Lecture 9: Outline

- Automatic versus interactive theorem proving
- Interactive theorem provers: proof state, logical statements and goals
- Forward and backward (goal-oriented) proof styles
- Features of interactive proving in the Rodin platform
- Homework: a flight ticket system

The need for automation

- For even moderately sized programs (or their models), the complexity of correctness proofs becomes very high
- Assistance may be provided by a tool which records and maintains the proof as it is constructed step by step, and thus ensures its soundness
- Of course, the tool itself should satisfy basic requirements – be reliable and secure, i.e., perform only sound logical inferences
- Moreover, the logic of the tool should be expressive enough to represent all statements/expressions of our language or domain

Automatic vs interactive (theorem) proving

- Different levels of automation: fully automatic, fully interactive, something in between
- Fully automatic proving is more like a "push-button" technology
- The result of fully automatic proving is either confirmation of a successful proof of the given statement/property or a failure
- If a failure occurs, it is not clear, whether the result is wrong/false or the prover was not good enough to complete the proof
- Examples: "behind the scenes" application of automatic provers in Rodin or model checking in general

Automatic vs interactive (theorem) proving

- Fully automatic: (if succeeds) completely automatically produces the verification result (proved theorem). Everything happens as in "black box", no control over the process
- Fully automatic: usually possible for well-understood and precisely defined concrete classes of tasks/properties, when the proving goes according to the completely established pattern/procedure
- Proving of a particular task becomes just a concrete instantiation (of the parameters) of such a generic proving procedure
- Example: proving of a generic chain of B-Method/Event-B refinements for driverless train systems by Siemens Transportation France

Automatic vs interactive (theorem) proving

- Fully interactive: the user completely controls the proving process. The computer system serves as checker/enforcer of the user proving commands
- Fully interactive: a proof becomes a program (in a specific language of proving commands). Proving scripts can be saved and reused
- Fully interactive: (advantage) very flexible, allows to experiment with any kinds of properties that can be formulated in the provers logic
- Fully interactive: (disadvantage) needs much more knowledge and expertise (more like "white box", where you can more effectively use it if you fully understand it)

Automatic vs interactive (theorem) proving

- Even in full-fledged interactive theorem provers, it is usually a mixture of bigger automatic steps (relying on integrated simplifiers, solvers, decision procedures) and smaller interactive steps (using only basic prover proof commands)
- Still, the main distinction of interactive proving comparing to the automatic one is that the current proof state is open and controlled by the user
- The user (by the available proof commands) modifies the proof state step-by-step until a new theorem is produced or the proof fails

Theorem provers

- There are general purpose interactive theorem provers (sometimes called proof assistants)
- Most actively developed or used: Isabelle, HOL (HOL-Light), Coq
- Can be used to define and reason about variety of domains as mathematical theories
- Functional programming languages (e.g., PolyML, OCaml) are used to manipulate proof state and write proof scripts
- The Rodin platform is not a general purpose prover and quite limited as an interactive theorem prover

Interactive theorem provers: hierarchical architecture

- Internal logic: the basic mathematical theories (like predicate calculus or set theory) that are used to define new notions and their properties
- Proof (meta) logic: a collection of axioms (accepted without proving) + the logical rules describing how new correct statements (theorems can be produced from old ones
- Pre-defined or exported libraries of theorems and automated proving procedures (solvers, simplifiers, ...)

Theorems (logical sequents)

- Standard form of logical statement (a theorem if proved to be true):

$$H_1, H_2, ..., H_3 \ \vdash \ C$$

- Here $H_1, H_2, ..., H_n$ are the hypotheses (assumptions). All logical (true or false) expressions
- C – the conclusion (logical expression)
- Example:
  even $x \ \vdash \ (x = 2) \vee \neg$prime x

  or, equivalently,

  $\vdash \ \forall x.$ even $x \ \Rightarrow \ (x = 2) \vee \neg$prime x

Different ways of proving a theorem

- We can construct a new theorem in different ways
- It can be a given axiom $\Rightarrow$ it is a theorem right away (without a proof)
- It can be constructed from other theorems or axioms by combining them together in some way. Then it is called *forward proof*
- It can stated as a goal (potential theorem) and then simplified/split until all the parts are immediately provable (from existing already theorems). Then it is called *backward proof*

Forward proof

- Produces a new theorem from existing ones by combining them in some way
- Proving commands: inference rules
- Inference rule – a parameterised function that takes theorems and other parameters and returns a new theorem

Forward proof: examples

- Combining two theorems by conjunction: for

$$H_{11}, H_{12}, ..., H_n \vdash C_1$$

and

$$H_{21}, H_{22}, ..., H_m \vdash C_2$$

the inference rule CONJ produces

$$H_{11}, H_{12}, ..., H_n, H_{21}, H_{22}, ..., H_m \vdash C_1 \wedge C_2$$

- Merging two theorems into one

Forward proof: examples

- Specialising an universally quantified theorem: for

$$H_1, H_2, ..., H_n \vdash \forall x.\ C[x]$$

where $x \in \mathbb{N}$, the inference rule SPEC 0 produces

$$H_1, H_2, ..., H_n \vdash C[0]$$

with all x replaced with 0

- Generates a theorem for a specific case from a general (universally quantified) case

Forward proof: examples

- Modus ponens

$$H_1, H_2, ..., H_n \vdash P \Rightarrow Q$$

and

$$H_1, H_2, ..., H_m \vdash P$$

where $m \leq n$, the inference rule MP produces

$$H_1, H_2, ..., H_n \vdash Q$$

- Combines an implicative theorem with a theorem that proves its antecedent

Backward (goal oriented) proof

- Starts with potentially true statement (goal)
- The same standard form of logical statement (a theorem if proved to be true):

$$H_1, H_2, ..., H_3 \ -? \ C$$

- Proof commands: tactics. Used to modify/simplify/split the current goal until it can be proved in one step
- Tactics are functions that take a goal (logical sequent) + (possibly) some other parameters and produce a list of goals that are sufficient to prove the original one

Backward proof: examples

- Often, dual to the corresponding inference rules
- Splitting a conjunctive goal into two: for

$$H_1, H_2, \ldots, H_n \; \text{–?} \; C_1 \;\wedge\; C_2$$

the tactic CONJ_TAC produces

$$H_1, H_2, \ldots, H_n \; \text{–?} \; C_1$$

and

$$H_1, H_2, \ldots, H_n \; \text{–?} \; C_2$$

- Splitting a goal into two subgoals

Backward proof: examples

- Specialising an existentially quantified goal: for

$$H_1, H_2, ..., H_n \;\text{–?}\; \exists x.\ C[x]$$

where $x \in \mathbb{N}$, the inference rule EXISTS_TAC 0 produces

$$H_1, H_2, ..., H_n \;\text{–?}\; C[0]$$

with all x replaced with 0

- Generates a goal for a specific suggested case from a general (existentially quantified) case

Backward proof: examples

- Splitting a goal into two, by considering alternative cases: for

  $$H_1, H_2, ..., H_n \; {-?} \; C$$

  the tactic BOOL_CASES_TAC "x=0" produces

  $$H_1, H_2, ..., H_n, x = 0 \; {-?} \; C$$

  and

  $$H_1, H_2, ..., H_n, x \neq 0 \; {-?} \; C$$

- Splitting a goal into two subgoals, by adding extra assumptions for the cases considered

Working with the proof state

- The proof state contains, at least, the current unproved subgoals
- Usually it also stores the proof tree, with all previous proof branches and proof commands
- Applying a tactic in interactive proof modifies the proof state
- With the whole proof tree available, it is possible backtrack to some previous state and try again

Mixing the backward and forward proof styles

- In backward proof, we can combine both proving styles
- For instance, we can add new assumptions if these assumptions are based on proved theorems
- If the existing assumptions (considered as theorems) can produce new theorems by applying some inference rules, the resulting theorems can be added as new assumptions
- Example: if a universally quantified assumption $\forall x.C[x]$ can be specified for concrete x value, say x_0, then the new assumption $C[x_0]$ can be added

Other more advanced stuff

- Tacticals: merging tactics together
- Simplifiers, rewriters, decision procedures, model checkers
- Other proving styles: Mizar, window inference
- A big number of contributions: many theories, libraries, automated tools

Interactive proving in the Rodin platform

- Quite limited comparing to full fledged interactive theorem provers such as HOL, Isabelle, or Coq
- Rodin mostly relies on already integrated or possibly added as plug-ins automatic solvers (predicate provers pp and npp, mini-lemma prover/simplifier ml, the plug-in for integrating external SMT solvers, the plug-in for bridging with Isabelle)
- Interactive proving based on applying proof commands (tactics) for working with an unsolved current goal

Interactive proving in the Rodin platform (cont.)

- A separate Eclipse perspective – Proving
- Shows the current goal, hypothesis, proof tree, proof information in subwindows
- Provides access to automatic Rodin (installed) provers as well as simple proof tactics
- Proofs can be "pruned" in the Proof Tree view, essentially backtracking to some previous proof step

Interactive proving in the Rodin platform (cont.)

Some tactics:

- lasso – catching additional hypothesis for all the identifiers (names) in the goal
- ah (add hypothesis) – stating a simple property, which starts a separate proof. If successfully proven, the property is then added as extra hypothesis. An example of forward proof
- dc (deduce cases) – splits the proof for separate cases of the given expression (like the *BOOL_CASES_TAC* before)
- If the expression is boolean, two cases (it is true or false) are separately assumed and added as extra hypotheses. If the expression is an enumerated set, all different set values are assumed, splitting the proof into the respective number of cases

Interactive proving in the Rodin platform (cont.)

More tactics are hidden under red-underlined symbols in the hypotheses or the goal:

- Underlined $\forall$ symbol in a hypothesis. If clicked, allows to instantiate the hypothesis by submitting concrete values for the quantified variable (forward proof on a hypothesis)
- Similarly, underlined $\exists$ symbol in a goal. If clicked, allows to instantiate the goal by suggesting a concrete value for quantified variable
- Underlined $=$ in an equational hypothesis. If clicked, allows to rewrite the goal according to the hypothesis equation
- Underlined $\subset$ or $\subseteq$. If clicked allows to replace, e.g., a set inclusion $s1 \subseteq s2$ with the equivalent expression $\forall x.\ x \in s1 \Rightarrow x \in s2$
- ...

Customising Event-B tactics in Rodin

- Rodin preferences: Event-B: Sequent Prover : Auto/Post Tactic
- Allows to combine simple tactics and automatic provers, saving as separate profiles
- Such saved scripts can be chosen to apply instead of the installed default simplifying/proving procedures