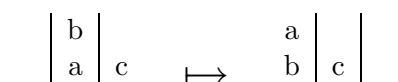


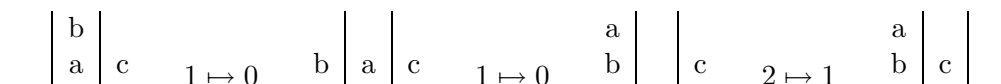
TP3 — Gestion d'entrepôt robotisée

Tous savent qu'un entrepôt servant à stocker des boîtes de toutes sortes doit être régulièrement remis en ordre afin de pouvoir s'y retrouver. En effet, les livraisons arrivant à et partant de l'entrepôt ont tendance à semer le désordre. La tâche considérée dans ce travail consiste donc à remettre de l'ordre dans un entrepôt.

Voyons donc en termes graphiques quelle est la tâche à accomplir. Tout d'abord, notons que l'entrepôt à réaménager est fait en longueur : il ne contient que des piles de boîtes les unes à côté des autres. Les boîtes sont symbolisées par des lettres. Un réaménagement consiste à transformer la disposition actuelle de l'entrepôt (dessin de gauche) en la disposition désirée (dessin de droite).



Comme notre entrepôt est très *hi-tech*, une pince robotisée fait les efforts à notre place. Celle-ci descend du plafond et est capable de saisir une boîte et de la déposer ailleurs. Nous supposons que les piles ne peuvent se trouver qu'à certains endroits bien définis. Ainsi, l'entrepôt présenté dans l'exemple plus haut comporte exactement trois emplacements pour les piles de boîtes (même si certains de ces emplacements ne contiennent aucune boîte à certains moments). Le réaménagement de l'entrepôt à l'aide de la pince se fait en mouvements élémentaires : il s'agit de déplacer une boîte d'une pile à l'autre. On n'a qu'à spécifier la pile de départ et la pile d'arrivée et la pince exécute le déplacement de la boîte concernée. Évidemment, c'est toujours la boîte du dessus de la pile de départ qui est saisie par la pince et lorsque la boîte est déposée, elle l'est sur le dessus de la pile d'arrivée. Si nous reprenons notre exemple, la séquence de mouvements suivants, $1 \mapsto 0$, $1 \mapsto 0$, $2 \mapsto 1$, permet de faire le réaménagement désiré. Notez que les piles sont numérotées à partir de 0 (zéro).



Le travail que vous devez faire en Prolog consiste à écrire deux relations reliées au réaménagement d'entrepôt. Il s'agit des relations `'verif'` et `'solve'`. Dans les deux cas, ce sont des relations ternaires qui contiennent des triplets `'(E1,S,E2)'` tels que `'E1'` est la disposition actuelle de l'entrepôt, `'E2'` est la disposition désirée et `'S'` est une séquence de mouvements qui permet de passer de `'E1'` à `'E2'`. La différence entre les deux relations vient du fait que `'verif'` sert à vérifier qu'une séquence permet de passer de la disposition actuelle

à la disposition désirée. On l'utilise avec ses deux premiers arguments fixés (ne contenant pas de variable). Le troisième peut être une variable ou non. Tandis que la relation 'solve' sert à découvrir une séquence à partir des deux dispositions d'entrepôts. On l'utilise donc avec les premier et dernier arguments fixés et avec une variable comme deuxième argument.

Voyons la représentation Prolog des données pour notre problème. Un mouvement est représenté à l'aide d'une structure 'move(*i*,*j*)', où *i* est le numéro de la pile de départ et *j* est le numéro de la pile d'arrivée. Une séquence de mouvements est représentée à l'aide d'une liste de structures. Ainsi, la séquence de mouvements effectuée dans notre exemple s'écrit :

```
[move(1,0),move(1,0),move(2,1)]
```

Les boîtes sont représentées par des atomes Prolog (typiquement une simple lettre minuscule). Des boîtes représentées par le même atome sont considérées interchangeables. Les piles sont représentées par des listes de boîtes où la boîte du dessus est la *première* boîte de la liste. L'entrepôt est représenté par une liste de piles où la pile numéro 0 est au début de la liste. Ainsi, nos deux dispositions d'entrepôts de l'exemple s'écrivent :

```
[[], [b,a], [c]]
[[a,b], [c], []]
```

Voici donc des exemples des trois scénarios normaux d'utilisation de vos relations :

```
| ?- verif([], [b,a], [c], [move(1,0),move(1,0),move(2,1)],
          [[a,b], [c], []]).
yes
| ?- verif([[c], [b,c,a], [a,b]], [move(2,0),move(1,0)], E).
E = [[b,a,c], [c,a], [b]] ? ;
no
| ?- solve([], [b,a], [c], S, [[a,b], [c], []]).
S = [move(1,0),move(1,0),move(2,1)] ;
no
```

Notez qu'il est acceptable (et compréhensible) que la relation 'solve' n'accepte pas toutes les stratégies qui permettent de passer d'une disposition d'entrepôt à l'autre. En effet, par souci d'efficacité, 'solve' peut se concentrer sur l'établissement d'une séquence particulière ou d'une famille de séquences. Tout ce qui est demandé à votre relation 'solve', c'est de découvrir au moins une séquence valide. Elle pourrait en générer d'autres (à l'aide de ';') si vous le désirez. Cependant, elle ne doit pas refuser de fournir une réponse ou boucler à l'infini lorsqu'il y a moyen de passer d'une disposition d'entrepôt à l'autre. Des temps de calcul exagérément longs ne sont pas acceptables non plus. Par exemple, dépasser 10 secondes pour trouver une solution à notre petit exemple est exagéré. S'il n'existe pas de moyen de passer d'une disposition d'entrepôt à l'autre, votre relation peut faire n'importe quoi. Enfin, la longueur des séquences trouvées par votre relation n'a pas d'importance.

Pour qu'il existe une séquence permettant de passer d'une disposition ('E1') à l'autre ('E2'), les trois conditions suivantes doivent être respectées :

- 'E1' et 'E2' doivent comporter le même nombre de piles ;
- sans égard à l'ordre, 'E1' et 'E2' doivent contenir les mêmes boîtes ; et
- un des trois cas suivants s'applique :
 - les entrepôts comportent soit 0, soit au moins 3 piles ;
 - les entrepôts comportent 1 pile et elles sont identiques (car aucun mouvement n'est possible) ; ou
 - les entrepôts comportent 2 piles et les boîtes correspondent si on les énumère dans l'ordre suivant : d'abord du bas de la pile 0 vers le haut et ensuite du haut de la pile 1 vers la bas (car seuls des échanges entre les dessus des deux piles sont possibles).

Vous devez remettre votre programme en un seul fichier (nommé 'tp3.pl') d'ici le 20 avril 23h59 à l'aide de la commande suivante :

```
remise ift2030 tp3 tp3.pl
```

N'oubliez pas de mettre vos noms au début du programme. Vous devez faire le travail en équipes de deux. Notez que le style de programmation est important : votre programme doit être bien structuré et bien commenté. Il est bien important de vous en tenir à la logique pure dans la mesure du possible. Une utilisation abusive de la négation, du *cut* (!) ou d'autres prédicats méta-logiques est considérée comme un mauvais style de programmation. Vous devez aussi remettre un rapport au professeur ou aux démonstrateurs d'ici la date d'échéance. Donnez une description concise de l'implantation des différentes relations que vous aurez écrites. Ce rapport risque d'être plus volumineux que les deux précédents. Soyez sûrs de bien expliquer votre implantation. La pénalité pour les retards est de 10% par jour.