Université BORDEAUX I **Année 1992-93**
Session de février

Maîtrise d'informatique

Modèles et Techniques de Programmation (J.P.Braquelaire)

Durée: 2 heures
Tous documents autorisés

Première partie *(6 points)*

On considère un interprète SCHEME utilisant une stratégie d'évaluation *normale* (et non la stratégie applicative habituelle).

1) On considère la définition suivante de la fonction *ou*:

```
(define or (lambda (x y) (if x x y)))
```

Expliquer à travers un exemple simple pourquoi cette définition n'est pas satisfaisante.

2) Une première idée pour éviter le phénomène de duplication de calcul (en particulier dans le cas d'expressions à effet de bord) est d'utiliser un environnement local. Améliorer la définition précédente au moyen d'un `let`.

3) En fait, nous allons voir que cette solution ne marche pas. On suppose tout d'abord que la forme spéciale `let` n'est pas une primitive du système, mais qu'elle est réécrite par macro-expansion: l'expression `(let ((`x_1 v_1`)` $\cdots$ `(`x_k v_k`))` $\langle expr \rangle$`)` est réécrite en

`((lambda (`x_1 $\cdots$ x_k`)` $\langle expr \rangle$`)` v_1 $\cdots$ v_k`)`

Donner la forme *macro-expansée* de la fonction `or` de la question précédente et expliquer pourquoi le problème de duplication de code n'est pas résolu.

4) Le problème de la duplication se pose, que la forme spéciale `let` soit une primitive ou une macro. En effet, dans l'expression `(let ((x e))` $\cdots$`)`, le symbole `x` est lié à l'expression `e` non encore évaluée. Proposer une solution à ce problème basée sur l'utilisation dans l'interprète de plusieurs stratégies d'évaluation. On proposera une syntaxe permettant cela.

Deuxième partie *(7 points)*

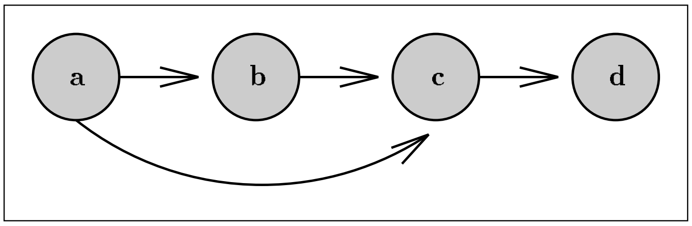
Le but de cette partie est d'exprimer l'évaluation d'une clause PROLOG en utilisant les clauses PROLOG comme formalisme. On définit pour cela le prédicat `clause`, de l'une des trois formes suivantes:

clause($\langle terme \rangle$)	:	$\langle terme \rangle$ est un fait.
clause($\langle terme \rangle_1$, $\langle terme \rangle_2$)	:	$\langle terme \rangle_1$ est la tête d'une clause, et $\langle terme \rangle_2$ est la queue de cette clause.
clause($\langle terme \rangle$, $\langle liste \rangle$)	:	$\langle terme \rangle$ est la tête d'une clause, et $\langle liste \rangle$ est la liste des termes de la queue de cette clause.

Par exemple le paquet de clauses

```
clause(arc(a,b)).
clause(arc(a,c)).
clause(arc(b,c)).
clause(arc(c,d)).
```

définit le graphe



1) Définir, au moyen de `clause`, la clause `chemin(X,Y)`, vraie s'il existe un chemin de `X` à `Y`.

2) Écrire le prédicat `prouver`, s'appliquant sur une clause (une liste de clauses) et calculant si elle est vraie. Par exemple, le calcul de `prouver(chemin(a,d)).` produit `yes`, et celui de `prouver(chemin(a,X)).` parcourt les chemins issus de `a`.

3) Que produit le calcul de `prouver(chemin(a,X))` ? (On suppose que l'on refuse chaque solution proposée par PROLOG.)

Troisième partie *(7 points)*

Le but de cette partie est d'étudier la définition d'une bibliothèque de saisie interactive de lignes de textes, permettant des fonctionnalités d'édition de type *emacs* au moyen de clés: édition de la ligne en cours de saisie (par exemple: `C-d` et `DEL` pour supprimer un caractère, `C-b`, `C-f`, `M-b`, `M-f`, `C-a`, ... pour déplacer le curseur, `C-p`, `M-<`, `C-n`, `M->`, `C-r`, `C-s`, ... pour parcourir l'historique des lignes validées, `C-m` et `C-j` pour valider la ligne, ...). On supposera que les clés sont liées à des fonctions particulières de la bibliothèque appelées *fonctions interactives*. Voici quelques unes des fonctionnalités que devra offrir cette bibliothèque:

– lecture d'une chaîne de longueur quelconque (ceci se fera en appelant une fonction `lineRead` retournant la chaîne lue);

– redéfinition des liaisons (la redéfinition est effectuée de façon statique, par le programme utilisant cette bibliothèque);

– filtrage des caractères saisis (par exemple, tous les caractères n'appartenant pas à l'alphabet `[0-9.-+]` sont refusés, et au plus huit caractères sont acceptés);

– définition de nouvelles *fonctions interactives* utilisables pour les liaisons (par exemple, conversion de toutes les minuscules en majuscules);

– ajout d'une chaîne à l'historique des commandes, et obtention de l'historique;

– ...

Donner les grandes lignes de l'implémentation en C de cette bibliothèque et le prototype (nom et type de la fonction, et type des paramètres), et la description de la mise en œuvre des principales fonctions de l'interface. Par exemple:

> **char *readline(char *prompt)** : cette fonction lit une chaîne de longueur quelconque dont elle retourne une copie allouée dynamiquement au moyen de....

```
clause(chemin(X,Y), arc(X,Y)).
clause(chemin(X,Y), [arc(X,Z), chemin(Z,Y)]).

prouver([A|B]) :- prouver(A), prouver(B).
prouver([]).
prouver(A) :- clause(A).
prouver(A) :- clause(A,B), prouver(B).

| ?- prouver(chemin(a,X)).

X = b ? ;

X = c ? ;

X = c ? ;

X = d ? ;

X = d ? ;

no
| ?-
```