Towards Intelligent Inductive Proof Engineering

Bernhard Gramlich
Fachbereich Informatik
Universität Kaiserslautern
D-6750 Kaierslautern
Germany
gramlich@informatik.uni-kl.de

Abstract

This paper deals with inductive theorem proving (ITP for short). It does not provide new theoretical results but analyses existing ITP methods from an AI point of view. The presentation is based on the *implicit* ITP approach, i.e. ITP using the well-developed framework of rewriting and completion techniques for systems of equations and rewrite rules. We think that the relevant practical problems for successful (partially) automated ITP are essentially the same as or at least closely related to those occurring in the more conventional framework of *explicit* ITP using schemas. The theoretical foundations of implicit ITP are briefly reviewed focussing on the central ideas as well as on important operationalization issues. Moreover, a brief comparison of explicit and implicit ITP approaches is included. In particular, we clarify some criticisms raised against the implicit ITP approach. The main part of the paper is devoted to a thorough discussion of central ITP problems from the viewpoint of system designers and users. We point out and exemplify the necessity of linking together the whole process of formalizing, modelling and structuring abstract (equational) specifications of algorithms and corresponding (inductive) properties to be verified. Crucial aspects of the whole specification and proof engineering process are isolated and discussed, in particular conceptual and proof-technical ones. We argue that such an analysis (which of course has to be continued and deepened) is necessary for an adequate integration and combination of intelligent user-guided and machine-supported automated inductive reasoning. Finally the main theoretical and practical problems as well as promising perspectives for future work are sketched, in particular concerning architectural and design principles for future generation inductive theorem provers.

Key words: inductive theorem proving, proof engineering, equational reasoning, term rewriting, completion.

1 Introduction

Equational reasoning is fundamental for many fields of computer science like functional programming, abstract data type specifications, program synthesis and verification. Within these applications one is usually interested in a standard initial model of a given equational axiom system and not in all its models. Proof methods for this initial model, i.e. for inductive theorem proving (abbreviated as ITP in the sequel), are usually based on induction schemas, e.g. for structural induction (cf. [BM79]).

Within the last decade an alternative approach based on rewriting and completion techniques has been developed (cf. e.g. [HH80]) and refined in many ways in the meanwhile (cf. e.g. [JK86], [Fri86], [Göb87], [Bac88], [Red90], [Gra90b]). Based on this theoretical framework which is briefly reviewed we shall focus on crucial aspects of the whole specification and proof engineering process occurring in practical applications.

Indeed, our extensive practical experiments with an elaborated implementation of the method (cf. [Gra90a], [GL91]) have shown that such a comprehensive point of view of the problem is really necessary for successfully tackling non-trivial proof problems as they naturally occur when verifying non-trivial inductive properties of (equationally defined) functional programs or abstract data type specifications, for instance.

The rest of this paper is organized as follows. In section 2 we briefly recall the main theoretical results for completion and rewriting based inductive theorem proving. Important operationalization and control aspects are summarized in a systematic manner. Furthermore, an overview of $\mathcal{UNICOM}$, an implementation of the method, is provided. A brief comparison with other (classical) approaches is given in section 3. The viewpoint of inductive theorem proving as a challenging engineering task is developed and exemplified in section 4. Finally, before concluding, open problems and promising perspectives for future research are discussed in section 5.

2 Theoretical Foundations and Operationalization Aspects

2.1 Theoretical Foundations

We shall omit most definitions and assume familiarity with the basic notions, definitions and results about (first order) terms, rewriting systems and equational logic (cf. e.g. [HO80], [Bac87]).

The inductive theory induced by a set E of equations is given by $ITh(E) := \{s = t \mid \sigma s \overset{*}{\leftrightarrow}_E \sigma t$ for all ground substitutions $\sigma\}$. Equations from $ITh(E)$ are called inductive theorems of E. These notions generalize easily to the case where we consider a term rewriting system (TRS for short) R instead of a set E of equations.

A TRS R is terminating if $\rightarrow_{\mathcal{R}}^{+}$ is a well-founded (strict partial) ordering. A reduction ordering $>$ is a well-founded ordering on terms which is monotonic w.r.t. replacement ($s > t \Longrightarrow u[s] > u[t]$) and substitution ($s > t \Longrightarrow \sigma s > \sigma t$). A TRS R is said to be $>$-ordered (and thus terminating) if $l > r$ holds for every rule $l \rightarrow r \in R$. R is confluent if ${}_{\mathcal{R}}^{*}\!\!\leftarrow \circ \rightarrow_{\mathcal{R}}^{*} \subseteq \rightarrow_{\mathcal{R}}^{*} \circ {}_{\mathcal{R}}^{*}\!\!\leftarrow$. It is ground confluent if it is confluent on ground terms. R is (ground) convergent if it is terminating and (ground) confluent. By $CP(R, R')$ we denote the set of critical pairs obtained from overlapping the rules of R into those of R'. If p is a non-variable position of a term l then $CP(R, p, l \rightarrow r)$ stands for the set of critical pairs obtained by overlapping R into $l \to r$ at position p. Accordingly $CP(R, P, l \to r)$ denotes the set of critical pairs obtained by overlapping R into $l \to r$ at all non-variable positions $p \in P$ from l. We speak of a critical peak $(\sigma l[\sigma r']_{l' \to r'} \leftarrow \sigma l[\sigma l'] \to_{l \to r} \sigma r)$ when taking into account the corresponding superposition term $\sigma l[\sigma l']$ belonging to the critical pair $(\sigma l[\sigma r'], \sigma r)$, too. For some set C of equations we write $C^{\leftrightarrow}$ for $\{l \to r, r \to l \mid l = r \in C\}$.

For a given TRS R a term s is inductively (R-) reducible iff all its ground instances are (R-) reducible. An equation $s = t$ is inductively (R-) reducible iff σs or σt is (R-) reducible for every ground instance $\sigma s = \sigma t$ of $s = t$ with $\sigma s \not\equiv \sigma t$ ($\equiv$ means syntactical equality). For a given ground convergent TRS R which is $>$-ordered by some reduction ordering $>$ we say that a set C of equations (inductive conjectures) is provably inconsistent iff C contains an equation $s = t$ with $s > t$ and s not inductively reducible or an equation $s = t$ with $s \not\equiv t$ and $s = t$ not inductively reducible.

A proof in $E \cup R$ consists of proof steps of the form $t_i \leftrightarrow_E t_{i+1}, t \to_{\mathcal{R}} t_{i+1}$ or $t_R\!\leftarrow t_{i+1}$. Two proofs of the form $s \overset{*}{\leftrightarrow}_{E \cup R} t$ and $u \overset{*}{\leftrightarrow}_{E' \cup R'} v$ are said to be equivalent if $s \equiv u$ and $t \equiv v$. Proofs of the form $\to_{\mathcal{R}}^* \circ {}_{\mathcal{R}}^{*}\!\!\leftarrow$ are called rewrite proofs. A proof ordering is a (strict partial) ordering $\gg$ on proofs. It is said to be a proof reduction ordering if it is well-founded and monotonic w.r.t. replacement, substitution and embedding. A proof P (in $E \cup R$) of the form $s \overset{*}{\leftrightarrow}_{E \cup R} t$ is said to be (equivalently) simplifiable into a proof P' (in $E' \cup R'$) if P' is of the form $s \overset{*}{\leftrightarrow}_{E' \cup R'} t$ with $P \gg P'$ (see [Bac87] for a more detailed introduction of equational proofs).

Using these notations inductive validity can be characterized as follows (cf. [Gra89], [Gra90b]):

Theorem 2.1 *Let $>$ be a reduction ordering, R a $>$-ordered ground convergent TRS, L a set of inductive theorems, i.e. $L \subseteq ITh(R)$, and C a set of inductive conjectures. Then: $C \subseteq ITh(R)$ iff*

(1) C is not provably inconsistent, and

(2) all ground instances of critical peaks corresponding to $CP(R, C^{\leftrightarrow})$ are equivalently simplifiable with $R \cup C \cup L$ (w.r.t. $>^p$).

Here $>^p$ is a proof reduction ordering which compares complexities of proofs using $R \cup C \cup L$ by comparing the corresponding multisets of the complexities of all occurring C-steps using essentially the double multiset extension $\ggg$ of the reduction ordering $>$[1] (cf. [Gra89]).[2] Condition (1) is decidable and can be easily tested in canonical theories with free constructors ($\mathcal{F}_0 \subseteq \mathcal{F}$ is a set of **free constructors** w.r.t. R iff every ground term $s \in \mathcal{T}(\mathcal{F})$ is R-reducible to a unique ground constructor term $s' \in \mathcal{T}(\mathcal{F}_0)$). A sufficient operational criterion for condition (2) is to verify simplifiability of the critical peaks corresponding to $CP(R, C^{\leftrightarrow})$ instead of ground simplifiability. For base systems R which are only terminating but not necessarily ground confluent a slight modification of

[1] More precisely, $>^p$ may be constructed in a more refined way by taking a lexicographic combination of several different well-founded orderings which are partially based on the underlying reduction ordering $>$.

[2] A slightly more general version consists in taking the transitive closure of the union of the reduction relation $\to_{\mathcal{R}}$ and the proper subterm ordering $>_{st}$, i.e. $\succ_R := (\to_{\mathcal{R}} \cup >_{st})^+$ (the "decreasing ordering generated by R", cf. [Red90]) instead of $>$.

the above theorem still provides a sufficient criterion for inductive validity. This is detailed in [Gra89], [Gra90b].

The general idea underlying the above theorem and the whole approach is to consider the problem of proving inductive validity of a conjecture $s = t$ w.r.t. a base system R as a proof simplification task. Namely, for every ground instance $\sigma s = \sigma t$ of $s = t$, considered as a proof consisting of one conjecture step, find an equivalent proof consisting of R-steps only. This positive point of view of the problem motivates the notion *positive* proof reduction ordering $>^{pos}$ mentioned above. Essentially it is the main difference to the *negative* approach for *proofs by consistency* as developed in [Bac88]. In the latter approach potentially existing inconsistency witnesses, i.e. invalid ground instances of conjectures are tried to be simplified until the inconsistency becomes obvious (in the sense of provable) or it is clear that there is no inconsistency, i.e. inductive validity is given. Within our positive framework we have to provide an appropriate noetherian ordering on proofs using base and conjecture steps such that (ground) proofs using only base steps are minimal. Then the aspired goal may be achieved gradually by finding equivalent but smaller proofs for every ground instance $\sigma s = \sigma t$ of $s = t$. The noetherian property of the proof ordering assures that such a proof simplification process cannot be performed ad infinitum. In fact, it is responsible for the inductive character of the whole method by allowing a conjecture to be applied in its own proof, but in *smaller* version. For the purpose of enabling proof simplification it may (seem to) be necessary to add new equations (conjectures) which are recursively processed in the same way and which are obtained essentially by simplification steps or the critical pair superposition mechanism. Such sets of deduced equations which suffice for assuring ground simplifiability of a given conjecture are called covering sets in [Bac88]. Indeed, processing a conjecture means covering it, either by simplification or by critical pair construction.

Completion and rewriting based inductive theorem proving roughly spoken proceeds as follows: Given a ground convergent base system R and a set C of inductive conjectures, C is first tested on provable inconsistency. If it is not provably inconsistent (and not directly simplifiable by base rules or lemmas) then (ground) simplifiability of critical peaks obtained by overlapping R into C is tried to be established. Intuitively, the superposition of base rules into a given conjecture $s = t$ provides a case analysis whose completeness is guaranteed because the conjecture is not provably inconsistent. The resulting critical pairs (called a covering set in [Bac88] and [Gra90b]) are in a sense new candidate conjectures which are recursively processed in the same way and which assure the desired property that every ground instance of $s = t$ is simplifiable. This process may stop successfully (all original and deduced conjectures are inductive theorems), detect a contradiction, i.e. provable inconsistency (at least one conjecture in C is not inductively valid) or run forever.

In order to obtain a better and precise comprehension of the theoretical basis as well as to derive concrete proof procedures one can make a clear distinction between logical and operational (control) aspects. The logical component can be formalized by an inference system as it has been done e.g. in [Bac88]. Given a base specification together with a corresponding reduction ordering such a system of inference rules describes possible transformation steps for pairs $(\mathcal{C}, \mathcal{L})$ or triples $(\mathcal{C}, \mathcal{H}, \mathcal{L})$. Here, $\mathcal{C}, \mathcal{H}$ and $\mathcal{L}$ denote sets of actual conjectures, already processed conjectures and inductive lemmas, respectively.[3]

[3]The distinction between processed and not yet processed conjectures may be considered to be an abstract implementation of a marking concept for treating inductive conjectures.

We shall not deepen this point here. Instead we shall focus more on operational aspects concerning refinements, optimizations and the control structure used for inference rule application.

2.2 Refinements and Optimizations

Essentially there are three main sources for refining and optimizing this implicit[4] ITP approach.

1. In order to minimize covering sets it is useful to restrict the number of critical pairs to be considered to so-called inductively complete (sets of) positions (cf. [Fri86], [Küc87]). Moreover, since ground subconnectedness (cf. e.g. [Küc87]) of these critical pairs is the main property to be established, it is worth-while refining the techniques for verifying this property. This can be done e.g. by weakening the goal of finding (ground) rewrite proofs for critical pairs (cf. [Göb87], [Küc87], [Gra89], [Gra90b]). Particularly important for practical success are very powerful simplification mechanisms because they partially reflect the degree of possible automation of the whole technique.

2. Instead of proceeding as economically as possible in deducing new equations (by simplification or critical pair construction) it is sometimes advantageous to perform additional potentially useful computations and exploit extra (inductive) knowledge of the domain of interest. Such refinements and optimizations may result in a considerable speed up of the proof process or even enable a successful proof whereas the spare technique fails due to non-termination. So-called inessential critical pairs (cf. [Küc87]) as well as generalization techniques (cf. [BM79]) fall into this category. Of course, these refinements also entail some disadvantages. Namely, a much more complicated management and book-keeping mechanism is required and moreover, some guessing mechanism or intelligent heuristic has to be provided for restricting the search space when producing new conjectures.

3. Finally, the combination and integration of the basic techniques for completion based inductive theorem proving and the corresponding refinements and optimizations into a concrete flexible and powerful implementation is a challenging task. In particular, the incorporation of good strategies and heuristics as well as the right mixture of automated and user-controlled reasoning are difficult problems.

Some of these points will be discussed in more detail in section 4.

2.3 $\mathcal{UNICOM}$: A Refined Completion Based Inductive Theorem Prover

Based on the above sketched theoretical framework we have implemented $\mathcal{UNICOM}$, a system for refined **UN**failing **I**nductive **COM**pletion[5] which is described in [Gra90c],

[4] The reason for using the terminology *implicit* and *explicit* ITP is explained later on in subsection 4.2.

[5] Here, *unfailing* indicates two aspects, namely, that the method can treat arbitrary possibly non-orientable conjectures, and secondly, that it is refutationally complete. The term *inductive completion* stands for completion procedures which are especially adapted for the purpose of ITP problems. Other similar notions used in literature are *inductionless induction* or *proof by consistency*. For our presentation we prefer the more general (and hopefully less confusing) notion of *implicit* ITP.

[GL91]. $\mathcal{UNICOM}$ is able to treat hierarchically structured many-sorted and constructor-based specifications of functions (rewrite programs) and inductive conjectures (equational properties to be proved). Input specifications have to satisfy the following conditions. Constructors must be declared and are required to be free. The left hand sides of the definition rules for a (n-ary) non-constructor symbol f have to be of the form $f(t_1, \ldots, f_n)$, where all t_i are constructor terms. The system comprises the following tools:

The **PARSER** checks the syntax of input specifications, imports used subspecifications and produces an internal representation.

The **CHECKER** tests the function definitions for completeness and consistency using the syntactical restrictions mentioned above. In particular, termination of the definition rules is established by automatically generating a suitable recursive path ordering with status which is also used for subsequent inductive proofs. Ground convergence is established by investigating critical pairs. A special feature of the implemented completeness test allows to identify minimal complete sets of defining rules for the same function symbol, i.e. alternative but equivalent function definitions.

The **COMPILER** provides a means for rapid prototyping by translating correct specifications (only the definition part) into executable LISP code.

The central part of the system is the **PROVER** which tries to prove or disprove the inductive conjectures of the current specification. $\mathcal{UNICOM}$ admits arbitrary possibly non-orientable (equational) conjectures. The following are some characteristic features of $\mathcal{UNICOM}$:

- Parallel independent inductive proofs can be performed according to the different possibilities of choosing complete positions in conjectures together with corresponding minimal complete function definitions.

- Inessential critical pairs may be computed as potentially useful auxiliary conjectures.

- Simplification (reduction) may be performed modulo the AC-properties of some operators.

- A simple generalization technique (via minimal non-variable common subterms) is available.

- Elimination of subsumed non-orientable conjectures is integrated.

- Non-equational inductive knowledge about free constructors is used for speeding up the proof or disproof of conjectures.

- Various user interface parameters allow for switching on/off optional features (e.g. generalization, computation of inessential critical pairs) and enable a fully automatic or more or less strongly user-controlled running mode.

- A couple of different trace levels are available in order to enable a more or less detailed inspection of the running proof process.

- A general navigation mechanism allows arbitrary navigation in the current proof tree which is useful for backtracking and (manually) focussing on certain interesting subproblems to be dealt with next.

- All logically relevant steps and actions of the proof process may be (automatically) recorded. This information provides the basis for (automatically) obtaining various more or less detailed and flexibly structured descriptions of a succuessfully finished, interrupted or failed proof attempt. In particular, for the analysis of failed proof attempts this information facility has turned out to be very valuable.

Numerous experiments with $\mathcal{UNICOM}$ involving a couple of non-trivial examples, e.g. various sorting algorithms, have been performed and evaluated which led to some substantial modifications and improvements of the system (cf. [Gra90a], [GL91]).

3 Comparison with Explicit ITP

The purpose of the following considerations is twofold. First we want to try to relate various versions of rewriting and completion based techniques for ITP to each other as well as to more classical approaches using some kind of (e.g. structural) schemas. We shall only sketch some of the most important aspects. We think that a substantial amount of work still seems to be necessary in order to obtain a convincing and comprehensive – perhaps uniform – framework for ITP techniques, at least for some restricted cases like equational logic. Such a uniform framework could be very useful, not only for isolating and understanding the central issues and notions used in different approaches, but also for clarifying their interrelations, advantages and disadvantages from a practical/operational point of view.

The second point we want to address here is to discuss some wide-spread misunderstandings converning the rewriting and completion based approach to ITP as presented in chapter 2. In particular, we claim that a couple of critical arguments raised against this approach (e.g. in [GG88]) do not persist to a thorough and precise analysis.

Perhaps the best-known and most successful ITP system up to now still is the Boyer/-Moore theorem prover (cf. [BM79], [BM88]). An alternative approach was initiated in [Mus80], [Gog80], [HH80] which was termed *inductionless induction* in [Lan80], [Lan81]. Since then a lot of progress in this direction has been achieved. For instance, abstracting from the special and (computationally) very convenient case of free constructors (cf. [HH80]) the method has ben generalized to arbitrary (convergent) base specifications (cf. [Pau84], [JK86]).

Optimizations concerning the minimization of covering sets have been developed in [Fri86], [Göb87], [Küc87]. This was done on the one hand side by restricting the critical pairs to be considered to inductively complete (sets of) positions, thus leading to a kind of linear proof procedure. On the other hand side the fact that only ground reasoning is relevant for inductive validity could be exploited for designing refined ground simplifiability (subconnectedness) criteria. Moreover, using the general framework of proof orderings (cf. [Bac87]) and unfailing completion techniques the restriction that conjectures have to be orientable w.r.t. the underlying reduction ordering could be removed (cf. [Bac88], [Gra89], [Gra90b]).

Finally the general paradigm of *proof by consistency* was thoroughly analyzed and developed in [KM87], [ZKK88], [Zha88]. This work was coupled with an investigation on the kind of possible model semantics underlying any intended notion of inductive validity. Indeed, one of the main drawbacks of the common initial semantics approach is the treatment of partiality, i.e. of partially defined functions. There is a certain lack of incremental behaviour which – roughly spoken – prevents the initial algebra approach from being monotonic w.r.t. consistent extensions. In [Zha88] this disadvantage has led to the fruitful notion of *constructor models* which – considered from an algebraic model-theoretic point of view – are not proper models because they allow partial functions on concrete domains. But it should be noted that these kinds of difficulties and drawbacks are not specific to the rewriting and completion based approach but are in general problematic for any kind of ITP approach.

Interesting variations of proof by consistency techniques have further been explored in [Pla85] and [KNZ86].

Within the framework of equational reasoning and rewriting techniques the classical approach to ITP using induction schemas has been discussed e.g. in [GG88].

Keeping close to rewriting and completion techniques the positive approach as sketched in chapter 2 and its relations to classical ITP have been worked out in [Red90] and [Gra89], [Gra90b]. In [Red90] the resulting method called *term rewriting induction* essentially performs inductive proofs w.r.t. the underlying noetherian reduction ordering $>$ (or more precisely for readers familiar with the approach: the decreasing order generated by $>$). In [Gra89], [Gra90b] a similar positive approach was developed exploiting the general, very powerful and flexible framework of proof orderings which of course heavily rely on the underlying reduction orderings. Both in [Red90] and [Gra89], [Gra90b] the inductive nature of the discussed methods as well as the close relationship to classical ITP approaches are pointed out. Hence the term *inductionless induction* for this class of approaches does not seem to be adequate any more, at least for the *positive* variants of rewriting and completion based ITP. Perhaps a better notion for distinguishing these aproaches would be to speak of explicit ITP (using schemas) and implicit ITP (using rewriting and completion techniques). This terminology will be used below.

Let us now turn briefly to the criticisms raised against the latter one. In particular, we want to address some wide-spread misunderstandings concerning the implicit ITP approach as sketched in section 2. We claim that some critical arguments raised again this approach (e.g. in [GG88]) do not persist when submitted to a thorough and precise analysis. We only pick out the most important points in [GG88]:

- The argument that explicit ITP has a simpler theoretical basis than implicit ITP may be partially true but is due to the fact that the latter provides a very general framework. In particular, it owns the great advantages that a powerful, precise and well-understood notion of simplification is available. In classical approaches the lack of a well-understood and practically applicable theoretical basis (in particular concerning a precise notion of simplification) is one of the main drawbacks concerning operationalization and automation issues.[6]

- Some important arguments against implicit ITP concerning its restricted applicability are not justified. For instance, neither (sufficient) completeness nor convergence of base specifications is really necessary for the approach to be applicable. Even non-termination is not a principle obstacle. Often it can be dealt with by using unfailing completion techniques and reasoning modulo some subtheory encapsulating the reason for non-termination. But clearly things are much more easier provided "nice"

[6] However, substantial progress in this direction seems to be possible (cf. [Wal92]).

properties like confluence and termination are given. Moreover, these problems have to be dealt with as well (and are by no means easier) in the classical explicit ITP approach.

- The argument that within explicit ITP proofs are easier to follow and failures easier to analyse does not hold either but of course the ability to understand precisely and interpret adequately what is going on presumes a certain amount of familiarity with and intuition about the basic concepts and mechanisms involved in implicit ITP. Instead we think that both approaches can profit quite a lot from each other in the future.

- Finally, a last point we would like to mention is an important advantage of implicit ITP. Namely, as shown in [Bac88], the implicit ITP approach is – in contrast to explicit ITP – refutationally complete (under some reasonable assumptions), i.e. invalid conjectures are eventually detected. This is not only a nice theoretical property, but may also be practically relevant because it can be exploited for finding out wrong conjectures, hopefully very early.

4 Inductive Theorem Proving as a Challenging Engineering Task

Very often ITP tasks are not given as isolated fixed problems but occur in a more general setting. For instance, in the field of abstract data type specifications usually a lot of basic knowledge (definitions and inductive lemmas), e.g. about data types like boolean algebra, natural numbers, lists etc. is available. When specifying non-trivial algorithms in an elegant and compact manner auxiliary functions usually play an important role. On the other hand the formalization of desired (inductive) properties of such algorithms – for instance partial correctness – also relies on the basic knowledge as well as on possibly newly introduced auxiliary functions and/or predicates. As we shall point out, exemplify and discuss, the ease or difficulty of successfully tackling the resulting verification problems is initimately tied not only to the technical details and features of the applied ITP technique but also to the above mentioned formalization and specification process. To this end, let us consider a non-trivial practical example taken from the case study in [Gra90a], namely the specification and verification of various sorting algorithms with $\mathcal{UNICOM}$.

Assume that enough basic knowledge about boolean algebra, natural numbers with the usual ordering relation $\leq$ and lists of natural numbers is available. Moreover, since sorting algorithms inherently require some kind of conditional reasoning, we have to provide means for specifying conditional operations and properties. Within this example we restrict ourselves to purely equational reasoning by encoding conditional constructs into unconditional ones using ternary if-then-else operations. Equations specifying defined functions and inductive conjectures will always be interpreted as directed from left to right, i.e. as rewrite rules. Furthermore we tacitly assume (sufficient) completeness and consistency of the considered specifications. In particular, we assume the existence of appropriate reduction orderings for the corresponding rewrite rule systems.[7]

[7]For the sake of readability we shall make free use of a mixed prefix and infix notation with usual priority rules for interpreting missing brackets.

4.1 Conceptual Aspects of Intelligent Specification and Proof Engineering

Let us now consider the problem of specifying and verifying (partial[8]) correctness of sorting algorithms. To this end let us assume that a sufficiently complete and consistent equational specification of a function *sort*: $list \to list$ for sorting lists of natural numbers (in ascending order) is given. Then we may ask for a formal specification of correctness of *sort*. Intuitively it is clear that correctness is characterized by the following two properties which must hold for every concrete list l of natural numbers:

(I) $sort(l)$ is ordered (ascendingly).
(II) l and $sort(l)$ have the same multiset of elements

These two properties may be formalized in quite different ways as we shall point out and discuss now. For property (I) it seems natural to introduce an ordered-predicate which is modelled by a boolean valued function $ord : list \to bool$ and to require

$$ord(sort(l)) \quad = \quad true.$$

A first specification of being an ordered list is obtained by defining recursively[9]

$$\text{(Ia)} \quad \begin{array}{lcl} ord(e) & = & true \\ ord(c(n,e)) & = & true \\ ord(c(m,c(n,l))) & = & m \leq n \wedge ord(c(n,l))\,. \end{array}$$

But other specifications may also be conceivable and – concerning proof-technical aspects – even more advantageous. For instance we may – in some sense redundantly – require that the first element of a given list is not only less than or equal to the next element (provided there exists one) but less than or equal to all remaining list elements. This leads to the following specification with an auxiliary operation $\leq_{nl}$: $nat \times list \to bool$.

$$\text{(Ib)} \quad \begin{array}{lcl} ord(e) & = & true \\ ord(c(n,l)) & = & n \leq_{nl} l \wedge ord(l) \\ m \leq_{nl} e & = & true \\ m \leq_{nl} c(n,l) & = & m \leq n \wedge m \leq_{nl} l \ . \end{array}$$

Still another less explicit and not constructor-based definition using the auxiliary operations $\leq_{nl}$ (as above) and $\leq_{ll}$ (for comparing two lists) would be

$$\text{(Ic)} \quad \begin{array}{lcl} ord(app(l_1,l_2)) & = & ord(l_1) \wedge ord(l_2) \wedge l_1 \leq_{ll} l_2 \\ e \leq_{ll} l & = & true \\ c(n,l_1) \leq_{ll} l_2 & = & n \leq_{nl} l_2 \wedge l_1 \leq_{ll} l_2 \ . \end{array}$$

It is rather straightforward to prove that the definitions (Ia), (Ib) and (Ic) are indeed equivalent.

The second correctness property (II) of sorting algorithms may be formalized in quite different ways. A first possibility is to introduce the property that a list is a permutation of another list via the operation $perm : list \times list \to bool$ which uses auxiliary functions

[8]For our purpose we assume totality of the function definitions, i.e. we do not treat here the problem of verifying termination of the corresponding rewrite programs.

[9]Here, the constructors e and c stand for the empty list and the usual *cons*-operation.

$del : nat \times list \rightarrow list$ and $\in: nat \times list \rightarrow bool$ for deleting one occurrence of an element in a given list and for membership test, respectively:[10]

$$\text{(IIa)} \quad \begin{array}{lcl} perm(e,e) & = & true \\ perm(e,c(n,l)) & = & false \\ perm(c(m,l_1),l_2) & = & m \in l_2 \wedge perm(l_1, del(m,l_2)) \\ m \in e & = & false \\ m \in c(n,l) & = & m =_{nat} n \vee m \in l \\ del(m,e) & = & e \\ del(m,c(n,l)) & = & if_{list}(m =_{nat} n, l, c(n, del(m,l))) \ . \end{array}$$

The correctness property (II) for sorting algorithms may then be formalized by

$$perm(l, sort(l)) \quad = \quad true.$$

The above definition of $perm$ is in a sense a strongly algorithmic one since the computation of $perm(l_1,l_2)$ for concrete lists l_1,l_2 of natural numbers proceeds by successively considering the first element m of l_1, testing for membership of m in l_2 and deleting (the first occurrence of) m in l_1 and l_2. Moreover, the structure of the definition does not reflect the symmetrical aspect of permutative equivalence, i.e. the property that two lists are the same up to a permutation of their arguments. This symmetry stated by

$$perm(l_1,l_2) \quad = \quad perm(l_2,l_1)$$

does indeed hold for the above definition, but a formal proof of it is non-trivial (cf. [Gra90c]).

As an alternative where symmetry becomes obvious consider the following definition using a function $oc : nat \times list \rightarrow nat$ which counts the number of occurrences of some natural number in a given list.

$$\text{(IIb)} \quad \begin{array}{lcl} oc(m,e) & = & 0 \\ oc(m,c(n,l)) & = & if_{nat}(m =_{nat} n, s(0), 0) + oc(m,l) \ . \end{array}$$

Then permutative equivalence of two lists l_1, l_2 is formalized by

$$oc(n,l_1) \quad = \quad oc(n,l_2) \ .$$

To ensure that this definition is indeed equivalent to the former version using $perm$ we have to verify that

$$(L) \quad perm(l_1,l_2) = t \quad \Longleftrightarrow \quad \forall n : oc(n,l_1) = oc(n,l_2)$$

is an inductive theorem of E with E consisting of the defining equations for all function symbols involved. Note that (L) is non-trivial, too. Moreover, it does not have equational form.[11] From (L) it can easily be deduced now that $perm$ (considered as a binary relation) is reflexive, symmetric and transitive.

[10] Additional auxiliary functions used here are $=_{nat}$ for the test of equality on natural numbers as well as if_{list} which stands for the ternary if-then-else operation with boolean-valued condition and list-valued alternatives.

[11] For a proof of it see [Gra90c].

Comparing the definitions (IIa) and (IIb) concerning proof-technical aspects the first one seems to be better for such cases where we know something about the first elements of the two lists to be compared. This is the case for instance for *insertion_sort* and *min_sort*. Definition (IIb) however is better suited for verifying *quick_sort* and *merge_sort* (see below), where the first element of a sorted list is not directly visible from the definition of sorting.

Finally, let us mention that the property of permutative equivalence could also be specified by explicitly computing the multiset of list elements and defining inclusion and equality for multisets (of elements).

Let us now consider for illustration *merge_sort*. The idea of sorting by merging is to partition every list with more than one element into two parts containing approximately the same number of elements. For that purpose we use two auxiliary functions $split1, split2$: $list \to list$. $Split1$ collects the elements occurring at odd positions and $split2$ those which occur at even positions. We shall make use of (Ia) and exploit (Ic) for an appropriate decomposition of the verification problem for merge_sort. The algorithm and the correctness predicates for sorting by merging, modelled by *merge_sort*: $list \to list$, are specified as follows:

$$\begin{array}{lllll}
(1) & merge_sort(e) & = & e \\
(2) & merge_sort(c(n,e)) & = & c(n,e) \\
(3) & merge_sort(c(m,c(n,l))) & = & merge(\ merge_sort(c(m,split1(l))), \\
 & & & \qquad merge_sort(c(n,split2(l)))\ ) \\
(4) & merge(e,l) & = & l \\
(5) & merge(l,e) & = & l \\
(6) & merge(c(m,l_1),c(n,l_2)) & = & if_{list}(m \leq n,\ \ c(m,merge(l_1,c(n,l_2))), \\
 & & & \qquad\qquad c(n,merge(c(m,l_1),l_2))\ ) \\
(7) & split1(e) & = & e \\
(8) & split1(c(m,e)) & = & c(m,e) \\
(9) & split1(c(m,c(n,l))) & = & c(m,split1(l)) \\
(10) & split2(e) & = & e \\
(11) & split2(c(m,e)) & = & e \\
(12) & split2(c(m,c(n,l))) & = & c(n,split2(l)) \\
(13) & ord(e) & = & true \\
(14) & ord(c(m,l)) & = & m \leq_{nl} l \wedge ord(l) \\
(15) & m \leq_{nl} e & = & true \\
(16) & m \leq_{nl} c(n,l) & = & m \leq n \wedge m \leq_{nl} l \\
(17) & oc(m,e) & = & 0 \\
(18) & oc(m,c(n,l)) & = & oc(m,l) + if_{nat}(m =_{nat} n, s(0), 0)\ \ .
\end{array}$$

The correctness properties to be established are

$$\begin{array}{llll}
(L_1) & ord(merge_sort(l)) & = & true\ , \\
(L_2) & oc(n,merge_sort(l)) & = & oc(n,l)\ .
\end{array}$$

For verifying (L_1) we have to exploit the fact that the function merge is indeed defined such that merging two ordered lists yields an ordered list which – again in slightly generalized form – is caught by the auxiliary lemma

$$(L_3) \quad ord(merge(l_1,l_2)) \quad = \quad ord(l_1) \wedge ord(l_2)\ \ .$$

This property in turn needs some more subsidiary lemmas to be proved. (L_3) corresponds closely to the intuition behind sorting by merging, namely that merging two ordered lists again produces an ordered list. (L_3) indeed contains more information because it also states that whenever a list l which is constructed by merging l_1 and l_2 is not ordered then l_1 or l_2 is not ordered, either. For successfully verifying (L_3) we additionally have to exploit the second correctness property, namely permutative equivalence of a list and its sorted version. Moreover, we must use the knowledge that the result of comparisons like $m \leq_{nl} l$ does not depend on the order of the elements in l. Note that this knowledge is implicitly visible within the definitions of $\leq$ and $\leq_{nl}$, due to the AC-property of boolean conjunction. The combination of these issues clearly motivates and justifies the introduction of subsidiary lemmas like

$$m \leq_{nl} merge\_sort(l, l') \quad = \quad m \leq_{nl} l \wedge m \leq_{nl} l' \ .$$

The situation concerning the verification of the permutative equivalence property is quite similar. In an analogous style most auxiliary lemmas used can be explained and motivated. In particular, the design decision to model permutative equivalence via a counting function oc for element occurrences is closely related to the nature of the problem to be solved. This connection is due to the presence of the AC-operator $+$ within the definition of oc which corresponds to the fact that the order of elements is irrelevant when counting occurrences.

The proof of (L_2) requires two more natural lemmas, namely the decomposition property

$$(L_4) \quad oc(n, merge(l_1, l_2)) \quad = \quad oc(n, l_1) + oc(n, l_2)$$

and the combination property

$$(L_5) \quad oc(n, split1(l)) + oc(n, split2(l)) \quad = \quad oc(n, l) \ .$$

In fact, what has been presented above is the top-level reasoning of an intelligent specification and proof engineering process. Many fruitless efforts and impasses have not been mentioned.

The problems that had to be tackled are essentially twofold. On the one hand the specification of the involved algorithms as well as of their corresponding correctness properties has to be carefully designed. Moreover, the resulting proof problems have to be adequately structured and prepared. On the other hand the actual mechanically supported proof process for specific conjectures may be technically quite challenging, even in the case that the available inductive knowledge (auxiliary lemmas) in principle suffices. This problem is due to various proof-technical degrees of freedom of the underlying proof method discussed below.

Summarizing we can say the following: Whenever we want to prove some property of an algorithm whose specification involves supplementary functions we usually have to expolit auxiliary knowledge about these underlying functions. This becomes true in an even stronger sense when we proceed mainly in a top-down design and verification style.[12] Then the supplementary functions are constructed such that certain intended properties

[12] In practice a purely top-down approach is is not realistic because we mostly need some basic knowledge, e.g. about the underlying basic data types.

are indeed satisfied. Of course these properties have to be kept in mind and may be used when trying to establish some verification condition of the main algorithm.

Whether an automation of such meta-level reasoning steps is possible (at least partially) is one of the most challenging AI problems in inductive theorem proving. The preceding discussion should render it obvious that a very demanding kind of *specifiation and proof engineering* seems indeed to be necessary for preprocessing, structuring and preparing inductive proof problems in such a way that they are "tractable by machine".

4.2 Technical Aspects of Intelligent Proof Engineering

Even in the case that a specification and verification problem has been carefully modelled and structured the remaining proof tasks may be quite challenging. In fact, some proof-technical aspects and details have turned out to be crucial for practically successful verification with a system like $\mathcal{UNICOM}$. These issues are discussed now.

4.2.1 Improving Simplification by Using AC-Rewriting

Originally simplification in $\mathcal{UNICOM}$ was implemented essentially by ordinary rewriting. As a consequence, the mechanical proofs of even simple inductive lemmas were often quite tedious and complicated, if not impossible, in particular for lemmas involving boolean and natural number reasoning. This problem was mainly due to the fact that the AC-properties of operators like $\wedge$, $\vee$ and $+$ could not be mechanically exploited for simplification. The integration of AC-rewriting into $\mathcal{UNICOM}$ did indeed solve this problem.[13] An important practical aspect when incorporating AC-rewriting into the simplication mechanism concerns so-called extension rules which are necessary to obtain an AC-simplification mechanism which is sufficiently general for practical applications.

4.2.2 Non-Reducing Proof Simplification Steps

According to the main theorem underlying completion based ITP (cf. section 2) the essence of the method consists in assuring (ground) simplifiability of critical proofs obtained by overlapping definition rules into conjectures. The easiest way to establish this simplifiability property clearly is to find rewrite proofs for the corresponding critical pairs by means of ordinary reduction to normal forms that coincide. A significant practical improvement resulting in an increase of simplification power is obtained by using AC-rewriting which can still be performed automatically. But even this enhanced technique does not solve the problem in all cases. The reason is that it is sometimes necessary to transform an intermediate result by applying a rewrite rule in the inverse direction in order to enable a next reducing simplification step. Currently such a kind of reasoning with non-reducing simplification steps cannot be performed mechanically by $\mathcal{UNICOM}$. In fact the difficulties involved are quite obvious. From a theoretical point of view it has to be verified that the overall complexity of the constructed proof is smaller than that of the critical peak itself. And practically, the question arises when and how such non-reducing guessing

[13] In fact, our practical experience with $\mathcal{UNICOM}$ based on numerous non-trivial examples has shown that in most cases more than 90 % of run time is used for reduction (including matching which is the central operation performed during reduction). This was the main motivation for designing and implementing a new and efficient AC-matching algorithm (cf. [GD88]) which is based on constraint propagation techniques.

steps should be performed. To be precise it must be noted that the problem of verifying proof complexities already occurs when using AC-rewriting. Here we need AC-compatible reduction orderings[14] for ensuring theoretical correctness. Such orderings are currently not available in $\mathcal{UNICOM}$. Nevertheless by inspecting the proofs produced it can often be verified by hand that the relevant conditions are indeed satisfied.

Summarizing one can say that in order to obtain simplified proofs (for critical instances of the main conjecture derived by the critical pair mechanism) the simple variant of allowing only reducing, i.e. complexity decreasing, computation steps is a sound one. But as soon as some steps are allowed to be non-reducing or even complexity increasing, the problem becomes much more difficult and requires intelligent heuristics because otherwise the search space grows dramatically. A careful analysis of the ordering restrictions easily shows that the only crucial steps for which a decreasing complexity has to be verified are the "inductive" ones, i.e. those where already processed conjectures are applied. This fact corresponds nicely to the explicit ITP approach where (sound) induction schemas and appropiately instantiated versions (induction bases and induction steps) of the concrete conjectures are computed and tried to be verified. Within this explicit induction framework the ordering conditions are in a sense compiled into sound induction schemas which also provide the available induction hypotheses *explicitly*.[15] Verifying the base and induction steps is usually done in a purely deductive fashion[16] and no ordering restrictions have to be obeyed any more.

4.2.3 Exploiting Simplification Indeterminism

Another practically very important and subtle point concerning completion based inductive theorem proving is the question of how to control the simplification process. Sometimes it may be the case that some conjecture can in principle be simplified to a trivial equation with an appropriate sequence of simplification steps (using definition rules, lemmas and already processed conjectures). But the given (explicit or implicit) simplification strategy returns with a non-trivial normalized conjecture. This phenomenon is due to the fact that the rules which are used for simplification do not constitute a confluent system in general. Whereas the confluence property usually holds for the set of definition rules (in most cases it is not only ground confluent but even confluent) it is in general violated if additional inductive lemmas and conjectures are taken into account. Hence it is very important to perform the simplification steps in an intelligent goal-directed way. One general heuristic which has turned out to be very useful in many examples and which is implemented in $\mathcal{UNICOM}$ roughly proceeds as follows. The rules available for simplification are partitioned into definition rules, inductive lemmas and the actual conjecture. The

[14] Cf. e.g. [Ste90].

[15] Note that this fact is closely related to a subtle, but important difference between explicit and implicit ITP. In the latter approach induction hypotheses are not provided explicitly but are available (after the corresponding conjecture has been processed, i.e. "covered") in a more general (compared to classical explicit ITP) *generic* form. This means that they may be applied as rewrite rules (or equations), i.e. in arbitrarily instantiated version, provided that the required ordering restrictions are satisfied. Note moreover, that this phenomenon is the main technical reason why we prefer the terms *explicit* and *implicit* induction. Clearly implicit ITP also performs an induction – and hence is not *inductionless* (!) – but w.r.t. a different well-founded ordering. This aspect has been clarified in [Red90] and [Gra90a].

[16] Of course, one may again use inductive reasoning in this deductive proof process if it turns out that there is no hope of success. From a more abstract point of view this case corresponds to nested induction.

highest priority for simplification is assigned to the conjecture itself which corresponds to the intuition that an induction hypothesis should be applied as early and as often as possible. Next it is checked whether a lemma can be used for simplification. If this is not possible, either, then it is attempted to apply a definition rule for one of the function symbols involved. Moreover, the available lemmas are ordered decreasingly with respect to their estimated importance. Of course, this heuristic which in a sense is still rather crude, might be refined in various ways, e.g. by allowing nested priorities or, more generally, by means of a dynamic and context-dependent priority mechanism. But it is obvious that it would become much more complicated and moreover, it is by no means clear how to do this in a non-trivial and adequate fashion.

4.2.4 How to Choose Inductively Complete Positions

For a given inductive conjecture $\mathcal{UNICOM}$ computes all positions which are inductively complete, i.e. suffice for constructing critical pairs. Depending on a system parameter either all corresponding proof attempts are then automatically developed in parallel or the user is asked to choose one for continuing. Whereas the fully automatic variant proceeding in parallel and performing conceptually independent proof attempts is theoretically quite elegant, it usually causes severe efficiency problems in practice. In many cases certain inductively complete positions are completely inappropriate for the intended goal whereas other choices seem to be more promising.

Within the framework of classical inductive theorem proving a lot of work has been devoted to recursion analysis for finding appropriate induction terms and induction schemas for a given conjecture (cf. [BM79], [Bun88]). We are convinced that such a sophisticated analysis can be carried over to the completion based approach and provide useful heuristics for supporting or even automating crucial steps like an intelligent selection process for inductively complete positions. But the details concerning this transfer of results and techniques from classical inductive theorem proving into our context still have to be worked out.[17]

4.2.5 Handling Conditional Reasoning

Within the presented example of sorting by merging conditional reasoning is clearly important. Our decision to model conditional properties and actions by encoding via ternary *if-then-else* operations was mainly motivated by pragmatic reasons. First of all there still exist various severe theoretical and practical problems concerning rewriting and completion techniques for proper conditional systems. And secondly, we actually wanted to find out what can be achieved within the purely equational approach when using a powerful implementation incorporating various refinements and optimizations of the basic method. And indeed, by making extensive use of numerous basic schematic properties of *if-then-else* operations encouraging experimental results have been obtained. Nevertheless there seems to be a growing consensus nowadays that for many problems a properly conditional approach separating the condition from the conclusion part would be more adequate and

[17] In order to gain more insight into the problem and practical experience we are currently implementing (and integrating into $\mathcal{UNICOM}$) a first prototype version for handling this problem which is partially based on the analysis described in [BM79] for the explicit ITP approach.

natural. Further research is needed to developed a theoretically well-founded and practically applicable extension of the underlying rewriting, simplification and completion techniques for properly conditional systems which is specialized to inductive theorem proving. Some encouraging progress along this line of research has already been achieved, e.g. in [Gan87], [KR90], [BG90], [Wir91], [WG92] and [GS92]. But successful and widely applicable running systems are still lacking.

5 Problems and Perspectives

In the following we shall summarize some of those problems arising in (rewriting and completion based) ITP which seem to be the most important ones for future work, both from a theoretical and a practical point of view. Moreover, we discuss existing approaches or at least ideas for tackling these problems. The main problems to be solved include among others the following topics:[18]

(1) **Non-Terminating Base Systems and Non-Free Constructors**
The theoretical foundations of the approach presented in section 2 rely on the assumption that the base system with respect to which one wants to prove inductive theorems is terminating (and ground confluent). This assumption is indeed very often satisfied in practical examples. But there are also interesting applications where this property does not necessarily hold, e.g. for some natural specification of sets. As already mentioned one possibility in such cases often is to reason modulo the non-terminating part of the specification (e.g. AC-axioms). First encouraging results in this direction have been obtained in [JK86]. One severe problem here is the fact that the accordingly generalized property of inductive reducibility modulo some set of equations becomes undecidable in general. And even for decidable subcases the efficiency of decision algorithms is still a problem. Concerning decidability and efficiency questions the problem of non-terminating base systems also exists for classical explicit ITP methods (when trying to compute and verify soundness of induction schemes).

Another very difficult problem arising in both implicit and explicit ITP approaches has to do with the type of constructors used. In almost all running systems the constructors are assumed to be free, i.e. there are no relations between constructor terms. Even stronger, a rather restrictive form of constructor discipline is usually required for defining new functions (cf. e.g. [BM79], [Gra89]). The main reason for such strong (syntactical) requirements are essentially twofold. Firstly, the problem of verifying well-definedness (in particular consistency, termination and completeness) of newly introduced defined functions becomes much simpler.[19] Secondly, the computation of sound and operationally feasible induction schemas (in explicit ITP) or inductively complete positions (in implicit ITP) is greatly simplified, too.

[18] We omit here the more technical but nevertheless important problem of when and how additional inductive knowledge should be (automatically) generated. For more details concerning this problem of lemma generation and generalization techniques the interested reader is referred to e.g. [BM79], [TJ89], [Lan89], [Gra89] and [Ave91].

[19] But note that even under such strong restrictions as mentioned these problems are still very hard or even undecidable.

(2) **Incorporating Domain Specific Knowledge and Methods**
In some applications the general approach of pure equational reasoning may be too crude. For instance, boolean or logical reasoning should not be performed by means of equational encoding but by appropiate built-in mechanisms or by using a more general formal language like first order logic with equality (cf. (3) below). Moreover, it may be the case that depending on the domain of interest and the type of application additional domain specific knowledge and methods, e.g. special decision procedures, are available. Although this specific knowledge and the corresponding methods can in principle be formulated equationally this may be inadequate or inefficient. Consider for instance polynomial arithmetic or systems of (linear) inequalities. Various specific techniques and algorithms for these fields are well-known and should be used. Thus, a combination of the general equational approach with other techniques better suited for particular cases seems to be useful and promising.[20] For the theoretical foundation and practical realization of this combination and integration of different methods a lot of work still has to be done.

(3) **Enriching Expressive Power**
A natural formalization of many specification and verification problems often requires more than pure equations. In most cases this is not a principal problem but rather a question of adequacy. For example one may consider order-sorted or conditional equation systems or even full first-order logic for that purpose. Although some substantial progress within these fields has been achieved in the last years (cf. e.g. [BG91], [GS92], [Wir91], [WG92]) many theoretical and practical problems remain open, in particular concerning verification tasks, e.g. inductive theorem proving. Another kind of enriching expressive power consists in allowing parameterized specifications which should permit to perform a kind of parametrized verification. Thus one might be able to prove correctness of generic specifications and algorithms provided the actual parameters satisfy certain restrictions. Some considerable progress in this direction has been achieved but mainly concerning the specification and programming language aspects and not so much with respect to verification methods (but see [Kir91], [Bec92]). This is reflected in practice by the lack of corresponding running systems with powerful and (at least partially) automated ITP components. A severe obstacle may also consist in the fact that a precise theoretical understanding of the kind of model semantics underlying the intended notion of inductive proof is not yet clear enough (cf. e.g. [Zha88], [KM87]).

(4) **Proof Organisation and Control**
From a practical point of view we think that the most important problem consists in improving the whole **specification and proof engineering** process. The theoretical foundations, the technical possibilities (e.g. inference rules) and the degrees of freedom in rewriting and completion based ITP have been well-understood and clearly worked out. The remaining practical problem is to exploit this knowledge for constructing powerful, flexible and widely applicable software environments for specification and verification. We consider $\mathcal{UNICOM}$ as well as some other systems as a first promising step towards this challenging goal. Possible and necessary

[20] See e.g. [BM88], where the authors discuss the integration of a decision procedure for a fragment of linear arithmetic into their ITP system.

improvements include better means for organizing, structuring and controlling the whole proof process. For instance meta-levels and concepts for formalizing strategies and heuristics are necessary (cf. e.g. [Bun88], [BvHSI90], [Hut90], [Nip89]). This is a very challenging task, since looking at the literature, a widely shared consensus about an adequate and sufficiently precise terminology in this field currently does not exist. Notions like heuristics, strategies, tactics, plans and methods are used in a rather informal and vague style. But probably this is not very surprising because the problems involved are very hard. Summarizing, we think that a lot of conceptual research and clarification concerning the adequate notions and levels of reasoning still has to be done here.

(5) **System Support**
Further important practical and lower-level aspects concern refined techniques for constructing, managing and using large knowledge-bases. Moreover, a comfortable user-interface and refined techniques for extracting and representing human- oriented relevant information would considerably support the overall engineering process.[21] In particular, this includes a well-designed control and interaction language together with a flexible and realistic conceptual model of interaction between a more or less competent human user (as specification and proof engineer) and the machine assistant providing as much mechanical support as possible.

The general concept underlying such a system should ideally take into account a couple of facilities/features which – for human oriented reasoning – are very useful, in particular:

- Logging features: e.g. a flexible trace facility, script files for batch mode and reproducing proofs, filters for specifying how detailed a log should be.
- Navigation facilities: Using an abstract proof search tree as underlying abstract data structure, means for backtracking, undo facilities, arbitrary navigation within proof search trees and focussing mechanisms should be available in order to facilitate proof management. In particular, the user should be enabled to obtain complete control over any single step if (s)he really wants to have it.[22]
- Relative proofs: Adequate abstraction and structuring mechanisms for verification tasks should be provided allowing e.g. high-level reasoning w.r.t. (not yet proved) assumptions. Such features constitute a necessary step in the direction of intelligent knowledge-based proof planning techniques. Of course, any step in this direction would entail a couple of theoretical and practical complications, e.g. a much more complicated data and knowledge-base management and the related question of how and when to guarantee or obtain consistency again.
- Comfortable user-interface: Clearly better means for communication and I/O including e.g. graphical representations as well as other useful interface features should be provided.

[21] For instance, one might think of designing and using specially adapted forms of hypertext systems as it has been tried e.g. in [KMN91].

[22] This would mean that the corresponding ITP system exhibits a (more or less) complete transparency of its internal behaviour.

In recently implemented modifications of $\mathcal{UNICOM}$ some of these aspects, in particular concerning better log and trace features, navigation facilities as well as user-interface improvements, have been taken into account. But a lot of conceptual and practical problems remain open.

6 Conclusion

We have given a brief summary of implicit (rewriting and completion based) inductive theorem proving. The central ideas and some of the most important operationalization issues have been pointed out and discussed. Moreover, a brief comparison of explicit and implicit ITP techniques has been presented. In particular, some arguments against and advantages of the implicit ITP approach have been clarified.

Concerning practical applications central conceptual and proof-technical aspects have been pointed out and exemplified. Moreover, the viewpoint of inductive theorem proving as a challenging specification and proof engineering task has been developed and discussed. Resulting and remaining problems and some perspectives which we think are the most important and promising ones have been sketched.

Clearly, the nature of this discussion has been rather abstract. But nevertheless we hope that these considerations may contribute to a better understanding of what future generation systems for inductive theorem proving should and could look like.

Acknowledgement: I would like to thank Ulrich Kühler for useful hints and detailed criticisms on a draft version of this paper.

References

[Ave91] J. Avenhaus. Proving equational and inductive theorems by completion and embedding techniques. In R.V. Book, editor, *Proc. of the 4th Int. Conf. on Rewriting Techniques and Applications*, volume 488 of *Lecture Notes in Computer Science*, pages 361–373. Springer Verlag, 1991.

[Bac87] L. Bachmair. *Proof Methods for Equational Theories.* PhD thesis, University of Illinois, Urbana Champaign, 1987.

[Bac88] L. Bachmair. Proof by consistency in equational theories. In *Proc. 3rd IEEE Symposium on Logic in Computer Science*, pages 228–233, 1988.

[Bec92] K. Becker. Inductive proofs in specifications parameterized by a built-in theory. SEKI Report SR-92-02, Dept. of Comp. Science, Univ. of Kaiserslautern, 1992.

[BG90] L. Bachmair and H. Ganzinger. On restrictions of ordered paramodulation with simplification. In M. Stickel, editor, *Proc. of the 1oth Int. Conf. on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 427–441. Springer, 1990.

[BG91] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. Technical Report MPI-I-91-208, Max-Planck-Institut für Informatik, Saarbrücken, 1991.

[BM79] R.S. Boyer and J.S. Moore. *A Computational Logic*. Academic Press, 1979.

[BM88] 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.

[Bun88] A. Bundy. The use of explicit proof plans to guide inductive proofs. In E. Lusk and R. Overbeek, editors, *Proc. of the 9th Int. Conf. on Automated Deduction*, volume 310 of *Lecture Notes in Computer Science*, pages 111–120. Springer, 1988.

[BvHSI90] A. Bundy, F. van Haremelen, A. Smaill, and A. Ireland. Extensions to the rippling-out tactic for guiding inductive proofs. In M.E. Stickel, editor, *Proc. of the 1oth Int. Conf. on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 132–146. Springer-Verlag, 1990.

[Fri86] L. Fribourg. A strong restriction of the inductive completion procedure. In E. Kott, editor, *Proc. of the 13th Int. Conf. on Automata, Languages and Programming*, volume 226 of *Lecture Notes in Computer Science*, pages 105–116. Springer, 1986.

[Gan87] H. Ganzinger. Ground term confluence in parametric conditional equational specifications. In F.J. Brandenburg, G. Vidal-Naquet, and M. Wirsing, editors, *Proc. of the 4th Int. Symp. on Theoretical Aspects of Computer Science*, volume 247 of *Lecture Notes in Computer Science*, pages 286–298. Springer, 1987. LNCS 247.

[GD88] B. Gramlich and J. Denzinger. Efficient AC-matching using constraint propagation. SEKI Report SR-88-15, Dept. of Comp. Science, Univ. of Kaiserslautern, 1988.

[GG88] S.J. Garland and J.V. Guttag. Inductive methods for reasoning about abstract data types. In *Proc. of ACM Symp. on Principles of Programming Languages*, pages 219–228. acm press, 1988.

[GL91] B. Gramlich and W. Lindner. A guide to UNICOM, an inductive theorem prover based on rewriting and completion techniques. SEKI Report SR-91-17, Dept. of Comp. Science, Univ. of Kaiserslautern, 1991.

[Göb87] R. Göbel. Ground confluence. In P. Lescanne, editor, *Proc. of the 2nd Int. Conf. on Rewriting Techniques and Applications*, volume 256 of *Lecture Notes in Computer Science*, pages 156–167. Springer, 1987.

[Gog80] J.A. Goguen. How to prove algebraic inductive hypotheses without induction. In W. Bibel and R. Kowalski, editors, *Proc. of the 5th Int. Conf. on Automated Deduction*, volume 87 of *Lecture Notes in Computer Science*, pages 356–373, 1980.

[Gra89] B. Gramlich. Inductive theorem proving using refined unfailing completion techniques. SEKI Report SR-89-14, Dept. of Comp. Science, Univ. of Kaiserslautern, 1989.

[Gra90a] B. Gramlich. Completion based inductive theorem proving: A case study in verifying sorting algorithms. SEKI Report SR-90-04, Dept. of Comp. Science, Univ. of Kaiserslautern, 1990.

[Gra90b] B. Gramlich. Completion based inductive theorem proving: An abstract framework and its applications. In L.C. Aiello, editor, *Proc. of the 9th European Conf. on Artificial Intelligence*, pages 314–319. Pitman Publishing, London, 1990.

[Gra90c] B. Gramlich. Unicom: A refined completion based inductive theorem prover. In M.E. Stickel, editor, *Proc. of the 1oth Int. Conf. on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 655–656. Springer-Verlag, 1990.

[GS92] 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.

[HH80] G. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. In *Proc. of the 21st Conf. on Foundations of Computer Science*, pages 96–107, 1980. also in JCSS 25(2), pp. 239-266, 1982.

[HO80] G. Huet and D.C. Oppen. Equations and rewrite rules: A survey. In Ronald V. Book, editor, *Formal Languages, Perspectives And Open Problems*, pages 349–405. Academic Press, 1980.

[Hut90] D. Hutter. Guiding induction proofs. In M.E. Stickel, editor, *Proc. of the 1oth Int. Conf. on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 147–161. Springer-Verlag, 1990.

[JK86] J.-P. Jouannaud and E. Kounalis. Automatic proofs by induction in equational theories without constructors. In *Proc. Symposium on Logic in Computer Science*, pages 358–366. IEEE, 1986. also in Information and Computation, vol. 82(1), pp. 1-33, 1989.

[Kir91] H. Kirchner. Proofs in parameterized specifications. In R.V. Book, editor, *Proc. of the 4th Int. Conf. on Rewriting Techniques and Applications*, volume 488 of *Lecture Notes in Computer Science*, pages 174–187. Springer-Verlag, 1991.

[KM87] D. Kapur and D.R. Musser. Proof by consistency. *Artifial Intelligence*, 31:125–157, 1987.

[KMN91] D. Kapur, D.R. Musser, and X. Nie. The TECTON proof system and its programmed hypertext interface. draft version, 1991.

[KNZ86] D. Kapur, P. Narendran, and H. Zhang. Proof by induction using test sets. In *proc. 8th CADE*, volume 230 of *Lecture Notes in Computer Science*, pages 99–117. Springer, 1986. LNCS 230.

[KR90] E. Kounalis and M. Rusinowitch. Mechanizing inductive reasoning. In *Proc. 8th National Conference on Artificial Intelligence*, pages 240–245. American Association For Artificial Intelligence, MIT Press, 1990.

[Küc87] W. Küchlin. Inductive completion by ground proof transformation. In *Proc. of Coll. on the Resolution of Equations in Algebraic Structures*, 1987.

[Lan80] D. Lankford. Some remarks on inductionless induction. Technical Report MTP-11, Math. Dept., Louisiana Tech. Univ., Ruston, 1980.

[Lan81] D. Lankford. A simple explanation of inductionless induction. Technical report, Math. Dept., Louisiana Tech. University, Ruston, 1981.

[Lan89] S. Lange. Towards a set of inference rules for solving divergence in Knuth-Bendix completion. In K.P. Jantke, editor, *Proc. of the 2nd Int. Workshop on Analogical and Inductive Inference*, volume 397 of *Lecture Notes in Artificial Intelligence*, pages 304–316, Reinhardsbrunn Castle, GDR, 1989. Springer-Verlag.

[Mus80] D.R. Musser. On proving properties of abstract data types. In *Proc. of the 7th ACM Symposium on Principles of Programming Languages*, pages 154–162, 1980.

[Nip89] T. Nipkow. Equational reasoning in ISABELLE. *Science of Computer Programming*, 12:123–149, 1989.

[Pau84] E. Paul. Proof by induction in equational theories with relations between constructors. In B. Courcelle, editor, *Proc. of the 13th Coll. on Trees in Algebras and Programming.* Cambridge University Press, 1984.

[Pla85] D. Plaisted. Semantic confluence tests and completion methods. *Information and Control*, 65:182–215, 1985.

[Red90] U.S. Reddy. Term rewriting induction. In M.E. Stickel, editor, *Proc. of the 1oth Int. Conf. on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 162–177. Springer, 1990.

[Ste90] J. Steinbach. Improving associative path orderings. In M.E. Stickel, editor, *Proc. of the 1oth Int. Conf. on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 411–425. Springer-Verlag, 1990.

[TJ89] M. Thomas and K.P. Jantke. Inductive inference for solving divergence in Knuth-Bendix completion. In K.P. Jantke, editor, *Proc. of the 2nd Int. Workshop on Analogical and Inductive Inference*, volume 397 of *Lecture Notes in Artificial Intelligence*, pages 288–303, Reinhardsbrunn Castle, GDR, 1989. Springer-Verlag.

[Wal92] C. Walther. Computing induction axioms. In *Proc. of the Conference on Logic Programming and Automated Reasoning, St. Petersburg*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1992. to appear.

[WG92] C.-P. Wirth and B. Gramlich. A constructor-based approach for positive/negative conditional equational specifications. SEKI Report SR-92-10, Dept. of Comp. Science, Univ. of Kaiserslautern, 1992.

[Wir91] C.-P. Wirth. Inductive theorem proving in theories specified by positive / negative conditional equations. Master's thesis, Dept. of Comp. Science, Univ. of Kaiserslautern, 1991.

[Zha88] H. Zhang. *Reduction, Superposition and Induction: Automated Reasoning in an Equational Logic*. PhD thesis, Rensselaer Polytech. Inst., Dept. of Comp. Sci., Troy, NY, 1988.

[ZKK88] H. Zhang, D. Kapur, and M.S. Krishnamoorthy. A mechanizable induction principle for equational specifications. In E. Lusk and R. Overbeek, editors, *Proc. of the 9th Int. Conf. on Automated Deduction*, volume 310 of *Lecture Notes in Computer Science*, pages 162–181. Springer, 1988.