Rapport de stage

Synet

3 juin 2002 au 12 juillet 2002

Igor Rosenberg

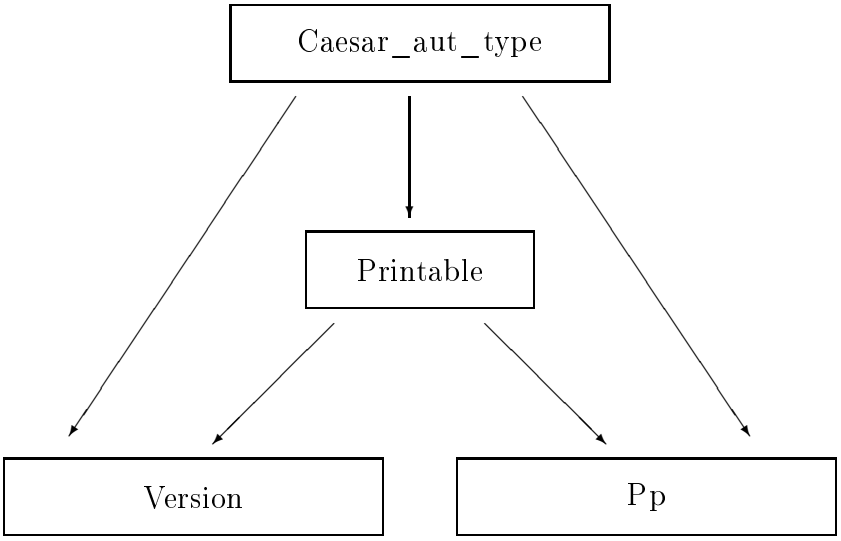
1 Préambule

Ceci est juste un descriptif de mon action sur le code du logiciel synet. Il est destiné à retrouver rapidement mes modifications, et permettre une compréhension rapide de mes choix lors de l'écriture de fonctions.

2 structure

J'ai créé deux répertoires : un pour les fonctions de base, **basics/**, et un pour le module d'expressions régulières, **regexp/**. Chaque repertoire est compilé par un Makefile qui lui est propre : on obtient deux nouveaux modules, `basics.cma` et `passerelle.cma`.

Module basics



3 Génération d'un langage (-rl)

3.1 Idée générale

On s'intéresse à la synthèse d'un réseau dont l'ensemble des parcours possibles est égal à un langage, donné sous forme d'automate. Lorsque ce n'est

pas possible, on souhaite obtenir le réseau dont le langage est la meilleure approximation par au-dessus du langage initial. La principale différence avec la génération d'un réseau à partir d'un automate est que l'on ne vérifie plus la séparation état-état (ssp).

3.2 Modifications

Lorsque l'on n'est pas dans le cadre du langage, on appelle la fonction `Synthesize.synthesize_isomorphism`. Pour les langages, on appelle la fonction `Synet.synthetiser_rayons_langage` qui elle-même renvoie sur `Synthesize.synthesize_language`. Pour ne garder que l'axiome essp, j'ai enlevé les lignes qui concernaient le ssp.

Il a fallu changer la représentation des états pour pouvoir appliquer le résultat de `Regions.base_depliage` et de `Regions.base` sur les mêmes fonctions (dépliage transformait les états en couples nom/numéro). Une implémentation avec des objets a donc été réalisée, entrainant des modifications conséquentes dans les fichiers `Caesar_aut`, `Caesar_aut_type` et la création de `Printable` . Pour contourner le problème d'égalité entre objets, une table de hachage est initialisée. La création d'un état s'effectue avec la fonction `make_entier`, `make_chaine` ou `make_identificateur`. Elle vérifie que l'état n'est pas déjà référencé, et le cas échéant, lui alloue une nouvelle cellule dans la table de hachage.

3.3 Strucure avec les objets - Caesar_aut_type

3.3.1 les objets sont tous printable

class t : Printable.t
En fait chaque objet a une méthode qui s'appele to_string :
val to_string : t $\mapsto$ string

3.3.2 différents objets / différentes utilisations

class identificateur : string $\mapsto$ t
class chaine : string $\mapsto$ t
class entier : int $\mapsto$ t
class couple : string $\mapsto$ int $\mapsto$ t

3.3.3 différents objet / différentes initialisations

val make_identificateur : string $\mapsto$ t
val make_chaine : string $\mapsto$ t

val make_entier : int $\mapsto$ t
val make_couple : string $\mapsto$ int $\mapsto$ t

3.3.4

Chaque type (entier, chaine...) a sa propre table de hachage (construite au fur et à mesure des besoins avec make_...), pour permettre l'unicité des identificateurs, à contenu égal.

4 Expressions régulières (-a)

Une extension utile de la synthèse modulo égalité de langage est de pouvoir donner le langage non plus sous forme d'automate, mais sous forme d'expression régulière. J'ai trouvé sur internenettes un programme permettant d'analyser des expressions régulières :

`http://www.lri.fr/~marche/regexp/`

Il est sous licence LGPL, et s'il est utilisé dans un logiciel, il oblige le logiciel tout entier à être sous GPL.

Le module `Passerelle` cache toute l'implémentation de regexp : synet n'utilise que la fonction

`lire_regexp` : *string* $\mapsto$ *"automate" (état ini* $\times$ *liste de transitions).*

4.1

synet -ar

synet.ml
. . ->synthetiser_expression_reguliere
. . . . ->Passerelle.lire_regexp
. . . . . ->Final.regexp_to_auto
. . . . ->Synthesize.synthesize_language

4.2 Drapeaux

Vu qu'une expression régulière implique la description d'un langage, l'option **-a** génère une synthèse sans ssp. Je n'ai pas implémenté l'option sans le paramètre **-r**, donc `synet -alr` $\Leftrightarrow$ `synet -ar` mais `synet -a` (sans le r) génère une erreur.

4.3 Pas d'identificateurs...

Le programme pour les expressions régulières est basée sur la recherche un caractère à la fois, et la reconnaissance d'identificateurs va être très délicate : la plupart des fonctions du module rendent un entier qui désigne le code ascii du caractère. Il faudrait construire une table de hachage reconnaissant tous les identificateurs (composés d'une seule lettre, ou non). Le code étant écrit pour reconnaître des plages de caractères ([a-z]), les modifications risquent de prendre du temps. Le plus simple à mon avis serait de réécrire complètement les fonctions de création du premier automate, avant sa réduction modulo les epsilon-transitions.

Pour prendre en compte cette nouvelle fonctionnalité, j'ai modifié les fichiers `Commande, synet.ml, Environment`. Le paramètre **-a** sur la ligne de commande est particulier, car le nom habituellement en fin de ligne n'est plus le nom d'un fichier décrivant un automate mais le nom d'un fichier contenant une expression régulière.

4.4 Mes ajouts dans regexp

Le module regexp, tel que je l'ai trouvé sur internet, pose plusieurs problèmes :

- pour diminuer la taille de l'automate, deux transitions (0,a,1) et (0,b,1) sont fusionnées pour donner (0,a b,1).
- si plusieurs caractères se suivent, ils sont réduits en un interval : (0,a,1) (0,b,1) et (0,c,1) donne (0,a-c,1).
- des transitions interdites sont générées : pour 'a|c', regexp génère (0,a,1) (0,b,-1) et (0,c,1) où -1 est un état puits.

Solution : pour retrouver un automate utilisable dans synet, j'ai rajouté des fonctions dans `Passerelle` pour étaler ces transitions :
`separate_automaton` : *"automate"* $\mapsto$ *"automate"* qui étale comme il faut l'automate généré par regexp. Une partie de l'algorithme fait une scission basée sur le séparateur *espace*. Donc si jamais la reconnaissance des identificateurs est implémentée, cela interdira les identificateurs contenant des espaces. Ce n'est pas un gros problème, mais il est important de noter ce léger défaut. J'ai aussi étalé la relation 'a-f', et supprimé les transitions vers l'état puits.

Il y a peut-être d'autres vices cachés...

4.5 conclusions sur regexp

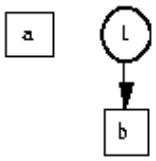
Tel quel, le module regexp me paraît inutilisable : je n'ai pas fait assez de tests pour garantir les résultats obtenus, et le nombre d'effets indésirables que j'ai repéré est trop grand, il rique donc d'y en avoir qui m'ont échappé. Quand je testais, j'ai trouvé qu'une bonne manière de vérifier est de générer une grosse expression régulière (programme `synet-v2/test/genreg`) et de regarder s'il n'y a pas de faute dans la syntaxe du réseau généré (un exemple) :

```
[~/synet-v2/src (13886K)]$./synet -arx pb.reg
The transition system is separated, w.r.t. essp .
Synthesized net:     -a-d-f-q-t-w:=1
graph parser: syntax error near line 16
context:  >>> t- <<< w [shape=box,label="t-w"];
```

5 Réseaux non bornés (-b)

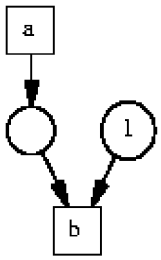
A l'aide du paramètre **-b** sur la ligne de commande, on autorise la synthèse de réseaux non bornés. Par exemple, le réseau issu de aa^*ba^* n'est pas borné.

sans le paramètre -b



Event-state separation failures : Event b and state 0 :0 are not separated.
Synthesized net : -b :=1

avec le paramètre -b



The transition system is separated, w.r.t. essp .
Synthesized net : -b :=1
$+a-b$

5.1 Modifications

J'ai surtout touché aux équations de `Regions` . Les lignes modifiées se repèrent assez bien grâce à la condition `if bb then` . J'ai aussi modifié `commande.ml` pour la lecture du paramètre, ainsi que `synet.ml` qui applique les fonctions en leur rajoutant le booléen **bb** : borné ou non.

6 Makefile

- basics/Makefile : j'arrive pas à tout effacer. Par exemple, `basics.cma`
- src/Makefile : le ocamlsynet est mal compilé : il faut rajouter malheureusement les options **-I regexp/ -I basics/**. Quelque chose est à revoir là. A cause de la nouvelle version de cplex, il a fallu changer les chemins, verifier que yen a pas en trop maintenant...

```
[~/synet-v2/src (13948K)]$ocamlsynet
        Objective Caml version 3.04

# Caesar_aut_type.make_entier 1;;
Unbound value Caesar_aut_type.make_entier
# exit 1;;


[~/synet-v2/src (13948K)]$ocamlsynet -I basics/ -I regexp/
        Objective Caml version 3.04

# Caesar_aut_type.make_entier 1;;
- : Caesar_aut_type.t = <obj>
# exit 0;;
```

7 Tests

Pour tester le **-rl**, j'ai essayé sur quelques automates. La synthèse est correcte sur de petits exemples (2-3 états) et sur des horreurs (les automates de Phillipe : quinzaine d'états). N'ayant pas les outils nécessaires pour fabriquer le graphe de marquage, je ne me suis pas aventuré plus loin.

La synthèse d'expression régulière est à prendre avec des pincettes. La dernière version ne m'a pas donné d'erreur, mais je n'ai pas eu le temps de faire une batterie de tests exhaustive. Le programme `test/genreg` est bien utile : il génère des expressions régulière d'une taille donnée, ce qui permet déjà quelques test. En fait il m'a manqué l'outil inverse : à partir d'un rdp, retrouver le graphe de marquages...

Mes modifications n'étant pas assez testées, les resultats obtenus ne sauraient être garantis.