Pour une transposition de la grammaire formelle d'un langage de programmation en un outil et objet d'apprentisage

Patrick Wang[1][0000−0003−3117−8189] and Sébastien Jolivet[2,3][0000−0003−3915−8465]

[1] Haute école pédagogique du canton de Vaud, Lausanne, Suisse
patrick.wang@hepl.ch
[2] IUFE, Université de Genève, Genève, Suisse
[3] LDAR, Université de Paris, France
sebastien.jolivet@unige.ch

Résumé Cet article s'intéresse aux difficultés que peuvent rencontrer les élèves lors de l'étape d'implémentation d'un algorithme avec un langage de programmation. Celles-ci peuvent être expliquées par une trop grande différence de niveau de précision des instructions entre le pseudo-code ou logigramme utilisé pour décrire l'algorithme et le langage utilisé pour l'implémenter, une mauvaise maîtrise de la syntaxe du langage de programmation utilisé, ou tout simplement des connaissances disciplinaires lacunaires. Pour pallier ces difficultés, nous proposons d'analyser, sous l'angle de la dialectique outil-objet, les avantages de l'enseignement des règles de la grammaire formelle du langage de programmation utilisé par les élèves du secondaire. Ainsi, les règles de grammaire en tant qu'outils peuvent aider les élèves à écrire des programmes syntaxiquement corrects. Et dans le même temps, ces règles en tant qu'objets de savoir peuvent être appliquées pour mettre en avant des constructions algorithmiques classiques pour résoudre des problèmes usuels.

Keywords: Algorithmique · Programmation · Grammaire formelle · Dialectique outil-objet · Didactique de l'informatique

1 Introduction

La science informatique s'inscrit dans les programmes scolaires de nombreux pays en Europe. À travers cette discipline, des thématiques liées aux quatre dimensions identifiées par Dowek [3] (algorithme, machine, langage et information) sont enseignées. Chacune de ces thématiques présente des difficultés d'apprentissage spécifiques, et nous nous focalisons dans cet article sur celles rencontrées par les élèves lors de la conception puis de l'implémentation d'un algorithme avec un langage de programmation donné.

Nous mettons en évidence plusieurs explications possibles aux difficultés en lien avec le passage d'un algorithme à un programme. Comme piste d'exploration, nous proposons d'enseigner les règles de syntaxe d'un langage de programmation sous l'angle de la dialectique outil-objet [2]. Puis, nous interrogeons la

pertinence et l'intérêt pédagogique de cette approche dans le cadre de l'enseignement de la science informatique au secondaire.

2 Conception et implémentation d'algorithmes

De nombreuses définitions existent pour caractériser ce qu'est un algorithme. Dans la suite de cet article, nous utilisons celle proposée par Gérard Berry : "Un algorithme est un procédé conceptuel d'arrangement en séquence ou en parallèle de très nombreuses opérations individuellement très simples pour atteindre un but précis, lui arbitrairement complexe" [4]. Cette définition fait apparaître les notions d'*opérations* et de *simplicité*, mais aucune caractérisation d'une *opération simple* n'est donnée.

Lors de la conception d'un algorithme, de telles opérations sont souvent décrites par du pseudocode ou un logigramme. Mais le niveau d'abstraction adopté par l'une ou l'autre de ces représentations peut être significativement plus élevé que celui utilisé par les langages de programmation communément enseignés au secondaire. Cette différence de niveau constitue précisément une première explication aux difficultés rencontrées par les élèves lors du passage d'un algorithme au programme. Pour illustrer notre propos, nous pouvons considérer la permutation de deux valeurs stockées dans deux variables avec les pseudocodes suivants qui peuvent être considérer comme équivalents :

```
permuter a et b
```

```
temp <- a
a <- b
b <- temp
```

Le pseudocode sur la gauche adopte un niveau d'abstraction élevé qui explicite le résultat de l'instruction tandis que celui de droite se rapproche du niveau des instructions d'un langage de programmation. En particulier, la variable temporaire de stockage y est explicitée, et sans connaissance préalable de la nécessité de cette variable, l'implémentation du pseudocode de gauche peut s'avérer difficile pour un élève.

Cet exemple fait écho au besoin exprimé par Du Boulay d'enseigner des *structures*, c'est-à-dire des "patrons ou méthodes qui peuvent être utilisés pour atteindre des objectifs à petite échelle, comme par exemple le calcul d'une somme d'entiers en utilisant une boucle" [4, p.54]. Ainsi, une seconde explication possible aux difficultés rencontrées par les élèves pendant l'implémentation d'un algorithme concerne un enseignement peut-être lacunaire de ces structures.

Enfin, une dernière difficulté dans l'implémentation d'un algorithme concerne la maîtrise de la syntaxe du langage de programmation utilisé. En effet, la littérature à ce sujet montre que les erreurs de syntaxe comptent parmi les erreurs les plus communément répertoriées dans les travaux des élèves [1].

4. Définition formulée en introduction de la leçon inaugurale de Claire Mathieu au Collège de France (3min39) : `https://www.college-de-france.fr/site/claire-mathieu/inaugural-lecture-2017-11-16-18h00.htm`

Pour aider les élèves à surmonter les difficultés mises en avant dans cette section, nous proposons de considérer la grammaire formelle d'un langage comme objet de savoir à enseigner, proposition explicitée dans la section suivante.

3 Enseigner la grammaire formelle d'un langage

3.1 Définition et exemple de la grammaire de Python

La grammaire d'un langage de programmation regroupe un ensemble de règles non-ambigües permettant de dire si les lignes de code qui constituent un programme sont syntaxiquement valides ou non. Sans connaissance de ces règles, il est virtuellement impossible pour un élève d'implémenter un algorithme dans un langage de programmation donné (voire même de le déboguer).

Pour illustrer notre propos, nous allons prendre l'exemple du langage Python dont la grammaire est disponible en ligne dans son intégralité [5]. Cette grammaire fait apparaître trois règles importantes dont découlent les concepts qu'un élève de secondaire doit apprendre : `statement`, `expression`, et `atom`.

La règle `statement` spécifie la syntaxe d'une instruction "simple" ou "composée", dont font partie les affectations de variables, les structures de contrôle, et les définitions de fonctions. La règle `expression` spécifie la syntaxe de toutes les opérations calculant une valeur littérale. Nous y retrouvons donc les opérations arithmétiques, de comparaison et booléennes, mais aussi les appels de fonctions (puisqu'elles retournent par défaut la valeur `None`). Enfin, la règle `atom` spécifie la syntaxe d'un identifiant ou d'une valeur littérale telle que les valeurs numériques, booléennes ou les chaînes de caractères. Python y inclue également les structures de données usuelles que sont les tuples, listes, dictionnaires et ensembles.

On constate que ces trois règles de grammaire recouvrent une grande partie des concepts à enseigner. Pourtant, les règles elles-mêmes n'apparaissent pas dans les programmes ou plans d'études. La section suivante argumente pour un enseignement explicite de ces règles s'appuyant sur une dialectique outil-objet.

3.2 Grammaire formelle comme outil et objet d'apprentissage

La définition de la dialectique outil-objet donnée par Douady s'inscrit dans un contexte mathématique [2], mais reste pertinente si transposée au contexte de l'informatique. Dès lors, "pour un concept [informatique], il convient de distinguer son caractère "outil" et son caractère "objet". Par outil nous entendons son fonctionnement scientifique dans les divers problèmes qu'il permet de résoudre. Par objet, nous entendons le concept [informatique], considéré comme objet culturel ayant sa place dans un édifice plus large qui est le savoir savant à un moment donné, reconnu socialement" [2, p.9-10].

Pour illustrer notre propos, nous pouvons considérer la règle `if_stmt` (dans cet exemple, `named_expression` est assimilable à une expression et `block` correspond au corps de l'instruction conditionnelle avec indentation) :

5. `https://docs.python.org/3/reference/grammar.html`

```
if_stmt:
  | 'if' named_expression ':' block elif_stmt
  | 'if' named_expression ':' block [else_block]
```

En tant qu'objet de savoir, cette règle permet d'expliciter qu'une instruction conditionnelle attend une expression dont l'évaluation résulte en une valeur booléenne. Par extrapolation, nous pourrions même ajouter que cette règle laisse entrevoir qu'il existe en Python une relation entre valeurs booléennes et numériques. En tant qu'outil, nous pouvons envisager de lister l'ensemble des valeurs booléennes pouvant être implémentées. Par exemple, et uniquement avec les opérateurs arithmétiques et de comparaison, un élève peut construire un test d'égalité entre deux littéraux, déterminer un extremum entre deux valeurs, déterminer l'ordre alphabétique entre deux chaînes de caractères, ou encore déterminer si un entier est divisible par un autre. Ces exemples montrent en quoi l'enseignement des règles de syntaxe peuvent aider les élèves à mieux comprendre le langage de programmation qu'ils utilisent ainsi qu'à faire face à des problèmes algorithmiques classiques.

4 Conclusion

Dans cet article, nous avons proposé de considérer les règles syntaxiques comme outil et objet d'apprentissage pour aider les élèves dans la conception et l'implémentation d'algorithmes. Ces règles ne sont pour le moment pas enseignées et nous pensons qu'un effort de transposition didactique est nécessaire pour les adapter aux élèves du secondaire. Nous allons donc poursuivre ces travaux en effectuant cette transposition didactique et en identifiant l'ensemble des constructions algorithmiques qu'un élève doit maîtriser et qui découlent des règles de grammaire.

Références

1. Denny, P., Luxton-Reilly, A., Tempero, E., Hendrickx, J. : Understanding the syntax barrier for novices. In : Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education. pp. 208–212. ITiCSE '11, Association for Computing Machinery, New York, NY, USA (Jun 2011). https://doi.org/10.1145/1999747.1999807
2. Douady, R. : Jeux de cadres et dialectiques outil-objet dans l'enseignement des Mathématiques. Une réalisation dans tout le cursus primaire. Ph.D. thesis, Université paris VII (Oct 1984)
3. Dowek, G. : Les quatre concepts de l'informatique. In : Sciences et Technologies de l'information et de La Communication En Milieu Éducatif : Analyse de Pratiques et Enjeux Didactiques. pp. 21–29. Athènes : New Technologies Editions (2011)
4. Du Boulay, B. : Some Difficulties of Learning to Program. Journal of Educational Computing Research **2**(1), 57–73 (Feb 1986). https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9