On Strategies for Inductive Theorem Proving

Bernhard Gramlich
TU Wien, Austria

Outline

- Background
  - Basics about induction
  - (A)TP versus (A)ITP
  - Approaches to and assumptions about (A)ITP
- Strategies in (A)ITP
  - Why are strategies so important for (A)ITP?
  - Which are the relevant strategic issues?
- State-of-the-art
  - Systems for Induction
  - Successes and Failures
- Problems and Challenges
  - Technical / Logical Problems
  - Control Issues
  - Software Engineering Issues
- Theses
- Conclusion

Basics about induction (1)

- What is inductive?
  - *proof techniques* can be inductive (using some form of well-founded induction) (syntactical, proof-theoretic)
  - *notion of validity* of statements may be inductive (semantic)
  - *learning process* may be inductive (another field!)
- Where does induction occur (in computer science)?
  - almost everywhere!
  - recursively defined data structures
  - programs / specifications correspond to concrete (classes of) models
  - usually (implicitly) assumed: no junk
  - properties of specifications / program verification: only models of interest relevant (often: standard model)

Basics about induction (2)

- syntactically / proof-theoretically (well-founded induction)

$$\frac{\forall x.\,[\,(\,\forall y.\,[\,y < x \Rightarrow P(y)\,]\,) \Rightarrow P(x)\,]}{\forall x.P(x)}$$

  with $<$ well-founded order (on domain of x)

- semantically / model-theoretically (inductive validity)
  - $\vdash F$ (derivable, deductive theorem) (assuming given *axioms* and *inference rules*)
  - $\mathcal{F} \vdash \mathcal{G}$ (syntactic entailment)
  - $\models F$ (holds in all models of initial axioms)
  - $\mathcal{M} \models F$ (holds in a particular (standard) model $\mathcal{M}$ of initial axioms)
  - $\mathbf{K} \models F$ (holds in a particular class of models of initial axioms)
  - $\mathcal{F} \models \mathcal{G}$ (semantic entailment)

Basics about induction (3)

- properties
  - soundness: $\vdash F \;\Rightarrow\; \models F$
  - completeness: $\models F \;\Rightarrow\; \vdash F$
  - induction: semantic definition usually corresponds to infinitary syntactic property, e.g.:

$$T(\Sigma)/_{=_E} \models s = t \iff \forall \sigma \in \; GSub\colon E \vdash s\sigma = t\sigma$$

  - however: notion of inductive validity may also be mixed
    - constructor theorems [Zhang'88]
    - validity in all (maximal) consistent extensions [Kapur/Musser'87]
    - monotonic notions of inductive validity [Wirth/Gramlich'94]
  - in these cases: tradeoff adequacy – complexity (e.g.: simple correspondence between syntax and semantics may be lost)
  - motivation: partiality / non-monotonicity phenomena

(A)TP versus (A)ITP (1)

theorem proving tasks considered

- TP = (general) *theorem proving*
- ATP = *automated* theorem proving
- ITP = *inductive* theorem proving
- AITP = *automated inductive* theorem proving

applications

- (A)TP
  - mathematics (algebra, logic, axiomatization of structures, ...)
  - computer science (boolean reasoning, parameterized specifications with loose semantics, specifications with no excluded models)
- (A)ITP
  - mathematics (logic, validity in standard models, model theory)
  - computer science (abstract data types, specifications and programs relying on natural semantics, verification of their properties)

(A)TP versus (A)ITP (2)

what are the differences?

- $ITh(E) \supseteq Th(E)$, $ITh(E) \not\subseteq Th(E)$ (otherwise: ω-complete)
- (A)TP: calculi sound and complete, $Th(E)$ r.e., cut elim.
- (A)ITP: calculi sound but (necessarily) incomplete, $ITh(E)$ not r.e., no cut elimination

differences in terms of proof search space

- (A)TP
  - search space (tree) finitely branching
  - counterexamples (via model building) may help, but are not essential
  - proof search control challenging
- (A)ITP
  - search space (tree) infinitely branching
  - counterexamples are very much essential (to eliminate wrong conjectures and cut useless branches)
  - proof search control extremely challenging

Approaches to and assumptions about (A)ITP (1)

some features (logic and framework)

- first-order vs. higher-order
- full first-order vs. universal fragment
- unsorted vs. many-sorted vs. order-sorted
- notion of inductive validity
- constructor vs. destructor style induction
- fixed vs. lazy induction ordering (generation) [e.g. Protzen'94]
- based on resolution, natural deduction, ..., saturation, ...
- equational vs. non-equational
- explicit induction [e.g. Bundy'01] vs. implicit induction (inductionless induction, proof by consistency) [e.g. Comon'01]

Approaches to and assumptions about (A)ITP (2)

some features (system architecture, purpose, control)

- stand-alone tool vs. component in bigger system
- homogeneous vs. heterogeneous tool
- built-in theories vs. explicit handling
- subtools for specialized proof tasks (decision procedures . . .)
- intended functionality (yes/no vs. justifications, proof objects, reuse, incrementality, modularity)
- general purpose reasoning system vs. specialized tool for particular problem domain
- experimental tool vs. high-fidelity system (certification)
- post- vs. pre- vs. integrated development
- desired degree of automation / interaction
- proof search control architecture / concept / language

Approaches to and assumptions about (A)ITP (3)

setting here (for simplicity)

- Φ inductive consequence of E (first-order clauses), iff, for every Herbrand interpretation $\mathcal{H}$:

$$\mathcal{H} \models E \Rightarrow \mathcal{H} \models \Phi$$

- axioms (specifications) E: Horn clauses with equality
- then unique (smallest) Herbrand interpretation I_E of E exists
- moreover, for positive clauses c:

$$c \text{ inductive consequence of } E \text{ iff } I_E \models c$$

however, also in other settings

- the analysis is essentially the same
- the basic problems are the same
- only their technical appearance is different

Outline

- Background
  - Basics about induction
  - (A)TP versus (A)ITP
  - Approaches to and assumptions about (A)ITP
- Strategies in (A)ITP
  - Why are strategies so important for (A)ITP?
  - Which are the relevant strategic issues?
- State-of-the-art
  - Systems for Induction
  - Successes and Failures
- Problems and Challenges
  - Technical / Logical Problems
  - Control Issues
  - Software Engineering Issues
- Theses
- Conclusion

Why are strategies so important for (A)ITP?

main reasons

- incompleteness of (inductive) proof methods
- structure and size of search space (infinitely branching in several dimensions)
- recursive nature of inductive proof attempts
- difficulty of measuring progress
- control of search space
  - what should be done (attempted) next?
  - when should a proof attempt be considered to be failed (hopeless)?
  - what to do in this case?
  - when should backtracking be applied?
  - when and how to generalize?
  - when and how to simplify (how far)?
  - when and how to start induction (generate induction schema)?
  - how to make induction hypothesis applicable (goal-directed)?
  - when and how to perform case analysis?

Which are the relevant strategic issues? (1)

typical inductive proof structure

- try non-inductive methods
  - inconsistency test, counterexample test
  - ATP attempt (without induction)
  - simplify as much as possible wrt. current database of definitions and lemmas, ...
  - but: simplifiability may require inductive arguments!
- try induction
  - analyze recursion structure in conjecture and involved functions
  - generate candidate(s) for appropriate induction schemas based on this analysis (involving some look-ahead)
  - perform induction
    - split into cases
    - simplify
    - make induction hypothesis applicable (cross-fertiziling, rippling [Bundy et al'89+] ...)
  - generalize conjecture
  - generate (auxiliary) lemma

Which are the relevant strategic issues? (2)

essential strategic control issues

- when to test for inconsistency? for the existence of counterexamples?
- when to try ATP without induction?
- simplification infinitely branching/critical
  - when?
  - how? using which definitions / lemmas? in which order?
  - (recursively) use induction to verify applicability of lemmmas? $C = D[l\sigma]$, $l \rightarrow r \Leftarrow c \in L$, $c\sigma$?
  - simplify to normal form?
  - inverse simplification (expansion)? how far? non-termination!
- induction infinitely branching/critical
  - compute and select appropriate induction scheme
  - generate corresponding proof tasks

Which are the relevant strategic issues? (3)

essential strategic control issues (cont'd)

- case analysis — infinitely branching/critical
  - when? how? according to which criteria?
  - many cases $\longrightarrow$ less general, fewer cases $\longrightarrow$ more general
  - how to verify individual cases?
- generalization — infinitely branching/critical
  - when? how? syntactical/semantical? directly/indirectly?
    e.g.: $(A_1 \lor A_2 \Leftarrow B_1 \land B_2)\sigma$ into $A_1 \Leftarrow B_1$
  - using look-ahead?
- lemma generation / speculation — infinitely branching/critical
  - when? how? for what purpose?
  - goal-directedness?
  - does lemma suffice? or only help?
  - organizational: top-down (relative ITP) or bottom-up (proofs are final)

Which are the relevant strategic issues? (4)

organizational strategic issues (cont'd)

- concerning the data- and knowledge base (lemmas, . . .)
  - in case of successful proof attempts
    - which (intermediate) lemmas should be kept (stored)?
    - in which form? in which order?
    - as what kind of knowledge (rewrite, type, definition, . . .)?
    - in structured (hierarchical / graph) form or flat (non-linked)?
    - should knowledge base be modified (simplified) wrt. newly proved inductive theorem?
  - in case of failed proof attempts?
- concerning the control structure and knowledge base
  - compute recursion analysis when introducing new definitions
  - memory and history mechanism (e.g. for avoiding loops)
  - overall proof search control structure
  - layered model of proof search control (via strategies and heuristics)

Outline

- Background
  - Basics about induction
  - (A)TP versus (A)ITP
  - Approaches to and assumptions about (A)ITP
- Strategies in (A)ITP
  - Why are strategies so important for (A)ITP?
  - Which are the relevant strategic issues?
- State-of-the-art
  - Systems for Induction
  - Successes and Failures
- Problems and Challenges
  - Technical / Logical Problems
  - Control Issues
  - Software Engineering Issues
- Theses
- Conclusion

State-of-the-art

assessment / comparison difficult

- no (regular) competition of (A)ITP systems
- no (widely accepted and used) benchmarks
- underlying logics and intended usage rather diverse
- amount of automation / interaction?
- usage typically requires quite considerable expertise

general observations

- full automation generally not (yet?) successful
- typical usage: (specification and) proof engineering with
  - human guidance for modelling, proof structure / ideas, critical
  - human guidance for lower-level control if necessary critical
  - human failure analysis (with few automatic support) critical
- tradeoff automation – interaction (wrt. efficiency, success rate, required expertise, flexibility of control, ...)

Systems for Induction (1)

prominent systems (maintained)

- ACL2 (NQTHM successor) [Kaufmann/Manolios/Moore'02]
  - efficient functional PL + ITP system
  - impressive collection of non-trivial examples
  - relatively high automation
- PVS [SRI]
  - provides mechanized support for formal spec. and verification
  - based on classical, typed higher-order logic
  - partially automated, framework for decision procedures
- VSE/INKA [Hutter et al, DFKI]
  - tool for supporting the formal software development process
  - one focus: inductive proofs
  - sophisticated search control strategies (during induction)
- ISABELLE/HOL [Nipkow/Paulson et al]
  - generic theorem proving environment and proof assistant
  - main application: formalization of math. proofs/ formal verif.
  - very flexible, much interaction with expert user required

Systems for Induction (2)

other systems (some)

- RRL [Kapur/Zhang]
  - rewrite-based, first-order
  - different inductive proof techniques, high automation
- Oyster/CLAM [Bundy et al]
  - tactic-based proof editor based on Martin-Löf constructive type theory + proof planner
  - meta language for constructing customized tactics for individual conjectures, especially for rippling
- QuodLibet [Kühler/Wirth]
  - specification language and ITP system for data types with partial operations
  - flexible control, user-oriented, with some automation
- PerfectDeveloper [Crocker;Escher Tech.]
  - program development and verification tool for generating Java or C++ programs
  - Hoare-style pre-/post-condition approach, partially automated

Successes and Failures

successes

- computer supported specification and verification of complex systems / relationships possible, e.g.
  - prime factorization theorem
  - undecidability of halting problem
  - specification and verification of microprocessors
- often revealed errors in initial specifications / conjectures

failures

- in general low automation degree
- inductive specification and proof engineering is tedious and difficult
- automatic support for failure analysis unsatisfactory
- building-in intelligence much more difficult than expected (by optimists)

Outline

- Background
  - Basics about induction
  - (A)TP versus (A)ITP
  - Approaches to and assumptions about (A)ITP
- Strategies in (A)ITP
  - Why are strategies so important for (A)ITP?
  - Which are the relevant strategic issues?
- State-of-the-art
  - Systems for Induction
  - Successes and Failures
- Problems and Challenges
  - Technical / Logical Problems
  - Control Issues
  - Software Engineering Issues
- Theses
- Conclusion

Problems and Challenges (1)

Technical / Logical Problems

- building-in knowledge (when? how?)
- structuring / modularizing specifications and proof tasks
- extending decidable cases (classes)
- combining ITP system with tools for special purposes
  - systems for particular data types / theories
  - decision procedures for restricted theories
  - combination mechanisms
  - logical / operational interface?
- better methods for goal-directed reasoning
- better methods for look-ahead based reasoning
- generalization: when? why? how?
- generation of (auxiliary) lemmas: when? why? how?
- how to recognize, analyze and deal with failure?
- how to make ITP more robust (monotonic, semantic)?

Problems and Challenges (2)

Control Issues

- good strategies/heuristics are vital in ITP to generate and deal with reasonable proof attempts
- basic question: what to do next, in view of the history, the current data and the knowledge base?
- overall proof search model of an (A)ITP system
  - is necessarily complex; must be flexible
  - needs automation and interaction (for proof engineering)
  - must allow/support interrupts, inspection, failure analysis and relative proving
  - should guarantee correctness requirements
  - should be compatible with user interaction, navigation (in search tree), information extraction, generation of proof objects
  - must have different layers (for different types of reasoning)
  - should allow the integration of/in other tools for subtasks/as subsystem
  - needs to integrate strategic/heuristic user input

Problems and Challenges (3)

Software Engineering Issues

- how to design/implement/apply a structured control concept for proof search that
  - is user-friendly, fully transparent, intelligible, flexible, extensible and modifiable
  - generates complete proof objects
  - has a programmable strategy/heuristics language with clearly defined semantics
  - allows efficient proof engineering in real time
  - allows unsafe reasoning (relative to unproved lemmas)
  - allows a high degree of automation
  - enables human user to quickly test/implement/model key ideas
  - is able to integrate new tools / subtools (e.g. decision proc.)
  - can easily be specialized to specific domains
  - has an appropriate system for mainting / adapting / its (large!) knowledge base
- HCI: how to do all this in a smart way as to interaction?

Outline

- Background
  - Basics about induction
  - (A)TP versus (A)ITP
  - Approaches to and assumptions about (A)ITP
- Strategies in (A)ITP
  - Why are strategies so important for (A)ITP?
  - Which are the relevant strategic issues?
- State-of-the-art
  - Systems for Induction
  - Successes and Failures
- Problems and Challenges
  - Technical / Logical Problems
  - Control Issues
  - Software Engineering Issues
- Theses
- Conclusion

Theses

- AITP in the near future will only be successful for
  - very specialized domains (e.g., with fixed axiomatizations)
  - for very restricted classes of conjectures
- substantial progress?
  - increased robustness (more monotonic, semantics based)
  - modularization, structuring and interaction of theories, proofs, proof search
  - appropriate framework(s) to model (and implement) strategic proof search control
    - expressive
    - flexible: extensible, adaptable, programmable
    - well-defined semantics
    - layered (different levels)
  - progress will take time, breakthroughs are unrealistic

Conclusion

- (A)ITP extremely important and ubiquitous as proof tasks
- (A)ITP hopeless without sophisticated strategic guidance
- (A)ITP will remain a very challenging specification and proof engineering process
- necessary
  - more foundational research (decidable case, decision procedures, proof search models and architectures, strategies/heuristics)
  - more specialized (interesting) problem domains and (A)ITP systems
  - another point of view: (A)ITP as integrated combined development (specification + verification) engineering that
    - needs human guidance for high(er)-level decisions and key ideas
    - provides as much automatic support as possible