Learning Programs from Traces using Version Space Algebra

Tessa Lau, Pedro Domingos et Daniel S. Weld

Fiche de lecture par Louis-Noël Pouchet, 6 mars 2006

Résumé

Cet article propose un cadre formel pour l'apprentissage de programmes depuis des traces d'exécution, à l'aide de l'algèbre de l'espace des versions [1]. Cette méthode permet d'apprendre des programmes complets à l'aide d'exemples de traces et d'une spécification dans le langage cible.

La méthode proposée est validée expérimentalement, et étendue à l'apprentissage de programmes de traitement de textes (programmation par démonstration) sans identifiant explicite d'étapes.

1 Introduction

La programmation par démonstration, ou le reverse-engineering, seraient facilités si on pouvait induire des programmes depuis leurs traces d'exécution. Des essais ont été faits sur des langages fonctionnels ou déclaratifs (LISP, PROLOG), et il faudrait une méthode pour des langages procéduraux comme le C++ ou Java.

L'approche proposée est basée sur l'algèbre d'espaces de version de Lau et al [2], qui étend l'espace des versions de Mitchell [3]. On doit imposer un biais grâce à une grammaire.

2 Algèbre d'espaces de versions

L'algèbre d'espaces de version [2] est une méthode pour composer des espaces de version ensemble, afin d'en créer un plus complexe. Les opérateurs sont les suivants :

- union (l'union de deux espaces),
- jointure (le produit cartésien de deux espaces),
- transformation (la conversion des domaines et images des fonctions).

3 Formulation du problème

On définit :

- l'état (l'environnement visible du système),
- la trace (une séquence d'états résultant de l'exécution du programme depuis l'état initial),
- le biais (le langage dénotant l'ensemble des programmes possibles - une grammaire).

Le contenu d'un état affecte la difficulté d'apprentissage. Les états peuvent être selon quatre configurations : incomplet (des variables importantes sont cachées), complet, complet et contenant un identifiant unique pour chaque pas du programme, et totalement visible (on dispose de toutes les informations).

4 Apprentissage de programmes depuis des traces

On définit le langage reconnu par l'apprenant à l'aide d'une grammaire (incluant les contrôles de flux comme les boucles et les conditionnelles), ce qui définit l'espace des hypothèses. On va ensuite apprendre en ne conservant que les hypothèses qui sont consistantes, c'est à dire celles qui dans un état S_i produisent l'état S_{i+k} à l'exécution.

5 Apprentissage sans identificateur d'étapes

Dans la programmation par démonstration, on ne peut pas demander à l'utilisateur de spécifier un pas de programme pour chaque action.

On va supposer que le programme est composé d'une seule boucle, avec un nombre fixé d'instructions à l'intérieur. Le programme d'apprentissage va automatiquement segmenter la trace afin de trouver le bon nombre d'itérations.

Par exemple, avec SMARTedit, si l'utilisateur effectue trois actions, l'ensemble des hypothèses contiendra les programmes effectuant une, deux et trois actions. Le programme d'apprentissage enlèvera rapidement les hypothèses contenant une et deux actions, celles-ci ne permettant pas d'atteindre la trace.

6 Travaux liés et conclusion

Ce travail est en relation avec la programmation automatique, et un travail futur sera de voir l'applicabilité de la méthode à la programmation logique inductive.

Le problème de l'apprentissage de programmes depuis des traces a été formalisé, dans le cadre de l'algèbre d'espaces de versions. Le biais d'apprentissage (donné par une grammaire) permet de définir l'espace des hypothèses. Ce système ne gère toutefois pas le bruit dans les exemples, ce qui constitue la suite du travail.

7 Critique

Cet article propose une méthode très intéressante pour l'apprentissage de programmes. Elle permet d'apprendre des programmes procéduraux, à partir d'une trace facilement obtenable dans le contexte du reverse-engineering.

On est toutefois très loin d'un apprentissage de programmes très complexe, puisqu'il faut définir une grammaire donnant la forme générale du programme à apprendre afin de définir l'espace des hypothèses. L'approche est séduisante, mais ne permettra à mon avis jamais d'apprendre des programmes en situation réelle (par exemple un algorithme complexe comme la déterministation d'automates finis).

Références

1. Lau T., Domingos P., Weld D. S., Learning programs from traces using version space algebra Proc. of K-CAP, (2003), 36–43.
2. Lau T., Domingos P., Weld D. S., Version space algebra and its application to programming by demonstration Proc. of ICML, (2000), 527–534.
3. Mitchell T., Generalization as search *Artificial Intelligence 18*, (1982), 203–226.