Implantation d'un OS pour téléphone cellulaire à l'aide d'automates

À travers ce projet vous découvrirez les bases du fonctionnement d'un OS (*Operating System* = système d'exploitation) et vous apprendrez à utiliser des automates pour concevoir un téléphone dans lequel des actions sont associées aux touches.

1 Description du projet

L'objectif du projet est de réaliser un simulateur capable de faire fonctionner un automate à pile et de l'utiliser pour concevoir une implantation de quelques fonctionnalités d'un téléphone cellulaire. L'utilisation d'automates oblige à spécifier la réaction de l'application pour chaque événement possible. Cette obligation évite les bugs du « baby player » où l'application entre dans un état non spécifié lorsqu'un enfant actionne au hasard les touches d'une interface.
On prend comme exemple le cas d'un téléphone portable mais on pourrait faire le même exercice sur le menu d'un site web ou toute autre interface.

Le projet : modélisation d'un téléphone cellulaire à l'aide d'un automate à pile

L'alphabet de l'automate est constitué des touches du clavier. On suppose qu'il dispose du clavier numérique, alphabétique et de touches spéciales comme [ON]. [OFF], [CALL], [PIN], [ENTER], [COMPOSE], ...
Plusieurs degrés de difficulté et les notes associées
≥ 12 :
- un moteur d'automate
- un automate avec stockages des données dans une pile et qui modélise les fonctionnalités suivantes :
  - on/off
  - verification du code pin=1718 avant de pouvoir passer des appels
  - appel possible vers 18, 17, S O S même sans code pin
  - composition/appel d'un numéro à 6 chiffres

≥ 14 : exigences précédentes mais :
- pour les automates à pile
- vérification du code pin = $a^{n+1}.b^{n+1}$
- bloquer le téléphone après deux essais infructueux de pin

≥ 15 : exigences précédentes mais
- vérification du code pin = palindrome avec marque $*$ au milieu

≥ 17 : exigences précédentes mais
- pour les automates à pile non déterministe
- vérification du code pin = palindrome sans marque au milieu

≥ 18 : exigences précédentes mais
- vérification du code pin = palindrome sans marque au milieu contenant les symboles 1 7 1 8 (dans cet ordre mais potentiellement avec d'autres symboles au milieu).
  Exemple1 : a1b71c8c17b1a fait l'affaire et il est difficile de deviner qu'elle est votre mot de passe
  Exemple2 : 127731787137721 fait l'affaire
  Exemple3 : 1718171 est le mot le plus court qui satisfait les conditions pour être un pin valide

2 Principe de fonctionnement

On considère un automate qui mémorise les événements passés (ceux qui lui sont utiles) dans une pile. Les nœuds de l'automate sont représentés par des entiers. Les transitions de l'automate portent à la fois un événement déclencheur, une séquence d'action à effectuer et éventuellement un préfixe de pile qui permet d'exprimer une contrainte que doit satisfaire la pile de l'automate pour effectuer le transition (cf. cours sur les automates à pile). Par exemple voici une transition (sans préfixe)

qui indique que dans le nœud initial 1 l'appui sur la touche [ON] provoque l'exécution de la séquence d'actions qui fait "booter" le téléphone puis demander le code pin, l'automate passe alors au nœud 2.

2.1 Modélisation et fonctionnalités à réaliser

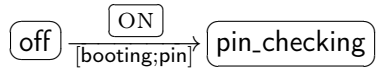
Les types permettant de modéliser un automate à pile sont fournis dans le fichier `sequelette.ml`.
Les fonctionnalités à réaliser sont rappelées en début de fichier.
Une fois le simulateur d'automate réalisé (les fonctions `one_step`) ; le projet consiste à réaliser les fonctionnalités demandés à l'aide d'un automate.

Les actions = command sequence Vous êtes libres de concevoir des actions aussi complexes que vous le souhaitez à condition qu'elles respectent le type `command`. Je vous conseille de vous limiter à quelques actions simples que vous pourrez combiner pour réaliser des actions plus évoluées.

Les nœuds des automates correspondent aux différents modes du téléphone. Il est possible d'utiliser des noms plutôt que des entiers pour les nœuds de l'automate. Par exemple,

1. off,
2. pin_checking,
3. composing,
4. calling,
5. ...

Ainsi, dans la transition prise comme exemple le nœud 1 serait nommé `"off"` et le nœud 2 serait nommé `"pin_checking"` par exemple.



Un processus en train d'exécuter l'automate est caractérisé par le nœud de l'automate sur lequel il se trouve et sa pile (stack).

Le système complet est constitué de tous les processus actif. Le comportement de chaque processus est défini par un automate. Il doit être possible de créer plusieurs instances d'un même automate.

2.2 Fonctionnement du simulateur d'automates

Votre implantation devra prendre en entrée un scenario sous la forme d'une séquence d'événements (= appui sur une touche). Votre moteur de simulation devra se comporter de la manière suivante :

1. afficher le mode/nœud courant (le nom du mode/nœud doit etre explicite) de chacun des processus actif,
2. afficher l'événement courant
3. faire réagir chacun des processus à l'événement courant :
   (a) déterminer les transitions possibles
   (b) exécuter les actions correspondantes. Si vous traitez le cas d'automate non-déterministe vous serez amené à créer des processus : un par transition possible si plusieurs transitions sont possibles sur un même événement.
   (c) mettre à jour le nœud de chacun des processus (suppression des processus qui ne peuvent réagir à l'événement courant)

2.3 Visualisation

Pour suivre l'évolution du système vous devez être en mesure d'afficher à chaque transition du système :
- les processus actifs
- l'état de chaque processus = nœud courant de chaque processus + sa pile
- le reste du scenario

Si vous utilisez l'interprète OCAML (plutôt que le compilateur) il ne sera pas utile de réaliser une fonction d'affichage pour suivre le nœud du système. Il suffira de demander à l'interprète la valeur de la variable qui représente l'état du système (le `system_state` courant). L'interprète affichera alors toutes les informations nécessaires :

```
system_state =
{ inputs    = ["enter" ; "off"];
  processes = [ {id = 1; state = {node = 0 ; stack = ["pin"] }} ;
                {id = 2; state = {node = 18; stack = ["S" ; "0" ; "S"] }}
              ]
}
```

La variable `system_state` est ici un enregistrement à deux champs :
- `input` montre le scenario = séquence de touches tapée par l'utilisateur et pas encore traitée par votre automate
- `processes` montre l'état des processus actifs = leur nœud courant et l'état de leur pile

3 Consignes et évaluation du projet

Conseils

1. Commencez par faire un automate très simple afin de mettre au point votre simulateur.
2. Une fois que vous avez confiance dans votre simulateur, réaliser des versions de téléphone intégrant de plus en plus de fonctionnalités.

Soyez à la fois créatifs (proposez un téléphone innovant avec une nouveau genre de mot de passe) mais réalistes (ne compliquez pas inutilement le moteur du simulateur). Et soignez votre codage. Plus votre implantation sera claire, lisible et puissante, meilleure sera votre note.

3.1 À rendre

- Un document de quelques pages décrivant les spécificités de votre téléphone, les modes et les symboles utilisés
- une implantation en OCAML
- un jeu de tests qui permet d'atteindre chacun des modes et de revenir au mode d'accueil.

3.2 Soutenance de 10min + 5 min de questions

La soutenance aura lieu lors du dernier TD. Attention 10 min c'est très court !
- 5 min de présentation des choix d'implantation
- 5 min de démo sur des scénarios préparés à l'avance qui montrent les fonctionnalités intéressantes de votre automate
- 5 min de questions techniques