Vérification automatique de la correction de strings formés dynamiquement

TIPE

Maire Grégoire

2023

Motivations

- En ville, les métros, voiture, avions ou autres transports en commun nécessitent l'assurance que leurs programmes soient corrects
- Longs programmes : besoin de preuves automatisées
- Mon but: automatiser les preuves d'invariants sur les strings dans les programmes

Motivations

- Plus précisément, mon intérêt : construction dynamique d'un string dans un programme.
- Exemple d'application: Vérification d'un message formatté d'un métro à une unité de contrôle

Limitation

Ce qu'on aurait envie de faire : récolter l'ensemble des états possibles d'un string à la fin du code d'un programme. Mais ce n'est pas toujours possible :

Limitation : Théorème de Rice [2]

Toute propriété sémantique non triviale d'un programme est indécidable (impossible à vérifier automatiquement).

- D'où l'utilisation d'une méthode pour contourner la limitation : *l'interprétation abstraite*, consistant en une sur-approximation des états possibles d'arrivée.
- Il est toujours possible de sur-approximer (pas indécidable).
- Peut mener à des faux-négatifs, mais jamais à des faux-positifs. Mathématiquement correct !

Interpétation abstraite

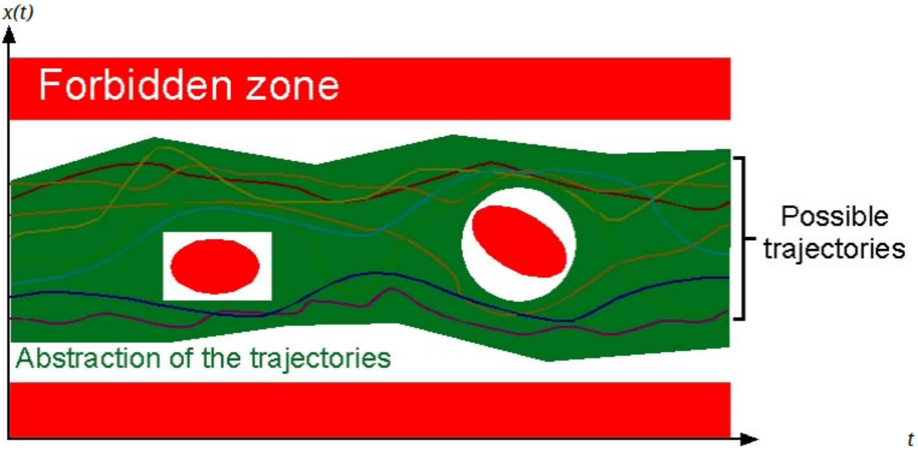

Figure: Visualisation de l'interprétation abstraite [1]

Choix d'un domaine abstrait

Pour ce faire, on définit un *domaine abstrait* pour représenter l'ensemble des valeurs prises par le string

Domaines déjà existants:

- Domaine sur les préfixes : "l'ensemble de tous les strings commençant par..."
- Domaine sur les facteurs communs: "l'ensemble de tous les strings ayant comme facteur..."

Création d'un domaine inexploré: la représentation par automates.

- Précis
- Plus coûteux

Un exemple concret

```
// ALPHABET = {0, 1}

str = new_string("0");

while(COND1) {
    if (COND2) {
        concat(str, "01");
    }
    else {
        concat(str, "100");
    }

    concat(str, "1");
}

// INVARIANT À PROUVER : [str.as_binary % 3 = 0]
```

Figure: Un programme construisant des nombres congru à 0 modulo 3

Prouvons sa correction automatiquement !

Table des matières

Table des matières

Structure d'automates

```
type 'a t = {
  nb: int; (* nombre d'états : numérotés 0, 1, ..., nb-1 *)
  sigma: 'a array; (*alphabet*)
  i: etat_t list;  (*états initiaux*)
  f: etat_t list;  (*états finaux*)
  delta: (etat_t * 'a, etat_t list) Hashtbl.t (*fonction de transitions*)
}
```

Figure: Structure choisie pour représenter les automates

- Affichage (par Graphviz)
- Acceptation d'un mot
- Complété, Émondé, Déterminisé, Complémentaire
- Union, Intersection.

Langage de test

On considère un string en particulier qu'on construit.
Pour appliquer la théorie, création d'un mini-langage, avec certaines primitives de string intégrées.
Syntaxe (inductive) du langage:

$$
\begin{aligned}
stat := \; & stat \; ; \; stat \\
\mid \; & \textbf{if } cond \textbf{ then } stat \textbf{ else } stat \\
\mid \; & \textbf{while } cond \textbf{ do } stat \textbf{ done} \\
\mid \; & \textbf{skip} \\
\mid \; & \textbf{assert}(cond) \\
\mid \; & stringop
\end{aligned}
$$

$$
\begin{aligned}
cond := \; & \textbf{unknown} \\
\mid \; & b \in Bool \\
\mid \; & \textbf{not } cond \\
\mid \; & cond \wedge cond \\
\mid \; & cond \vee cond
\end{aligned}
$$

$$
\begin{aligned}
stringop := \; & \textbf{new\_string}(str) \\
\mid \; & \textbf{push\_left}(str) \\
\mid \; & \textbf{push\_right}(str)
\end{aligned}
$$

Le langage en pratique

- En pratique, seulement un AST mais je présenterai les exemples en code directement.
- Le programme d'introduction peut ainsi être représenté de la sorte:

```
new_string("0");
while unknown do
    if unknown then push_right("01") else push_right("100");
    push_right("1")
done
```

Figure: Programme d'introduction avec la syntaxe créée

Sémantique de récolte des états

- But: récolter l'ensemble des états possibles du string au fur et à mesure du programme
- Concept : récolte séquentielle des états par une fonction de transfert pour amener à l'étape suivante de l'algorithme

Cas du while

Un cas ardu se pose : la récolte d'états après un *while*.
Lorsqu'on arrive sur un while, on effectue les procédures suivantes :

- Récolte de l'intégralité des opérations sur le string dans le while
- S'il ne contient que des `push_right`, alors on ajoute une boucle à l'automate. (point fixe des états)

```
new_string("b");

while unknown do push_right("a") done
```

Figure: Code d'exemple

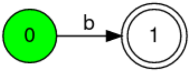

Figure: Après 0 entrée dans le while

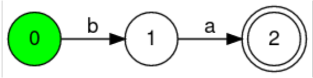

Figure: Après 1 entrée dans le while

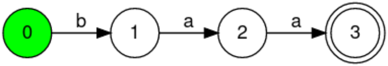

Figure: Après 2 entrées dans le while

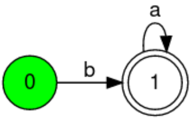

Figure: Approximation du point fixe du while

Approximation *correcte* car on récupère au moins les états accessibles.

Table des matières

Application sur le programme d'introduction :

```
new_string("0");
while unknown do
    if unknown then push_right("01") else push_right("100");
    push_right("1")
done
```

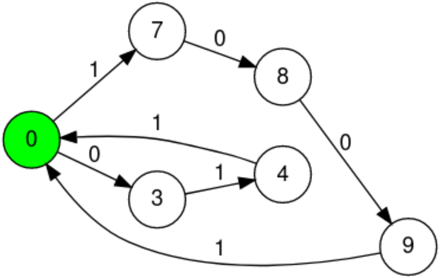

Figure: Automate généré

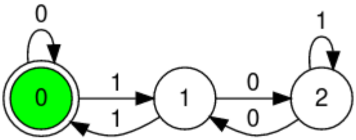

Figure: Automate de référence

On vérifie automatiquement que le langage de l'automate généré est inclus dans celui de l'automate de référence.

Table des matières

Explosion d'états

Considérons le programme suivant :

```
new_string("");

if unknown then push_right("a")
else push_right("b");

if unknown then push_right("a")
else push_right("b");

// [...]

if unknown then push_right("a")
else push_right("b");

if unknown then push_right("a")
else push_right("b");
```

Explosion d'états

Explosion exponentielle des états par rapport au nombre de `if`:

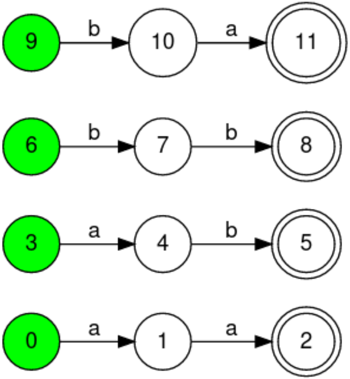

Figure: Automate généré pour 2 `if` successifs

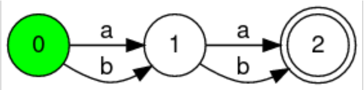

Figure: Un autre automate reconnaissant le même langage

Solution à l'explosion d'états

- Solution possible: minimisation de l'automate

Algorithme de Brzozowski [3]

On pose $t(\mathcal{A})$ l'automate transposé de $\mathcal{A}$, $d(\mathcal{A})$ le déterminisé de $\mathcal{A}$.
Alors : $\mathcal{A}' = d(t(d(t(\mathcal{A}))))$ est l'automate minimal de $\mathcal{A}$.

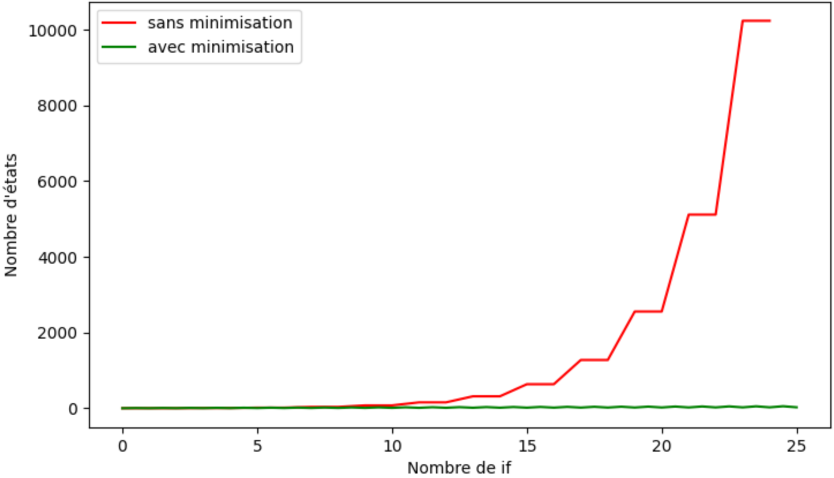

Figure: Evolution du nombre d'états en fonction du nombre de if

Table des matières

Conclusion

- Construction d'un nouveau domaine abstrait
- Application sur certains programmes
- Mise en évidence et amélioration de certaines limitations

Table des matières

Énoncé du théorème

Soit f une fonction booléenne totale prenant en entrée un algorithme, telle que f est non-triviale, et f conserve l'équivalence sémantique.
Alors f est non-calculable.

Soient $\langle F \rangle = \langle \texttt{while (true)} \rangle$. et $\langle B \rangle$ tel que $f(\langle B \rangle) \neq f(\langle F \rangle)$
On pose tr: $(\langle A \rangle, e) \longmapsto \langle x \mapsto A(e); B(x) \rangle$
Soient A un algorithme et e une entrée de A.
Si $A(e)$ finit: $tr(A, e) \sim \langle B \rangle$ donc $f(tr(\langle A \rangle, e)) = f(\langle B \rangle)$
Sinon: $tr(\langle A \rangle, e) \sim \langle F \rangle$ donc $f(tr(\langle A \rangle, e)) = f(\langle F \rangle)$
Ainsi:

$$\begin{aligned} f(tr(\langle A \rangle, e)) = f(\langle B \rangle) &\iff A(e) \text{ termine} \\ &\iff Arret(\langle A \rangle, e) \end{aligned}$$

D'où une réduction du problème $Arret$ (resp. $\neg Arret$).

Annexe: Preuve de la correction de minimisation

Algorithme de Brzozowski [3]

On pose $t(\mathcal{A})$ l'automate transposé de $\mathcal{A}$, $d(\mathcal{A})$ le déterminisé par parties de $\mathcal{A}$ obtenu en ne gardant que les états accessibles.
Alors on a : $\mathcal{A}' = d(t(d(t(\mathcal{A}))))$ est l'automate minimal de $\mathcal{A}$.

Cet automate reconnaît le même langage que $\mathcal{A}$ sans soucis.
Soient P et P' deux états distincts de $\mathcal{A}'$.
Par définition, P et P' sont deux ensembles-états de l'automate $d(t(\mathcal{A}))$.
Soit R un état de $d(t(\mathcal{A}))$, tel que $R \in P$ et $R \notin P'$. R est aussi un ensemble-état, qui vérifie, par définition du transposé,
$R = \{q \mid q \xrightarrow{w} f$ dans $\mathcal{A}$, avec $f \in F\}$ pour un certain mot w. On en déduit que dans $\mathcal{A}'$, il existe un chemin étiquetté par w vers un état final, alors que ce n'est pas le cas pour P'. Donc P et P' ne sont pas des états équivalents.

Bibliographie

1 Visualisation de l'interprétation abstraite: https://www.di.ens.fr/ cousot/AI/IntroAbsInt.html

2 Théorème de Rice: H. G. Rice, Classes of Recursively Enumerable Sets and Their Decision Problems, Transactions of the American Mathematical Society, volume 74, numéro 2, mars 1953

3 Algorithme de Brzozowski :Janusz A. Brzozowski, *Canonical regular expressions and minimal state graphs for definite events*, Proceedings of the Symposium on Mathematical Theory of Automata, Polytechnic Institute of Brooklyn, April 1962, New York, Wiley, 1963, p. 529-561

4 Domaines de strings existant: https://pm.inf.ethz.ch/publications/CostantiniFerraraCortesi11.pdf