Learning Programs from Traces using Version Space Algebra

Tessa Lau, Pedro Domingos et Daniel S. Weld

6 mars 2006

Outline

Introduction

Domaine : Apprentissage de programmes.

Objectifs :

- Proposer un cadre général d'apprentissage de programme.
- Induire des programmes depuis des traces d'exécution.
- Induire des programme donnés par démonstration.

Introduction

Domaine : Apprentissage de programmes.

Objectifs :

- Proposer un cadre général d'apprentissage de programme.
- Induire des programmes depuis des traces d'exécution.
- Induire des programme donnés par démonstration.

Introduction

Domaine : Apprentissage de programmes.

Objectifs :

- Proposer un cadre général d'apprentissage de programme.
- Induire des programmes depuis des traces d'exécution.
- Induire des programme donnés par démonstration.

Contexte de cette recherche

- Apprentissage de programme dans des langages procéduraux (C++, Java).
- Utilisation de l'algèbre d'espaces de versions.
- Utilisation d'une grammaire pour exprimer le biais.
- Exploration efficace et exhaustive de l'espace des hypothèses.

Contexte de cette recherche

- Apprentissage de programme dans des langages procéduraux (C++, Java).
- Utilisation de l'algèbre d'espaces de versions.
- Utilisation d'une grammaire pour exprimer le biais.
- Exploration efficace et exhaustive de l'espace des hypothèses.

Contexte de cette recherche

- Apprentissage de programme dans des langages procéduraux (C++, Java).
- Utilisation de l'algèbre d'espaces de versions.
- Utilisation d'une grammaire pour exprimer le biais.
- Exploration efficace et exhaustive de l'espace des hypothèses.

Contexte de cette recherche

- Apprentissage de programme dans des langages procéduraux (C++, Java).
- Utilisation de l'algèbre d'espaces de versions.
- Utilisation d'une grammaire pour exprimer le biais.
- Exploration efficace et exhaustive de l'espace des hypothèses.

Algèbre d'espaces de versions

Objectif : composer des espaces de versions simples pour en former un plus complexe.

Opérateurs :

- union,
- jointure,
- transformation.

Formulation du problème [1/2]

Définitions :

- **Etat** : l'environnement visible du système durant l'exécution.
- **Trace** : une suite d'états générée par un programme.
- **Biais** : le langage dénotant l'ensemble des programmes possibles.

Formulation du problème [2/2]

Formulation du problème d'apprentissage

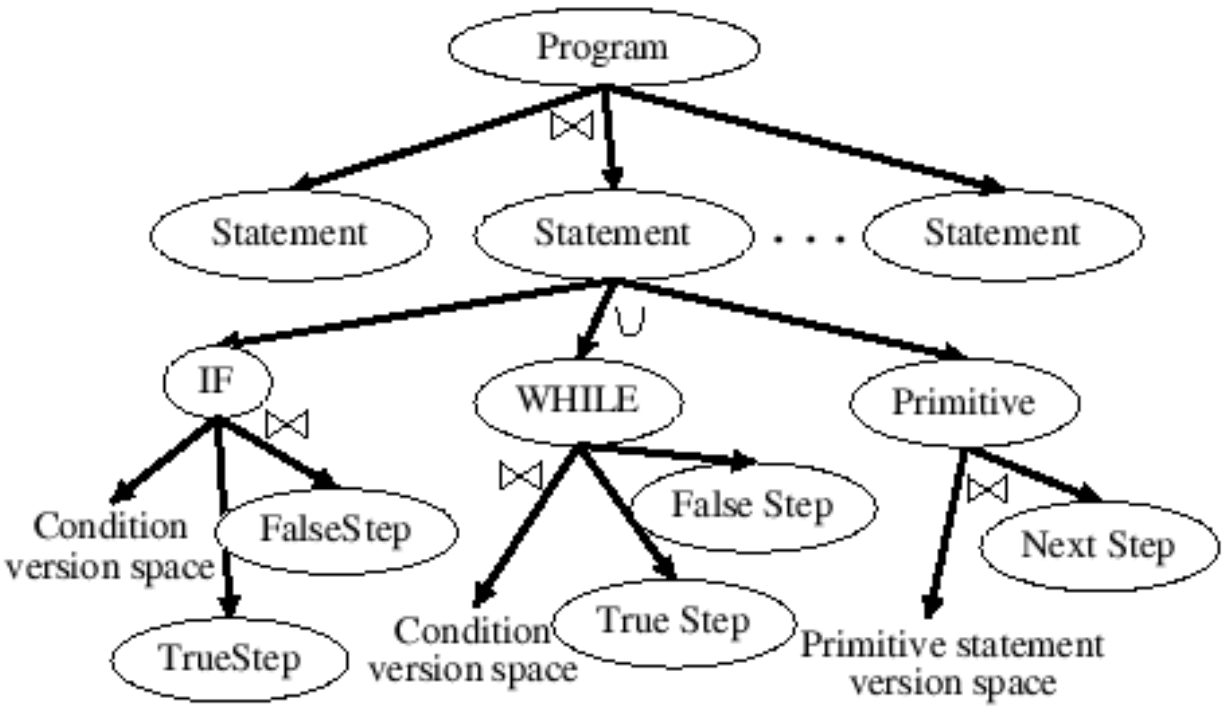
Etant donné un langage et une ou plusieurs traces d'un programme cible, renvoyer un programme cohérent avec tous les exemples d'apprentissage et qui se généralise correctement aux futurs exemples.

Exemple de biais

Example (Biais d'apprentissage)

```
<Program> ::= <Statement> [; <Program> ]
<Statement> ::= <PrimitiveStatement>
<Statement> ::= IF <Condition> THEN <Program>
                                   ELSE <Program>
<Statement> ::= WHILE <Condition> DO <Program>
```

Exemple d'espaces de versions



Exemple de trace

Prog step	Var values i	j	k	Label
1	18	12	0	S_0
2	18	12	0	S_1
3	18	12	6	S_2
4	12	12	6	S_3
1	12	6	6	S_4
2	12	6	6	S_5
3	12	6	0	S_6
4	6	6	0	S_7
1	6	0	0	S_8
5	6	0	0	S_9

(a)

```
1: while j > 0:
2:         k = i mod j
3:         i = j
4:         j = k
```

(b)

Programmation par démonstration

Objectif : apprendre des programmes dont on ne connais pas les pas d'exécution.

Exemple : SMARTedit, un programme d'apprentissage de traitement de texte.

Méthode : Faire une segmentation automatique du programme.

Résultat : Plus performant que le système original.

Programmation par démonstration

Objectif : apprendre des programmes dont on ne connais pas les pas d'exécution.

Exemple : SMARTedit, un programme d'apprentissage de traitement de texte.

Méthode : Faire une segmentation automatique du programme.

Résultat : Plus performant que le système original.

Programmation par démonstration

Objectif : apprendre des programmes dont on ne connais pas les pas d'exécution.

Exemple : SMARTedit, un programme d'apprentissage de traitement de texte.

Méthode : Faire une segmentation automatique du programme.

Résultat : Plus performant que le système original.

Programmation par démonstration

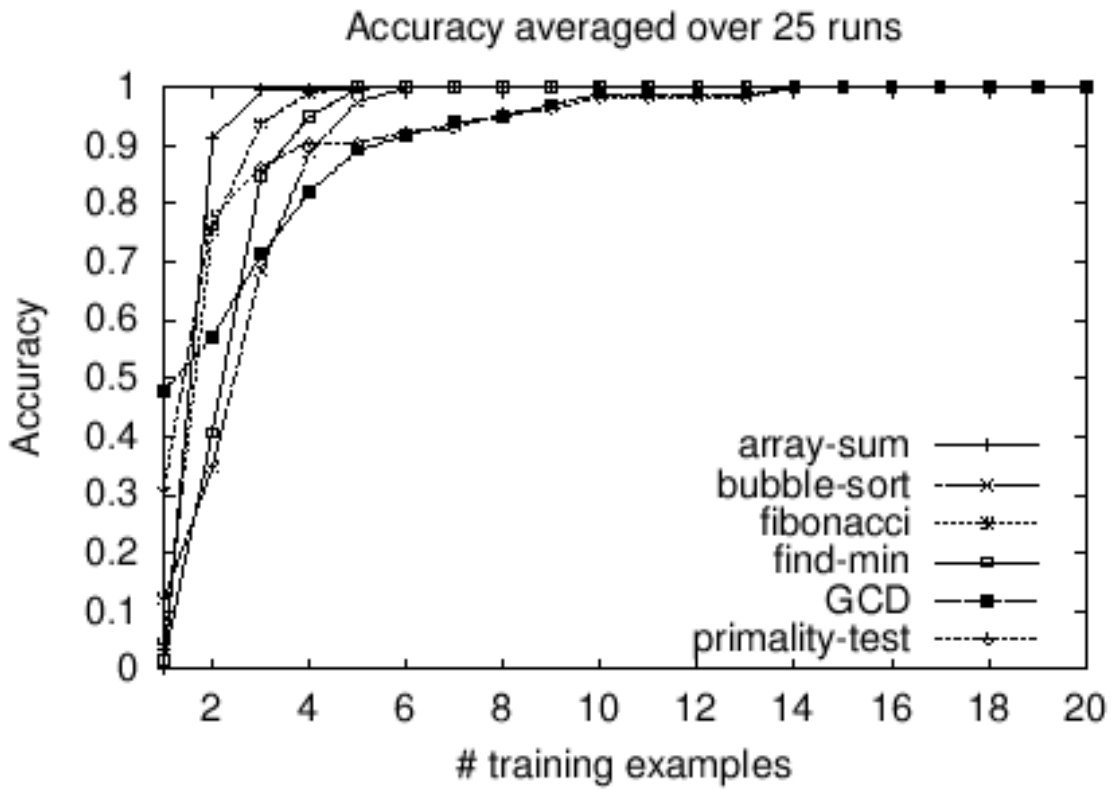
Objectif : apprendre des programmes dont on ne connais pas les pas d'exécution.

Exemple : SMARTedit, un programme d'apprentissage de traitement de texte.

Méthode : Faire une segmentation automatique du programme.

Résultat : Plus performant que le système original.

Résultats


Accuracy averaged over 25 runs
Accuracy
training examples
array-sum
bubble-sort
fibonacci
find-min
GCD
primality-test
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
2
4
6
8
10
12
14
16
18
20

Ouvertures

- Travail très proche de la programmation automatique.
- Application possible à la programmation logique inductive.
- Nécessité de supporter le bruit dans les exemples ?

Ouvertures

- Travail très proche de la programmation automatique.
- Application possible à la programmation logique inductive.
- Nécessité de supporter le bruit dans les exemples ?

Ouvertures

- Travail très proche de la programmation automatique.
- Application possible à la programmation logique inductive.
- Nécessité de supporter le bruit dans les exemples ?

Conclusion [1/2]

Ce travail propose :

- une formalisation du problème d'apprentissage de programmes à l'aide de l'algèbre d'espaces de version,
- une méthode pour apprendre des programmes depuis des traces,
- une généralisation aux programmes sans identifiants de pas,
- une validation expérimentale de la méthode.

Conclusion [1/2]

Ce travail propose :

- une formalisation du problème d'apprentissage de programmes à l'aide de l'algèbre d'espaces de version,
- une méthode pour apprendre des programmes depuis des traces,
- une généralisation aux programmes sans identifiants de pas,
- une validation expérimentale de la méthode.

Conclusion [1/2]

Ce travail propose :

- une formalisation du problème d'apprentissage de programmes à l'aide de l'algèbre d'espaces de version,
- une méthode pour apprendre des programmes depuis des traces,
- une généralisation aux programmes sans identifiants de pas,
- une validation expérimentale de la méthode.

Conclusion [1/2]

Ce travail propose :

- une formalisation du problème d'apprentissage de programmes à l'aide de l'algèbre d'espaces de version,
- une méthode pour apprendre des programmes depuis des traces,
- une généralisation aux programmes sans identifiants de pas,
- une validation expérimentale de la méthode.

Conclusion [1/2]

Ce travail propose :

- une formalisation du problème d'apprentissage de programmes à l'aide de l'algèbre d'espaces de version,
- une méthode pour apprendre des programmes depuis des traces,
- une généralisation aux programmes sans identifiants de pas,
- une validation expérimentale de la méthode.

Conclusion [2/2]

Travail à venir :

- génération automatique des espaces des versions depuis la grammaire,
- utilisation d'autres méthodes d'apprentissage,
- réduire le nombre de traces nécessaires.