clôture réflexive-transitive  $\rightarrow^*$ , notamment.

On appellera *instance* d'un but  $?A$  tout atome clos de la forme  $A\rho$ , où  $\rho$  est un environnement. On appellera  $A\rho$  plus précisément une  $\rho$ -instance de  $A$  lorsqu'on veut mentionner  $\rho$  explicitement.

12. Montrer que la règle de résolution est correcte, au sens où, pour tout environnement  $\rho$ , si toutes les  $\rho$ -instances du côté droit  $C\theta \cup \{?A_1\theta, \dots, ?A_n\theta\}$  sont dans  $\text{lfp}(T_P)$ , alors toutes les  $\theta\rho$ -instances du côté gauche  $C \cup \{?B\}$  sont aussi dans  $\text{lfp}(T_P)$ .
13. En déduire que la sémantique opérationnelle de Prolog est correcte : si  $\{?B\} \rightarrow^* \emptyset$ , alors il existe une substitution  $\theta$  telle que, pour tout environnement  $\rho$ ,  $B\theta\rho$  est dans  $\text{lfp}(T_P)$ . On construira  $\theta$  explicitement.

Dans ce cas, Prolog répond **yes** suivi de la restriction de  $\theta$  aux variables de  $B$ .

14. En adaptant la question 11, montrer que la sémantique opérationnelle de Prolog est aussi adéquate : si  $B\rho$  est une instance de  $?B$  dans  $\text{lfp}(T_P)$ , alors  $\{?B\} \rightarrow^* \emptyset$ . On montrera plus précisément que si  $B\rho$  est une instance de  $?B$  dans  $\text{lfp}(T_P)$ , alors il existe un nombre fini d'étapes  $\{?B\} = C_0 \xrightarrow{\theta_1} C_1 \xrightarrow{\theta_2} \dots \xrightarrow{\theta_m} C_m = \emptyset$  et un environnement  $\rho'$  tel que  $\rho = \theta_1\theta_2 \dots \theta_m\rho'$ .

## 4 Datalog

Datalog est la restriction de Prolog où tous les symboles de fonctions sont d'arité 0. On rappelle que pour  $c$  d'arité 0, on note préférentiellement  $c$  le terme qu'on devrait noter  $c()$ .

En Datalog, les seuls termes clos sont les constantes. Les termes soit les variables et les constantes. On supposera qu'il n'y a qu'un nombre fini de constantes.

Un tel programme Datalog est ce qu'on appelle une *base de données déductive*. (Vous comprendrez pourquoi en cours de bases de données.)

15. On rappelle que Prolog, et ce sera aussi le cas pour Datalog, a une sémantique opérationnelle non déterministe. Ceci a la conséquence curieuse suivante : il existe un programme Datalog  $P$  et un but  $?B$  qui ont (au moins) deux exécutions :

- une qui termine,  $\{?B\} \rightarrow^* \emptyset$  ;
- une qui ne termine pas,  $\{?B\} = C_0 \xrightarrow{\theta_1} C_1 \xrightarrow{\theta_2} \dots \xrightarrow{\theta_m} C_m \xrightarrow{\theta_2} \dots$

Exhiber un tel programme Datalog  $P$  et un tel but  $?B$ .

16. Montrer que pour tout programme Datalog  $P$ ,  $\text{lfp}(T_P)$  est fini et calculable. Par calculable, on entend qu'il existe un algorithme *qui termine* qui prend  $P$  en entrée et calcule  $\text{lfp}(T_P)$ .

## 5 Magic sets

On s'intéresse à la question de savoir quelles sont les instances  $B\rho$  de  $?B$  qui sont dans  $\text{lfp}(T_P)$ . L'avantage de l'algorithme suggéré en question 16, dit « algorithme en avant », est qu'il termine pour tout programme Datalog : une fois calculé  $\text{lfp}(T_P)$ , il suffit de trouver les

instances de  $?B$  qui sont dans  $\text{lfp}(T_P)$ , ce qu'on peut faire soit en énumérant bêtement toutes les instances de  $?B$  et en testant leur appartenance à  $\text{lfp}(T_P)$ , soit en unifiant successivement  $B$  à chaque élément de  $\text{lfp}(T_P)$ . L'inconvénient de l'algorithme en avant est qu'il procède complètement à l'aveuglette, et n'est pas du tout guidé par la connaissance de  $B$ .

On peut à la place partir de  $?B$ , et utiliser la question 14 : on cherche « en arrière » toutes les exécutions  $\{?B\} = C_0 \xrightarrow{\theta_1} C_1 \xrightarrow{\theta_2} \dots \xrightarrow{\theta_m} C_m = \emptyset$  qui terminent, et on retourne toutes les instances de  $B\theta_1\theta_2 \dots \theta_m$ . Ceci est bien guidé par la connaissance de  $B$ , mais ne termine pas en général.

Notre but est d'obtenir un algorithme qui offre le meilleur des deux solutions.

On se fixe un but  $?B$  et un programme Prolog  $P$ . (Nous ne nous limiterons pas à Datalog ici.) Les prédicats de  $B$  et de tous les atomes de  $P$  sont supposés être tous dans l'ensemble  $\Pi$ .

On commence par dupliquer chaque prédicat : pour chaque prédicat  $p \in \Pi$ , on crée un nouveau prédicat *magic- $p$*  hors de  $\Pi$ . Si  $p$  et  $q$  sont distincts, alors *magic- $p$*  et *magic- $q$*  sont distincts. De plus,  $p \in \Pi$  est distinct de tout nouveau prédicat *magic- $q$* ,  $q \in \Pi$ .

Pour chaque atome  $A = p(t_1, \dots, t_n)$ ,  $p \in \Pi$ , on notera *magic- $A$*  l'atome *magic- $p$* ( $t_1, \dots, t_n$ ).

Pour chaque clause définie  $C = (A :- A_1, \dots, A_m)$  de  $P$ , on appellera :

- $A :- \text{magic-}A, A_1, \dots, A_m$ . la *clause sortante* de  $C$  ;
- $\text{magic-}A_i :- \text{magic-}A$ ,  $1 \leq i \leq m$ , les *clauses entrantes* de  $C$ .

Le nouveau programme *magic- $P/B$*  est par définition l'ensemble formé des clauses entrantes et des clauses sortantes de toutes les clauses définies de  $P$ , plus le fait *magic- $B$* . Il est formé sur le langage des prédicats  $\Pi \cup \{\text{magic-}p \mid p \in \Pi\}$ .

17. Montrer que tout atome clos  $p(t_1, \dots, t_n)$  de  $\text{lfp}(T_{\text{magic-}P/B})$  tel que  $p \in \Pi$  (donc, pas de la forme *magic- $q$*  pour aucun  $q$ ) est dans  $\text{lfp}(T_P)$ .
18. Montrer que pour tout atome clos  $A = p(t_1, \dots, t_n)$  avec  $p \in \Pi$  et tel que *magic- $A$*  est dans  $\text{lfp}(T_{\text{magic-}P/B})$ , si  $A \in \text{lfp}(P)$  alors  $A \in \text{lfp}(T_{\text{magic-}P/B})$ .
19. En déduire que les instances  $B\rho$  de  $?B$  qui sont dans  $\text{lfp}(T_P)$  et celles qui sont dans  $\text{lfp}(T_{\text{magic-}P/B})$  sont les mêmes.
20. Expliquer l'intérêt de la construction sur l'exemple suivant :

$$\begin{aligned} B &= p(a)., \\ P &= \{p(a)., q(X, Z) :- q(X, Y), q(Y, Z)., q(a, b)., q(b, c)., q(c, c)., q(c, a).\}, \end{aligned}$$

où l'on a trois constantes  $a, b, c$ . (A titre d'intuition, les clauses définies portant sur le prédicat  $q$  calculent la clôture transitive du graphe d'arêtes  $a \rightarrow b, b \rightarrow c, c \rightarrow c, c \rightarrow a$ .)