Overview of the KeY System
Contents of Second Part of Course

- Overview of KeY
- UML and its semantics
- Introduction to OCL
- Specifying requirements with OCL
- Modelling of Systems with Formal Semantics
- Propositional & First-order logic, sequent calculus
- OCL to Logic, horizontal proof obligations, using KeY
- Dynamic logic, proving program correctness
- Java Card DL
- Vertical proof obligations, using KeY
- Wrap-up, trends
Philosophy of KeY Tool

Formal Methods must *and can* be integrated into commercial processes, tools and languages for software development.
Philosophy of KeY Tool

Formal Methods must *and can* be integrated into commercial processes, tools and languages for software development

Integrated tool for
- Modeling
- Development
- Formal specification
- Formal verification

of object oriented programs
Technologies Used in this Course

Standard language for Modeling of Software

- Unified Modeling Language — UML  (Borland Together)
  Visual language for OO Modeling
  Standard of Object Management Group (OMG)

- Object Constraint Language — OCL
  Formal textual language for requirements specification
  UML sub-standard

Modern industrial programming language

Java (Card)

Logic, Automated Deduction

First-order logic, Dynamic Logic, Theorem proving
KeY plugin for Eclipse IDE

Java Modeling Language — JML

Formal Interface Specification Language for Java
Wide-spread in academic projects

Automatic Translation from OCL into English

To come soon:

Visual symbolic debugger based on symbolic program execution
The KeY System

Lightweight Usage of Formal Methods

English → OCL/UML → JML → Logic → Taclets

FM expert → Logic xp → Wizard

Borland Together CC
Eclipse IDE
KeY Plugin
KeY Plugin
JML Browser

OCL/NL Tool

KeY Prover

Synthesis of Proof Obligations

OCL/FOL Translation
JML/FOL Translation

Rule Base
The KeY System — UML Design, Java Coding
Target Language: Java Card
Target Language: Java Card

Java Card

Sun’s Java dialect for smart cards and embedded systems
Target Language: Java Card

Java Card

- Sun’s Java dialect for smart cards and embedded systems

Java Card is relevant target language for verification:

- Restrictions admit complete coverage (more later)
- Applications smallish
Target Language: Java Card

Java Card

- Sun’s Java dialect for smart cards and embedded systems

Java Card is relevant target language for verification:

- Restrictions admit complete coverage (more later)
- Applications smallish

- Applications safety & security critical
- Often impossible to update smart cards/embedded systems
Conventional CASE-Based Development

UML Model
Conventional CASE-Based Development

UML Model

CASE Tool

Java (Card)
(partial implementation)
Conventional CASE-Based Development

UML Model

CASE Tool

Java (Card)
(partial implementation)

Java (Card)
The KeY System

Lightweight Usage of Formal Methods → FM expert → Logic xp → Wizard

English → OCL/UML → JML → Logic → Taclets

Borland Together CC → Eclipse IDE

OCL/NL Tool → KeY Plugin → KeY Plugin → JML Browser

OCL/FOL Translation → JML/FOL Translation

Synthesis of Proof Obligations

KeY Prover

Rule Base
The KeY System — Formal Specification

Lightweight Usage of Formal Methods ➔ FM expert ➔ Logic xp ➔ Wizard

English ➔ OCL/UML ➔ JML ➔ Logic ➔ Taclets

Borland Together CC ➔ Eclipse IDE ➔ KeY Plugin ➔ KeY Plugin ➔ JML Browser

OCL/NL Tool ➔ KeY Plugin ➔ JML/FOL Translation

JML/FOL Translation

Synthesis of Proof Obligations

OCL/FOL Translation

KeY Prover

Rule Base
Specification Language: OCL

UML has textual specification language as sub-standard:

Object Constraint Language (OCL)
- OOP-like syntax, ASCII
- designed for easy navigation within UML class diagrams, etc.
- strongly typed
- formal semantics: translation to typed FOL

OCL expressions reduce legal instances of underlying UML diagram
- Class invariants
- Pre-/postconditions of operations and methods

Permits formal specification of functional requirements

Other specification languages: Alloy, JML, RSL, ...
Formal Specification in OCL

UML Model → CASE Tool → Java (Card) (partial implementation) → OCL
Parsing OCL

UML Model

CASE Tool

Java (Card)
(partial implementation)

Java (Card)

OCL

OCL Parser
Univ Dresden

AST
Parsing OCL

UML Model

CASE Tool

Java (Card)
(partial implementation)

Java (Card)

XML API

OCL

OCL Parser
Univ Dresden

AST
Where to Take Formal Specification from?

- UML Model
- CASE Tool
- Java (Card)
  (partial implementation)
- Java (Card)
- XML API
- OCL
- OCL Parser
  Univ Dresden

AST
(Library of Patterns and Idioms)
The KeY System

Lightweight Usage of Formal Methods ↔ FM expert ↔ Logic xp ↔ Wizard

English ↔ OCL/UML ↔ JML ↔ Logic ↔ Taclets

Borland Together CC
Eclipse IDE
KeY Plugin
KeY Plugin
JML Browser

OCL/NL Tool

OCL/FOL Translation
JML/FOL Translation

Synthesis of Proof Obligations

Rule Base

KeY Prover
The KeY System — Verification

- Lightweight Usage of Formal Methods
- English
- OCL/UML
- KeY Plugin
- Borland Together CC
- Eclipse IDE
- JML
- KeY Plugin
- JML Browser
- Logic xp
- Wizard
- OCL/NL Tool
- JML/FOL Translation
- Synthesis of Proof Obligations
- Rule Base
- KeY Prover

Tasks of Verification Component
Tasks of Verification Component

- Translation OCL $\Rightarrow$ logic

Need to simplify resulting formulas
Tasks of Verification Component

- Translation OCL $\Rightarrow$ logic

- Synthesize FOL formulas from horizontal verification tasks

Eg, structural subtyping
Tasks of Verification Component

- Translation OCL $\Rightarrow$ logic
- Synthesize FOL formulas from horizontal verification tasks
- Synthesize Java Card DL formulas from vertical verification tasks

Eg, method invariants, total correctness
Tasks of Verification Component

- Translation OCL $\implies$ logic

- Synthesize FOL formulas from horizontal verification tasks

- Synthesize Java Card DL formulas from vertical verification tasks

- Driving the deduction component
Tasks of Verification Component

- Translation OCL $\Rightarrow$ logic
- Synthesize FOL formulas from horizontal verification tasks
- Synthesize Java Card DL formulas from vertical verification tasks
- Driving the deduction component
- Correctness management
Translation, Horizontal Verification

UML Model → CASE Tool → Java (Card) (partial implementation) → Java (Card)

Patterns/Idioms → OCL (partial specification) → OCL

XML API → OCL Parser

AST → Translation

First order logic

OCL Parser

Univ Dresden

First order logic

formula synthesis (horiz. verif.)
Translation, Vertical Verification

UML Model
CASE Tool
Java (Card) (partial implementation)
Java (Card)
RecodeR
Univ Karlsruhe
AST

Patterns/Idioms

OCL (partial specification)
OCL
OCL Parser
Univ Dresden
AST
Translation
formal synthesis (vert. verif.)

Java Card DL
Deductive Verification

UML Model

CASE Tool

Java (Card) (partial implementation)

Java (Card)

RecodeR
Univ Karlsruhe

AST

XML API

OCL (partial specification)

OCL Parser
Univ Dresden

AST

Translation
formula synthesis (vert. verif.)

Java Card DL

Interactive/Automated Theorem Prover

Formal proof

Deductive Verification

UML Model

CASE Tool

Java (Card) (partial implementation)

Java (Card)

RecodeR
Univ Karlsruhe

AST

XML API

OCL (partial specification)

OCL Parser
Univ Dresden

AST

Translation
formula synthesis (vert. verif.)

Java Card DL

Interactive/Automated Theorem Prover

Formal proof
The KeY System

Lightweight Usage of Formal Methods → FM expert → Logic xp → Wizard

English → OCL/UML → JML → Logic → Taclets

Borland Together CC → Eclipse IDE

KeY Plugin → KeY Plugin → JML Browser

OCL/NL Tool

OCL/FOL Translation → JML/FOL Translation

Synthesis of Proof Obligations

KeY Prover

Rule Base

The KeY System — Proving

Lightweight Usage of Formal Methods

- FM expert
- Logic xp
- Wizard

English

OCL/UML

- JML

- Logic
- Taclets

OCL/NL Tool

Borland Together CC

Eclipse IDE

KeY Plugin

KeY Plugin

JML Browser

OCL/FOL Translation

JML/FOL Translation

Synthesis of Proof Obligations

Rule Base

KeY Prover
The Program Logic: Dynamic Logic

Syntax

- Modal operators $[p]$ and $⟨p⟩$ for each Java (Card) program $p$
- Statements about final state of $p$
The Program Logic: Dynamic Logic

Syntax

- Modal operators \([p]\) and \(\langle p \rangle\) for each Java (Card) program \(p\)
- Statements about final state of \(p\)

Semantics

- \([p] F\): If \(p\) terminates normally, then \(F\) holds in the final state
  (partial correctness)

- \(\langle p \rangle F\): \(p\) terminates normally and \(F\) holds in the final state
  (total correctness)
Java vs. Java Card

Java features that are not present in Java Card
Java vs. Java Card

Java features that are not present in Java Card

- **Threads**
  - unrestricted concurrency
Java vs. Java Card

Java features that are not present in Java Card

- **Threads**
  - unrestricted concurrency

- **Floating point arithmetic**
  - IEEE standard 754 is huge . . .
Java vs. Java Card

Java features that are not present in Java Card

- **Threads**
  - unrestricted concurrency

- **Floating point arithmetic**
  - IEEE standard 754 is huge . . .

- **Dynamic class loading**
  - Implementation must be known before verification
Java vs. Java Card

Java features that are not present in Java Card

- **Threads**
  - unrestricted concurrency

- **Floating point arithmetic**
  - IEEE standard 754 is huge . . .

- **Dynamic class loading**
  - Implementation must be known before verification

- **Graphical/buffered I/O**
  - formal specification Swing classes?
Java vs. Java Card

Java features that are not present in Java Card

- **Threads**
  - unrestricted concurrency
- **Floating point arithmetic**
  - IEEE standard 754 is huge . . .
- **Dynamic class loading**
  - Implementation must be known before verification
- **Graphical/buffered I/O**
  - formal specification Swing classes?