TER

Implémentation du Spécial K

Sylvain Beucler, Julien Charles, Pierre Châtel, Cyril Rodas

Table des matières

Avant-Propos

Le but de ce TER est l'intégration dans l'environnement PLT DrScheme[1] d'un langage à base de règles les plus proches possibles de celles utilisées par Emmanuel Kounalis dans ses cours d'algorithmique à la Faculté des Sciences de Nice Sophia-Antipolis.

Les objectifs et les enjeux liés à l'implémentation de ce langage seront détaillés dans les sections qui vont suivre. Nous nous attacherons à évoquer une partie des objectifs, notamment pédagogiques, qui découlent de ce sujet.

1 Introduction

1.1 Objectifs

L'objectif principal de ce travail est l'intégration avancée dans DrScheme [FLATT04]. Ce dernier est le « laboratoire à langage » de l'équipe PLT. Disponible en téléchargement et utilisé avec succès depuis de nombreuses années dans de certaines universités lors de la formation des élèves, il est fourni principalement avec une branche du langage Scheme, mais il est aussi possible de s'en servir avec d'autres langages par les biais d'un mécanisme d'extension (où de « plugin »). C'est justement un de ces « language pack » qu'il s'agit ici de développer.

Le choix de DrScheme comme interface utilisateur et de Scheme comme langage hôte s'est effectué pour les raisons évoquées ci-dessus et aussi pour éviter de réinventer la roue. Nous bénéficierons ainsi d'un environnement de développement éprouvé ainsi que la puissance de Scheme: il s'agit d'un des langages qui présente le plus de similitudes avec les concepts qui seront développés dans SpécialK. L'essentiel du travail va donc porter sur **l'écriture d'un compilateur SpécialK vers Scheme**. Il est à noter qu'un TER du même type avait déjà été proposé, mais il avait alors été réalisé en Java (qui lui est loin d'avoir une quelconque similitude avec SpécialK) et l'interface était basée sur Swing: il n'est actuellement pas utilisé pour l'enseignement.

Il ne faut donc pas négliger la **motivation pédagogique**: le produit final doit pouvoir être utilisé par Emmanuel Kounalis lors de la réalisation de ses TPs machine. L'objectif est donc d'obtenir une réalisation la plus intuitive possible: fournir l'ensemble des fonctionnalités attendues pour l'enseignement sans pour autant nuire à la compréhension du système. On dénombre ainsi l'exécution « pas à pas » des algorithmes, des facilités de visualisation graphique des arbres et des tableaux ainsi qu'une documentation du langage étoffée. L'ensemble devra être le plus robuste possible car il sera utilisé par des étudiants; il faudra aussi proposer un système de gestion des erreurs dans les algorithmes qui ne mette pas en défaut le compilateur.

Un autre objectif sera la **mise en valeur de certaines caractéristiques des programmes propres à l'algorithmique de premier cycle universitaire**: on fournira donc un moyen de comparer le temps d'exécution des algorithmes ainsi qu'une bibliothèque de fonctions utilitaires sur les types de données (tris, parcours d'arbres,...).

1.2 Fournitures

Ceci est une liste non exhaustive de ce qui sera fourni à l'issue de la période de développement:

-1- http://www.drscheme.org

- « Language pack » pour DrScheme et ses bibliothèques
- Un ensemble d'exemples d'algorithmes usuels écrits en SpécialK
- La documentation (en ligne) du langage

1.3 Définitions et acronymes

- **Analyse lexicale**: Terme emprunté à la linguistique. L'analyse lexicale consiste à différencier les différents lexèmes d'un texte.
- **Analyse syntaxique**: L'analyse syntaxique sert à déterminer si les lexèmes obtenus après l'analyse lexicale sont bien ordonnés.
- **Algorithme**: Suite d'étapes pour réaliser un but.
- **Clauses:** Une clause est une expression composée en générale d'une partie gauche (tête) et d'une partie droite (queue). Lors de l'exécution d'un programme composé de clauses, le flot de données est unifié avec la tête de la clause, et on remplace les données par la queue de la clause. Ce mécanisme est principalement utilisé dans Prolog (le langage de programmation logique). Les clauses ont évoluées en des expression à trois parties avec l'arrivée des langages de programmation logique à flots de données parallèles, lorsqu'on s'est aperçu que le mécanisme d'unification n'est pas suffisant pour représenter tous les modèles de données [GUES87]. On ajoute une condition appelée garde devant la clause, pour empêcher d'exécuter une clause que l'unification aurait sélectionnée.
- **Compilateur**: Programme pouvant transformer un texte écrit en un langage de programmation A vers un texte écrit dans un langage de programmation B. Pour faciliter la compilation il existe des outils consacrés pour effectuer l'analyse lexicale et l'analyse syntaxique.
- **DrScheme**: L'interface graphique de PLT Scheme, une implémentation de Scheme par PLT. Elle est extensible facilement, on peut utiliser d'autres langages que le Scheme avec cette interface. Par exemple, on peut éditer du Java ou de l'Algol 60 avec cette interface.
- **Expressivité d' un langage**La facilité avec laquelle on peut écrire des algorithmes dans un langage.
- **Fonction**: Relation qui associe à un ensemble d'entrées un ensemble de sorties.
- **Garde**: Condition, permettant de choisir entre différentes clauses. Le terme est hérité des langages Parlog [BOR91] et Strand88 [STRAND88].
- **Langage abstrait**: Langage proche des mathématiques. On utilise la récursivité pour y définir des structures. Le langage d'E. Kounalis en est assez proche.
- **Langage à flots de données: L**angage qui repose sur la notion de clauses. A chaque étape de l'exécution les données empreinte une clause. La métaphore d'un flot que l'on détourne vers les différentes clauses est immédiate.
- **Langage fonctionnel**: Langage dans lequel le mécanisme principal utilisé pour exprimer les algorithmes est la récursivité.
- **Langage logique**: Langage utilisant la logique des prédicats pour exprimer ses algorithmes.
- **Langage multi-paradigmes:** Langage utilisant des fonctionnement de différents paradigmes de programmation. En général, ces langages utilisent des constructions des langages logiques et des

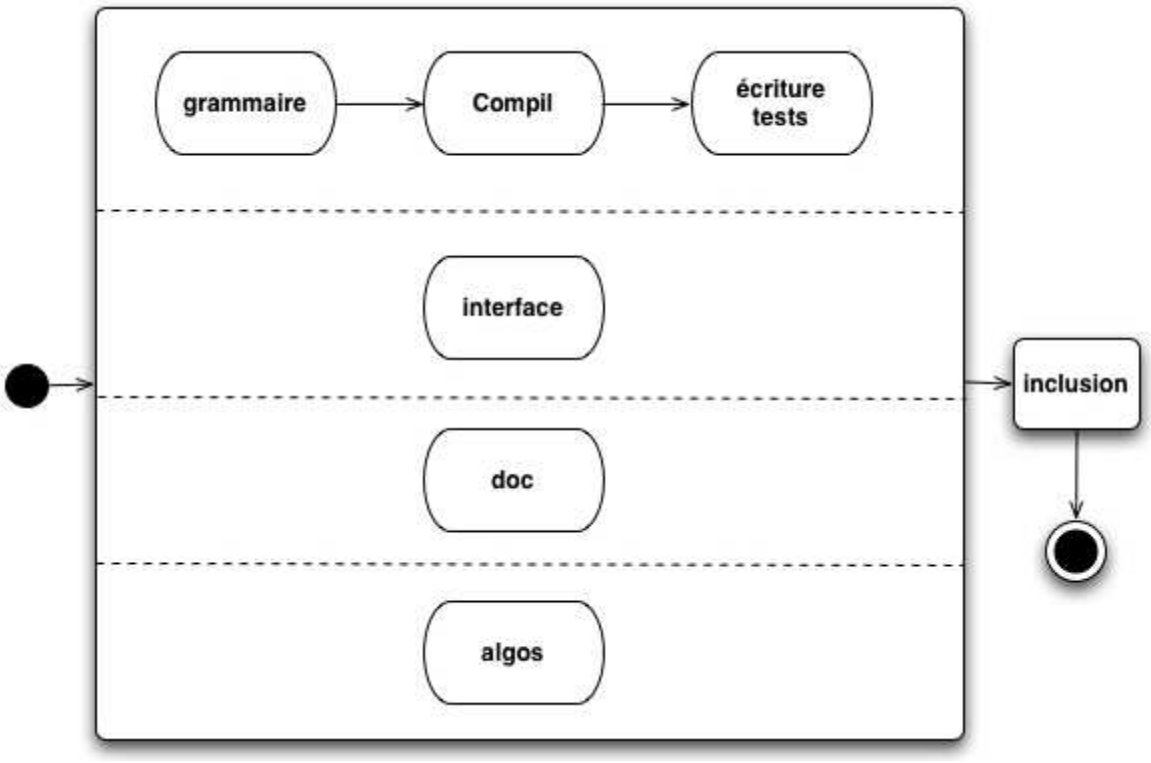
langages fonctionnels, ils sont alors difficilement compréhensibles pour les non-initiés, mais ils ont une expressivité importante.

- **Langage séquentiel/ impératif**: Un langage où l'on exprime les algorithmes préférentiellement de manière séquentielle.
- **Language pack**: Dans DrScheme, on rajoute des langages à l'aide de Language packs, qui sont donc des modules de langages pour DrScheme.
- **Lexème**: Ensembles de caractères. Plus petite unité syntaxique.
- **Lex/Yacc**: Les parsers les plus connus. Lex sert à effectuer l'analyse lexicale, et Yacc l'analyse syntaxique.
- **Parser**: ou Analyseur, utilitaire permettant d'effectuer l'analyse lexicale, et l'analyse syntaxique.
- **PLT Scheme**: l'ensemble des outils scheme fournis par PLT, à ne pas confondre avec mzscheme.
- **Récursivité**: Manière de définir des fonctions à partir d'elle-même.
- **Scheme**: Langage de programmation développé à partir du langage Lisp.
- **Unification**: le mécanisme de faire correspondre deux structures de données ensemble. L'unification réussit si les deux structures de données ont la même forme; sinon elle échoue. Si l'unification réussit, on obtient une association entre les éléments de la première structure et ceux de la seconde structure. Par exemple en unifiant les deux liste (a, b, c) et (1, 2, 3) on obtient les associations {a, 1}, {b, 2} et {c, 3}. [BAA99]

2 Organisation du projet

2.1 Processus

2.1.1 Décomposition du projet dans le temps



Le tableau suivant correspond à la répartition temporelle des différentes étapes de développement du TER:

Étape	***Attribution***
Grammaire (ajouter les indices de placement)	Effectué
Compilation de la grammaire -> production de Scheme	1 sem
Exécution pas-à-pas (adaptation du compilateur, utilisation des continuations)	1 sem
Gestion des erreurs	Doit être prévu à la base dans la partie compilation. 1 sem
Chronomètre d'exécution	...
Interface graphique pour la représentation des données	1 sem
Écriture d'un ensemble d'algorithmes	Base initiale, puis adaptation aux changements éventuels de la grammaire, et extension pour le jeu de tests
Écriture de la documentation	Base au début, puis mise à jour tout au long du développement

2.1.2 Justification du modèle de développement

Certaines tâches sont indépendantes les unes des autres, elle peuvent donc êtres développées séparemment. C'est par exemple le cas de certains éléments de l'interface graphique qui sont indépendants du SpécialK car basés sur le langage hôte (Scheme dans notre cas).

Ainsi on peut imaginer on peut déterminer 2 parties importantes: d'un côté le coeur du langage, composé de la mise en place de la grammaire, la compilation de la grammaire, de l'exécution et de la gestion des erreurs (à l'exécution). De l'autre on a la partie des éléments ne faisant pas partie du moteur SpécialK *stricto sensu*, c'est à dire l'interface graphique - indispensable mais indépendante; l'écriture d'un jeu d'algorithmes de base pour les étudiants; et l'écriture de la documentation.

La seconde partie est en faite constituée d'éléments pouvant être développés de manière autonome. Le compilateur (la partie 1) peut être développé en même temps que l'interface graphique. La documentation dépend d'une certaine façon de l'interface graphique et de la définition du langage; mais si la partie théorique est bien définie au préalable on peut l'écrire indépendemment des parties qu'elle décrit.

2.2 Organisation Structurelle

Le principales étapes du développement de ce projet et leurs attributions sont les suivantes:

Étape	***Réalisation***
Grammaire (ajouter les indices de placement)	Julien Charles et Cyril Rodas
Compilation de la grammaire -> production de Scheme	Julien Charles et Pierre Châtel
Exécution pas-à-pas (adaptation du compilateur, utilisation des continuations)	Julien Charles et Pierre Châtel, Cyril Rodas
Gestion des erreurs	Julien Charles et Pierre Châtel
Chronomètre d'exécution	...
Interface graphique pour la représentation des données	Julien Charles et Pierre Châtel
Écriture d'un ensemble d'algorithmes	Sylvain Beucler
Écriture de la documentation	Sylvain Beucler

2.3 Limites et interfaces

De part le choix de conception qui à été fait, les limites au développement de ce projet sont plus temporelles que techniques, il est donc à souhaiter que des tierces-personnes étendent ses fonctionnalités même à l'issue du TER.

Malgré tout, on peut dénombrer certaines limitations dues au manque de maturité de certaines parties de DrScheme. Par exemple le « stepper » (qui permet l'exécution étape par étape du code) ou encore le module de « profiling » qui sont intégrés à DrScheme, et qui devraient pouvoir être utilisées sans modification lourde par Spécial K, ne seront pas utilisés car elles ne présentes pas encore les

capacités d'extensibilité prévues dans le futur.

3 Gestion

3.1 Objectifs et priorités

Les objectifs principaux sont les suivants:

- **La mise au point du coeur du projet**: c'est à dire la *grammaire*, la *compilation*, la *gestion des erreurs* et le *pas à pas*. Cette partie doit présenter le moins de « bugs » possible, et le code doit être propre afin de permettre une meilleure maintenance du code.
- **La documentation** qui doit être la plus complète possible.

Bien qu'importantes, les parties suivantes, seront traités de manière secondaire:

- *L'interface*. Le mot d'ordre sera « concision, et efficacité ». On ne programmera que ce qui est essentiel à la représentation des données, mais de manière efficace.
- L'écriture du jeu d'*algorithmes* d'exemples.

3.2 Hypothèses dépendances et contraintes

3.2.1 Hypothèses

Nous avons pris la décision de distribuer notre implémentation sous forme de « language pack » pour l'IDE DrScheme. Ces « langages pack » sont des plugins permettant de programmer dans un certain langage à l'aide de l'interface de DrScheme. Il existe déjà des plugins pour certains langages comme Algol60.

Nous fournirons une documentation du langage implémenté (exemples, syntaxe...).

Vu que nous devons traduire le langage SpécialK vers Scheme nous écrirons la totalité des programmes en Scheme et nous utiliserons pour l'analyseur la bibliothèque lex/yacc.

3.2.2 Contraintes

Le langages SpécialK est une implémentation de langage utilisé en cours par Emmanuel Kounalis. Les utilisateurs étant les étudiants de MI2 et Licence, la syntaxe doit se rapprocher le plus possible de celle qu'ils ont l'habitude d'utiliser.

Le code écrit en spécialK doit être traduit et interprété en Scheme. La lisibilité du code produit n'est pas une priorité. On cherchera un compromis entre optimisation et lisibilité.

Nous devons fournir un chronomètre d'éxecution simple à utiliser ainsi qu'un stepper et la possibilité de représenter graphiquement des structures de données simples.

3.2.3 Dépendances

Le logiciel fournit est un « langage pack ».

3.3 Gestion du risque et Moyens de contrôle

Le premier risque est que la grammaire ne fonctionne pas. Malheureusement on ne peut pas se le permettre. Il faudra donc mettre en place de nombreux tests au cours du développement pour éviter les erreurs.

Le second risque est la non complétude de la gestion des erreurs (celles de l'analyse lexicale, syntaxique et sur le plan d'exécution). Il faut être capable de faire correspondre un erreur avec sa position dans le code SpécialK.

Un risque plus général serait que le langage produit ne satisfasse pas les attentes d'E.Kounalis et des étudiants: on pourra mettre ne place une évaluation par les étudiants des différentes versions en développement.

Pour la construction du langage pack nous prendrons comme exemple le travail effectué sur les langages packs existants. Cependant, il n'est pas à exclure que l'on pourrait choisir de construire nous même une IDE simple si la construction du langage pack est trop contraignante.

La facilité d'utilisation de l'application et de la documentation (et son utilité) est aussi un objectif à respecter. Un soin particulier devra y être apporté.

4 Techniques

4.1 Méthodes et outils employés

Le logiciel proprement dit sera développé intégralement en Scheme, et DrScheme semble être un bon outil pour travailler sur DrScheme.

Pour écrire SpécialK nous utilisons le package ParserTools présent dans PLT Scheme. Il s'agit d'un outil basé sur le modèle de Lex et Bison produisant du code Scheme à la place de code C. On l'utilise pour écrire la grammaire et effectuer l'analyse syntaxique. Un fois l'analyse syntaxique terminée; on obtient un arbre décoré représentant le programme analysé. La gestion des erreurs au niveau de l'analyse syntaxique s'effectue principalement grâce aux facilités offertes dans *parser-tools*.

Les besoins en interface graphique (MrEd) et en portabilité (MzScheme) sont également assurés par DrScheme. Tout comme en Java, il n'y a pas besoin de configurer le logiciel pour un système en particulier.

La distribution se fera sous la forme d'une archive propre à DrScheme (fichier .plt) qui contient un script d'installation pour s'intégrer dans l'environnement DrScheme de l'utilisateur de manière transparente.

Enfin, la gestion des sources durant le développement se fera via CVS: il s'agit d'un système que tous les membres du projet savent utiliser au moins de manière basique, et reste d'une puissance suffisante pour l'utilisation que nous en ferons. Pour un projet d'étalant sur plus de temps, ou faisant intervenir plus de personnes, un système plus souple notamment au niveau de la réorganisation des fichiers tel que Subversions ou GNU Arch aurait été préférable, quitte à passer un peu de temps en formation. Plusieurs d'entre nous possédons une connexion permanente au réseau Internet, de sorte que nous pourrons installer un serveur CVS facilement, probablement avec un système de sauvegarde distante via rsync. Utiliser le réseau de la faculté aurait pu être intéressant, mais son manque de fiabilité et le manque de contrôle que nous aurions aurait été problématique.

4.2 Documentation

SpécialK devra disposer d'une aide en ligne. Dans l'optique de l'intégration maximale à DrScheme, nous avons décider de générer une documentation semblable à celle de cet outil: l'équipe PLT utilise un ensemble (un brin anarchique) de documents LaTeX, convertis en HTML à l'aide de l'outil tex2page (écrit en Scheme, cela va de soi) ainsi que de quelques scripts. Cet ensemble n'étant pas considéré comme de bonne qualité par ses auteurs, nous n'avons malheureusement pas pu y avoir accès, mais nous tenterons de nous conformer à cette organisation.

L'ensemble LaTeX + tex2page propose les avantages suivants:

- Une fois la documentation produite, elle peut être affichée par le navigateur léger intégré à DrScheme ou tout autre navigateur de l'utilisateur.

- LaTeX permet une bonne gestion de liens ou références croisées.

- DrScheme effectue ses recherches dans un fichier d'index et de mots-clef plutôt que dans l'ensemble du texte. On peut les créer à partir des fichiers .idx de LaTeX.

- L'ensemble de la présentation finale est pris en charge par tex2page, ce qui permet à l'auteur de la documentation de se concentrer sur le contenu.

- La documentation pourra être imprimée.

Le wiki installé à la faculté nous avait paru dans un premier temps un bon moyen de gérer la documentation, de par ses outils intégrés et accessibles via un simple nagivateur. Cependant, il présente l'inconvénient de ne pas être très structuré (l'organisation est réalisée au cas par cas à l'aide de liens pour ne pas contraindre les utilisateurs et garder le système simple) et de manquer de moyens d'exportation sans besoin de post-traitement. On l'utilisera donc plutôt en temps que forum pour déterminer collectivement certaines points de décision (comme nous l'avons fait pour la grammaire), réservant CVS pour la gestion des fichiers finaux.

Enfin, on fournira une description, au format encore indéfini, sur l'architecture du projet et son organisation interne, dans la lignée des fichiers HACKING des projets libres, à destination du corps enseignant qui aura éventuellement à adapter et améliorer SpécialK à l'avenir.

5 Calendrier

Nous disposons d'un mois à temps complet pour travailler sur le projet, à partir de la fin des examens.

Le but est de réaliser sous 3 semaines maximum un produit exploitable, même s'il manque de stabilité. On pourra utiliser le temps restant pour améliorer la robustesse du produit, et réaliser des tests « grandeur nature » si possible avec les étudiants de DEUG-MI2 et licences informatiques, probablement sur la base de test bénévole, l'utilisation en TPs notés étant problématique.

Cette version pourrait en outre être présentée à l'équipe de PLT et initier son intégration dans les distributions standards de DrScheme, tout comme le module Algol60 par exemple. Une version anglaise de la documentation sera nécessaire.

L'organisation reste relativement simple, car il n'y a pas de dépendances importantes. Une grande partie de la documentation et des exemples peut-être réalisée dès à présent, il faudra simplement tenir ces éléments à jour, pendant que le coeur du système, le traducteur et ses outils, sera développé.

6 Contraintes non fonctionnelles

Les contraintes techniques sont relativement souples, car un faut déjà un ordinateur décent, surtout en mémoire, pour lancer l'usine à gaz DrScheme, ce qui est le cas des postes de travail de la faculté. Si le temps le permet, on tentera néanmois de permettre l'utilisation de SpécialK dans MzScheme (l'interpréteur Scheme de DrScheme), permettant à l'utilisateur d'avoir un environnement tel qu'Emacs+mzscheme+Quack, plus léger (Quack est un mode Emacs dédié à Scheme avec des spécificités pour PLT Scheme).

Un autre point lié à la technique est le temps de traitement, qui devra être inférieur au projet KTA de l'année passée.

Enfin, pour faciliter l'intégration avec DrScheme, le logiciel sera diffusé sous la même licence, la GNU LGPL. Il s'agit d'une licence libre avec copyleft, excepté qu'il est permis à un logiciel propriétaire de l'utiliser sous certaines conditions.

Index lexical

Bibliographie

FLATT04: John Clement, Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi, Fostering Little Languages, 2004
GUES87: Iréne Guessarian, José Meseguer, On the axiomatization of “if-then-else”, 1987
BOR91: Edon Börger & Elvinia Riccobene, A Formal specification of PARLOG, 1991
STRAND88: Ian Foster, Strand: a Practical Parallel Programming Language, 1990
BAA99: Franz Baader, Unification theory, 1999