Analyse de traces d'exécution de programmes informatiques : Application au jeu sérieux Prog&Play

S. Meresse[1], M. Muratet[1,2], and A. Yessad[1]

[1] Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, 4 place Jussieu 75005 Paris, France
[2] INS HEA, 58-60 Avenue des Landes, 92150 Suresnes, France
{mathieu.muratet, amel.yessad}@lip6.fr

Abstract. Cet article présente un travail de recherche sur l'analyse de traces d'exécution de programmes informatiques réalisés par des étudiants dans le cadre du jeu sérieux Prog&Play. Afin de garantir la généricité du processus d'analyse de traces et son indépendance vis à vis du langage et de l'environnement de programmation, nous ne basons pas l'analyse sur les programmes sources produits par les étudiants mais plutôt sur les traces d'exécution de ces programmes. L'objectif de cette recherche est de reconstituer la sémantique d'un programme écrit par un étudiant à partir de la trace d'exécution de celui-ci et d'utiliser cette sémantique pour fournir un *feedback* adapté aux étudiants. Nous présentons dans cet article notre positionnement vis à vis des travaux qui nous ont inspirés à la fois pour l'analyse des traces et la production du feedback.

Keywords: Analyse de trace, Prog&Play, Jeux sérieux, Compression de traces, Alignement de traces

1 Introduction

Prog&Play [1] est une bibliothèque de fonctions qui, intégrée à un moteur de jeux de STR (Stratégie Temps Réel), permet à l'apprenant d'écrire un programme informatique qu'il va pouvoir exécuter pour réaliser des actions dans le jeu (déplacer des unités, attaquer un ennemi, soigner un allié...). Prog&Play est disponible en 6 langages de programmation (Ada, C, Java, OCaml, Python et Scratch) laissant la possibilité à chaque enseignant de choisir le paradigme et le langage de programmation le plus adapté à ses besoins. Dans ce contexte, nous voulons intégrer un module d'analyse au jeu permettant la génération automatique de feedbacks à destination des apprenants pour les renseigner sur la qualité de leurs solutions.

Etant donné que Prog&Play est compatible avec plusieurs langages de programmation et n'impose pas d'environnement de programmation, une approche qui serait basée sur l'analyse statique des programmes impliquera nécessairement le développement de modules spécifiques à chaque langage et environnement de programmation. L'approche que nous avons envisagée ne consiste donc pas à analyser la production de l'étudiant (son code source) mais à analyser la trace d'exécution de son programme afin d'être indépendant du langage et de l'environnement de programmation utilisé. Cependant, de telles traces peuvent être volumineuses et sont souvent pauvres d'un point de vue

sémantique car seuls les appels aux fonctions de la bibliothèque Prog&Play sont tracés. Nous pouvons ainsi dégager deux questions de recherche :

1. Comment reconstituer au mieux la sémantique du code source d'un programme informatique écrit par un apprenant à partir de sa trace d'exécution brute et partielle ?
2. Comment, à partir d'une trace d'exécution ainsi reconstituée, générer automatiquement un feedback pertinent et utile pour un apprenant ?

2 Etat de l'art et positionnement

Cette section présente les travaux de recherche que nous avons explorés ou sur lesquels nous nous sommes basés pour mettre en place notre solution.

Dans un premier temps, nous avons étudié la méthode d'analyse dynamique de Safyallah et Sartipi [2] qui a été proposée pour extraire des patrons de séquences du code source d'un logiciel afin d'en identifier des fonctionnalités spécifiques. Pour cela, un ensemble de cas d'utilisation du logiciel impliquant une fonctionnalité spécifique est défini (pour un logiciel de dessin, tous les cas d'utilisation impliquant la fonctionnalité *Tracer une ligne* par exemple). Ces cas d'utilisation sont ensuite simulés dans le but de produire un ensemble de traces d'exécution. Enfin, les fonctions impliquées dans un cas d'utilisation (ou dans tous les cas d'utilisation) sont identifiées en utilisant du *sequential pattern mining*. Par analogie, si nous considérons comme fonctionnalité une action programmable à l'aide de la bibliothèque Prog&Play et comme cas d'utilisation un programme écrit par un joueur, une première forme d'analyse pourrait être envisagée en appliquant la méthode d'analyse dynamique afin d'identifier les patterns de fonctions utilisés pour chaque action. Cela nécessiterait cependant de lister toutes les actions possibles et d'écrire pour chacune un nombre très important de programmes permettant de la réaliser dans le jeu. Nous avons exploré cette solution mais nous avons conclu très rapidement que la nature des traces traitées dans Prog&Play ne se prête pas à une approche *data mining* qui requiert de grandes quantités de données pour être efficiente.

Dans un second temps, nous nous sommes intéressés aux méthodes de compression de traces utilisées dans différents domaines d'application et à de nombreuses fins [3,4]. Les bénéfices recherchés par ce type d'algorithmes consistent à réduire la taille des données et à faciliter l'extraction de connaissances. La compression semble donc être une solution adaptée pour répondre à notre première question. Il convient de préciser que dans le cadre de notre travail la compression des traces vise à faciliter une analyse a posteriori. Il est donc essentiel que les traces obtenues en sortie puissent être exploitées sans qu'une étape de décompression soit nécessaire. Des algorithmes de compression traditionnels comme RLE (*Run Length Encoding*) ou DEFLATE, qui visent à réduire l'espace de stockage nécessaire quitte à rendre les données compressées inexploitables en l'état, ne conviennent donc pas.

Par ailleurs, d'autres types d'algorithmes permettent de détecter et de supprimer de façon plus spécifique les répétitions contiguës de séquences de traces dues aux boucles et aux appels récursifs présents dans un programme. Sur la base des travaux de Hamou-Lhadj et Lethbridge [5], nous avons développé une solution adaptée à la compression de traces brutes issues de l'exécution du programme informatique d'un apprenant. Une fois cette solution implémentée et testée, l'ojectif suivant était d'en extraire une sémantique

qui servira à construire un *feedback* adapté aux apprenants. Or, la liberté d'approche offerte à l'apprenant pour résoudre un niveau du jeu rend cette tâche complexe. Nous avons donc cherché des solutions qui, tout en restant génériques, permettent d'injecter aisément des connaissances expertes sur un niveau particulier afin d'améliorer la qualité des feedbacks générés.

Concernant cette deuxième question de recherche, plusieurs approches symboliques ou numériques ont été proposées dans la littérature pour répondre à des problématiques similaires à la notre. Dans [6], une ontologie est utilisée pour inférer des connaissances à partir d'une séquence d'observés. Les traces recueillies sont ainsi traduites en une suite d'événements eux-mêmes associés aux concepts d'une ontologie. Cette approche symbolique a donc l'avantage d'offrir la possibilité de raisonner sur une trace grâce à un moteur d'inférence. Dans le cas de Prog&Play, les classes et les relations qui décrivent les éléments présents dans l'univers du jeu peuvent être facilement définies (par exemple, la classe *Unité* est la classe mère des classes *Assembleur*, *Octet*, *Bit*, etc.). Il serait donc possible de créer une base de faits (des triplets RDF) à partir d'une trace brute. Cependant, cette approche comporte également de nombreuses difficultés : Comment construire une ontologie modélisant (avec différents niveaux d'abstraction) toutes les actions programmables par le joueur pour résoudre une mission ? Comment découper la trace compressée en une suite d'événements et surtout comment associer un événement à une classe de l'ontologie à partir des traces qui lui sont associées ? Toutes ces questions nous ont amenés à conclure que l'application de cette approche à notre contexte serait trop complexe. Nous nous sommes donc orientés vers une approche basée sur l'alignement de deux traces compressées : celle du joueur et celle d'un expert qui servira de trace référence pour le niveau joué.

En bio-informatique, différents algorithmes sont utilisés pour aligner deux (ou plusieurs) séquences de macromolécules biologiques (ADN, ARN ou protéines) dans le but d'identifier des régions homologues ou similaires. L'alignement par paires (deux séquences uniquement) peut être : local (une sous-chaîne de la première séquence est aligné avec une sous-chaîne de la seconde) ; semi-global (une séquence est alignée avec une sous-chaîne de l'autre séquence) ; ou global (l'alignement s'effectue sur toute la longueur des séquences). L'algorithme de Smith et Waterman [7] permet de trouver un alignement local optimal entre deux séquences tandis que l'algorithme de Needleman et Wunsch [8] permet de déterminer un alignement global ou semi-global optimal en temps polynomial.

Ces deux algorithmes reposent sur le paradigme de la programmation dynamique qui est une méthode permettant de résoudre des problèmes d'optimisation. Cette méthode est elle-même basée sur le *principe d'optimalité de Bellman* qui stipule que toute solution optimale s'appuie elle-même sur des sous-problèmes résolus localement de façon optimale. Il est donc possible de déduire la (ou une) solution optimale d'un problème en combinant des solutions optimales d'une série de sous-problèmes. Nous avons adapté l'algorithme de Needleman-Wunsch pour réaliser un alignement semi-global de deux traces compressées (celle du joueur et celle d'un expert) et détecter des écarts entre ces deux traces. Il est alors possible de fournir un feedback à l'apprenant qui lui permettra d'identifier les erreurs présentes dans son programme.

3 Conclusion

Le travail de recherche abordé dans le cadre de cet article comporte deux questions de recherche principales: d'une part, la reconstitution de la sémantique des programmes des apprenants en se basant sur leurs traces d'exécution et d'autre part, l'utilisation des traces reconstituées pour générer automatiquement des feedbacks aux apprenants sur la qualité des leurs programmes.

Pour répondre à la première question, nous nous sommes inspirés du travail de Hamou-Lhadj et Lethbridge [5] pour mettre au point un algorithme capable de compresser une trace brute produite lors de l'exécution d'un programme faisant appel à la bibliothèque Prog&Play et conforme au modèle de traces que nous avons défini. Malgré l'information limitée disponible, nous avons constaté qu'il était possible de reconstituer dans certains cas (même complexes) la structure du programme d'origine.

En ce qui concerne la seconde question, nous avons développé un algorithme d'alignement de traces compressées adapté de celui de Needleman et Wunsch [8]. Cet algorithme est utilisé pour identifier parmi un ensemble de traces compressées expertes du niveau joué celle qui permet d'obtenir le meilleur alignement avec la trace compressée de l'apprenant. Grâce à cet alignement, il est ainsi possible d'identifier les points de divergences entre les deux structures de programme reconstituées afin de générer automatiquement un *feedback* à destination de l'apprenant. La qualité de ce *feedback* repose bien évidemment sur la qualité de la compression des traces d'exécution, réalisée de façon heuristique, mais également sur le niveau de contribution des experts qui ont la possibilité d'injecter des informations sémantiques pouvant améliorer la fiabilité du processus d'analyse .

References

1. Mathieu Muratet. *Conception, réalisation et évaluation d'un jeu sérieux de stratégie temps réel pour l'apprentissage des fondamentaux de la programmation*. PhD thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier, 2010.
2. Hossein Safyallah and Kamran Sartipi. Dynamic analysis of software systems using execution pattern mining. In *14th IEEE International Conference on Program Comprehension (ICPC'06)*, pages 84–88. IEEE, 2006.
3. EN Elnozahy. Address trace compression through loop detection and reduction. In *ACM SIGMETRICS Performance Evaluation Review*, volume 27, pages 214–215. ACM, 1999.
4. Martin Burtscher, Ilya Ganusov, Sandra J Jackson, Jian Ke, Paruj Ratanaworabhan, and Nana B Sam. The vpc trace-compression algorithms. *IEEE Transactions on Computers*, 54(11):1329–1344, 2005.
5. Abdelwahab Hamou-Lhadj and Timothy C Lethbridge. Compression techniques to simplify the analysis of large execution traces. In *Program Comprehension, 2002. Proceedings. 10th International Workshop on*, pages 159–168. IEEE, 2002.
6. Olivier Georgeon. Analyse de traces d'activité pour la modélisation cognitive: application à la conduite automobile [pdf]. 2008.
7. Temple F Smith and Michael S Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.
8. Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.