Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 125 (2005) 5–43
www.elsevier.com/locate/entcs

Strategic Issues, Problems and Challenges in Inductive Theorem Proving

Bernhard Gramlich[1]

Fakultät für Informatik, TU Wien
Favoritenstr. 9 – E185/2, A–1040 Wien, Austria

Abstract

(Automated) *Inductive Theorem Proving* (ITP) is a challenging field in automated reasoning and theorem proving. Typically, *(Automated) Theorem Proving* (TP) refers to methods, techniques and tools for automatically proving *general* (most often first-order) theorems. Nowadays, the field of TP has reached a certain degree of maturity and powerful TP systems are widely available and used. The situation with ITP is strikingly different, in the sense that proving inductive theorems in an essentially automatic way still is a very challenging task, even for the most advanced existing ITP systems. Both in general TP and in ITP, strategies for guiding the proof search process are of fundamental importance, in automated as well as in interactive or mixed settings. In the paper we will analyze and discuss the most important strategic and proof search issues in ITP, compare ITP with TP, and argue why ITP is in a sense much more challenging. More generally, we will systematically isolate, investigate and classify the main problems and challenges in ITP w.r.t. automation, on different levels and from different points of views. Finally, based on this analysis we will present some theses about the state of the art in the field, possible criteria for what could be considered as *substantial progress*, and promising lines of research for the future, towards (more) automated ITP.

Keywords: Inductive theorem proving, automated theorem proving, automation, interaction, strategies, proof search control, challenges.

1 Background and Overview

Theorem proving tasks are ubiquitous in computer science, e.g., in fields like program specification, transformation and verification. Most often the given, generated or resulting proof tasks are not *general* ones (in the sense, that validity in **all** models of the underlying logic specification is to be established),

[1] Email: gramlich@logic.at www: http://www.logic.at/staff/gramlich/

doi:10.1016/j.entcs.2005.01.006

but more specific ones, where (in)validity is to be shown only in some subclass of models (or in a unique *standard model* provided it exists). The class of formulae being valid in such restricted classes of models is usually much bigger than the one consisting of the general theorems. Typically, any reasonable approach for proving such properties – *inductive theorems* – needs some kind of *induction* (which is one source of explanation why such theorems are called *inductive theorems*). [2] In fact, in computer science applications, in particular in (formal foundations of) software engineering and reasoning about specifications and programs, most often the specifier, programmer or user is not interested so much in all models of a given specification (or axiomatization), but rather in a specific (and often unique) natural model that is induced by the specification and that closely corresponds to her/his intuition. This means that for proving properties (general) TP is usually insufficient and can only be used as a kind of first default approach. If a conjecture is provable via TP, then it is also an inductive theorem, but if it is not provable by TP (or even proved to be not valid in general), it may still be an inductive theorem.

Automated TP has turned out to be fairly successful nowadays (in its range of applications), whereas this does not really hold for ITP, more precisely for automated ITP. Hence, a better understanding of the differences may also be helpful for improving ITP approaches and techniques.

The paper is primarily non-technical and high-level, focusing on relevant issues in (automated) ITP, that make the problem difficult. We assume some basic knowledge, terminology and notations in first-order logic. Though the presentation is essentially non-technical, to really understand, appreciate or criticize the analysis developed, especially the conclusions and interrelations, we conjecture that some familiarity with ITP in theory and practice will be necessary, since without own experience the problems in ITP are typically underestimated. For the interested reader, we give a lot of references to works in the field that may serve as a starting point for a more detailed study of various aspects and approaches in ITP. Although the links to the literature are fairly comprehensive, we do not claim any kind of completeness with this bibliography.

[2] In fact, most often also the definition of *inductive theorem* can be given in such a way that it has an *inductive nature*, in the sense that from many – typically infinitely many (ground) – instances of a formula a more general one is (inductively) deduced, cf. e.g. [12]. Furthermore, it should be noted that the kind of *induction* (as proof method) we are dealing with here should not be confused with *induction* in the field of *inductive learning* as it is used for instance in the *ALT (Algorithmic Learning Theory)* conferences.

As main contributions we consider the systematic analysis and evaluation of the main problems and challenges in ITP. They finally yield an assessment of the state of the art in the field, with some theses about what could be considered as *substantial progress*, with corresponding promising lines of research for the future, towards (more) automated ITP.

Before going into details let us mention a couple of papers and proceedings, that are not explicitly discussed later on, where certain issues related to proof search in TP and ITP have been discussed in some depth. One major forum for such work is the workshop on *Strategies in Automated Deduction*, held since 1997, cf. [90], [93], [91], [92], [22]. Another source is the workshop on the *Automation of Inductive Proof*, held irregularly since the beginning of the 1990s, but with at most informal proceedings (cf. e.g. [108]). Further literature includes [45,48], [49], [64], [65], [66], [68], [86,87,88,89], [104,106,110], [119], [120], [125], [130], [135], [147], [149], [154], [155], [157,158,159], [161,162], [171], [172], [175], [177], [178], [196], [203], [231], [233], [234].

The plan of the paper is as follows. In this section we will deal with some basics about induction, with TP versus ITP, and with different approaches to and assumptions about ITP. In Section 2 we will in detail cover and discuss the relevant strategic issues and problems in ITP. In Section 3 we will give an assessment of the current state of the art and of some prominent ITP systems used nowadays (again without claiming completeness), together with a summary of successes and failures. Finally, in Section 4 we will isolate, summarize and interrelate what we consider to be the main problems and challenges in ITP, giving rise to a few theses in Section 5 about the future, and about promising lines of research / criteria for substantial progress. These theses may be quite debatable and may not be widely shared, but there is quite some evidence for them. If they were to entail a lively discussion in the research community, this would already be a very positive consequence.

1.1 Basics about Induction

As already mentioned, the notion *inductive* is ambiguous in its usage, though the underlying concept – going from specific instances to something more general – is the same. *Proof techniques* can be inductive insofar as they employ *induction schemata* based on *induction principles*. Also the notion of *validity of statements* may be inductive. There exist other notions and areas where *induction* and *inductive processes* make sense, like (algorithmic) *inductive learning* mechanisms. We are concerned with the former two, namely inductive proofs of theorems (more precisely, conjectures) or proofs of inductive properties. Before having a closer look at what this means more precisely,

let us consider where induction in this sense occurs (in computer science and mathematics). The easy answer is: Almost everywhere! For instance, when reasoning about recursively defined domains or data structures with functions defined on them, or about programs and specifications, then most often not only general TP is needed, but more, namely ITP. Especially in computer science where logic-based specifications are often (explicitly or implicitly) assumed to constructively specify and/or model some system or computation task, one is often interested not in all conceivable models of a given specification, but only in certain ones (i.e., in some restricted class of models) or even only in a distinguished particular one (a *standard model*) capturing the essential properties that one has in mind. Such restrictions can often be characterized by requiring that every element in some model considered has to be *term generated*. In other words, elements in models need to have a syntactic counterpart in the specification. In a sense, this is a very natural requirement in a computer science context for very many, though not all, applications. Unfortunately, in the literature on verification and theorem proving the distinction between (general) validity and inductive validity (theorems and inductive theorems, respectively) is sometimes not appropriately mentioned or even ignored. For certain purposes this is quite problematic, since, as we shall discuss, the proof-technical implications are far-reaching.

The very general *principle of well-founded induction*, for proving some property P depending on a parameter x, may be stated in the form of an inference rule as follows:

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

with $<$ a well-founded order (on the domain of x). The typically used induction principles and schemas (*natural* induction, *structural* induction, *course-of-values* induction etc.) are all obtained from this general principle by appropriate instantiations.

Next let us briefly review basic notions and relationships between syntax and semantics, especially between derivability and validity (in a first-order logic framework). Typically, based on some set of some set of *axioms* and some some set of *rules*, the resulting *inference system* defines $\vdash F$, (syntactic) *derivability* of a formula F. If $\vdash F$ is derivable, it is a *deductive theorem* (of the underlying set of axioms in the given inference system). *Syntactic entailment* or *relative derivability* of F from some set $\mathcal{G}$ of assumptions (considered as additional axioms) is denoted by $\mathcal{G} \vdash F$, in which case F is called a *deductive consequence* of $\mathcal{G}$.

One the semantic side, one usually has a notion of *interpretation* of a given

logical specification. An interpretation satisfying a specification $\mathcal{G}$ is a *model* of $\mathcal{G}$. A formula is F *valid* ($\models F$) if it holds in all models of the initial axioms. *Semantic entailment* or *relative validity* of F from some set $\mathcal{G}$ of assumptions (considered as additional axioms) is denoted by $\mathcal{G} \models F$, where F is called a *semantic consequence* of $\mathcal{G}$. Abusing notation, the fact that F holds in some particular model $\mathcal{M}$ of the initial axioms (or in some class $\mathcal{K}$ of models of the initial axioms) is also denoted by $\mathcal{M} \models F$ (or $\mathcal{K} \models F$, respectively).

Now, for obvious reasons — in logic and (automated) TP — one strives for the equivalence of (semantic) validity and (syntactic) derivability, and for effective and efficient ways to establish the latter. In first-order logic, any reasonable inference system is well-known to be both *sound* ($\vdash F \Rightarrow \models F$) and *complete* ($\models F \Rightarrow \vdash F$). Unfortunately, this nice (finitary) correspondence between validity and derivability is in general lost in ITP. We will not discuss this phenomenon in depth here, but mention only one particular simple case. Consider some set E of (implicitly universally quantified) equations over some signature Σ. Then, *inductive* or *initial* validity of some equation $s = t$ is given by $T(\Sigma)/_{=_E} \models s = t$, where $T(\Sigma)/_{=_E}$ is the ground term algebra factored through the congruence induced by E. To capture this notion of inductive validity syntactically, i.e., deductively or in terms of derivability, requires an infinitary characterization:

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

Here, the substitution σ ranges over all *ground substitutions* over the given signature. To solve this infinitary proof task one typically uses an (appropriate) *inductive* proof technique, to show that $E \vdash s' = t'$ holds indeed for all ground instances $s' = t'$ of $s = t$.

In the above case, it is at least possible to characterize inductive validity by an infinite conjunction of derivability statements, i.e., to relate (inductive) validity and derivability in a clear way. This need not always be so easily the case. In fact, this depends very much on the notion of inductive validity that is used. In various cases, there are some good reasons to choose a version of inductive validity that deviates from the standard way of defining it. We briefly mention here only one aspect and give a few pointers to corresponding literature. The basic motivating aspect to proceed differently is an undesirable *non-monotonicity* phenomenon. To illustrate this, consider an equational specification E for addition given by $0+x = x$, $s(x)+y = s(x+y)$. Then it is easy to verify (prove) that $(*)$ $x + 0 = x$ is an inductive theorem of E, i.e., it holds in the initial algebra $T(\Sigma)/_{=_E}$. Now we enrich the original specification by a (consistent) definition for subtraction, via $x - 0 = x$ and $s(x) - s(y) = x - y$ yielding E'. However, somehow contrary to intuition,

now $(*)$ is no longer an inductive theorem of E'! The (essential) reason is that the enrichment has introduced new *junk* terms in the initial model which destroy the desired property. To wit, substituting e.g. $0-s(0)$ for x, we obtain $(0-s(0))+0 = 0-s(0)$ which cannot be proved in E'. Here, the enrichment was *consistent* (i.e., no previously distinct elements have been identified), but *incomplete* in the sense that the new operation was only *partially defined* (in terms of the "old data constructors").

Starting with the work of [128,127] (on *proof by consistency* and *completion of incomplete specifications*), and of [232], [235] who used so-called *constructor models*, this line of reasoning — to adopt, define and use "more monotonic" versions of inductive validity — has been further discussed, developed and refined subsequently, e.g. in [223], [230,229], [169], [8]. We will not discuss these approaches in detail, but it should be noted that the above non-monotonicity phenomenon has serious consequences in ITP. Typically, one either imposes a very strict specification discipline thus avoiding partiality and non-monotonicity problems (this approach is adopted in most current ITP systems), or one has to work with a more involved framework that is more liberal, but also technically more complicated. So, in some sense there is a trade-off between adequacy and feasibility of the existing approaches.

1.2 TP vs. ITP

Next we shall discuss a bit things in common of and differences between TP and ITP, where depending on the context we assume that the goal is to be as automatic as possible.

1.2.1 Comparison from a Logical Point of View

Suppose E is a first-order specification with equality such that it admits some unique *standard model*.[3] Let us denote the set of all valid formulæ in E by $Th(E)$, and the set of inductively valid formulæ, i.e., those that are true in the standard model of E, by $ITh(E)$. Then, from a logical point of view we can make the following observations:

(1) We have $Th(E) \subseteq ITh(E)$, but in general $ITh(E) \not\subseteq Th(E)$.
(2) Any reasonable calculus for $Th(E)$ is sound and complete, and $Th(E)$ is (in general) undecidable but semi-decidable, i.e., $Th(E)$ is recursively enumerable but not its complement. Moreover, *cut-elimination* is possible.

[3] For instance, if E is a set of (implicitly universally quantified) positive-conditional equations where the conditions are conjunctions of equations, then the existence of such a unique standard model (the *initial algebra*) is guaranteed.

(3) Any reasonable calculus for $ITh(E)$ is sound, but in general necessarily incomplete.[4] $ITh(E)$ is in general neither decidable nor semi-decidable, i.e., not even recursively enumerable. Moreover, *cut-elimination* is not possible ([167]).

Actually, regarding (1), in combination with (2) and (3), there are cases where $ITh(E)$ coincides with $Th(E)$, hence where $ITh(E)$ becomes semi-decidable in particular. One case where the latter obviously holds is given when the inclusion $Th(E) \subseteq ITh(E)$ is non-strict. For instance, if E is purely equational, then E is called ω*-complete* if an equation is valid iff it is inductively valid. In such ω-complete (equational) specifications, trying to prove inductive validity of a conjecture is the same as trying to prove its (general) validity. Unfortunately, such cases where ITP and TP coincide are only possible for very restricted cases and settings, and do not apply in most verification problems where ITP is involved. Yet, it should be noted that there are known cases where inductive validity of certain formulae is decidable, as well as concrete decision procedures for certain sub-tasks. Such knowledge and corresponding effective decision procedures are crucial ingredients of state of the art TP- and ITP-systems, cf. e.g. [199], [63], [216], [184,185], [209], [6], [121], [141], [126], [78,79], [140], [219,220].

Concerning ω-complete specifications which have interesting applications for instance in *process algebra*, we refer to e.g. [98], [96], [1], [95], [170], [18], [71].

Regarding (2) and (3), these relationships and properties show fundamental differences between proving validity and proving inductive validity that go back to pioneering works in theoretical computer science and mathematical logic, cf. e.g. [80], [97], [217,218], [156], [167]. In particular, w.r.t. (3), there is no first-order proof system with induction that is complete for $ITh(E)$.The impossibility of cut-elimination ([167]) also has severe consequences, and constitutes a substantial difference to general TP. In essence, this means that in ITP auxiliary lemmas (to be guessed) can in general not be avoided.

Concerning the fact that $ITh(E)$ is not even recursively enumerable, this indicates (among other aspects) that one should, when trying to prove an inductive conjecture, make sure that the positive proof attempt can possibly be successful, by (also) searching for a contradiction and thus excluding as early as possible false conjectures.

[4] In essence, this is due to Gödel's first incompleteness theorem that states that in any formal (deductive) system for arithmetic there are formulæ that are true but unprovable (cf. [80], [97]).

1.2.2 Comparison in Terms of Proof Search Aspects

Here we only give a first rough account of essential differences between TP and ITP regarding the problem of proof search. Some of these aspects will be detailed later on.

From an abstract point of view the proof search, the proof search tree construction and proof search control can be characterized as follows:

- TP:
  - The proof search tree is **finitely branching**, for any reasonable (sound and complete) underlying inference system like *ordered resolution* cf. e.g. [11] and paramodulation-based theorem proving (cf. e.g. [186]).
  - Searching for and constructing *counterexamples* in the proof search process (for instances via *model building* techniques, cf. e.g. [69], [197]) **may help** but is in some sense **not essential** for successful proofs.
  - The proof search control in (first-order) TP is (known to be) **challenging**, especially when trying to be efficient but still complete.
- ITP:
  - The proof search tree is **infinitely branching**, for any reasonable (sound) underlying inference system, and for several different reasons.
  - Searching for and constructing *counterexamples* in the proof search process is **very much essential** and **highly important** in practice.
  - The proof search control in (first-order) ITP is **extremely challenging**, for almost every non-trivial inductive conjecture, and sometimes even for trivial ones.

That the proof search tree is infinitely branching in ITP, as opposed to TP, is in essence due to the incompleteness of inductive reasoning power. A basic source for this infinitary character is already given by the fact that usually there are infinitely many (sound) induction schemas that might serve for a proof attempt. Additionally, as a well-known consequence in practice, one often has to introduce various *guessing steps*, like *generalizing conjectures*, inventing auxiliary *inductive lemmas* (to be proved as well) and introducing *inductive case splittings*. These inductive guessing steps imply that often it is not clear whether the actual conjecture treated can indeed be an inductive theorem. Hence, to eliminate wrong conjectures and cut useless branches, it is vital to simultaneously try to disprove conjectures, e.g., by searching for counterexamples. [5] The combination of these aspects entails that in ITP

[5] Somehow, it seems that in the automated reasoning community, the theme of (explicit) *disproving* of conjectures, including model building and the construction of counterexamples, has not yet received the attention it deserves (see, e.g., [200]). For a recent attempt to focus on this theme see e.g. [3].

systems with some substantial amount of automation the proof search control is extremely challenging.

1.3 Approaches to and Assumptions about ITP

For the purpose of the paper it is not really necessary to present and discuss the existing approaches to ITP in appropriate depth. This would also be clearly beyond the scope and require much more space. However, we want to mention at least some issues that are important in practice and that have consequences for the design and implementation of ITP systems.

First let us consider some issues and aspects regarding the logic and the framework in different approaches.

- First-order vs. higher-order logic: Though higher-order logic is more expressive, for the purpose of (automation of) ITP, first-order logic is already challenging enough.
- Full first-order vs. universal fragment: Most ITP systems and approaches concentrate on universally quantified first-order formulae. Allowing existential quantification (or arbitrary quantifier alternations) clearly complicates things considerably. Typically, existential inductive conjectures are approached by trying to constructively find witnesses and prove that they satisfy the respective inductive property, cf. eg [39,42], [20], [148], [105], [57], [152], [166], [198], [174].
- Unsorted vs. many-sorted vs. order-sorted: Including / requiring a more strict typing discipline may help a lot to avoid useless computations in proof search, but of course also puts restrictions on the modeling of systems and specifications. The typing concepts of ITP systems are quite diverse, ranging from rather untyped (e.g. [39]) to strongly typed systems (e.g. [190]).
- Partiality and the notion of inductive validity: The treatment of *partiality*, i.e., of only partially defined new functions is an important issue (see also the discussion above). Allowing unrestricted partiality when extending specifications often invalidates most previous inductive theorems. Hence, either one has to adapt the notion of inductive validity (cf. e.g. [235], [128,127], [230,229], [169]), or to make partiality non-critical, for instances by simply forbidding it, or by making specifications total in some well-defined way (cf. e.g. [39], [41], [176]). In any case, the treatment of partiality has far-reaching consequences for the framework and the whole ITP process.
- Constructor vs. destructor style induction: Concerning the representation of basic data structures, one may either constructively represent data object with *constructor* functions, or destructively extract the relevant informa-

tion via *destructor* (or *selection*) functions, like for lists with constructors *nil*, *cons* and destructors *car*, *cdr* (cf. e.g. [39]). Normally, the representations and proofs can be easily translated from a constructor to a destructor framework and vice versa, however, from a proof-technical point of view it is advisable to stick to one of these dual frameworks.

- Fixed vs. lazy induction ordering (generation): Most approaches to ITP start a proof attempt for an inductive conjecture by analyzing the formulae (and the underlying specification) and devising an appropriate (and by then **fixed**) induction schema for it according to which they then proceed. An alternative approach is to start the proof without a fixed induction schema, but rather gather information along the proof attempt about which induction schema would turn the reasoning indeed into a correct inductive proof. This latter approach has been pursued e.g. in [201], and is also partially incorporated in [169], [8]. Of course, the generality and elegance of this latter approach has its price. Namely, correctness of the overall inductive reasoning becomes non-trivial and has to be established a posteriori, and — more severely — proof-technically the search becomes more difficult since goal-directedness, which usually heavily relies on an existing fixed induction schema, is not easily obtained anymore.
- Underlying (deductive) logical framework: This may be based on a couple of different approaches (for first-order logic with equality), like ordered resolution, superposition, paramodulation, etc.. It may be saturation-based or not, equational or non-equational, based on clausal normal forms or on arbitrary clauses etc., with all advantages and disadvantages known for these approaches. All these issues are certainly relevant for proof search in ITP, but will not be discussed here.
- Explicit vs. implicit induction: The early works on (automated) ITP (cf. e.g. [36,37,39,38,42,43], [223]) all use explicit induction, in the sense that for a given inductive conjecture a concrete induction schema is explicitly computed on which the subsequent reasoning is based. Starting with the pioneering works of [182], [81], [101,102], an alternative approach of *implicit* or *"inductionless"* induction was developed which is based on a *proof by consistency* principle. The basic idea of the method roughly works as follows (in the context of clauses with equality), cf. [60]: Given a specification (set of clauses) E and a set of inductive conjectures C, one adds C to E and tries to derive an inconsistency. If this turns out to be impossible, then the clauses in C are inductive consequences of E. Here, "inconsistency" is understood w.r.t. an appropriate axiomatization of the *standard* model of E. "Turning out to be impossible" means that one has to devise appropriate strategies of *inductive saturation* such that using these strategies

the inductive saturation process hopefully terminates (with or without an inconsistency) in many cases. In the 1980s and later on the approach of [182], [81], [101,102] was extended, refined and generalized in various ways, cf. e.g. [192], [122,123], [72,74], [9], [83], [76,77], [62], [60]. Also, a couple of other related approaches somewhere in between explicit an implicit induction were developed, e.g., [100], [204], [44], [127,128], [133], [235], [131,132], [160], [31], [29], [26], [27], [230,229], [169], [227]. There was and still is an on-going debate about explicit vs. implicit induction and combinations thereof, in particular concerning the strengths and weaknesses of and the relationships between these approaches. But it seems clear that, regarding the essential problems in the proof search process in ITP, both approaches face essentially the same problems. For that reason we will not go into details about the latter aspects and relationships here.

Next, for the sake of completeness let us mention a few (though not all) important aspects regarding ITP systems, especially in connection with the system architecture, its features, purpose, and its proof search control.

- Stand-alone tool vs. component in bigger system: Clearly, this property has consequences for the design and the architecture.
- Homogeneous vs. heterogeneous tool: A homogeneous tool may be much easier to implement, understand and maintain, but is likely to be much less powerful.
- Built-in theories vs. explicit handling: How to handle pre-defined basic data types and their properties, as well as certain problematic properties of functions (like associativity plus commutativity) is a crucial aspect of any implementation.
- Subtools for specialized proof tasks (decision procedures etc.): Such subtools should certainly be available, but their integration is typically non-trivial.
- Intended functionality (yes/no vs. justifications, proof objects, reuse, incrementality, modularity) : The intended functionality is of course extremely important in the design process, and often makes different systems very hard to compare.
- General purpose reasoning system vs. specialized tool for a particular problem domain: Obviously, such a decision also has far-reaching consequences.
- Experimental tool vs. high-fidelity system (certification): To develop a system of the latter type usually amounts to much more work and care with the relevant design decisions.
- Post- vs. pre- vs. integrated development: The type of the envisaged reasoning tasks (including inductive proofs) to be tackled with the system and the

overall development model (e.g., *first program, then verify*; or, *first specify and verify, then program*) obviously affect the system design and architecture.

- Desired degree of automation / interaction: Clearly, a high degree of (desired) automation implies a much more sophisticated proof search model than an interactive style of reasoning.
- Proof search control architecture / concept / language: For ITP systems with at least some degree of automation the architecture of the proof search control, the underlying concepts and languages for describing and guiding the proof search are crucial components to realize a system with the intended automatic capabilities.

For the rest of the paper, let us fix the theoretical (logical) setting for ITP in first-order logic with equality (although also for other reasonable settings the resulting analysis would be the same), partially following [60]: Let E be a set of first-order sentences. A first-order formula ϕ is an *inductive consequence* of E iff, for every *Herbrand interpretation* $\mathcal{H}$, $\mathcal{H} \models E$ implies $\mathcal{H} \models \phi$. If E is a set of Horn clauses with equality, then there exists a unique smallest Herbrand model of E (or *standard*) model $\mathcal{I}_E$. In this case, one does not need to consider all the Herbrand interpretations, but only the smallest ones: If E is a set of Horn clauses and c a positive clause, then c is an inductive consequence of E iff $\mathcal{I}_E \models c$. Note that inductive consequences of E should not be confused with elements in the theory of $\mathcal{I}_E$. The inductive theory is contained in $\mathcal{I}_E$, but not conversely (because negative statements expressing that certain elements in $\mathcal{I}_E$ are distinct, do not hold in all Herbrand interpretations of E). **In the sequel when proving or disproving inductive conjectures we always have in mind validity in $\mathcal{I}_E$, and inductive theorems are formulæ that are valid in the standard model $\mathcal{I}_E$.** In the ITP setting we assume moreover, that the database contains the basic specification E with all the axioms and definitions, a set of already proved (or otherwise established) inductive lemmas L, and the current set of inductive conjectures C (allowing C to contain more than one element is convenient in practice since often additional conjectures are generated during a proof attempt).

In other reasonable basic logical settings for ITP, the subsequent analysis of the basic problems in ITP will most probably be very similar. This also concerns the aspect, whether the underlying induction concept is explicit or implicit as discussed above. Although the precise analysis clearly depends on the framework used, the essential technical and proof search problems are the same. Yet, their syntactical and technical appearance may be different.

2 Strategies in ITP

Here we will discuss in some depth (though non-technically) the crucial strategic and proof control aspects in ITP, where (implicitly) the goal in mind is always to get a high degree of automation.

2.1 Why are Strategies so Important in ITP?

As we have argued previously (see Section 1), ITP is in some sense much more difficult that general TP, hence this should also be reflected in the proof search process. So, why are (good) strategies so important for ITP? Some of the main reasons are

- the incompleteness of ITP methods,
- the structure and the size of the search space (infinitely branching in several dimensions, see below),
- the recursive nature of inductive proof attempts,
- the difficulty of measuring progress within an ITP attempt
- the difficulty of controlling / guiding the proof search process in an adequate and intelligent way. This difficulty comprises a lot of more detailed steps and decisions:
  - 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 (in a goal-directed manner)?
  - When and how to perform a case analysis?

2.1.1 Essential Strategic Control Issues

Let us consider more precisely what are relevant strategic issues. For that we look at a typical structure of an inductive proof attempt:

- Try non-inductive methods (when and how?):
  - Testing for inconsistency, generation of counterexamples.
  - Start a TP attempt (without induction).
  - Simplify as much as possible w.r.t. 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.
  - · Generalize conjecture.
  - · Generate (auxiliary) lemma.

Now, what are the crucial and essential strategic control issues in the above list? First of all, the following questions are obvious, but not their answer:

- When should one test for inconsistency, and how? When should one search for a counterexample, and how?
- When should one decide to try general TP without induction, and when not?

The next list of actions is central and of fundamental importance for every reasonable ITP system, and all these actions are **critical**[6] and — considered as choice options in the search tree — **infinitely branching**:

(1) Simplification **(infinitely branching[7] and critical)**
  (a) When?
  (b) How? Using which definitions and lemmas? In which order?
  (c) Recursively use induction to verify the applicability of conditional lemmas?
  (d) Simplify to normal form?
  (e) Inverse simplification (expansion)? How far?

(2) Induction **(infinitely branching and critical)**
  (a) Compute and select appropriate induction schema.
  (b) Generate corresponding proof tasks.
  (c) Make induction hypothesis applicable, e.g. by *cross-fertilization*, *rippling*, and other *difference reduction* techniques.

[6] By *critical* we mean here, that without such an (appropriate) action (or a similar one) the whole proof attempt may be hopeless.

[7] Actually, w.r.t. a finite database of definitions, lemmas and current conjectures, the first step of such a simplification action is usually only finitely branching, because there are only finitely many possibilities. However, many-step simplification is infinitely branching when simplification can be non-terminating (in this case, "simplification" has only an intuitive meaning, not a formal one), or when during simplification other infinitely branching operations, like induction for the verification of the condition of some conditional lemma, are allowed.

(3) Case analysis **(infinitely branching and critical)**
   (a) When?
   (b) How? According to which criteria?
   (c) How to verify individual cases?

(4) Generalization **(infinitely branching and critical)**
   (a) When?
   (b) How? For what purpose?
   (c) How to avoid over-generalization?

(5) Lemma generation / speculation **(infinitely branching and critical)**
   (a) When?
   (b) How? For what purpose?
   (c) Organizational: Top-down (relative ITP) or bottom-up (proofs are absolute)?

Some remarks and comments about this list seem in order (cf. also [47] for a discussion of some aspects of the infinite branching behaviour in ITP).

First of all, it is clear that proof search with infinitely branching steps of different types can in principle be sequentialized (assuming only countably many choices), i.e., simulated by a finitely branching tree. However, in practice and w.r.t. the nature of the problem this would be completely unsatisfactory and infeasible.

Regarding simplification (1), there is typically a very high degree of non-determinism (b) of what could be simplified and how, w.r.t. the current database. Also, the notion of "simplification" is not always clear, since well-foundedness of such transformations need not be ensured. If it is guaranteed, e.g., by imposing some well-founded ordering on terms and formulae, then certain desirable transformation steps (expansion or inverse simplification) are not allowed anymore. In the presence of permutative properties like commutativity, guaranteeing termination of simplification processes is difficult and extra efforts have to be made to avoid circular (or more general forms of non-terminating) reasoning. On the other hand, it is also well-known that in many examples certain transformations have to be "non-simplifying" to lead to a final success. The power of simplification is considerably increased by recursively allowing induction (c), e.g., to verify the applicability of a conditional lemma of the form $l \to r \Leftarrow c$ on the actual formula $C = D[l\sigma]$, by inductively proving $c\sigma$. Since simplification is most often non-confluent, and also frequently non-terminating, it is highly non-trivial to determine how far one should simplify, cf. (e), (f). Experience shows, that simplifying too much can also prevent a proof to be found, as well as the other way round.

Induction (2) is perhaps the most intensively investigated operation. Initiated by [39], the analysis of recursive definitions and proofs of well-definedness (i.e., termination proofs) is a fairly well understood area. In [39], the computation and selection of appropriate induction schemas for a given inductive conjecture is a two-stage process. First, at the time of definition introduction for some function f, the definition, in particular the recursion structure, is analyzed and the resulting knowledge about the recursion structure of f and the reasons for termination of its recursion are stored (in the internal knowledge base). Then, when trying induction on some formula, all defined function symbols in the conjecture give rise to candidate induction schemas that are obtained from the generic information in the knowledge base together with actual information extracted from the conjecture. This way a lot of candidate induction schemas are computed, which are then *merged* as far as possible and ranked, according to some quality criteria, until one final candidate remains. The strategies and heuristics of [39,42] (and of its successor [151]) have been amazingly successful and impressive. Other works, variations and explicit versions of recursion analysis and induction schema generation include e.g. [210], [51,50], [222,222]. Given a concrete (sound) induction schema, the generation of corresponding proof tasks (b) is normally straightforward. However, what is challenging and at the heart of inductive reasoning, are goal-directed techniques to make the induction hypothesis applicable. [8] Many ideas, strategies and heuristics are known for this step, and more generally for reasoning based on *difference reduction* and *proof plans*, cf. e.g. *cross-fertilization* ([39]), and *rippling* (cf. e.g. [45], [46], [52], [50], [54], [53], [165,166], [164], [47], [103], [107], [112], [113], [109], [14,15,17,16], [65], [67]), [136].

Case analysis (3) is also a very challenging topic and subtask in ITP. Especially, when should one initiate a case analysis, and if so, how? And according to which criteria should the generation cases be done? Furthermore, if the case analysis is not obviously complete, in the sense that all cases are covered, [9] verifying completeness involves in general again inductive reasoning, hence the full power (and difficulty) of induction. The questions of when and how to apply case analysis (in ITP) are also very difficult questions where not so much literature exists (although any ITP system has heuristics for doing so), cf. e.g. [39,42], [151], [30]. A related question is how fine-grained the case distinction should be. The more special some case is, the more information is available

[8] In general, an induction schema for a conjecture consists of several base cases and several induction steps, all of which have to be successfully processed.

[9] Completeness of some case analysis with cases $c_1, \ldots, c_n$ can always be enforced by adding a case for the negation of the disjunction of the c_i. However, sometimes one rather wants to use a semantic distinction where completeness has to be explicitly (inductively) verified.

to prove the property in this case. On the other hand, it is a well-known phenomenon – like in induction in general – that more specialized statements may be more difficult to prove, as compared to more general versions! Some of the techniques for (2) above can also be useful to plan and attempt (hopefully) reasonable case analyses.

Generalization of inductive conjectures (4) and lemma generation (5), which are closely related, are often unavoidable, if a proof is to be found at all. Of course, the search space is infinitely branching w.r.t. such an operation. For lemma generation this is obvious. In the case of generalization of some conjecture C to be proved, this is also easy to see. If C is just an equation (or another universally quantified literal), then there are only finitely many possibilities to do a syntactic generalization (according to the well-founded instantiation ordering on formulae). However, as soon as one allows semantic generalizations,[10] there are infinitely many possibilities. Both generalization and lemma generation are highly error-prone in the sense that they may lead to false conjectures whereas the original conjecture was inductively true. Hence, it is extremely important to provide mechanisms in order to avoid over-generalization and to avoid the generation of false "lemmas". Typically, both operations are only applied with some more or less concrete goal in mind, namely, to solve a failure of a previous proof attempt. However, to what extent such a step might really help in the context of the actual conjecture processed, remains to be found out and verified, either a priori or a posteriori. To date, there is some but not really much literature about concrete heuristics and strategies for (4) and (5), and more generally on how to do this (automatically) in practice and to integrate these features into the overall ITP system, cf. e.g. [35,36,37,39,42], [146], [75], [118], [225], [138], [126], [141], [163], [219,220].

2.1.2 Organizational Strategic Issues

Next let us discuss (more modestly: ask questions and state desirable properties) some of the most important organizational (and strategic) issues in ITP. Again this list is necessarily incomplete, and we want to focus on some issues that we find important.

(i) Concerning the data- and knowledge base (definitions, conjectures, lemmas, ...):
   (a) In case of successful proof attempts:

[10] By this we mean any statement inductively implying C, for instance if C is conditional, i.e., of the form $D \Leftarrow c$ and we can prove $c \Rightarrow d$ as well as $D \Leftarrow d$, then the latter (inductively) generalizes $D \Leftarrow c$. More generally, in clauses generalization can be done by weakening the succedent part and/or strengthening the antecedent part.

Which (intermediate) lemmas should be kept (stored)? In which form? In which order? And as what kind of knowledge (rewrite / simplification lemma, type information, definition, ...)?
Should the current data and knowledge base be modified / simplified w.r.t. to the new knowledge, or not? And if the former, how?

(b) In case of (definitely) failed proof attempts:
What are the consequences for the next steps?
What can (internally) be learnt from this failure? How can it be exploited for a more promising attempt?
If the original conjecture has been falsified, how can it be corrected, e.g. via generalization.
If only the proof attempt has failed, how could this failed attempt trigger a more promising one?

(c) In case of neither successful nor failed proof attempts:
This situation occurs in the middle of a proof, and there is always the question what to do next.
Should the attempt be continued (as planned)?
Should the attempt be abandoned, because there seems to be not much hope of successfully completing it?
Should there be some kind of backtracking, in order to resume proving at an earlier branching point with another decision?

(ii) Concerning the control structure and the knowledge base:

(a) When introducing new definitions, the recursion analysis should yield appropriate control knowledge about the recursion schema of the newly defined functions.

(b) There should be internal memory and history mechanisms, e.g., for avoiding circular reasoning and loops.

(c) The proof search control model should somehow be layered and fairly flexible, in order to enable a realization of many different heuristics and strategies.

Most of the issues mentioned above are highly non-trivial, and in existing ITP systems their solutions often appear a bit ad hoc. For instance, in most of the major ITP systems, once an induction schema has been chosen, and once the subsequent proof attempt has failed, then the whole proof attempt is considered to have failed. An alternative point of view could be, at that point, to resume the process with another, possibly sightly lower ranked, induction schema. Similarly, backtracking rarely occurs in the existing ITP systems. Once a certain operation has failed, not many attempts are made to backtrack and proceed differently. Such a rigid proof search process may have dramatic consequences. For instance, a proof attempt of some inductive conjecture

may go through, provided we are able to apply some simplification lemma at a certain point. But, in order to verify an application condition of the lemma (which holds indeed), special emphasis has to be put on this effort, whereas with the standard way of simplifying we simply do not succeed in verifying this applicability condition and fail. Similar phenomena occur, when formulae are simplified too much, or when certain transformations (entailing a danger of non-termination) are forbidden. In a sense, it is not really surprising why current ITP systems (with a substantial degree of automation) have a rather rigid proof search model and not much flexibility concerning the overall control structure (w.r.t., for instance, the possibility of backtracking). We think that the main reason for this actually is the inherent complexity and difficulty of making the right decisions at the right points, of predicting the further outcome of a proof attempt, and of judging/ranking/estimating correctly the consequences of current decisions.

3 State of the Art

Let us give some comments on the current state of the art in ITP, based on some evidence from the literature and also from personal experience with ITP and different ITP systems. Of course, it is difficult to make general statements and give a comprehensive overview. We do not really claim any kind of completeness here nor that there exists a consensus about these issues, but rather aim at some conclusions which ideally should trigger further discussions.

3.1 Assessment, Comparison, Contests?

As to the state of the art, an assessment and comparison of different ITP systems is very difficult. The reasons for this situation are manifold, and include at least the following:

- There exists no (regular) competition of ITP systems.
- There are no (widely accepted and used) benchmarks.
- The underlying logics and the intended usage of the systems are rather diverse.
- The degree of automation / interaction and the underlying philosophy are often incomparable.
- The usage typically requires quite considerable expertise, or even high familiarity with the system.

Especially the first two aspects above indicate a major difference to the first-order theorem prover community. There, the yearly system competitions

(cf. e.g. [213], [214]) and the corresponding databases of benchmarks (cf. e.g. [215], [212], [211]) have been very inspiring and motivated a lot of fruitful developments. In ITP, nothing comparable exists (for reasons which may have become a bit more clear in the presentation above), though there have been discussions and attempts in this direction, cf. e.g. [111], [108].

Let us conclude this part with some general observations:

- Full automation in ITP is generally not (yet?) successful.
- The typical usage of such systems is *(specification and) proof engineering* with
  - human guidance for modeling, proof structure and proof ideas,
  - human guidance for lower-level control if necessary,
  - human failure analysis (with few automatic support), and with a
- tradeoff automation – interaction (w.r.t. efficiency, success rate, required expertise, flexibility of control, etc.).

3.2 Successes and Failures

With certain ITP systems, especially the Boyer-Moore theorem prover NQTHM ([39,42], [145]) and its successor ACL2 ([152,153,151,150]), remarkable results have been achieved. The successes (with a relatively high degree of automation, where, however, a substantial amount of human specification and proof engineering is necessary) include in particular the following aspects:

- The computer supported specification and verification of certain complex systems and relationships is indeed possible, e.g., of
  - the prime factorization theorem,
  - the undecidability of the halting problem,
  - the specification and verification of microprocessors and hardware components,
  - the specification and verification of compilers and of model checking algorithms.
- Often errors in initial specifications and conjectures could be revealed (with human help, from failed proof attempts).

On the other hand, as "failures" one may still consider e.g. the following aspects:

- The degree of automation is in general still rather low.
- Inductive specification and proof engineering continues to be rather tedious and difficult, and requires a substantial amount of expertise and training (by the human user), as well as a relatively high familiarity and most often

precise knowledge about the used ITP system.

- The automatic support for failure analysis is in general unsatisfactory.
- Building-in intelligence has turned out to be much more difficult than expected (by optimists, at least).

3.3 Systems for Induction

For the sake of illustration let us mention and briefly discuss some of the existing and available ITP systems.

The most successful, widely used, and maintained systems include:

- **ACL2** ([152,153,151,150]), the successor of NQTHM ([39,42], [145]): ACL2 is both a powerful and efficient functional programming language and an ITP system. Its inductive theorem prover deals with the universal fragment of first-order logic. There exists an impressive collection of non-trivial examples (cf. e.g. [150]). The system has a relatively high degree of automation and sophisticated strategies, heuristics and mechanisms for induction.
- **PVS** ([190,191], [189], [70], [180] [179]): PVS provides mechanized support for formal specification and verification. It is based on a powerful framework, namely classical, typed higher-order logic. Reasoning in PVS, especially ITP, is partially automated, and the system includes a modern framework for decision procedures.
- **VSE/INKA** ([115], [183], [7], [114]): VSE/INKA is a comprehensive tool for supporting the formal software development process. Inductive proofs are one focus area of the system, with sophisticated search control strategies for certain subtasks (during induction). But the system has also substantial support for the whole development process.
- **Isabelle/HOL** ([194,195], [187], [188], [94], [173,67]): Isabelle/HOL is a (logical framework and a) generic theorem proving environment as well as a proof assistant. Its main application is the formalization of mathematical proofs and of logical systems, as well as formal verification. The system is very powerful and flexible, however, usually a lot of user interaction with an expert user is required.

Further ITP systems which are also available, but more experimental and not that sophisticated, include e.g. (again, this list is by no means complete)

- **RRL** ([142,143,144], [235], [131], [234], [133], [2], [129,134]): RRL is rewrite-based, first-order, and incorporates different inductive proof techniques. It is fairly easy to use and to experiment with. Its working mode is highly automatic.

- **Spike** ([28], [32], [31], [29], [26], [208]): Spike is also rewrite- and saturation-based, and incorporates different inductive proof proof techniques including test set induction. It is based on conditional equations, and works mainly automatically.
- **Oyster/CLAM** ([55], [50], [205], [46,47], [51,50]): Oyster/CLAM is a tactic-based proof editor based on Martin-Löf constructive type theory, together with a proof planner. It incorporates a meta language for constructing customized tactics for individual conjectures, based especially on rippling techniques.
- **QuodLibet** ([229,230], [169], [168], [8], [227]): QuodLibet is the successor system of **UNICOM** ([82,84,83]. It is both a specification language and an ITP system for data types with partial operations. It has a flexible control, is user-oriented, but with some automation facilities. It allows *lazy* induction proofs and has a sophisticated concept for managing (possibly simultaneous) proof attempts.

4 Problems and Challenges

In this section we try to summarize in a concise way what we think are the main problem and challenges in ITP. First we consider **technical, conceptual and logical problems and challenges**:

- Building-in knowledge: When and how to do it?
- Structuring / modularizing specifications and proof tasks: How to do this as automatically as possible?
- Extending decidable cases (classes), cf. also [78,79], [126], [6]: How, and in which direction?
- How to combine ITP system with tools for special purposes, like
  · systems for particular data types and theories,
  · decision procedures for restricted theories,
  · combination mechanisms?
- How to get better methods for goal-directed reasoning?
- How to achieve better methods for look-ahead based reasoning?
- Generalization: When, why, and how to do it?
- Generation of (auxiliary) lemmas: When, why and how?
- How to recognize, analyze and deal appropriately with failure?
- How to make ITP more robust (monotonic, semantic)? [11]

[11] By *monotonic* we mean here that, when extending or enriching (inductive) specifications

Next we look at **problems and challenges regarding control issues** (of course, there is a big overlap with the above mentioned aspects, too):

- Good strategies and heuristics are vital in ITP to generate and deal with reasonable proof attempts. The question is how to improve currently employed strategies and heuristics substantially.
- How to achieve progress w.r.t. the basic (but ubiquitous) question in ITP of what to do next, in view of the history, the current data and the knowledge base?
- The overall proof search model of an ITP system
  - is necessarily complex, and should be very flexible;
  - needs both automation and interaction (for proof engineering);
  - should allow for / support interrupts, inspection, failure analysis and relative [12] proving;
  - should guarantee correctness requirements, i.e., be sound together with the inference machine;
  - should be compatible with user interaction, navigation (in the search tree), information extraction, and generation of proof objects (for possible later off-line verification in the sense of *proof checking*);
  - should also have different layers (for different types of reasoning);
  - should allow the integration of other tools for sub-tasks and the integration as a subsystem in other systems;
  - needs to be able to understand and integrate strategic and heuristic user input into the proof process.

Finally, from a *software engineering point of view*, problematic and challenging issues are the following:

- How to design / implement / apply a structured control concept for proof search that
  - is user-friendly, fully transparent, intelligible, flexible, extensible and modifiable;

---

in a consistent way, previously (automatically or partially automatically) generated proofs of inductive properties are still obtained after the extension/enrichment. In existing systems, this is often not the case, since the new data and knowledge items may disturb previous proof processes. By *more semantic* we refer to the fact that the success of proof attempts very often heavily depends on the exact syntactic form of a conjecture, and fails if a slightly different, but semantically equivalent, version is considered. Yet, for obvious reasons it would be nice, if this strong dependence on syntax could be reduced or ideally eliminated, at least for certain cases.

[12] By *relative proving* we mean reasoning relative to some yet unproved intermediate assumptions, that eventually also have to be verified, to obtain an overall proof. This way, temporarily invented auxiliary lemmas can be tested on whether they are indeed sufficient for making some main proof go through.

  - · generates complete proof objects;
  - · has a programmable strategy/heuristics language with clearly defined semantics, in which both high- low-level proof ideas can be easily expressed;
  - · 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 easily and quickly integrate new tools / subtools (e.g., decision procedures);
  - · can easily be specialized to specific domains;
  - · has an appropriate system for maintaining / adapting / its (large!) knowledge base?
- HCI: How to do all this in a smart way as to human computer interaction?

5 Some Theses

Here are a few theses about (automated) ITP for which there is no formal proof, but only some evidence: Theses:

- **In the near future, ITP will only be successful for**
  - · **very specialized domains (e.g., with fixed axiomatizations)**,
  - · **for very restricted classes of conjectures**.
- **ITP will continue to be a very challenging engineering process**.
- Regarding the research question "What could be considered to be substantial progress in ITP?", we think that
  - · **increased robustness (more monotonic, semantics based)**,
  - · **an improved modularization and improved structuring of theories, proofs, and proof search**, as well as
  - · **appropriate framework(s) to model (and implement) strategic proof search control**, that are
    expressive,
    flexible (extensible, adaptable, programmable),
    with well-defined semantics, and
    layered, and also
  - · **benchmarks for ITP problems, together with well-designed proposals for ITP contests**

  would be elements of such substantial progress.

In fact, we are convinced that substantial progress in ITP will take time, and that spectacular breakthroughs are unrealistic, in view of the enormous problems and the inherent difficulty of inductive theorem proving.

6 Conclusion

ITP problems are at the heart of many verification and reasoning tasks in computer science. We have tried to give a thorough, though only high-level, account of *induction* (in the sense of ITP), both regarding the possible approaches, methods, proof techniques and corresponding systems, as well as concerning the main problems in actual proofs, having in mind the goal of a high automation degree. In particular, we have worked out, isolated and discussed the crucial differences to general TP. This comparison also reveals or, better, explains, why automating inductive proofs is much harder than automating general theorem proving. We hope that this analysis and discussion contributes to a better understanding of the essence of ITP, initiates further debates in the ITP community and thus contributes a bit to the further development of the field.

Acknowledgements: This paper is an extended version of the abstract of an invited talk given at the STRATEGIES 2004 workshop. I'm very grateful to Maria Paola Bonacina and Thierry Boy de la Tour, the Program Co-Chairs of STRATEGIES 2004, for this kind invitation and for their helpful comments on a draft version of this paper. [13] Thanks also to the audience at STRATEGIES 2004 for a lively and inspiring discussion.

References

[1] J. Adàmek. Recursive data types in algebraically omega-complete categories. *Information and Computation*, 118(2):181–190, May 1995.

[2] R. Agarwal, D. Musser, D. Kapur, and X. Nie. The Tecton proof system. In R. Book, ed., *Proc. 4th Int. Conf. on Rewriting Techniques and Applications (RTA'91)*, LNCS 488, pp. 442–444. Springer, Apr. 1991.

[3] W. Ahrendt, P. Baumgartner, and H. de Nivelle, eds. *Proc. Workshop on Disproving: Non-Theorems, Non-Validity, Non-Provability (DISPROVING'04), in conjunction with IJCAR'04, Cork, Ireland, July 5, 2004*, 2004.

[4] A. Armando and S. Ranise. Termination of constraint contextual rewriting. In H. Kirchner and C. Ringeissen, eds., *Proc. 3rd Int. Workshop on Frontiers of Combining Systems (FROCOS 2000)*, LNCS 1794, pp. 47–61. Springer, Mar. 2000.

[5] A. Armando and S. Ranise. Constraint contextual rewriting. *Journal of Symbolic Computation*, 36(1–2):193–216, 2003. July-August 2003.

[6] A. Armando, M. Rusinowitch, and S. Stratulat. Incorporating decision procedures in implicit induction. *Journal of Symbolic Computation*, 34(4):241–258, 2002.

[7] S. Autexier, D. Hutter, H. Mantel, and A. Schairer. System description: inka 5.0 - a logic voyager. In H. Ganzinger, ed., *Proc. 16th Int. Conf. on Automated Deduction, Trento, Italy, July 7-10, 1999*, LNCS 1632, pp. 207–211. Springer, July 1999.

[13] and also for their patience regarding the final version

[8] J. Avenhaus, U. Kühler, T. Schmidt-Samoa, and C.-P. Wirth. How to prove inductive theorems? QUODLIBET! In F. Baader, ed., *Proc. 19th Int. Conf. on Automated Deduction (CADE'03), Miami Beach, FL, USA, July 28 – August 2, 2003*, LNCS 2741, pp. 328–333. Springer, 2003.

[9] L. Bachmair. Proof by consistency in equational theories. In *Proc. 3rd IEEE Symposium on Logic in Computer Science*, pp. 228–233, 1988.

[10] L. Bachmair and N. Dershowitz. Equational inference, canonical proofs, and proof orderings. *Journal of the ACM*, 41(2):236–276, 1994.

[11] L. Bachmair and H. Ganzinger. Resolution theorem proving. In J. Robinson and A. Voronkov, eds., *Handbook of Automated Reasoning*, volume 1, chapter 2, pp. 19–99. Elsevier and MIT Press, 2001.

[12] S. Baker, A. Ireland, and A. Smaill. On the use of the constructive omega-rule within automated deduction. In A. Voronkov, ed., *Proc. 3rd Int. Conf. on Logic Programming and Automated Reasoning (LPAR 1992), St. Petersburg, Russia, July 15-20, 1992, 1993*, LNCS 624, pp. 214–225. Springer, July 1992.

[13] S. Baker and A. Smaill. A proof environment for arithmetic with the omega rule. In J. Calmet and J. A. Campbell, eds., *Proc. 2nd Int. Conf. on Integrating Symbolic Mathematical Computation and Artificial Intelligence (AISMC'94), Cambridge, UK, August 3-5, 1994*, LNCS 958, pp. 115–130. Springer, Aug. 1994.

[14] D. Basin and T. Walsh. Difference matching. In D. Kapur, ed., *Proc. 11th Int. Conf. on Automated Deduction (CADE'92), Saratoga Springs, NY, USA, June 15-18, 1992*, LNCS 607, pp. 295–309. Springer, June 1992.

[15] D. Basin and T. Walsh. Difference unification. In R. Bajcsy, ed., *Proc. 13th Int. Conf. on Artificial Intelligence (IJCAI'93), Chambéry, France, August 28 -September 3, 1993*, pp. 116–122. Morgan Kaufmann, 1993.

[16] D. Basin and T. Walsh. A calculus for and termination of rippling. *Journal of Automated Reasoning*, 16(1/2):147–180, Mar. 1996.

[17] D. A. Basin and T. Walsh. A calculus for rippling. In N. Dershowitz and N. Lindenstrauss, eds., *Proc. 4th Int. Workshop on Conditional and Typed Rewriting Systems (CTRS 1994), Jerusalem, Israel, July 13-15, 1994*, LNCS 968, pp. 15–30. Springer, July 1995.

[18] J. A. Bergstra and J. Heering Which data types have omega-complete initial algebra specifications. *Theoretical Computer Science*, 124(1), Feb. 1994.

[19] N. Berregeb, A. Bouhoula, and M. Rusinowitch. Automated verification by induction with associative-commutative operators. In R. Alur and T. Henzinger, eds., *Proc. 8th International Conference on Computer Aided Verification (CAV'96), New Brunswick, NJ, USA, July 31 – August 3, 1996*, LNCS 1102, pp. 220–231. Springer, 1996.

[20] S. Biundo. A synthesis system mechanizing proofs by induction. In J. Boulay, D. Hogg, and L. Steels, eds., *Proc. 7th European Conf. on Artificial Intelligence (ECAI'86), Brighton, United Kingdom, July 20-25, 1986*, pp. 287–296. North-Holland, July 1987.

[21] S. Biundo, B. Hummel, D. Hutter, and C. Walther. The Karlsruhe induction theorem proving system. In J. Siekmann, ed., *Proc. 8th Int. Conf. on Automated Deduction (CADE'86), Oxford, England, July 27 – August 1, 1986*, LNCS 230, pp. 672–674. Springer, 1986.

[22] M. Bonacina and B. Gramlich, eds. *4th International Workshop on Strategies in Automated Deduction (STRATEGIES 2001) – Selected Papers*, volume 58 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2001.

[23] A. Bouhoula. Sufficient completeness and parameterized proofs by induction. In G. Levi and M. Rodríguez-Artalejo, eds., *Proc. 4th Int. Conf. on Algebraic and Logic Programming, Madrid (ALP'94)*, LNCS 850, pp. 23–40. Sept., Sept. 1994.

[24] A. Bouhoula. General framework for mechanizing induction using test set. In N. Foo and R. Goebel, eds., *Proc. 4th Pacific Rim International Conference on Artificial Intelligence (PRICAI'96), Cairns, Australia, August 26-30, 1996*, LNCS 1114, pp. 1–12. Springer, Aug. 1996.

[25] A. Bouhoula. Using induction and rewriting to verify and complete parameterized specifications. *Theoretical Computer Science*, 170(1–2):245–276, Dec. 1996.

[26] A. Bouhoula. Automated theorem proving by test set induction. *Journal of Symbolic Computation*, 23(1):47–77, Jan. 1997.

[27] A. Bouhoula and J.-P. Jouannaud. Automata-driven automated induction. *Information and Computation*, 169(1):1–22, 2001.

[28] A. Bouhoula, E. Kounalis, and M. Rusinowitch. SPIKE, an automatic theorem prover. In A. Voronkov, ed., *Proc. 3rd Int. Conf. on Logic Programming and Automated Reasoning (LPAR 1992), St. Petersburg, Russia, July 15-20, 1992, 1993*, LNCS 624, pp. 460–462. Springer, July 1992.

[29] A. Bouhoula, E. Kounalis, and M. Rusinowitch. Automated mathematical induction. *Journal of Logic and Computation*, 5(5):631–668, 1995.

[30] A. Bouhoula and M. Rusinowitch. Automatic case analysis in proof by induction. In *Proc. 13th Int. Conf. on Artificial Intelligence*, pp. 88–94, Chambéry Savoie, France, Apr. 1993.

[31] A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14(2):189–235, 1995.

[32] A. Bouhoula and M. Rusinowitch. SPIKE: A system for automatic inductive proofs. In V. Alagar and Maurice Nivat, eds., *Proc. 4th Int. Conf. on Algebraic Methodology and Software Technology (AMAST'95), Montreal, Canada, July 3-7, 1995*, LNCS 936, pp. 576–577. Springer, July 1995.

[33] R. Boulton and K. Slind. Automatic derivation and application of induction schemes for mutually recursive functions. In J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.-K. Lau, C. Palamidessi, L. Pereira, Y. Sagiv, and P. Stuckey, eds., *Proc. 1st Int. Conf. on Computational Logic (CL'00), London, UK, 24-28 July, 2000*, LNCS 1861, pp. 629–643. Springer, 2000.

[34] R. Boyer and J Moore. Program verification. *Journal of Automated Reasoning*, 1(1):17–23, 1985.

[35] R. Boyer and J S. Moore. Proving theorems about LISP functions. In N. Nilsson, ed., *Proc. 3rd Int. Conf. on Artificial Intelligence (IJCAI'73), Standford, CA, August 1973*, pp. 486–493. William Kaufmann, Aug. 1973.

[36] R. Boyer and J S. Moore. Proving theorems about lisp functions. *Journal of the ACM*, 22(1):129–144, Jan. 1975.

[37] R. Boyer and J S. Moore. A lemma driven automatic theorem prover for recursive function theory. In R. Reddy, ed., *Proc. 5th Int. Conf. on Artificial Intelligence (IJCAI'77), Cambridge, MA, August 1977*, pp. 511–519. William Kaufmann, Aug. 1977.

[38] R. Boyer and J S. Moore. Overview of a theorem-prover for a computational logic. In J. Siekmann, ed., *Proc. 8th Int. Conf. on Automated Deduction (CADE'86), Oxford, England, July 27 – August 1, 1986*, LNCS 230, pp. 675–678. Springer, 1986.

[39] R. S. Boyer and J S. Moore. *A Computational Logic*. Academic Press, 1979.

[40] R. S. Boyer and J S. Moore. Metafunctions: Proving them correct and using them efficiently as new proof procedures. In R. S. Boyer and J S. Moore, eds., *The Correctness Problem In Computer Science*. Academic Press, London, 1981.

[41] R. S. Boyer and J S. Moore. The addition of bounded quantification and partial functions to a computational logic and its theorem prover. *Journal of Automated Reasoning*, 4(2):117–172, June 1988.

[42] R. S. Boyer and J S. Moore. *A Computational Logic Handbook*, volume 23 of *Perspectives in Computing*. Academic Press, 1988. Formerly: Notes and Reports in Computer Science and Applied Mathematics.

[43] R. S. Boyer and J S. Moore. A theorem prover for a computational logic (keynote address). In M. Stickel, ed., *Proc. 10th Int. Conf. on Automated Deduction (CADE'90), Kaiserslautern, FRG, July 24-27, 1990*, LNCS 449, pp. 1–15. Springer, July 1990.

[44] F. Bronsard, U. Reddy, and R. Hasker. Induction using term orderings. In A. Bundy, ed., *Proc. 12th Int. Conf. on Automated Deduction (CADE'94), Nancy, France, June 26 – July 1, 1994*, LNCS 814, pp. 102–117. Springer, 1994.

[45] A. Bundy. Analysing mathematical proofs (or reading between the lines). In *Proc. 4th Int. Conf. on Artificial Intelligence (IJCAI'75) - Advance Papers, Tbilisi, Georgia, USSR, 3-8 September 1975*, pp. 22–28, 1975.

[46] A. Bundy. The use of explicit proof plans to guide inductive proofs. In E. Lusk and R. Overbeek, eds., *Proc. 9th Int. Conf. on Automated Deduction*, LNCS 310, pp. 111–120. Springer, 1988.

[47] A. Bundy. The automation of proof by mathematical induction. In J. Robinson and A. Voronkov, eds., *Handbook of Automated Reasoning*, volume 1, Part IV, chapter 13, pp. 845–911. Elsevier and MIT Press, 2001.

[48] A. Bundy. A critique of proof planning. In A. Kakas and F. Sadri, eds., *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II*, LNCS 2408, pp. 160–177. Springer, 2002.

[49] A. Bundy, F. Giunchiglia, A. Villafiorita, and T. Walsh. Abstract proof checking: An example motivated by an incompleteness theorem. *Journal of Automated Reasoning*, 19(3):319–346, Dec. 1997.

[50] A. Bundy, F. Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7(3):303–324, 1991.

[51] A. Bundy, F. Harmelen, J. Hesketh, A. Smaill, and A. Stevens. A rational reconstruction and extension of recursion analysis. In N. Sridharan, ed., *Proc. 11th Int. Conf. on Artificial Intelligence, Detroit, MI, USA, August 1989*, pp. 359–365. Morgan Kaufmann, 1989.

[52] A. Bundy, Harmelen, Frank van, A. Smaill, and A. Ireland. Extensions to the rippling-out tactic for guiding inductive proofs. In M. Stickel, ed., *Proc. 10th Int. Conf. on Automated Deduction*, LNCS 449, pp. 132–146. Springer, 1990.

[53] A. Bundy and V. Lombart. Relational rippling: A general approach. In *Proc. 14th Int. Joint Conf. on Artificial Intelligence, IJCAI 95, Montréal, Québec, Canada, August 20-25, 1995*, pp. 175–181. Morgan Kaufmann, 1995.

[54] A. Bundy, A. Stevens, F. Harmelen, A. Ireland, and A. Smaill:. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, 62(2):185–252, 1993.

[55] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam system. In M. E. Stickel, ed., *Proc. 10th Int. Conf. on Automated Deduction (CADE'90), Kaiserslautern, FRG, July 24-27, 1990*, LNCS 449, pp. 647–648. Springer, 1990.

[56] F. Cantu, A. Bundy, A. Smaill, and D. Basin. Experiments in automating hardware verification using inductive proof planning. In M. Srivas and A. Camilleri, eds., *Formal Methods in Computer-Aided Design, First International Conference, FMCAD '96, Palo Alto, California, USA, November 6-8, 1996, Proceedings*, LNCS 1166, pp. 94–108, 1996.

[57] J. Chazarain and E. Kounalis. Mechanizable inductive proofs for a class of forall exists formulas. In A. Bundy, ed., *Proc. 12th Int. Conf. on Automated Deduction (CADE'94), Nancy, France, June 26 – July 1, 1994*, LNCS 814, pp. 118–132. Springer, 1994.

[58] H. Comon. Inductive proofs by specification transformation. In N. Dershowitz, ed., *Proc. 3rd Int. Conf. on Rewriting Techniques and Applications (RTA'89)*, LNCS 355, pp. 76–91. Springer, Apr. 1989.

[59] H. Comon. Complete axiomatizations of some quotient term algebras. *Theoretical Computer Science*, 118(2):167–191, 1993.

[60] H. Comon. Inductionless induction. In J. Robinson and A. Voronkov, eds., *Handbook of Automated Reasoning*, volume 1, chapter 14, pp. 913–962. Elsevier and MIT Press, 2001.

[61] H. Comon and F. Jacquemard. Ground reducibility is exptime-complete. In *Proc. 12th Annual IEEE Symposium on Logic in Computer Science*, pp. 26–34, Warsaw, Poland, June 29 – July 2 1997. IEEE Computer Society Press.

[62] H. Comon and R. Nieuwenhuis. Induction=i-axiomatization+first-order consistency. *Information and Computation*, 159(1–2):151–186, May/June 2000.

[63] D. C. Cooper. Theorem proving in arithmetic without multiplication. *Machine Intelligence*, 7:91–99, 1972.

[64] L. Dennis, A. Bundy, and I. Green. Making a productive use of failure to generate witnesses for coinduction from divergent proof attempts. *Annals of Mathematics and Artificial Intelligence*, 29(1–4):99–138, 2000. Special Issue on Strategies in Automated Deduction, ed. by Bernhard Gramlich, Hélène Kirchner, and Frank Pfenning.

[65] L. A. Dennis and A. Smaill. Ordinal arithmetic: A case study for rippling in a higher order domain. In R. J. Boulton and P. B. Jackson, eds., *Proc. 14th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'01), Edinburgh, Scotland, UK, September 3-6, 2001*, LNCS 2152, pp. 185–200. Springer, 2001.

[66] E. Deplagne, C. Kirchner, H. Kirchner, and Q. Nguyen. Proof search and proof check for equational and inductive theorems. In F. Baader, ed., *Proc. 19th Int. Conf. on Automated Deduction (CADE'03), Miami Beach, FL, USA, July 28 – August 2, 2003*, LNCS 2741, pp. 297–316. Springer, 2003.

[67] L. Dixon and J. D. Fleuriot. Higher order rippling in IsaPlanner. In K. Slind, A. Bunker, and G. Gopalakrishnan, eds., *Proc. 17th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'04), Park City, Utah, USA, September 14-17, 2004*, LNCS 3223, pp. 83–98. Springer, Sept. 2004.

[68] R. Erickson and D. Musser. The AFFIRM theorem prover: Proof forests and management of large proofs. In W. Bibel and R. Kowalski, eds., *Proc. 5th Int. Conf. on Automated Deduction (CADE'80), Les Arcs, France, July 8-11, 1980*, LNCS 87, pp. 220–231. Springer, 1980.

[69] C. G. Fermüller, A. Leitsch, U. Hustadt, and T. Tammet. Resolution decision procedures. In J. Robinson and A. Voronkov, eds., *Handbook of Automated Reasoning*, volume 2, chapter 25, pp. 1791–1849. Elsevier and MIT Press, 2001.

[70] J.-C. Filliâtre, S. Owre, H. Rueß, and N. Shankar. ICS: Integrated canonizer and solver. In G. Berry, H. Comon, and A. Finkel, eds., *Proc. 13th International Conference on Computer Aided Verification (CAV 2001)*, LNCS 2101, pp. 246–249, Paris, France, July 2001. Springer.

[71] W. Fokkink and B. Luttik. An omega-complete equational specification of interleaving. In U. Montanari, J. Rolim, and E. Welzl, eds., *Proc. 27th International Colloquium on Automata, Languages and Programming (ICALP'00), Geneva, Switzerland, July 9-15, 2000*, LNCS 1853, pp. 729–743. Springer, July 2000.

[72] L. Fribourg. A strong restriction of the inductive completion procedure. In E. Kott, ed., *Proc. 13th Int. Conf. on Automata, Languages and Programming*, LNCS 226, pp. 105–116. Springer, 1986.

[73] L. Fribourg. On the use of conditional rewrite rules in inductive theorem proving. In S. Kaplan and J.-P. Jouannaud, eds., *Proc. 1st Int. Workshop on Conditional Rewriting Systems (CTRS'87)*, LNCS 308, pp. 56–61. Springer, 1988.

[74] L. Fribourg. A strong restriction of the inductive completion procedure. *Journal of Symbolic Computation*, 8:253–276, 1989.

[75] L. Fribourg. Automatic generation of simplification lemmas for inductive proofs. In V. Saraswat and K. Ueda, eds., *Proc. International Symposium on Logic Programming (ISLP'91), San Diego, California, USA, Oct. 28 - Nov 1, 1991*, pp. 103–116. MIT Press, 1991.

[76] H. Ganzinger and J. Stuber. Inductive theorem proving by consistency for first-order clauses. In *Informatik – Festschrift zum 60. Geburtstag von Günter Hotz*. Teubner Verlag, 1992.

[77] H. Ganzinger and J. Stuber. Inductive theorem proving by consistency for first-order clauses. In M. Rusinowitch and J. Remy, eds., *Proc. 3rd Int. Workshop on Conditional Term Rewriting Systems (CTRS'92), Pont-á-Mousson, France, July 8-10, 1992*, LNCS 656, pp. 226–241. Springer, 1993.

[78] J. Giesl and D. Kapur. Deciding inductive validity of equations. In R. Goré, A. Leitsch, and T. Nipkow, eds., *Proc. 1st Int. Joint Conf. on Automated Reasoning (IJCAR'01), Siena, Italy, June 18-23, 2001*, LNCS 2083, pp. 17–31. Springer, June 2001.

[79] J. Giesl and D. Kapur. Deciding inductive validity of equations. In F. Baader, ed., *Proc. 19th Int. Conf. on Automated Deduction (CADE'03), Miami Beach, FL, USA, July 28 – August 2, 2003*, LNCS 2741, pp. 17–31. Springer, 2003.

[80] K. Gödel. Über formal unentscheidbare Sätze der principia mathematica und verwandter Systeme. i. (German). *Monatsh. Math. Phys.*, 38:173–198, 1931.

[81] J. A. Goguen. How to prove algebraic inductive hypotheses without induction. In W. Bibel and R. Kowalski, eds., *Proc. 5th Int. Conf. on Automated Deduction*, LNCS 87, pp. 356–373, 1980.

[82] B. Gramlich. Completion based inductive theorem proving: A case study in verifying sorting algorithms. SEKI Report SR-90-04, Fachbereich Informatik, Universität Kaiserslautern, 1990.

[83] B. Gramlich. Completion based inductive theorem proving: An abstract framework and its applications. In L. Aiello, ed., *Proc. 9th European Conf. on Artificial Intelligence*, pp. 314–319. Pitman Publishing, London, 1990.

[84] B. Gramlich. Unicom: A refined completion based inductive theorem prover. In M. Stickel, ed., *Proc. 10th Int. Conf. on Automated Deduction*, LNCS 449, pp. 655–656. Springer, 1990.

[85] B. Gramlich. Towards intelligent inductive proof engineering. SEKI Report SR-92-01, Fachbereich Informatik, Universität Kaiserslautern, 1992.

[86] B. Gramlich. Experiences with the development, maintenance and enhancement of unicom. In D. Basin, F. Giunchiglia, and M. Kaufmann, eds., *Proc. CADE-12 Workshop "Correctness and Metatheoretic Extensibility of Automated Reasoning Systems"*, pp. 42–43, Nancy, France, June 1994.

[87] B. Gramlich. On evaluation criteria for inductive theorem proving systems. In R. Boyer, A. Bundy, D. Kapur, and C. Walther, eds., *Proc. 4th Int. Workshop on the Automation of Proof by Mathematical Induction, Dagstuhl-Seminar-Report*, July 1995.

[88] B. Gramlich. Design issues for inductive theorem proving systems. In *CADE-14 Workshop on Automated Induction Theorem Proving*, Townsville, North Queensland, Australia, July 1997.

[89] B. Gramlich. Strategic aspects in inductive theorem proving and proof engineering (extended abstract). In D. H. et al., ed., *Proc. FLoC'99 Workshop on the Automation of Proof by Mathematical Induction*, pp. 25–28, Trento, Italy, July 1999.

[90] B. Gramlich and H. Kirchner, eds. *Proceedings of the CADE-14 Workshop on Strategies in Automated Deduction* (70 pp., Townsville, North Queensland, Australia, July 1997. 70 pp.

[91] H. Kirchner, B. Gramlich and F. Pfenning, eds. *Proc. FLoC'99 Workshop on Strategies in Automated Deduction (STRATEGIES'99)*, 92 pp., Trento, Italy, July 1999.

[92] B. Gramlich, H. Kirchner, and F. Pfenning, eds. *Strategies in Automated Deduction (special issue)*. Annals of Mathematics and Artificial Intelligence, Volume 29, Number 1/4, 2000. Kluwer Academic Publishers, Feb. 2001.

[93] B. Gramlich and F. Pfenning, eds. *Proceedings CADE-15 Workshop on Strategies in Automated Deduction*, Lindau, Germany, July 1998. 74 pp.

[94] D. Griffioen and M. Huisman. A comparison of PVS and Isabelle/HOL. In *Proc. 11th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'98), Canberra, Australia, September 27 - October 1, 1998*, LNCS 1479, pp. 123–142. Springer, 1998.

[95] J. Groote. A new strategy for proving omega-completeness applied to process algebra. In J. Baeten and J. Klop, eds., *Proc. CONCUR'90, Theories of Concurrency: Unification and Extension, Amsterdam, The Netherlands, August 27-30, 1990*, LNCS 458, pp. 314–331. Springer, Aug. 1990.

[96] J. Heering. Partial evaluation and omega-completeness of algebraic specifications. *Theoretical Computer Science*, 43:149–167, 1986.

[97] J. v. Heijenoort. *From Frege to Gödel – A Sourcebook in Mathematical Logic, 1879–1931.* Source Books in the History of the Sciences. Harvard University Press, Cambridge, Massachusetts, 1967.

[98] L. Henkin. A generalization of the concept of omega-completeness. *Journal of Symbolic Logic*, 22(1):1–14, 1957.

[99] R. Hennicker. Context induction: a proof principle for behavioural abstractions. In A. Miola, ed., *Proc. International Symposium on Design and Implementation of Symbolic Computation Systems, DISCO '90, Capri, Italy, April 10-12, 1990*, LNCS 429, pp. 101–110. Springer, Apr. 1990.

[100] D. Hofbauer and R.-D. Kutsche. Proving inductive theorems based on term rewriting systems. In J. Grabowski, P. Lescanne, and W. Wechler, eds., *Proc. 1st Int. Workshop on Algebraic and Logic Programming (ALP'88), Gaussig, GDR, November 14-18, 1988*, LNCS 343, pp. 180–190. Springer, Nov. 1988.

[101] G. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. In *Proc. 21st Conf. on Foundations of Computer Science*, pp. 96–107, 1980. also in JCSS 25(2), pp. 239–266, 1982.

[102] G. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25(2):239–266, Oct. 1982.

[103] D. Hutter. Guiding induction proofs. In M. Stickel, ed., *Proc. 10th Int. Conf. on Automated Deduction*, LNCS 449, pp. 147–161. Springer, 1990.

[104] D. Hutter. Adapting a resolution calculus for inductive proofs. In B. Neumann, ed., *Proc. 10th European Conf. on Artificial Intelligence, ECAI 92, Vienna, Austria, August 3-7, 1992*, pp. 65–69. John Wiley and Sons, Chichester, Aug. 1992.

[105] D. Hutter. Synthesis of induction orderings for existence proofs. In A. Bundy, ed., *Proc. 12th Int. Conf. on Automated Deduction (CADE'94), Nancy, France, June 26 – July 1, 1994*, LNCS 814, pp. 29–41. Springer, 1994.

[106] D. Hutter. Using rippling for equational reasoning. In G. Görz and S. Hölldobler, eds., *Proc. 20th Annual German Conference on Artificial Intelligence (KI'96), Dresden, Germany, September 17-19, 1996*, LNCS 1137, pp. 121–133. Springer, Sept. 1996.

[107] D. Hutter. Coloring terms to control equational reasoning. *Journal of Automated Reasoning*, 18(3):399–442, June 1997.

[108] D. Hutter, ed. *Proc. FLoC'02 Workshop on Automation of Proofs by Mathematical Induction (IND-WS'99), July 6, 1999, Trento, Italy*, 1999.

[109] D. Hutter. Annotated reasoning. *Annals of Mathematics and Artificial Intelligence*, 29(1–4):283–222, 2000. Special Issue on Strategies in Automated Deduction, ed. by Bernhard Gramlich, Hélène Kirchner, and Frank Pfenning.

[110] D. Hutter. Deduction as an engineering science. *Electronic Notes in Theoretical Computer Science*, 86(1), June 2003. Final Proc. of 4th International Workshop on First-Order Theorem Proving (FTP 2003), June 12-14, 2003, Valencia, Spain.

[111] D. Hutter and A. Bundy. The design of the CADE-16 inductive theorem prover contest. In H. Ganzinger, ed., *Proc. 16th Int. Conf. on Automated Deduction, Trento, Italy, July 7-10, 1999*, LNCS 1632, pp. 374–377. Springer, July 1999.

[112] D. Hutter and M. Kohlhase. A coloured version of the λ-calculus. In W. McCune, ed., *Proc. 14th Int. Conf. on Automated Deduction*, LNCS 1249, pp. 291–305, Townsville, North Queensland, Australia, July 1997. Springer.

[113] D. Hutter and M. Kohlhase. Managing structural information by higher-order colored unification. *Journal of Automated Reasoning*, 25(2):123–164, 2000.

[114] D. Hutter, G. Rock, J. H. Siekmann, W. Stephan, and R. Vogt. Formal software development in the verification support environment (VSE). In J. N. Etheredge and B. Z. Manaris, eds., *Proc. 13nth International Florida Artificial Intelligence Research Society Conference (FLAIRS'00), May 22-24, 2000, Orlando, Florida, USA*, pp. 367–376. AAAI Press, 2000.

[115] D. Hutter and C. Sengler. INKA: The next generation. In M. McRobbie and J. Slaney, eds., *Proc. 13th Int. Conf. on Automated Deduction, New Brunswick, NJ, USA, July 30 – August 3, 1996*, LNCS 1104, pp. 288–292. Springer, 1996.

[116] A. Ireland. The use of planning critics in mechanizing inductive proofs. In A. Voronkov, ed., *Proc. 3rd Int. Conf. on Logic Programming and Automated Reasoning (LPAR 1992), St. Petersburg, Russia, July 15-20, 1992, 1993*, LNCS 624, pp. 178–189. Springer, July 1992.

[117] A. Ireland. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16(1–2):79–111, Mar. 1996.

[118] A. Ireland and A. Bundy. Extensions to a generalization critic for inductive proof. In J. S. Michael A. McRobbie, ed., *Proc. 13th Int. Conf. on Automated Deduction, New Brunswick, NJ, USA, July 30 – August 3, 1996*, LNCS 1104, pp. 47–61. Springer, 1996.

[119] A. Ireland and A. Bundy. Automatic verification of functions with accumulating parameters. *Journal of Functional Programming*, 9(2):225–45, 1999.

[120] A. Ireland, M. Jackson, and G. Reid. Interactive proof critics. *Formal Aspects of Computing*, 11(3):302–325, 1999.

[121] P. Janicic and A. Bundy. A general setting for flexibly combining and augmenting decision procedures. *Journal of Automated Reasoning*, 28(3):257–305, Apr. 2002.

[122] J.-P. Jouannaud and E. Kounalis. Automatic proofs by induction in equational theories without constructors. In *Proc. Symposium on Logic in Computer Science*, pp. 358–366. IEEE, 1986. also in Information and Computation, vol. 82(1), pp. 1–33, 1989.

[123] J.-P. Jouannaud and E. Kounalis. Automatic proofs by induction in theories without constructors. *Information and Computation*, 82:1–33, July 1989.

[124] J.-P. Jounnaud. Theorem proving languages for verification. In F. Wang, ed., *Proc. 2nd International Conference on Automated Technology for Verification and Analysis (ATVA'04), Taipei, Taiwan, ROC, October 31-November 3, 2004*, LNCS 3299, pp. 11–14. Springer, 2004.

[125] D. Kapur. An automated tool for analyzing completeness of equational specifications. In T. Ostrand, ed., *Proc. of the 1994 International Symposium on Software Testing and Analysis (ISSTA'94), August 17-19, 1994, Seattle, WA, USA*, volume Special Issue of *Software Engineering Notes*, pp. 28–43, Aug. 1994.

[126] D. Kapur. Rewriting, decision procedures and lemma speculation for automated hardware verification. In E. Gunter and A. Felty, eds., *Proc. 10th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'97), Murray Hill, NJ, USA, August 19-22, 1997*, LNCS 1275, pp. 171–182. Springer, Aug. 1997.

[127] D. Kapur and D. Musser. Inductive reasoning with incomplete specifications. preliminary report. In *Proc. of 1st IEEE Symposium on Logic in Computer Science (LICS'86)*, Cambridge, MA, 1986.

[128] D. Kapur and D. Musser. Proof by consistency. *Artificial Intelligence*, 31:125–157, 1987.

[129] D. Kapur, D. Musser, and X. Nie. The Tecton proof system. In V. Alagar, L. Lakshmanan, and F. Sadri, eds., *Formal Methods in Databases and Software Engineering, Proceedings of the Workshop on Formal Methods in Databases and Software Engineering, Montreal, Canada, 15-16 May 1992*, Workshops in Computing, pp. 54–79. Springer, 1993.

[130] D. Kapur, D. Musser, and X. Nie. An overview of the Tecton proof system. *Theoretical Computer Science*, 133(2):307–339, 1994.

[131] D. Kapur, P. Narendran, and H. Zhang. Proof by induction using test sets. In J. Siekmann, ed., *Proc. 9th Int. Conf. on Automated Deduction (CADE'88), Argonne, Illinois, USA, May 23-26, 1988*, LNCS 230, pp. 99–117. Springer, 1986.

[132] D. Kapur, P. Narendran, and H. Zhang. On sufficient completeness and related properties of term rewriting systems. *Acta Informatica*, 24:395–415, 1987.

[133] D. Kapur, P. Narendran, and H. Zhang. Automating inductionless induction using test sets. *Journal of Symbolic Computation*, 11(1/2):83–111, 1991.

[134] D. Kapur, X. Nie, and D. Musser:. An overview of the Tecton proof system. *Theoretical Computer Science*, 133(2):307–339, Oct. 1994.

[135] D. Kapur and N. Sakhanenko. Automatic generation of generalization lemmas for proving properties of tail-recursive definitions. In D. Basin and B. Wolff, eds., *Proc. 16th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'03), Rom, Italy, September 8-12, 2003*, LNCS 2758, pp. 136–154. Springer, Sept. 2003.

[136] D. Kapur and M. Subramaniam. Using linear arithmetic procedure for generating induction schemes. In P. Thiagarajan, ed., *Proc. 14th Conf. on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS'97), Madras, India, December 15-17, 1994*, LNCS 880, pp. 438–449. Springer, Dec. 1994.

[137] D. Kapur and M. Subramaniam. Automating induction over mutually recursive functions. In M. Wirsing and M. Nivat, eds., *Proc. 5th Int. Conf. on Algebraic Methodology and Software Technology (AMAST'96), Munich, Germany, July 1-5, 1996*, LNCS 1101, pp. 117–131. Springer, July 1996.

[138] D. Kapur and M. Subramaniam. Lemma discovery in automated induction. In M. McRobbie and J. Slaney, eds., *Proc. 13th Int. Conf. on Automated Deduction, New Brunswick, NJ, USA, July 30 – August 3, 1996*, LNCS 1104, pp. 538–552. Springer, 1996.

[139] D. Kapur and M. Subramaniam. New uses of linear arithmetic in automated theorem proving by induction. *Journal of Automated Reasoning*, 16(1–2):39–78, Mar. 1996.

[140] D. Kapur and M. Subramaniam. Extending decision procedures with induction schemes. In D. McAllester, ed., *Proc. 17th Int. Conf. on Automated Deduction, Pittsburgh, PA, USA, June 17-20, 2000*, LNCS 1831, pp. 324–345. Springer, June 2000.

[141] D. Kapur and M. Subramaniam. Automatic generation of simple lemmas from recursive definitions using decision procedures – preliminary report. In V. Saraswat, ed., *Advances in Computing Science – ASIAN 2003, Programming Languages and Distributed Computation, 8th Asian Computing Science Conference, Mumbai, India, December 10-14, 2003*, LNCS 2896, pp. 125–145. Springer, 2003.

[142] D. Kapur and H. Zhang. An overview of RRL: Rewrite rule laboratory. In N. Dershowitz, ed., *Proc. 3rd Int. Conf. on Rewriting Techniques and Applications (RTA'89)*, LNCS 355, pp. 513–529. Springer, 1989.

[143] D. Kapur and H. Zhang. An overview of the rewrite rule laboratory. *Journal of Computer and Mathematics with Applications*, 29(2):91–114, 1995.

[144] D. Kapur and H. Zhang. An overview of the rewrite rule laboratory. *Journal of Computer and Mathematics with Applications*, 29(2):91–114, 1995.

[145] M. Kaufmann. An interactive enhancement to the boyer-moore theorem prover. In E. Lusk and R. Overbeek, eds., *Proc. 9th Int. Conf. on Automated Deduction (CADE'88), Argonne, Illinois, USA, May 23-26, 1988*, LNCS 310, pp. 735–736. Springer, May 1988.

[146] M. Kaufmann. Generalization in the presence of free variables: A mechanically-checked correctness proof for one algorithm. *Journal of Automated Reasoning*, 7(1):109–158, Mar. 1991.

[147] M. Kaufmann. An informal discussion of issues in mechanically-assisted reasoning. In M. Archer, J. Joyce, K. Levitt, and P. Windley, eds., *Proceedings 4th International Workshop on the HOL Theorem Proving System and its Applications, August 1991, Davis, California, USA*, pp. 318–337. IEEE Computer Society, 1991.

[148] M. Kaufmann. An extension of the Boyer-Moore theorem prover to support first-order quantification. *Journal of Automated Reasoning*, 9(3):355–372, 1992.

[149] M. Kaufmann. Acl2 support for verification projects (invited talk). In C. Kirchner and H. Kirchner, eds., *Proc. 15th Int. Conf. on Automated Deduction, Lindau, Germany, July 5-10, 1998*, LNCS 1421, pp. 220–238. Springer, July 1998.

[150] M. Kaufmann, P. Manolios, and J S. Moore. *Computer-Aided Reasoning: ACL2 Case Studies*. Kluwer Academic Publishers, June 2000.

[151] M. Kaufmann, P. Manolios, and J S. Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, 2000.

[152] M. Kaufmann and J S. Moore. Design goals of ACL2. CLI Technical Report 101, Computational Logic, Inc., 1994.

[153] M. Kaufmann and J S. Moore. An industrial strength theorem prover for a logic based on Common Lisp. *IEEE Transactions on Software Engineering*, 23(4):203–213, Apr. 1997.

[154] M. Kaufmann and J S. Moore. Structured theory development for a mechanized logic. *Journal of Automated Reasoning*, 26(2):161–203, 2001.

[155] M. Kaufmann and P. Pecchiari. Interaction with the boyer-moore theorem prover: A tutorial study using the arithmetic-geometric mean theorem. *Journal of Automated Reasoning*, 16(3), June 1996.

[156] L. A. S. Kirby and J. Paris. Accessible independence results for Peano arithmetic. *Bull. Lond. Math. Soc.*, 14:285–293, 1982.

[157] T. Kolbe and C. Walther. Reusing proofs. In A. Cohn, ed., *Proc. 9th European Conf. on Artificial Intelligence (ECAI'94), Amsterdam, The Netherlands, August 8-12, 1994*, pp. 80–84. John Wiley and Sons, Aug. 1994.

[158] T. Kolbe and C. Walther. Patching proofs for reuse (extended abstract). In N. Lavrac and S. Wrobel, eds., *Proc. 8th European Conference on Machine Learning (ECML'95), Heraclion, Crete, Greece, April 25-27, 1995*, LNCS 912, pp. 303–306. Springer, 1995.

[159] T. Kolbe and C. Walther. Termination of theorem proving by reuse. In M. McRobbie and J. Slaney, eds., *Proc. 13th Int. Conf. on Automated Deduction, New Brunswick, NJ, USA, July 30 – August 3, 1996*, LNCS 1104, pp. 106–120. Springer, 1996.

[160] E. Kounalis. Testing for the ground (co-)reducibility property in term-rewriting systems. *Theoretical Computer Science*, 1:87–117, 1992.

[161] E. Kounalis and M. Rusinowitch. Mechanizing inductive reasoning. In *Proc. 8th National Conference on Artificial Intelligence*, pp. 240–245. American Association For Artificial Intelligence, MIT Press, 1990.

[162] E. Kounalis and M. Rusinowitch. Mechanizing inductive reasoning. *Bulletin of the European Association for Theoretical Computer Science*, 41:216–226, June 1990.

[163] E. Kounalis and P. Urso. Generalization discovery for proofs by induction in conditional theories. In A. Kumar and I. Russell, eds., *Proc. 12th International Florida Artificial Intelligence Research Society Conference (FLAIRS'99), May 1-5, 1999, Orlando, Florida, USA*, pp. 250–256. AAAI Press, May 1999.

[164] I. Kraan. Using the rippling heuristic in set membership proofs. In J. P. Bowen, M. G. Hinchey, and D. Till, eds., *Proc. 10th International Conference of Z Users (ZUM'97), Reading, UK, April 3-4, 1997*, LNCS 1212. Springer, Apr. 1997.

[165] I. Kraan, D. Basin, and A. Bundy. Middle-out reasoning for logic program synthesis. In D. Warren, ed., *Proc. 10th Int. Conf. on Logic Programming, June 21-25, 1993, Budapest, Hungary*, pp. 441–455. MIT Press, 1993.

[166] I. Kraan, D. Basin, and A. Bundy. Middle-out reasoning for synthesis and induction. *Journal of Automated Reasoning*, 16(1–2):113–145, 1996.

[167] G. Kreisel. Mathematical logic. In T. Saaty, ed., *Lectures on Modern Mathematics*, volume 3, pp. 95–195. J. Wiley & Sons, 1965.

[168] U. Kühler. *A Tactic-Based Inductive Theorem Prover for Data Types with Partial Operations.* PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, Infix, Sankt Augustin, 2000.

[169] U. Kühler and C.-P. Wirth. Conditional equational specifications of data types with partial operations for inductive theorem proving. In H. Comon, ed., *Proc. 8th Int. Conf. on Rewriting Techniques and Applications (RTA'97)*, LNCS 1232, pp. 38–52, Sitges, Spain, June 1997. Springer.

[170] A. Lazrek, P. Lescanne, and J.-J. Thiel. Tools for proving inductive equalities, relative completeness, and ω-completeness. *Information and Computation*, 81(1):47–70, 1990.

[171] J. C. López and R. Monroy. A rippling-based difference reduction technique to automatically prove security protocol goals. In C. Lemaître, C. A. Reyes, and J. A. González, eds., *Proc. 9th Ibero-American Conference on Artificial Intelligence, Puebla, México, November 22-26, 2004 (IBERAMIA'04)*, LNCS 3315, pp. 364–374. Springer, Nov. 2004.

[172] H. Lowe, A. Bundy, and D. McLean. The use of proof planning for co-operative theorem proving. *Journal of Symbolic Computation*, 25(2):239–261, 1998.

[173] L. Dixon and J. D. Fleuriot IsaPlanner: A prototype proof planner in isabelle. In F. Baader, ed., *Proc. 19th Int. Conf. on Automated Deduction (CADE'03), Miami Beach, FL, USA, July 28 – August 2, 2003*, LNCS 2741, pp. 279–283. Springer, 2003.

[174] P. Madden, A. Bundy, and A. Smaill. Recursive program optimization through inductive synthesis proof transformation. *Journal of Automated Reasoning*, 22(1):65–115, 1999.

[175] A. Manning, A. Ireland, and A. Bundy. Increasing the versatility of heuristic based theorem provers. In A. Voronkov, ed., *Proc. 4th Int. Conf. on Logic Programming and Automated Reasoning (LPAR 1993), St. Petersburg, Russia, July 13-20, 1993*, LNCS 698, pp. 194–204. Springer, July 1993.

[176] P. Manolios and J S. Moore. Partial functions in ACL2. *Journal of Automated Reasoning*, 31(2), 2003.

[177] R. Monroy, A. Bundy, and I. Green. Planning proofs of equations in ccs. *Automated Software Engineering¡*, 7(7):263–304, 2000.

[178] R. Monroy, A. Bundy, and A. Ireland. Proof plans for the correction of false conjectures. In F. Pfenning, ed., *Proc. 5th Int. Conf. on Logic Programming and Automated Reasoning (LPAR 1994), Kiev, Ukraine, July 16-22, 1994*, LNCS 822, pp. 54–68. Springer, 1994.

[179] L. M. de Moura, S. Owre, H. Rueß, J. M. Rushby, and N. Shankar. The ICS decision procedures for embedded deduction. In D. A. Basin and M. Rusinowitch, eds., *Proc. 2ndd Int. Joint Conf. on Automated Reasoning (IJCAR'04), Cork, Ireland, July 4-8, 2004*, LNCS 3097, pp. 218–222. Springer, July 2004.

[180] L. M. de Moura, S. Owre, H. Rueß, J. M. Rushby, N. Shankar, M. Sorea, and A. Tiwari. SAL 2. In R. Alur and D. Peled, eds., *Proc. 16th International Conference on Computer Aided Verification (CAV 2004), Boston, MA, USA, July 13-17, 2004*, LNCS 3114, pp. 496–500. Springer, July 2004.

[181] L. M. de Moura, H. Rueß, and M. Sorea. Bounded model checking and induction: From refutation to verification (extended abstract, category a). In W. A. Hunt Jr. and F. Somenzi, eds., *Proc. 15th International Conference on Computer Aided Verification (CAV 2003), Boulder, CO, USA, July 8-12, 2003*, LNCS 2725, pp. 14–26. Springer, July 2003.

[182] D. R. Musser. On proving properties of abstract data types. In *Proc. 7th ACM Symposium on Principles of Programming Languages, Las Vegas, Nevada, January 1980*, pp. 154–162, 1980.

[183] D. H. nad Bruno Langenstein, C. Sengler, J. H. Siekmann, W. Stephan, and A. Wolpers. Deduction in the verification support environment (VSE). In M.-C. Gaudel and J. Woodcock, eds., *Proc. 3rd International Symposium of Formal Methods Europe: Industrial Benefit and Advances in Formal Methods (FME'96), Co-Sponsored by IFIP WG 14.3, Oxford, UK, March 18-22, 1996*, LNCS 1051, pp. 268–286. Springer, Mar. 1996.

[184] G. Nelson and D. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1(2):245–257, 1979.

[185] G. Nelson and D. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356–364, Apr. 1980.

[186] R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In J. Robinson and A. Voronkov, eds., *Handbook of Automated Reasoning*, volume 1, chapter 7, pp. 371–443. Elsevier and MIT Press, 2001.

[187] T. Nipkow. Term rewriting and beyond – theorem proving in Isabelle. *Formal Aspects of Computing*, 1(4):320–338, 1989.

[188] T. Nipkow, L. C. Paulson, and M. Wenzel, eds. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, LNCS 2283. Springer, 2002.

[189] S. Owre, S. Rajan, J. M. Rushby, N. Shankar, and M. K. Srivas. PVS: Combining specification, proof checking, and model checking. In R. Alur and T. A. Henzinger, eds., *Proc. 8th International Conference on Computer Aided Verification (CAV'96), New Brunswick, NJ, USA, July 31 – August 3, 1996*, LNCS 1102, pp. 411–414. Springer, 1996.

[190] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, ed., *Proc. 11th Int. Conf. on Automated Deduction (CADE'92), Saratoga Springs, NY, USA, June 15-18, 1992*, LNCS 607, pp. 748–752. Springer, June 1992.

[191] S. Owre, J. M. Rushby, N. Shankar, and D. W. J. Stringer-Calvert. PVS: An experience report. In D. Hutter, W. Stephan, P. Traverso, and M. Ullmann, eds., *Proc. International Workshop on Current Trends in Applied Formal Methods (FM-TRENDS'98), Boppard, Germany, October 7-9, 1998*, LNCS 1641, pp. 338–345. Springer, Oct. 1998.

[192] E. Paul. Proof by induction in equational theories with relations between constructors. In B. Courcelle, ed., *Proc. Coll. on Trees in Algebra and Programming (CAAP'84)*. Cambridge University Press, 1984.

[193] E. Paul. Equational methods in first order predicate calculus. *Journal of Symbolic Computation*, 1(1):7–29, Mar. 1985.

[194] L. C. Paulson. Isabelle: The next seven hundred theorem provers. In E. L. Lusk and R. A. Overbeek, eds., *Proc. 9th Int. Conf. on Automated Deduction (CADE'88), Argonne, Illinois, USA, May 23-26, 1988*, LNCS 310, pp. 772–773. Springer, May 1988.

[195] L. C. Paulson. *Isabelle – A Generic Theorem Prover (with a contribution by T. Nipkow)*. Springer, 1994.

[196] L. C. Paulson. Proving properties of security protocols by induction. In *Proc. 10th Computer Security Foundations Workshop (CSFW '97), June 10-12, 1997, Rockport, Massachusetts, USA*, pp. 70–83. IEEE Computer Society, June 1997.

[197] N. Peltier. Model building with ordered resolution: Extracting models from saturated clause sets. *Journal of Symbolic Computation*, 36:5–48, 2003.

[198] B. Pientka and C. Kreitz. Instantiation of existentially quantified variables in inductive specification proofs. In J. Calmet and J. Plaza, eds., *Proc. 4th Int. Conf. on Artificial Intelligence and Symbolic Computation (AISC'98), Plattsburgh, New York, USA, September 16-18, 1998*, LNCS 1476, pp. 247–258. Springer, Sept. 1998.

[199] M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Sprawozdanie z I Kongresu Matematikow Krajow Slowianskich (Comptes Rendu I Congrès des Mathématiciens des Pays Slaves), Warszawa, 1929*, pp. 92–101, 1930.

[200] M. Protzen. Disproving conjectures. In D. Kapur, ed., *Proc. 11th Int. Conf. on Automated Deduction (CADE'92), Saratoga Springs, NY, USA, June 15-18, 1992*, LNCS 607, pp. 340–354. Springer, June 1992.

[201] M. Protzen. Lazy generation of induction hypotheses. In A. Bundy, ed., *Proc. 12th Int. Conf. on Automated Deduction (CADE'94), Nancy, France, June 26 – July 1, 1994*, LNCS 814, pp. 42–56. Springer, 1994.

[202] M. Protzen. Patching faulty conjectures. In M. McRobbie and J. Slaney, eds., *Proc. 13th Int. Conf. on Automated Deduction, New Brunswick, NJ, USA, July 30 – August 3, 1996*, LNCS 1104, pp. 77–91. Springer, 1996.

[203] Z. Qian. Structured contextual rewriting. In P. Lescanne, ed., *Structured Contextual Rewriting*, LNCS 256, pp. 168–179. Springer, May 1987.

[204] U. Reddy. Term rewriting induction. In M. Stickel, ed., *Proc. 10th Int. Conf. on Automated Deduction (CADE'90), Kaiserslautern, FRG, July 24-27, 1990*, LNCS 449, pp. 162–177. Springer, July 1990.

[205] J. Richardson, A. Smaill, and I. Green. System description: Proof planning in higher-order logic with Lambda-Clam. In C. Kirchner and H. Kirchner, eds., *Proc. 15th Int. Conf. on Automated Deduction, Lindau, Germany, July 5-10, 1998*, LNCS 1421, pp. 129–133. Springer, July 1998.

[206] J. A. Robinson and A. Voronkov, eds. *Handbook of Automated Reasoning*. Elsevier and MIT Press, 2001. in two volumes.

[207] M. Rusinowitch, S. Stratulat, and F. Klay. Mechanical verification of an ideal incremental ABR conformance. In E. Emerson and A. Sistla, eds., *Proc. 12th International Conference on Computer Aided Verification (CAV'00), Chicago, IL, USA, July 15-19, 2000*, LNCS 1855, pp. 344–357. Springer, July 2000.

[208] M. Rusinowitch, S. Stratulat, and F. Klay. Mechanical verification of an ideal incremental ABR conformance algorithm. *Journal of Automated Reasoning*, 30(2):53–177, Feb. 2003.

[209] R. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31(1):1–12, 1984.

[210] A. Stevens. A rational reconstruction of Boyer and Moore's technique for constructing induction formulas. In Y. Kodratoff, ed., *Proc. 8th European Conf. on Artificial Intelligence (ECAI'88), Munich, Germany, August 1-5, 1988*, pp. 565–570. Pitmann Publishing, London, Aug. 1998.

[211] G. Sutcliffe. System description: Systemon tptp. In D. A. McAllester, ed., *Proc. 17th Int. Conf. on Automated Deduction, Pittsburgh, PA, USA, June 17-20, 2000*, LNCS 1831, pp. 406–410. Springer, June 2000.

[212] G. Sutcliffe and C. B. Suttner. The TPTP problem library - CNF release v1.2.1. *Journal of Automated Reasoning*, 21:177–203, Oct. 1998.

[213] G. Sutcliffe and C. B. Suttner. The CADE-15 ATP system competition. *1999*, 23(1):1–23, July 1999.

[214] G. Sutcliffe, C. B. Suttner, and F. J. Pelletier. The IJCAR ATP system competition. *Journal of Automated Reasoning*, 28(3):307–320, Apr. 2002.

[215] G. Sutcliffe, C. B. Suttner, and T. Yemenis. The TPTP problem library. In A. Bundy, ed., *Proc. 12th Int. Conf. on Automated Deduction (CADE'94), Nancy, France, June 26 – July 1, 1994*, LNCS 814, pp. 252–266. Springer, 1994.

[216] A. Tarski. *A Decision Method for Elementary Algebra and Geometry (2nd ed.)*, volume III. University of California Press, Berkeley, 1951. 63 p.

[217] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. Lond. Math. Soc., II. Ser. 42*, pp. 230–265, 1936.

[218] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. a correction. *Proc. Lond. Math. Soc., II. Ser. 43*, pp. 544–546, 1937.

[219] P. Urso and E. Kounalis. "Term partition" for mathematical induction. In R. Nieuwenhuis, ed., *Proc. 14th Int. Conf. on Rewriting Techniques and Applications (RTA'03), Valencia, Spain, June 9-11, 2003*, LNCS 2706, pp. 352–366. Springer, June 2003.

[220] P. Urso and E. Kounalis. Sound generalizations in mathematical induction. *Theoretical Computer Science*, 323(1–3):443–471, Sept. 2004.

[221] C. Walther. Computing induction axioms. In A. Voronkov, ed., *Proc. 3rd Int. Conf. on Logic Programming and Automated Reasoning (LPAR 1992), St. Petersburg, Russia, July 15-20, 1992, 1993*, LNCS 624, pp. 381–392. Springer, July 1992.

[222] C. Walther. Combining induction axioms by machine. In R. Bajcsy, ed., *Proc. 13th Int. Conf. on Artificial Intelligence (IJCAI'93), Chambéry, France, August 28 -September 3, 1993*, pp. 95–101. Morgan Kaufmann, 1993.

[223] C. Walther. Mathematical induction. In D. Gabbay, C. Hogger, and J. Robinson, eds., *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 2, chapter 13, pp. 127–228. Oxford University Press, 1994.

[224] C. Walther. On proving the termination of algorithms by machine. *Artificial Intelligence*, 71(1):101–157, 1994.

[225] C. Walther and T. Kolbe. On terminating lemma speculations. *Information and Computation*, 162(1–2):96–116, October/November 2000.

[226] C. Walther and T. Kolbe. Proving theorems by reuse. *Artificial Intelligence*, 116(1–2):17–66, Jan. 2000.

[227] C.-P. Wirth. Descente infinie + deduction. *Logic Journal of the IGPL*, 12(1):1–96, Jan. 2004.

[228] C.-P. Wirth and K. Becker. Abstract notions and inference systems for proofs by mathematical induction. In N. Dershowitz and N. Lindenstrauss, eds., *Proc. 4th Int. Workshop on Conditional and Typed Rewriting Systems (CTRS 1994), Jerusalem, Israel, July 13-15, 1994*, LNCS 968, pp. 353–373. Springer, 1995.

[229] C.-P. Wirth and B. Gramlich. A constructor-based approach for positive/negative conditional equational specifications. *Journal of Symbolic Computation*, 17:51–90, 1994.

[230] C.-P. Wirth and B. Gramlich. On notions of inductive validity for first-order equational clauses. In A. Bundy, ed., *Proc. 12th Int. Conf. on Automated Deduction*, LNCS 814, pp. 162–176, Nancy, France, 1994. Springer.

[231] T. Yoshida, A. Bundy, I. Green, T. Walsh, and D. A. Basin. Coloured rippling: An extension of a theorem proving heuristic. In A. G. Cohn, ed., *Proc. 9th European Conf. on Artificial Intelligence (ECAI'94), Amsterdam, The Netherlands, August 8-12, 1994*, pp. 85–89. John Wiley and Sons, Aug. 1994.

[232] H. Zhang. *Reduction, Superposition and Induction: Automated Reasoning in an Equational Logic.* PhD thesis, Rensselaer Polytech. Inst., Dept. of Comp. Sci., Troy, NY, 1988.

[233] H. Zhang. Contextual rewriting in automated reasoning. *Fundamenta Informaticae*, 24(1/2):107–1–23, 1995.

[234] H. Zhang and X. Hua. Proving the Chinese Remainder Theorem by the cover set induction. In D. Kapur, ed., *Proc. 11th Int. Conf. on Automated Deduction (CADE'92), Saratoga Springs, NY, USA, June 15-18, 1992*, LNCS 607, pp. 431–445. Springer, June 1992.

[235] H. Zhang, D. Kapur, and M. Krishnamoorthy. A mechanizable induction principle for equational specifications. In E. Lusk and R. Overbeek, eds., *Proc. 9th Int. Conf. on Automated Deduction (CADE'88), Argonne, Illinois, USA, May 23-26, 1988*, LNCS 310, pp. 162–181. Springer, May 1988.