Module Description, available in: EN, FR

Advanced Programming Paradigms

General Information

Number of ECTS Credits

3

Module code

TSM_AdvPrPa

Valid for academic year

2020-2021

Last modification

2019-10-26

Responsible of module

Edgar Lederer (FHNW, edgar.lederer@fhnw.ch)

Explanations regarding the language definitions for each location:

- Instruction is given in the language defined below for each location/each time the module is held.
- Documentation is available in the languages defined below. Where documents are in several languages, the percentage distribution is shown (100% = all the documentation).
- The examination is available 100% in the languages shown for each location/each time it is held.

	Berne	Lausanne			Lugano	Zurich		
Instruction			**X** F 70%	**X** E 30%		**X** E 100%		
Documentation				**X** E 100%		**X** E 100%		
Examination			**X** F 100%			**X** E 100%		

Module Category

TSM Technical scientific module

Lessons

2 lecture periods and 1 tutorial period per week

Entry level competences

Prerequisites, previous knowledge

Good knowledge of object-oriented programming.

Brief course description of module objectives and content

A wealth of fascinating technologies exists alongside the ubiquitous object-oriented programming and the inadequate testing methods. This module introduces students to the most relevant of these emerging technologies from a general programming-language point of view.

Paradigms Besides object-orientation as today's mainstream programming paradigm, other quite different paradigms have been developed and brought to maturity during the last decades: in particular functional programming, but also logic and constraint programming. None of these paradigms is well suited to solving all the different kinds of problems but each has its own particular strengths in specific areas. Since modern software encompasses many such areas, simultaneous application of several paradigms seems appropriate, and consequently, their seamless integration into

multi-paradigm languages.

Types Programming languages with a rich and consistent type system make it possible to detect certain errors at the time of compilation already. Using the type system, invariants can be declared in one's own data types, which are then checked by the compiler. Programming in and with a strong type system can be regarded as the first step in the direction of program verification. With even stronger type systems, it is possible to formulate complete program specifications. The compiler does then, however, generally require support for verification.

Correctness Choosing the right programming paradigm for a given problem simplifies its solution, but does not guarantee its correctness – the most important of all software qualities. Such a guarantee requires, in addition to the actual implementation (the "How?"), a specification (the "What?") and proof of correctness (the "Why?"). Continuous research right from the very outset of computing has now led to a verification technology that is entering industrial application. Since object-oriented programming is ubiquitous, its specification and verification is of particular importance.

This module will provide:

- an overview of programming concepts, paradigms and languages;
- a comprehensive introduction to functional programming (using Haskell or Scala);
- an introduction to multi-paradigm programming, with special emphasis on types (using Scala, which is a combination of functional and object-oriented programming);
- an introduction to the theory and practice of specification and verification of imperative programs (as a basis for the verification of object-oriented programs - example Dafny) and/or of functional programs (example Coq).

Aims, content, methods

Learning objectives and acquired competencies

Students will acquire an understanding of the emerging paradigms, practical skills in modern functional, multiparadigm and type-full programming, and a basic understanding of the increasingly important field of software specification and verification.

Contents of module with emphasis on teaching content

Functional programming (6 weeks)

- Programming concepts, paradigms and languages.
- Absence of state, referential transparency, reasoning about programs.
- Eager versus lazy evaluation.
- Types and type inference, higher-order functions, concrete data types and pattern matching.
- Functors, applicative functors, monads.
- An application: interpreter for a small imperative programming language (IML).

Multi-paradigm and strong typed programming (4 weeks)

- Trait types (and Mixin composition as a variant on classical inheritance).
- Generic types (co- and contravariance for type parameters).
- Type classes and implicit parameters.
- Type-secure DSLs (Domain Specific Languages).

Program verification (4 weeks)

- Reliability via testing and verification.
- Hoare logic and weakest preconditions.
- Architecture of verification tools.
- An application: verification condition generator.
- A current verification tool: Dafny and/or Coq.

Teaching and learning methods

- Ex-cathedra teaching.
- Programming and verification exercises.

Literature

- Graham Hutton, Programming in Haskell, Second Edition, Cambridge, 2016.
- Miran Lipovaca, Learn You a Haskell for Great Good!, No Starch Press, 2011.
- Martin Odersky, Lex Spoon and Bill Venners, Programming in Scala, Artima, 2008.
- David Gries, The Science of Programming, Springer, 1981 (a classical text).
- José Bacelar Almeida et al., Rigorous Software Development, Springer, 2011.
- Federico Biancuzzi und Shane Warden, Masterminds of Programming: Conversations with the Creators of Major Programming Languages, O'Reilly, 2009 (for recreation).

Assessment

Certification requirements

Module does not use certification requirements

Basic principle for exams

As a rule, all the standard final exams for modules and also all repetition exams are to be in written form

Standard final exam for a module and written repetition exam

Kind of exam

written

Duration of exam

120 minutes

Permissible aids

Aids permitted as specified below:

Permissible electronic aids

No electronic aids permitted

Other permissible aids

a summary on at most 4 pages DIN A4 (= 2 sheets DIN A4, written by hand or electronically and printed out)

Special case: Repetition exam as oral exam

Kind of exam

oral

Duration of exam

30 minutes

Permissible aids

No aids permitted

Description du module, disponible en: EN, FR

Paradigmes de programmation avancée

Informations générales

Nombre de crédits ECTS

3

Code du module

TSM_AdvPrPa

Valable pour l'année académique

2020-2021

Dernière modification

2019-10-26

Nom du/de la responsable de module

Edgar Lederer (FHNW, edgar.lederer@fhnw.ch)

Explication des définitions de langue par lieu :

- Les cours se dérouleront dans la langue définie ci-dessous par lieu/exécution.
- Les documents sont disponibles dans les langues définies ci-dessous. Pour le multilinguisme, voir la répartition en pourcentage (100% = documents complets)
- L'examen est disponible à 100% dans chaque langue sélectionnée pour chaque lieu/exécution.

	Berne	Lausanne			Lugano	Zurich		
Leçons			**X** F 70%	**X** E 30%		**X** E 100%		
Documentation				**X** E 100%		**X** E 100%		
Examen			**X** F 100%			**X** E 100%		

Catégorie de module

TSM approfondissement technico-scientifique

Leçons

2 leçons et 1 leçon de pratique par semaine

Compétences préalables

Connaissances préalables, compétences initiales

Bonnes connaissances opérationnelles de la programmation orientée objet.

Brève description du contenu et des objectifs

Au-delà de la programmation orientée-objets devenue une norme et du *testing* insuffisant, il existe tout un univers parallèle de technologies fascinantes. Ce module présente un panorama des diverses technologies émergentes du point de vue des langages de programmation.

Paradigmes Outre la programmation orientée objet qui est le paradigme de programmation le plus courant actuellement, d'autres *paradigmes* assez différents ont été amenés à maturité au cours des dernières décennies : en particulier la programmation *fonctionnelle,* la programmation logique ou encore la programmation par contraintes. Aucun de ces paradigmes (pas même la programmation orientée-objet) n'est adapté pour résoudre de manière optimale tous les types de problèmes, bien que chacun ait son domaine d'action où il peut faire valoir ses atouts. Toutefois, étant donné que

les logiciels modernes englobent souvent bon nombre de ces domaines, l'application simultanée de plusieurs paradigmes semble appropriée. En conséquence, il existe depuis peu des langages de programmation multi-paradigmes.

Typage Les langages de programmation disposant d'un système de typage riche et consistant permettent d'identifier certaines erreurs dès l'instant de la traduction. Le système de typage permet de déclarer des invariances dans des types de données, qui seront ensuite contrôlées par le compilateur. La programmation dans et à l'aide d'un système de typage fort peut être considéré comme un premier pas vers la vérification d'un programme.

Exactitude Le choix du bon paradigme de programmation permet de simplifier la solution de tout problème donné mais ne garantit pas son exactitude, la première de toutes les qualités d'un logiciel. Une telle garantie requiert, en plus de l'implémentation (le «comment»), une spécification (le «quoi») ainsi qu'une preuve d'exactitude (le «pourquoi»). Les recherches menées depuis les origines de la science informatique ont abouti à une technologie de la vérification qui entre désormais dans sa phase d'application industrielle. Etant donné que la programmation orientée objet est omniprésente, sa spécification et sa vérification revêtent une importance particulière.

Le module donnera:

- un aperçu des concepts et des paradigmes sous-tendant les langages de programmation;
- une introduction à la programmation fonctionnelle (à l'aide de Haskell ou Scala);
- une introduction à la programmation multiparadigmes, avec un accent particulier donné aux types (utilisant Scala, un langage alliant programmation fonctionnelle et programmation orientée objet);
- une introduction à la théorie et à la pratique de la spécification et la vérification des programmes impératifs comme base de la vérification de programmes orientés objet (à l'aide de Dafny et/ou Coq).

Objectifs, contenus, méthodes

Objectifs d'apprentissage, compétences à acquérir

Les étudiants acquerront une vision des paradigmes émergents, des connaissances pratiques de la programmation fonctionnelle, type-full et multiparadigme, ainsi que les bases de la spécification et de la vérification, domaine en pleine expansion.

Contenu des modules avec pondération du contenu des cours

Programmation fonctionnelle (6 semaines)

- Concepts, paradigmes et langages de programmation
- Absence d'état, transparence référentielle, raisonnement à propos des programmes.
- Evaluation avancée ou retardée (*eager vs lazy*).
- Types et interférence du type.
- Fonctions de rang supérieur.
- Types de données concrètes et *pattern matching*.
- Une application : un interpréteur pour un petit langage de programmation impératif.

Programmation multiparadigmes et fortement typée (4 semaines)

- Les traits comme types (et composition Mixin comme variante à l'héritage traditionnel).
- Types génériques (covariance et contrevariance comme paramètres de typage).
- Classes de typage et paramètres implicites.
- Languages dédiés, DSL, (*Domain Specific Languages*) de typage sûr.

Vérification de programme (4 semaines)

- Fiabilité par les tests et la vérification.
- Logique de Hoare et calcul de plus faibles pré-conditions.
- Architecture des outils de vérification.
- Une application : générateur d'obligations de preuve VCG.
- Un outil de vérification actuel : Dafny et/ou Coq.

Méthodes d'enseignement et d'apprentissage

- Cours ex-cathedra.
- Exercices de programmation et de vérification.

Bibliographie

- Graham Hutton, Programming in Haskell, Second Edition, Cambridge, 2016.
- Miran Lipovaca, Learn You a Haskell for Great Good!, No Starch Press, 2011.
- Martin Odersky, Lex Spoon and Bill Venners, Programming in Scala, Artima, 2008.
- David Gries, The Science of Programming, Springer, 1981 (a classical text).
- José Bacelar Almeida et al., Rigorous Software Development, Springer, 2011.
- Federico Biancuzzi und Shane Warden, Masterminds of Programming: Conversations with the Creators of Major Pro- gramming Languages, O'Reilly, 2009 (for recreation).

Evaluation

Conditions d'admission

Le module n'utilise pas de conditions d'admission.

Principe pour les examens

En règle générale, tous les examens de fin de module réguliers et les examens de rattrapage sont organisés sous la forme écrite

Examen de fin de module régulier et examen écrit de répétition

Type de l'examen

écrit

Durée de l'examen

120 minutes

Aides autorisées

Les aides suivantes sont autorisées:

Aides électroniques autorisées

Aucune aide électronique autorisée

Autres aides autorisées

un résumé écrit sur 4 pages DIN A4 au plus (=2 feuilles DIN A4, écrites à la main ou électroniquement et imprimées)

Cas spécial: examen de répétition oral

Type de l'examen

oral

Durée de l'examen

30 minutes

Aides autorisées

Sans aides