Discussion of refinement-based approach

- Advantages:
  - quite close to programming language and programmer
  - special IDEs for programs, specifications and proofs
  - can smoothly integrate powerful logics and dedicated automated techniques
- Disadvantages:
  - expensive solution
  - not very flexible w.r.t. extensions
  - meta-logical aspects cannot be handled

Subsection 8.5.3

Extended static checking

Example systems

  - programming and specification language: Spec# (extension of C#)
  - specific focus on modularity of specifications
  - uses first-order ATP
  - also supports dynamic checking
  - verifier for single-threaded and multi-threaded C and Java programs
  - pre- and postconditions written in separation logic
  - user guides the proofs by so-called “lemma functions”
  - uses the SMT solver Z3
- BLAST: [mtc.epfl.ch/software-tools/blast/](mtc.epfl.ch/software-tools/blast/)
  - software model checker for C programs
  - checking temporal safety properties
  - uses CounterExample-Guided automatic Abstraction Refinement
  - succeeds or provides a counterexample or fails
Typical architecture for ESC

Spec# tool architecture:

Spec# (annotated C#)
Spec# Compiler
Annotated CIL
Translator
BoogiePL
VC Generator
Verification conditions
Automated Theorem Prover

Discussion of extended static checking

• Advantages:
  ▶ close to programming language and programmer
  ▶ good integration with normal IDEs
  ▶ in principle, no contact with the prover needed

• Disadvantages:
  ▶ specifications less expressive (why?), in particular w.r.t. abstraction
  ▶ error messages can be tricky if checking fails
  ▶ helping the prover can get difficult

Subsection 8.5.4

Specification and refinement

• Support the formal development from software models to programs
• Relate software models on different levels of abstraction
• Proof refinement properties by generating verification conditions
• Possibly several provers to discharge the VCs (automated and/or interactive)
Example systems

- **Event B**: [www.event-b.org/](http://www.event-b.org/)
  - correctness by construction in the tradition of VDM
  - system = software + environment: represented as transition systems
  - B notation following the Z notation
  - specific development and proof platform Rodin
  - programs are generated from most concrete model

- **KIV**: [www.informatik.uni-augsburg.de/lehrstuehle/swt/se/kiv/](http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/kiv/)
  - formal systems development and interactive verification
  - specification support:
    - functional aspects: abstract data types and HOL
    - state-based aspects: programs and abstract state machines
  - supports various kinds of refinements
  - sophisticated IDE for proof engineering

Discussion

- Software verification goes beyond program verification
- Other interesting aspects:
  - Correctness of software evolution steps
  - Correctness of refactorings
  - Correctness of compilers and programming tools

Subsection 8.5.5

**ATP: Automated theorem proving**

A rough classification:
The software verification tools use many techniques for automated proving, in particular:

- Superposition provers (e.g., SPASS, E)
- SMT solvers and model checkers (e.g., Z3, SPIN)
- Abstract interpretation and abstraction refinement