Introduction

Les langages de programmation ne sont que des approximations, rigoureuses mais incomplètes, du langage mathématique. Les langages généraux (non spécialisés à certains types de problèmes), sont complets au sens de Turing. Cela veut dire qu'ils permettent d'écrire tous les algorithmes calculables. Ils ne sont cependant pas tous équivalents, car ils diffèrent énormément par leur expressivité, c'est-à-dire leur capacité d'exprimer succinctement certains algorithmes. Nous sommes donc poussés à une recherche —sans fin ?— de nouvelles structures de programmation, plus abstraites, permettant de décrire des algorithmes de façon plus concise. Nous ne recherchons pas non plus l'expressivité à tout prix, et nous efforçons de n'introduire que des constructions essentielles, simples, bien formalisés et sures.

Donner la syntaxe d'un langage n'est qu'un préliminaire à sa définition. Pour parler de programme il faut donner un sens aux expressions du langage, c'est-à-dire définir leur sémantique. Par exemple, cela peut se faire en décrivant l'évaluation des expressions. Parce que pour être intéressant, les langages de programmation généraux sont complets au sens de Turing, il faut abandonner tout espoir de pouvoir décider de propriétés importantes telles que la terminaison d'un programme. Pour des raisons analogues, la bonne exécution d'un programme est souvent une propriété indécidable. La sûreté de l'exécution ne doit évidemment pas être abandonnée au profit de l'expressivité. Il est essentiel de conserver l'expressivité tout en se limitant à une sous-classe décidable des programmes qui s'évaluent normalement.

Le typage est un moyen de définir un sous-ensemble des expressions bien formées par un critère décidable, et en général simple. On attend du typage qu'il garantisse la sécurité de l'évaluation. Nous considérerons surtout des systèmes de typage implicites, c'est-à-dire que les types ne font pas partie des programmes, mais seront synthétisés.

Ainsi, au travers de la formalisation du langage ML, de son typage et de sa sémantique, ces notes de cours ont pour but principal de montrer l'apport du typage dans les langages de programmation et de fournir les techniques et outils nécessaires permettant d'étudier l'ajout au langage de nouvelles constructions, ainsi que d'aborder l'étude d'autres langages de façon rigoureuse.

L'unité du cours est le langage ML, et tous les aspects étudiés seront présentés à partir d'extensions de ce langage. On peut distinguer nettement trois parties. Les chapitres 1 à 3 formalisent le langage ML, en partant du noyau, puis en ajoutant de façon relativement modulaire les autres constructions. Les points importants sont le système de typage de ML, la synthèse des types, sa sémantique opérationnelle et la

sûreté du calcul garantie par le typage.

Dans une seconde partie (chapitre 4 à **??**) nous nous intéressons à des extensions avancées du langage, et essentiellement de son système de typage. Nous y décrivons le typage des objets enregistrements. Puis, nous ajoutons un mécanisme de sous-typage, et nous montrons que les propriétés essentielles du système de type de ML sont conservées. Nous décrivons très brièvement d'autres extensions du système de typage (types dynamiques, types existentiels et universels déclarés, types d'ordre supérieur, types intersections).

Enfin la troisième partie est consacrée à un aspect essentiel des langages de programmation : la modularité. Nous y présentons un système de module pour ML mais de façon aussi indépendante du langage ML que possible.

Tout au long de ce cours nous proposons des exercices. La plupart sont annotés de (*) à (***) selon leur niveau de difficulté, et nous en donnons un corrigé (parfois succinct) en annexe. D'autres ne sont pas annotés et ne sont pas corrigés. Ce sont soit des exercices très simples qui ne méritent pas de correction, ou bien au contraire des exercices plus long, mais pas nécessairement difficiles.