HAL
open science

Utilisation de traits sémantiques pour une méthodologie de construction d'un système d'aide dans un EIAH de l'algorithmique

Denis Bouhineau

To cite this version:

Denis Bouhineau. Utilisation de traits sémantiques pour une méthodologie de construction d'un système d'aide dans un EIAH de l'algorithmique. EIAH 2013 - 6e conférence sur les Environnements Informatiques pour l'Apprentissage Humain, May 2013, Toulouse, France. pp.141-152. hal-00862437

HAL Id: hal-00862437

https://hal.science/hal-00862437v1

Submitted on 16 Sep 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Utilisation de traits sémantiques pour une méthodologie de construction d'un système d'aide dans un EIAH de l'algorithmique

Travail sur les notions de jeux d'essai et de contre-exemples dans EDBA

Denis Bouhineau*

** Laboratoire d'Informatique de Grenoble (LIG)*
Université de Grenoble (Grenoble-I, Univ. J. Fourier)
Denis.Bouhineau@imag.fr

RÉSUMÉ. *Le recours à de « simples tests » sur des jeux d'essai pour estimer la correction d'un programme est une méthode largement répandue dans l'enseignement de l'algorithmique et dans les EIAH du domaine. L'usage habituel (hors EIAH) pour l'élaboration de ces jeux d'essai s'appuie sur la spécification de la fonction à réaliser et les structures des données utilisées. Pour une approche EIAH, il est proposé de prendre en compte également les erreurs habituelles et celles induites par l'exercice lui-même pour définir ces jeux d'essai. Pour cela, cet article propose un outil sémantique, basé sur les notions de jeux d'essai et de contre-exemples, qui permette une caractérisation simple de tous les programmes : les corrects (vis-à-vis de la résolution d'un problème donné) et les autres. Reposant sur cet outil, une méthodologie de définition incrémentale des jeux d'essai et des contre-exemples est proposée. Elle permet de prolonger les systèmes de test des exercices en systèmes de diagnostic des erreurs et d'aide à la correction de ces erreurs. L'observation d'une utilisation ponctuelle de cette méthodologie dans un enseignement standard de licence d'informatique donne des éléments de validation pratique de la méthode.*

MOTS-CLÉS : *aide, algorithmique, diagnostic, edba, erreur, jeux d'essai, sémantique, contre-exemple.*

Introduction

L'algorithmique et la programmation sont deux disciplines importantes des cursus universitaires en informatique pour lesquelles divers types d'évaluation des productions d'étudiants sont possibles : évaluation des modélisations informatiques des situations proposées ; évaluation de la correction, de la complétude, de l'efficacité des solutions programmées ; évaluation du style ou de la forme des programmes proposés ; évaluation de l'argumentaire autour des programmes pour en expliquer le fonctionnement, la correction, la complétude. Autant de formes d'évaluation que le domaine –l'informatique au sens large, allant au-delà du contexte universitaire- possède de critères d'évaluation. Dans le cadre universitaire, ces évaluations peuvent relever d'enjeux multiples et appartenir en conséquence à différents contextes : évaluation sommative en fin de semestre, évaluation formative en cours de formation, évaluation par compétences ponctuellement, pour ne citer que les principaux modes d'évaluation. Une étude plus détaillée de l'évaluation dans l'enseignement peut être trouvée dans [SCALLON 04]. L'automatisation de l'ensemble de ces types d'évaluation, dans chacun des contextes universitaires habituels pour la réalisation d'un EIAH complet de l'algorithmique n'est pas à l'ordre du jour.

Dans l'informatique comme industrie, l'évaluation correspond au test logiciel et vise à vérifier la conformité des dits logiciels aux cahiers des charges qui leur sont associés. Cela s'opère le plus souvent en deux temps ; le premier consiste à rédiger un cahier de tests comportant des scénarios d'utilisation du logiciel avec les comportements attendus ; le second temps consiste à vérifier, à la main quand cela n'est pas automatisable, que le logiciel se comporte de manière conforme aux scénarios décrits. Une typologie des tests pour l'informatique est disponible dans [LEGEARD et al. 09]. Considérés selon le contexte universitaire, ces tests s'apparentent à l'évaluation de la correction, de la complétude et de l'efficacité des solutions proposées. C'est essentiellement une évaluation de la sémantique des programmes, même si parfois des métriques sont associées aux codes (taille du code, des commentaires ; taux de code dupliqué) pour évaluer certains aspects syntaxiques.

Pour beaucoup d'EIAH de l'algorithmique ou de la programmation, l'évaluation reprend un schéma similaire d'évaluation sémantique par scénarios ou jeux d'essai, pour une raison simple [DOUCE et al. 05] : une fois la rédaction des scénarios d'usage effectuée, pour des tests simples, l'automatisation de la vérification de la conformité des programme proposés au scénario attendu est relativement facile. À l'heure de gloire de l'intelligence artificielle [JOHNSON & SOLOWAY 85], des évaluations cognitives automatisées ont été pratiquées, mais leur coût a toujours été vu comme un frein [SELF 88]. Parfois, ailleurs, une évaluation mêlant sémantique et syntaxe [SKUPIENE 10] est effectuée.

Probablement influencée par la pratique professionnelle, la construction des jeux d'essai dans les EIAH est souvent laissée aux soins de l'expert ou de l'enseignant concepteur des exercices. Cette construction s'appuie sur diverses formes de tests (tests de type boites noires[1] ou boites blanches[2] par exemple). La plupart de ces types de tests sont centrés sur la vérification de la correction des programmes proposés cherchant à les mettre en défaut à travers des cas généraux non triviaux ou des cas limites. Ainsi, après passation des tests, un programme est validé ou rejeté (parfois avec un taux d'acceptabilité dépendant de la proportion de jeux d'essai validés). Malgré leur grande efficacité, deux critiques au moins peuvent être formulées vis-à-vis de ces méthodes : tout d'abord, et c'est une critique

[1] Test de type « boite noire » : test proposé sans connaitre le code du programme testé ; ce test doit être conçu à partir des spécifications de la fonction à réaliser.

[2] Test de type « boite blanche » : test proposé en inspectant le code du programme testé pour en déceler la structure et produire des exécutions qui fassent fonctionner toutes les parties de ce code

récurrente dès qu'il y a un appel à un expert (on la retrouve pour l'élaboration des systèmes expert, tuteurs intelligents et même pour les systèmes à base de contraintes [OHLSSON & MITROVIC 06]), la qualité du système d'évaluation dépend beaucoup de la qualité des propositions de jeux d'essai faites par l'expert (sans pour autant que des outils lui soient fourni pour faciliter son travail ou pouvoir mesurer la qualité de ses propositions). Seconde critique importante, qui touche plus à notre domaine -celui des EIAH- les méthodes à base de jeux d'essai s'intéressent peu ou pas à un diagnostic cognitif ou sémantique des erreurs repérées par les tests en vue de leur correction. Tout au plus, la gravité des erreurs en termes de nuisance de fonctionnement est prise en compte. Ce ne sont pas des diagnostics d'erreur au sens EIAH, comme par exemple peuvent en fournir les tuteurs cognitifs [KOEDINGER & CORBETT 05] où, pour arriver à ce résultat, les erreurs sont modélisées et diagnostiquées au même titre que les connaissances (avec un coût). *S'exprime donc un besoin en EIAH pour la définition d'une méthodologie de construction de jeux d'essai qui prenne en compte les erreurs, sans reposer sur une expertise trop couteuse ou poussée et tout en conservant la simplicité de mise en œuvre des systèmes à base de jeux d'essai*. En affinant la prise en compte de l'erreur, il viendra ensuite la possibilité pour les EIAH d'exploiter cette reconnaissance des erreurs pour élaborer un modèle de l'apprenant ou un système d'aide à la correction d'erreurs.

La suite de la communication portera sur la définition d'un outil sémantique, extension de la notion de jeux d'essai, qui permette de prendre en compte la notion d'erreur. Une seconde partie concernera la définition d'une méthodologie de construction de jeux d'essai pour un système d'aide à la correction d'erreurs qui s'appuie sur cet outil sémantique. Une application pratique de cette méthodologie sera également présentée en dernière partie.

1. Notion de jeux d'essai. Limite pour le diagnostic d'erreur.

Un jeu d'essai, pour une fonction à réaliser « `f` », est la donnée d'un couple `(entrée,sortie)`, tel que `sortie=f(entrée)`. Pris isolément, un jeu d'essai ne dit pas grand-chose de la fonction « `f` », mais pour autant décrit précisément ce que la fonction « `f` » doit produire comme résultat pour la donnée `entrée`. C'est un trait sémantique ponctuel associé à « `f` ». Il est rare qu'un jeu d'essai unique puisse être considéré comme caractéristique d'une fonction. En général plusieurs jeux d'essai sont associés à une fonction. Par exemple, pour une fonction de tri croissant, les jeux d'essai suivants peuvent être considérés :

- `([ 1 ], [ 1 ])`
- `([ 1 2 3 ], [ 1 2 3 ])`
- `([ 0 2 1 4 ], [ 0 1 2 4 ])`
- `([ 1 5 2 4 3 2 6 ], [ 1 2 2 3 4 5 6 ])`

Si un algorithme ou programme « p » ne produit pas sur les entrées des jeux d'essai du tri les sorties de ces jeux d'essai, il ne peut prétendre réaliser la fonction « `tri-croissant` ». Mais s'il s'exécute correctement sur ces jeux d'essai en donnant les réponses attendues, il n'est pas certain non plus qu'il réalise la fonction. Depuis le mot célèbre et évident « Program testing can be used to show the presence of bugs, but never to show their absence! » [DIJKSTRA 69], les tests ne servent que comme moyen rapide et peu couteux d'avoir une première évaluation de la justesse d'une proposition pas un verdict absolu.

Dans la pratique, en particulier dans le domaine de l'enseignement, cette première évaluation est suffisante pour déceler la très grande majorité des algorithmes ou programmes erronés, que les erreurs portent soit sur des erreurs immédiates (mauvaise compréhension de la fonction à réaliser, erreur technique) soit sur des erreurs « rares » (cas d'entrées atypiques repérées par l'expert et évaluées dans les jeux d'essai) [BOUHINEAU & PUITG 11]. L'étude des résultats aux jeux d'essai semble même révéler une structure de treillis associée

aux erreurs ; par exemple, les programmes comportant des erreurs rares auront des comportements d'exécution proches de la correction. Mais les éléments fournis par l'analyse des résultats aux jeux d'essai ne sont pas suffisants pour aller plus loin dans l'exploitation de cette structure et diagnostiquer des erreurs : pour un ensemble de jeux d'essai donné, deux erreurs peuvent se confondre comme la pratique le montre régulièrement. Ainsi, pour le problème du tri, deux algorithmes « `place-minimum-en-première-position` » et « `place-maximum-en-dernière-position` » auront le même comportement vis-à-vis des jeux d'essai précédents, avec les deux premiers jeux d'essai validés et les deux suivant incorrects.

Nous parvenons à un double constat. En pratique, *la notion de jeux d'essai permet de valider de manière satisfaisante les exercices d'algorithmique, mais elle ne permet pas de diagnostiquer les erreurs*. Il y a pourtant là un paradoxe, diagnostiquer une erreur devrait être une opération similaire au diagnostic d'une solution. Aussi, on peut se demander s'il faut, pour arriver à diagnostiquer les erreurs, changer la notion de jeux d'essai : prendre plus de cas dans les jeux d'essai ou modifier la manière dont on utilise ces jeux d'essai.

2. Démarche systématique de test par jeux d'essai. Définition d'une sémantique observationnelle partielle

L'ensemble des jeux d'essai choisi précédemment pour l'exemple du tri était de petite taille (seulement quatre exemples). Dans un EIAH visant une pratique effective on peut imaginer que le concepteur d'exercices fournisse plus de jeux d'essai. Pour un concepteur d'exercices voulant clore son travail dans un temps raisonnable (une dizaine de minute), il semble que 7 soit un nombre de tests satisfaisant pour valider un programme (moyenne obtenue après introduction de 512 exercices sur EDBA un EIAH de l'algorithmique [BOUHINEAU 11]). Au-delà, le travail manuel devient trop important, d'autant plus s'il doit être répété pour chaque exercice et qu'il semble perdre en efficacité quand l'ensemble de jeux d'essai devient conséquent. Avec 7 tests au lieu de 4, les limites observées précédemment pour diagnostiquer des erreurs risquent d'être les mêmes. Aussi, essayons de nous passer du travail à la main du concepteur d'exercices et considérons que l'ensemble des jeux d'essai est produit automatiquement. Ainsi, il peut être de beaucoup plus grande taille. L'exécution des jeux d'essai mène-t-elle à une meilleure identification des programmes et à un diagnostic des erreurs ?

L'étude a été effectuée en prenant 140 algorithmes (constituant des problèmes donnés usuellement aux étudiants) et en produisant pour chacun une liste de jeux d'essai (en moyenne 240 jeux d'essai par algorithme) par énumération systématique des entrées possibles en partant des entrées les plus *simples* et en les associant aux sorties produites par l'algorithme sur ces entrées.

L'« énumération systématique des entrées en partant des plus simples » est un problème non trivial car c'est un problème mal posé ; que signifie « simple » ? Pour l'exemple des listes d'entiers strictement positifs, plusieurs définitions du « simple » sont possibles :

- 1ère définition : plus simple sera la liste avec la somme la plus petite. Ainsi l'énumération donne `[1], [1 1], [2], [1 1 1], [1 2], [2 1], [3], [1 1 1 1], [1 1 2], [1 2 1], [1 3], [2 1 1], [2 2], [3 1], [4], [1 1 1 1 1]`.
- 2nde définition : plus simple sera la liste dont le maximum entre la longueur et l'élément maximum de la liste sera le plus faible. Ainsi, l'énumération donne : `[1], [1 1], [1 2], [2], [2 1], [2 2], [1 1 1], [1 1 2], [1 1 3], [1 2 1], [1 2 2], [1 2 3], [1 3], [1 3 1], [1 3 2], [1 3 3]`.

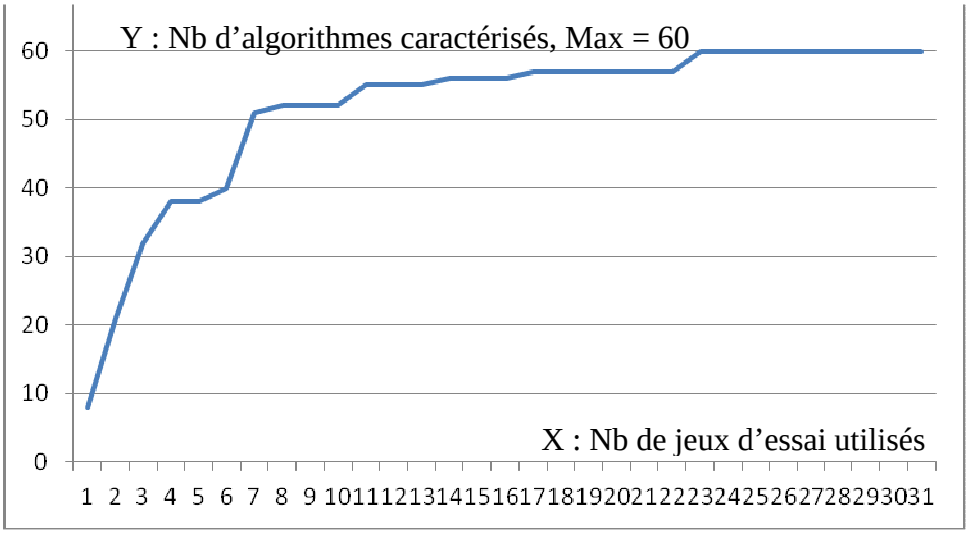

Figure 1. *Nombre d'algorithmes caractérisés en fonction du nombre de jeux d'essai utilisés pour les 60 algorithmes ayant en entrée une liste.*

Chaque énumération peut avoir des qualités et des défauts vis-à-vis de la caractérisation d'un algorithme particulier. Pour l'algorithme de tri, on peut souhaiter par exemple privilégier l'apparition des permutations sur les petits entiers. En conclusion, il n'y a pas d'énumération canonique ou idéale. Aussi, une énumération basée sur un ensemble d'énumérations automatiques a été utilisée pour comparer les 140 algorithmes[3].

Pour l'algorithme de tri, en prenant la seconde énumération, la liste de jeux d'essai produite que nous appellerons la signature de l'algorithme tri -réduite aux 16 premiers éléments- est donc : `([1], [1]), ([1 1], [1 1]), ([1 2], [1 2]), ([2], [2]), ([2 1], [1 2]), ([2 2], [2 2]), ([1 1 1], [1 1 1]), ([1 1 2], [1 1 2]), ([1 1 3], [1 1 3]), ([1 2 1], [1 1 2]), ([1 2 2], [1 2 2]), ([1 2 3], [1 2 3]), ([1 3], [1 3]), ([1 3 1], [1 1 3]), ([1 3 2], [1 2 3]), ([1 3 3], [1 3 3])`.

Plusieurs observations ont été effectuées lors de cette étude systématique au sujet de l'utilisation des jeux d'essai pour identifier un algorithme :

- Parmi les 140 algorithmes testés, ceux qui ont obtenu des ensembles de jeux d'essai identiques sont bien des algorithmes identiques (avec un nom différent). Par exemple : « `inverse-liste` » et « `retourne-liste` ». Il s'agissait donc de doublons dans la liste des 140. Ils étaient au nombre de 9 (vérification à la main).
- Parmi les réponses aux jeux d'essai, certaines sont plus courantes que d'autres. Par exemple, la liste [ ] et l'entier 0 sont des réponses courantes. Ces réponses sont donc parmi les moins discriminantes dans un jeu d'essai.
- Parmi les jeux d'essai, certains sont très courants, par exemple ([ ], [ ]), ou ([ ], rien[4]). En moyenne, étant donnée une entrée, pour le jeu d'essai le plus courant, la sortie associée à l'entrée apparait entre 8 et 19 fois (moyenne à 11 pour les listes). Ces jeux d'essai sont donc les moins discriminants.

---

[3] En outre, tous les algorithmes n'ont pas vocation à s'exécuter sur toutes les données. Outre le respect d'un profil, il existe dans certains cas des pré-conditions à respecter pour exécuter un algorithme. Ceci doit être également pris en compte dans une démarche systématique.

[4] « rien » signifie qu'il n'y a pas d'exécution possible pour l'entrée donnée, les pré-conditions de l'algorithme ne sont pas respectées.

- Certains algorithmes partagent plus de réponses avec les autres. Ainsi, l'algorithme « `identité` » est l'algorithme dont le comportement est le plus similaire aux autres parmi la trentaine de ceux dont le profil est `(entrée : liste, sortie : liste)`.
- Inversement : deux algorithmes différents ont toujours eu des comportements différents. Parmi les réponses, certaines sont rares ; parmi les algorithmes, certains sont atypiques.

Certaines conséquences peuvent être tirées vis-à-vis de l'utilisation de jeux d'essai systématiques pour l'identification d'algorithmes :

- La méthode d'identification d'un algorithme par jeux d'essai est une méthode efficace (cela confirme le résultat expérimental obtenu dans [BOUHINEAU & PUITG 11]). Pour les 60 algorithmes sur 140 qui avaient un profil avec en entrée une liste seule, une vingtaine de tests suffit pour distinguer ces 60 algorithmes (23 tests pour une énumération des listes ne cherchant pas à minimiser ce résultat) et seulement quelques tests permettent de distinguer la plupart des algorithmes, voir fig. 1 (ici, « quelques» signifie 7, comme le nombre moyen de tests « à la main »). La méthode d'identification d'un algorithme par jeux d'essai permet donc de définir un outil capturant une partie significative de la sémantique d'un algorithme à partir de l'observation de son comportement, c'est un outil de sémantique observationnelle partiel qui converge (quand l'observation s'étend à un plus grand nombre de jeux d'essai) vers un outil sémantique à part entière.

  D'autres outils sémantiques observationnels peuvent être imaginés en algorithmique. Par exemple, pour des observations plus faibles que celle donnée par l'observation des résultats de l'exécution : observation des coûts d'exécution d'un algorithme sur une donnée particulière. Hors de l'algorithmique, des outils similaires peuvent être imaginés, en géométrie ou en algèbre.

  Cet outil peut être utilisé dans un EIAH de l'algorithmique pour :

  - permettre la définition d'une base d'algorithmes interrogeable via un index sémantique. À ce jour, il n'existe d'index pour les algorithmes que prenant comme entrée le nom de l'algorithme (ce qui est peu sémantique). La mise en place d'un index sémantique a été entreprise de manière expérimentale dans EDBA selon cette méthode, sur la base des 140 exercices utilisés plus haut. Pour le jeu d'essai `(entrée : [1 2 3], sortie : [1 2 3])` en entrée de l'index, les algorithmes suivants sont identifiés : `identité`, `tri`, `enlève-0`, `supprime-doublon`, `aplatir-liste`, `double-0`, `inverse-première-décroissance`.

    En ajoutant `(entrée : [3 2 1], sortie : [1 2 3])` en entrée de l'index, le résultat est uniquement `tri`. Hors EIAH et algorithmique, ce type d'index existe déjà pour les suites numériques [SLOANE 07]. Il serait intéressant d'en définir un similaire pour les algorithmes.

    - déceler l'existence ou l'absence de doublons dans une base d'algorithmes (c'est un problème délicat quand la taille de la base est importante)
  - identifier les algorithmes utilisés dans une résolution d'un problème complexe selon une démarche top-down utilisant des sous-programmes aux noms indéterminés (reverse-engineering)

Ces outils sémantiques (index, reverse-engineering, recherche de doublons) sont importants pour les EIAH qui voudraient étendre leur utilisation au-delà d'une base de quelques exercices. Les démarches actuelles à base d'ontologies ou de taxonomies ou de mots clés réalisent difficilement ces fonctions.

- l'étude précédente montre aussi que tous les jeux d'essai ne sont pas équivalents (certains jeux d'essai sont moins discriminants) : la proportion de jeux d'essai validés par une proposition de solution donne habituellement une évaluation de la justesse de la proposition. Cette évaluation mène parfois à une note. Quelle est la justesse de cette note ? La vision globale fournie par l'étude systématique confirme l'expérience du correcteur de copie, ou de l'expert construisant son jeu d'essai : certains tests ne suffisent pas pour valider une solution ; ils ne devraient pas entrer dans le décompte d'une note, ou avec un poids très restreint. Ici l'argument n'est pas pédagogique, mais probabiliste. La chance pour que la réponse d'un algorithme à l'entrée [ ] soit la sortie [ ] est grande. L'évaluation à base de jeux d'essai devrait faire intervenir la nature des jeux d'essai. Pour s'appuyer sur cette évaluation et donner une note il faudrait procéder à une caractérisation des jeux d'essai utilisé. Sont-ils décisifs ? Importants ? Concernent-ils des comportements rares ? etc. L'apport d'une vision globale est nécessaire pour aider celui qui a pour tâche de définir ces jeux d'essai.

3. Extension de la notion de jeux d'essai, jeu d'essai positif *versus* jeu d'essai négatif, notion de contre-exemple.

Malgré les éléments de preuve apportés précédemment justifiant l'emploi de jeux d'essai pour déterminer la sémantique d'un programme (sémantique observationnelle partielle), l'utilisation de jeux d'essai risque de rester limitée au niveau de l'évaluation de la correction d'un programme et de ne pas fournir un diagnostic de l'erreur tant que la notion d'erreur n'est pas introduite. En première approche, nous disions que diagnostiquer une erreur est une opération similaire à identifier une solution. Pour diagnostiquer les erreurs il faudrait donc que la base des algorithmes de référence comporte les algorithmes erreurs attendus. Cela amène plusieurs remarques bien connues [SUAREZ & SISON 08] :

- en pratique, les erreurs sont beaucoup plus nombreuses que les solutions (en particulier parce que les erreurs peuvent se combiner). Ne risque-t-on pas une explosion de la taille de la base d'algorithmes (et du temps de calcul des évaluations) ?
- l'inventaire des erreurs n'est pas aisé. Les étudiants arrivent régulièrement à étonner leurs enseignants. La base risque donc d'être incomplète.
- mélanger algorithmes-solution et algorithme-erreur dans une base d'algorithmes peut être délicat. Ce statut « erreur » / « solution » n'est pas absolu, il est défini par rapport à un problème algorithmique donné. Faudrait-il définir autant de bases d'algorithme que d'exercices ?

Revenons à la définition de jeux d'essai et tentons un lien avec l'erreur via la notion de contre-exemple. Un jeu d'essai est un exemple de fonctionnement correct : lorsqu'un jeu d'essai ($\texttt{entrée}, \texttt{sortie}_{\texttt{pour_f}}$) défini pour une fonction à réaliser « `f` » n'est pas validé par une proposition d'algorithme « `P` », on peut en déduire que la proposition d'algorithme « `P` » ne réalise pas « `f` », mais il est difficile de dire quelle erreur a été commise, sauf à pouvoir faire un test positif de « `P` » vis-à-vis d'un algorithme « `erreur` » (ce que l'on essaie d'éviter, ne sachant comment avoir les algorithmes « `erreur` »). Inversons les valides et les invalides, les corrects et les incorrects, les positifs et les négatifs, les exemples et les contre-exemples.

Imaginons qu'il y ait également des jeux d'essai négatifs, c'est-à-dire des couples (entrée, sortie), définis pour une fonction à réaliser « f » comme des contre-exemples de bon fonctionnement de « f », c'est à dire tels que par définition sortie≠f(entrée). Nous noterons ces jeux d'essai négatifs ~(entrée, sortie). Choisissons de définir un jeu d'essai négatif pour « f » à partir d'un algorithme « erreur » donné : ~(entrée, $sortie_{erreur}$)défini avec $sortie_{erreur}$ = erreur(entrée) ≠ f(entrée). Reprenons, notre exemple avec « P ». Si « P » réussit le contre-exemple, on ne peut dire grand-chose sur le fait qu'il se comporte comme « f » sur « entrée », mais au moins il ne produit pas « $sortie_{erreur}$ » qui est assurement faux. Par contre, si « p » échoue sur le jeu d'essai négatif cela signifie que « P(entrée) ≠ $sortie_{erreur}$ » est faux, donc que « P(entrée) = $sortie_{erreur}$ », c'est-à-dire que « P » se comporte comme « erreur ». Le test négatif de « P » sur ~(entrée, $sortie_{erreur}$) pour « f » est équivalent à un test positif de « P » sur (entrée, $sortie_{erreur}$) pour « erreur ». Il tend à identifier « P » comme l'algorithme « erreur ».

Pour le tri croissant, si l'on souhaite diagnostiquer les erreurs d'algorithme qui donnent un tri décroissant, l'ensemble de jeux d'essai peut donc être le suivant :

- ([1], [1])
- ([1 2 3], [1 2 3])
- ~([1 2 3 4 5 6], [6 5 4 3 2 1])

En reprenant les arguments montrant le pouvoir des tests d'identification par jeux d'essai positifs, on arrive à montrer la capacité des tests par jeux d'essai négatifs (ou « contre-exemple ») pour identifier les erreurs. Ainsi, il est raisonnable de penser que l'algorithme qui échouera au dernier jeu d'essai ~([1 2 3 4 5 6], [6 5 4 3 2 1]) sera un algorithme qui aura confondu tri-croissant et tri-décroissant. Mais attention, l'utilisation de contre-exemple est à double tranchant : s'ils permettent d'identifier les erreurs, ils sont assez faibles pour identifier les solutions. Quand un programme réussit à un test négatif, cela ne dit pas grand-chose sur la question de la correction vis-à-vis de la solution. Sur l'exemple précédent, l'algorithme « identité » réussirait les trois tests. De plus un jeu d'essai négatif ne permet a priori d'identifier qu'une seule erreur et la construction de ce jeu d'essai s'est opérée en utilisant un algorithme erreur spécifique. Le problème de l'identification des erreurs est-il clos ? A-t-on vraiment avancé ? Oui si l'on considère que le contre-exemple peut exister sans avoir conceptualisé explicitement la notion d'erreur associée. Ce qui est effectivement possible. Cependant, pour que ces jeux d'essai négatifs soient parlants il faut que la référence à l'erreur soit explicitée à un moment. En soi, sans cette association à une erreur répertoriée, ces jeux d'essai négatifs risquent d'être peu utiles. Il semble donc qu'une intervention humaine soit encore nécessaire pour définir les jeux d'essai (positifs ou négatifs) et les caractériser. Une méthodologie reste à définir.

4. Méthodologie de construction dynamique et incrémentale de jeux d'essai pour un système d'aide à la correction d'erreurs.

Mettre en place un système à base de connaissances est un problème délicat. A priori, trois méthodes au moins peuvent coexister pour la définition de jeux d'essai (positifs et négatifs) : une méthode d'énumération systématique, une méthode utilisant une connaissance experte du domaine, une méthode reposant sur l'observation des productions d'étudiants par les enseignants et concepteurs d'exercices.

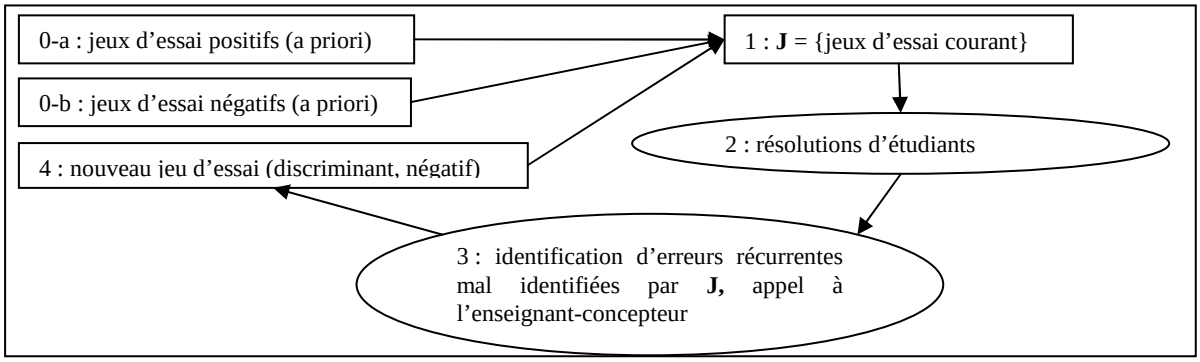

Figure 2. *Production dynamique d'un ensemble de jeux d'essai.*

L'énumération systématique est celle proposée dans la première partie. Elle a pour avantage d'être sûre et complète et donc de pouvoir apporter une grande aide à l'expert introduisant un exercice en lui épargnant une partie du travail, mais elle suppose la connaissance des algorithmes à identifier. Son intérêt est donc limité aux jeux d'essai positifs (l'algorithme à identifier étant la solution du problème). Le coût en termes de temps d'exécution et le manque d'expressivité ou de significativité d'une partie des jeux d'essai limitent cependant l'utilisation de cette méthode. Pour les jeux d'essai négatifs, les algorithmes à identifier sont trop nombreux, ce sont les algorithmes-erreurs ; le coût humain pour les identifier et les définir explicitement (les programmer) est trop important.

L'élaboration d'un ensemble de jeux d'essai par un expert semble la méthode la plus convaincante. Des jeux d'essai positifs significatifs et expressifs peuvent être définis par l'expert. La connaissance du domaine, et en particulier des erreurs typiques, permet à l'expert de définir également des jeux d'essai négatifs (i.e. contre-exemple) identifiant les erreurs les plus courantes. Toutefois, le coût peut être élevé. Par ailleurs, sans une étude plus approfondie de l'exercice, les erreurs spécifiques de l'exercice ne seront pas repérées. Enfin, il est probable qu'une part du temps consacré à la définition des contre-exemples soit inutile (les erreurs typiques ne sont pas nécessairement toujours présentes). Cette méthode ne peut donc être considérée comme idéale.

L'observation des premières résolutions d'étudiants pour identifier les erreurs récurrentes issues d'un exercice est à la base de la méthodologie de construction dynamique et incrémentale des jeux d'essai proposée dans cette partie. Ce retour venant du terrain nous semble indispensable pour construire un système d'évaluation d'exercices et réduit de beaucoup l'intervention de l'expert.

Dans une phase initiale, cf. fig. 2, un premier ensemble de jeux d'essai (positifs et négatifs) est produit par l'enseignant concepteur de l'exercice. Cet ensemble peut être limité et incomplet, il peut correspondre à l'ensemble de jeux d'essai habituellement utilisé dans les systèmes à base de tests. Certaines erreurs peuvent être indiscernables pour cet ensemble de jeux d'essai initial. Dans une seconde phase, la résolution de l'exercice par les étudiants donne des algorithmes (justes ou erronés) concrets sur lesquels une analyse automatique plus fine de la sémantique peut être effectuée. Cette étude des résolutions d'étudiants peut se prolonger avec l'identification automatique d'erreurs récurrentes mal repérées par l'ensemble de jeux d'essai initial. Quand cela arrive, une troisième phase est mise en œuvre avec un appel à l'enseignant concepteur de l'exercice pour que l'étude des résolutions concrètes explicite les erreurs actives commises dans la résolution de l'exercice et entraine l'introduction de nouveaux jeux d'essai discriminant spécifiques de ces erreurs. Les deux dernières phases peuvent être répétées tant que de nouvelles erreurs récurrentes se manifestent. L'ensemble des algorithmes-erreurs des étudiants, en particulier ceux représentatifs d'une erreur récurrente, est conservé tout le long de ce processus, permettant

l'élaboration a posteriori d'un catalogue des erreurs associées à l'exercice particulier et permettant de vérifier que les nouveaux programmes sont bien identifiés.

En parallèle de la construction dynamique et incrémentale de l'ensemble des jeux d'essai et de la base d'algorithmes erreur, un système d'aide à la correction d'erreur peut être mis en place en associant à chaque erreur repérée un message de diagnostic et d'aide spécifique (voir ci après).

5. Observation sur un cas pratique

La méthodologie de construction dynamique et incrémentale de jeux d'essai a été utilisée de manière informelle et expérimentale en septembre-novembre 2012 pour un ensemble de 28 exercices parmi 512, sur lesquels 18 étudiants ont travaillé (sur la base du volontariat parmi 36 étudiants). Par ailleurs, le dispositif sollicitait les étudiants lorsqu'ils passaient d'une résolution identifiée comme fausse à une résolution identifiée comme juste afin qu'ils rédigent un message d'aide « ***Bravo ! Vous avez bien progressé depuis la dernière tentative de résolution. Vous aviez une erreur ? Qu'avez-vous fait ? Quelle indication donneriez-vous à quelqu'un dans une situation similaire*** *? Ne soyez pas trop explicite : ne donnez pas la solution et pensez que celui qui vous lira ne connaitra peut-être pas votre code.* ». L'hypothèse poursuivie était que l'indication pouvait servir à la construction automatique d'un mécanisme d'aide à la correction d'erreur. Pour éviter de solliciter les utilisateurs trop souvent, lorsque le système comportait des indications pour une situation donnée, la sollicitation était faite avec une probabilité plus faible. Dans un dispositif plus complet, il est imaginé également que le concepteur de l'exercice puisse à tout moment observer les tentatives de résolution de son exercice, ajouter des indications relatives à une résolution observée et consulter/modifier les indications enregistrées pour son exercice par les apprenants.

Pour les 11 exercices les plus travaillés, les étudiants ont produit plus de 500 propositions de solution par exercice (en moyenne 60 propositions, maximum 194, minimum 15) et ont rédigé 30 propositions d'indication. Seulement 30 indications pour l'aide à la correction d'erreur contre plus de 500 propositions de résolution, cela peut sembler très peu ; en fait, parmi les 500 tentatives de résolutions, la plupart correspondent à des propositions fausses et n'ont pas déclenché une sollicitation d'indication. Relativement aux sollicitations, le nombre d'indications proposées par les apprenants est proche, en proportion d'un taux de réponse courant pour des enquêtes électronique : 15% à 30%.

Relativement à l'utilisation des jeux d'essai négatifs :

- Les ensembles de jeux d'essai initiaux (construits sans une grande analyse a priori, avec jeux d'essai positifs et négatifs) sont insuffisants pour l'identification de l'ensemble des erreurs.
  Prenons l'exemple d'un jeu d'essai négatif particulier, pensé pour une erreur E_1, ayant déclenché six diagnostics d'erreur. Dans trois cas, l'erreur constatée était effectivement E_1 ; dans les trois autres cas, l'erreur observée était différente, E_2. Ce nouveau cas d'erreur a été pris en compte avec un jeu d'essai négatif supplémentaire, ainsi les six diagnostics ont été corrects. L'ensemble de jeux d'essai prévoyait trois erreurs, il s'est avéré qu'il y en avait cinq (récurrentes).
- L'identification a priori des erreurs est utile, mais ne garantit pas l'apparition effective de ces erreurs. Dans deux tiers des cas seulement, les erreurs prévues ont été effectivement repérées dans les propositions de solution d'étudiants.

- Le diagnostic des erreurs est coûteux. Dans deux tiers des cas, les jeux d'essai négatifs sont activés rarement (<5%), ce qui confirme leur aspect pointu, mais avec un spectre étroit.

Relativement aux 30 propositions d'indications :

- Leur répartition indique une qualité moyenne/bonne de ces indications
  - 10 étaient vagues (« Bien lire l'énoncé », « Penser récursivité », …),
  - 9 étaient pertinentes (« La liste vide tu n'oublieras point », « 'charles' est une constante », …),
  - 4 étaient correctes mais peu utiles,
  - 3 étaient fausses ou pouvaient être mal interprétées.
- Si l'on observe les utilisateurs ayant fait ces indications, plus de la moitié des étudiants ayant utilisé le système ont répondu au moins une fois aux sollicitations mais rares sont ceux qui ont répondu plus de 2 fois. L'effectif étant faible, l'analyse prévue des profils des répondants (sur leur niveau d'expertise dans le système et la qualité de leurs indications) n'a pas été menée.

Conclusion

Plusieurs pistes de travail ont été explorées vis-à-vis de l'utilisation de jeux d'essai pour un EIAH de l'algorithmique : d'une part la mise en place d'énumérations automatiques et systématiques de jeux d'essai positifs permettant la définition d'une sémantique observationnelle partielle puissante dont les usages restent à définir ; d'autre part la définition de jeux d'essai négatifs étendant la notion de jeux d'essai et permettant l'identification d'erreurs spécifiques ; enfin la définition d'une méthodologie de construction d'ensembles de jeux d'essai associés à un système de diagnostic et d'aide. Les deux dernières pistes sont particulièrement tournées vers l'EIAH. La dernière suggère la construction de systèmes d'aide principalement autorégulés par les apprenants, avec une aide modérée d'enseignants concepteurs d'exercices. C'est une piste prometteuse reposant sur les technologies actuelles, privilégiant l'interactivité, la définition dynamique et incrémentale de systèmes logiciels ainsi que la collaboration entre apprenants et enseignants, piste qu'il serait intéressant d'étendre à d'autres domaines.

Références

[BOUHINEAU 11] Bouhineau, D. « *AJAX pour EIAH ? Choix technologique pour un EIAH de l'algorithmique : EDBA », Actes de la conférence EIAH'11*, Edition de l'UMONS, isbn 978-2-87325-061-4, Univ. Mons, Belgique, 2011.

[BOUHINEAU & PUITG 11] Bouhineau, D., Puitg, F., *« Évaluations de solutions d'exercices d'algorithmique . À la main versus automatiques par jeux d'essai. », Actes de la conférence EIAH'11*, Edition de l'UMONS, isbn 978-2-87325-061-4, Univ. Mons, Belgique, 2011.

[DIJKSTRA 69] Dijkstra, E. W., Notes on structured programming, Technological University Eindhoven, Netherlands Department of Mathematics, 1969.

[DOUCE et al. 05] Douce, C., Livingstone, D., Orwell, J., *« Automatic test-based assessment of programming: A review », Journal on Educational Resources in Computing*, (5)3, 2005.

[JOHNSON & SOLOWAY 85] Johnson, W.L., Soloway, E., *« PROUST: Knowledge-Based Program Understanding », IEEE Transaction on Software Engineering*, (SE-11)3, 1985.

[KOEDINGER & CORBETT 06] Koedinger, K. R., Corbett, A., *« Cognitive Tutors: Technology Bringing Learning Science to the Classroom. »*, The Cambridge handbook of the learning sciences, 2006.

[LEGEARD et al. 09] Legeard, B., Bouquet, F., Pickaert, P, *Industrialiser le test fonctionnel : des exigences métier au référentiel de tests automatisés*, Dunod, Paris, 2009.

[OHLSSON & MITROVIC 06] Ohlsson S., Mitrovic, A., *«Constraint-Based Knowledge Representation for Individualized Instruction»*, Computer Science and Information Systems, (3)1, 2006.

[SCALLON 04] Scallon, G., *L'évaluation des apprentissages dans une approche par compétences*, Edition de Boeck Université, Bruxelles, 2004.

[SELF 88] Self, J., *«Bypassing the intractable problem of student modelling»*, ITS 1988, Ablex, New Jersey, 1988.

[SKUPIENE 10] Skupiene, J., *«Improving the evaluation model for the lithuanian informatics olympiads»*, Informatics in education (9)1, Vilnius, 2010.

[SLOANE 07] Sloane, N. J. A., *«The On-Line Encyclopedia of Integer Sequences»*, LNCS 4573, Towards Mechanized Mathematical Assistants, Springer Berlin Heidelberg, 2007.

[SUAREZ & SISON 08] Suarez, M., Sison, R., *«Automatic Construction of a Bug Library for Object-Oriented Novice Java Programmer Errors»*, LNCS 5091, ITS'08, Montreal, 2008.