Structured Incremental Proof Planning

Stefan Gerberding and Brigitte Pientka

Technical University of Darmstadt
{stefan,pientka}@inferenzsysteme.informatik.th-darmstadt.de

Abstract. We extend our framework of *incremental proof planning*. By employing nested sets of meta-rules the formulation of strategies may be structured. By switching to another meta-rule set the planner can adjust to a new situation within the proof. The new meta-rule set represents a more specialized strategy better suited for the current situation. We define the semantics of our framework by an inference system.

1 Introduction

Automated theorem provers have to cope with tremendous search spaces. Lots of heuristics have been developed to guide the proofs and to prune the search space. We will focus on the domain of automated induction theorem proving, but our method is not limited to this domain. An automated induction theorem prover has various techniques at its disposal. Some are general purpose calculi. Others are specially tailored for certain situations within a proof: Rippling for the rewrite of the conclusion in the step-cases of an induction to make the induction hypothesis applicable [3], generalization, fertilization [1], the computation of induction axioms to choose an appropriate induction relation [5], or symbolic evaluation, case-splits, etc. Those techniques are modeled as *tactics* [4].

Different approaches are known to control the application of tactics. Some systems apply certain tactics in a *fixed order* (waterfall), e.g. NQTHM [1]. Any heuristics to control the application of the different inference mechanisms are hard-coded into the order of application. Other systems leave the control mostly to the user. This is either done implicitly by inserting hints or appropriate lemmata into the problem description or the user explicitly calls the desired tactic.

To overcome the control problem for tactics Bundy proposes the use of *proof plans* [2]. Proof plans can be generated automatically if *meta-reasoning* about tactics is possible. In Bundy's framework the proof plan is computed prior to any proof attempt, i.e. before executing any tactic. The result of the tactics are estimated or simulated for the computation of the new subgoals during proof planning. No information about the history of the proof and previous steps in the proof plan are available to the planner. The planner is *uninformed* apart from the tactics' pre- and postconditions; no further knowledge about the tactics and the domain of application is available. To specify strategy knowledge, the preconditions must be modified to reflect the new strategy. The proof plan is passed to the proof checker to verify that its execution yields a valid proof.

2 Incremental Proof Planning

These drawbacks are obsolete for *incremental proof planning* [6]. The knowledge about the domain of application, i.e. the strategy, and about tactics is declaratively represented by *meta-rules*: triples consisting of a precondition, an action, and a persistence condition. The knowledge about the partial proof and the proof plan is located in the *proof-tree*. Meta-rules can be structured in *meta-rule sets* providing a further level of abstraction and structuring. E.g. techniques for certain proof situations may be collected in some meta-rule sets. If the planner switches to another meta-rule set (as part of an action), it adopts the strategy to the current proof situation. Since the complete strategy information used by the proof planner is declaratively represented by the sets of meta-rules, the user may easily modify the strategy by supplying new sets of meta-rules.

In §2.1 we will briefly explain the framework of *incremental proof planning* introduced in [6]. We will extend our presentation by specifying a set of inference rules defining the semantics of incremental proof planning. In the remainder of this paper we will then focus on incremental proof planning with *meta-rule sets*. To demonstrate the power of the extended framework, we will model a strategy for induction theorem proving similar to lazy induction [11] by meta-rule sets.

2.1 Overview

The partial proof and the proof plan is represented by the *proof-tree*. All information in the proof-tree is available to the planner. A proof-tree is a labeled and-or-tree. The nodes' labels are closed first-order formulae. The root is labeled by the conjecture. A leaf is (*negatively*) *solved* iff its label is the formula true (false). An and-node resp. or-node is (negatively) solved iff (any) each successor resp. (each) any successor is (negatively) solved. A proof and a successful proof plan has been found for the conjecture iff the root node is solved. If the root node is negatively solved the conjecture is disproved.

Apart from the goal the label of a node N contains the state σ of N (among other information). σ indicates whether the goal G is positively or negatively solved, open, or unprovable by the system, etc. The states of the leaves are propagated through the proof-tree. The edges of a proof-tree are labeled by tactic names, indicating which tactic computed the child nodes, and additional attributes returned by the tactics (e.g. marks for the base and step-case of an induction). Furthermore, the tactics' results determine whether the node corresponding to the current goal becomes an and-node or an or-node.

We use the notation "$\wedge$" for object-level as well as meta-level conjunction; the semantic of the latter is defined by (1) in §2.2. $\equiv$ denotes syntactical equality. We will write lists $o_1, \ldots, o_n$ of variables, terms, etc. as o^*.

The planner's strategy is declaratively implemented by meta-rules. A meta-rule π has the syntactical form $C \Rightarrow A \looparrowleft P$ where C is the *precondition*, A is the *action*, and P is the *persistence condition*. The precondition C and the persistence condition P are conjunctions of atoms: $C_1(x_1^*) \wedge \ldots \wedge C_n(x_n^*)$. Each atom references a *meta-predicate* on proof-trees. x_i^* is a list of *meta-variables*, which are bound when evaluating the meta-predicate wrt. the proof-tree. The precondition-action pair $C \Rightarrow A$ is similar to a rule of an expert system.

The semantics of a meta-predicate $C_i(x_i^*)$ is given by a PQL-expression. PQL is a *proof-tree query language* developed for our framework to define all bindings to *meta-variables* x_i^*, s.t. $C_i(x_i^*)$ is satisfied for the given proof-tree. A meta-variable may be bound to positions, symbols, terms, formulae, proof-tree nodes, etc. depending on the type[1] of the formal parameter of the corresponding meta-predicate. Using PQL the user is able to specify appropriate meta-predicates, which are needed to express the desired strategy. For details we refer to [6, 9].

The condition $C \equiv C_1(x_1^*) \wedge \ldots \wedge C_n(x_n^*)$ is *satisfiable* (for the given proof-tree representing the current situation) iff a *consistent binding* for the meta-variables $x_1^*, \ldots, x_n^*$ exists, s.t. all meta-predicates are satisfied. A meta-variable binding is *consistent* iff for every x, y identical bindings[2] are implied by $x \equiv y$.

If the precondition C is satisfiable, the meta-rule is applicable and the planner can select the action A for execution. In the most simple case A is a single tactic $T(x^*)$. Since C is satisfiable, bindings for the meta-variables x^* exist. Such a binding is passed as parameter list to T. In general A is a sequence of tactics and calls to meta-rule sets (cf. §2.3) interleaved by optional continuation conditions: $T_1(x_1^*) \looparrowleft \mathrm{CC}_1(y_1^*), T_2(x_2^*) \looparrowleft \mathrm{CC}_2(y_2^*), \ldots, T_m(x_m^*)$: When T_1 completed successfully the satisfiability of the continuation condition CC_1 is checked and T_2 is executed when the condition is satisfied. An action fails explicitly if some T_i returns fail or if a continuation condition is not satisfied. In this case the (partly) executed action is retracted and a different binding for the same meta-rule or another meta-rule is considered. Information about the retracted actions may be stored in the proof-tree and utilized by meta-predicates and the planner.

After the completion of an action A the planner reasons about A's results, provided A did not fail explicitly and a persistence condition P is specified. If P is not satisfied for the proof-tree expanded according to A's results, then the action is retracted as for the explicit failure. Using persistence conditions we are able to identify goals not contributing to the proof.

2.2 Semantics

For the description of the formal semantics of incremental proof planning we need some notation: *Meta-variable bindings* are written like substitutions: v/t. We extend this notation to lists of meta-variables and terms: v^*/t^*. Set notation is used to denote the extension of a binding E: $E \cup v^*/t^*$. For a tactic T with formal parameters x^* and a meta-variable binding $E \ni x^*/t^*$, $T(E)$ denotes the instantiation of the formal parameters by t^*. $\hat{E}$ denotes a stack of meta-variable bindings. The push operation is written by concatenation: $\hat{E}E$. We use the same notation for stacks of meta-rule sets: $\hat{M}M$. Proof-tree expansion is denoted by $\circ$,

[1] For brevity we will omit any type information throughout this paper.

[2] This is a simplified description of consistency. Because of the various types a notion of equivalence is employed, which allows bindings of variables of different types to be consistent, e.g. a symbol s and a term whose leading symbol is s.

if cur is a proof-tree or a node cur $\circ G$ is the proof-tree obtained by expanding cur by the tree G. $P[\text{cur}]$ ambiguously identifies cur as some leaf of the tree P. We use sub-calculi $\vdash_c$ resp. $\vdash_a$ to model the evaluation of conditions resp. actions.

Conditions: The notion of satisfiability of a meta-predicate atom $\text{mp}(v^*)$ is defined by PQL [9], see also §2.1. The $\vdash_c$-calculus operates on triples consisting of a proof-tree, a meta-variable binding, and a condition. A c-derivation is a sequence of triples $\ldots, T_{i-1}, T_i, \ldots$, s.t. T_i is derived from T_{i-1} by an application of the inference rule (1). The corresponding inference relation is denoted by $\vdash_c$.

$$\frac{\langle P, E, \text{cond}\rangle}{\langle P, E \cup v^*/t^*, \text{cond} \wedge \text{mp}(v^*)\rangle} \quad \begin{array}{l}\text{if mp}(v^*) \text{ is satisfiable for } P,\ v^*/t^* \text{ is a} \\ \text{meta-variable binding s.t. mp}(t^*) \text{ is sat-} \\ \text{isfied for } P \text{, and } E,\ v^*/t^* \text{ are consistent.}\end{array} \tag{1}$$

Lemma 1. *Let P be a proof-tree, E, E' meta-variable bindings, and* cond *a conjunction of meta-predicate atoms. If $\langle P, E, \emptyset\rangle \vdash_c \langle P, E', \text{cond}\rangle$, then* cond *is satisfiable for P and E' defines the E-compatible meta-variable binding sufficient to satisfy* cond *for P.*

Actions: A meta-rule set induces an environment of meta-variable bindings, which is empty for the initial meta-rule set. Meta-rule sets are parameterized like tactics, cf. fig. 2 for an example. When a meta-rule set M is called its parameters define the environment E for M. The preconditions of meta-rules from M are evaluated extending the meta-variable bindings E. There are no restrictions for the hierarchy of calls of meta-rule sets.

Together with $\vdash$ defined later, the semantics of structuring meta-rule sets and the interaction of continuation conditions, persistence conditions, and sequencing is given by $\vdash_a$. The semantics of tactics is not modeled by $\vdash_a$. $\vdash_a$ operates on quintuples: The additional components are $\hat{M}M$ resp. $\hat{E}E$ representing the stack of meta-rule sets resp. the stack of parameter bindings induced by the meta-rule sets. M identifies the active meta-rule set (cf. (5)) and E defines M's parameter bindings. The third component represents the additional meta-variable bindings, s.t. the precondition guarding the action is satisfied.

An action consists of a single tactic or a call to a meta-rule set, or a sequence thereof, optionally interleaved by continuation conditions: $T_1(x_1^*) \looparrowleft \text{CC}_1(y_1^*), T_2(x_2^*) \looparrowleft \text{CC}_2(y_2^*), \ldots, T_m(x_m^*)$. The parameterization x_i^* of T_i must not depend on meta-variables y_j^*, because CC_j cannot alter the environment (cf. (3): the meta-variable bindings F' satisfying the condition do not alter the environment $E \cup F$). Continuation conditions and persistence conditions are conceptionally identical. So (3) defines both of them.

We assume that a tactic applied to some goal cur either explicitly fails by returning fail or yields a non-empty set G of subgoals, s.t. all of them (and-nodes) resp. one of them (or-nodes) is sufficient for cur. For simplicity we assume that any but the last tactic in a sequence computes only one new subgoal. We refer to §2.3 for the general case.

An a-derivation is a sequence of quintuples $\ldots, Q_{i-1}, Q_i, \ldots$, s.t. Q_i is derived from Q_{i-1} by an application of an inference rule (2)-(4). The corresponding inference relation is denoted by $\vdash_a$.

$$\frac{\langle \hat{M}, \hat{E}E, F, P[\mathrm{cur}], \mathrm{action}\rangle}{\langle \hat{M}, \hat{E}E, F, P[\mathrm{cur}\circ G], \mathrm{action}\ T\rangle} \quad \text{if cur} \neq \mathsf{fail} \text{ is a leaf and the application of tactic } T(F\cup E) \text{ to cur yields the subgoals } G \neq \mathsf{fail}. \tag{2}$$

$$\frac{\langle \hat{M}, \hat{E}E, F, P[\mathrm{cur}], \mathrm{action}\rangle}{\langle \hat{M}, \hat{E}E, F, P[\mathrm{cur}\circ G], \mathrm{action}\ T{\hookleftarrow}P\rangle} \quad \text{if } \langle P[\mathrm{cur}\circ G], F\cup E, \emptyset\rangle \vdash_c \langle P[\mathrm{cur}\circ G], F', \mathrm{pers}\rangle \text{ plus proviso (2).} \tag{3}$$

$$\frac{\langle \hat{M}, \hat{E}, F, P, \mathrm{action}\rangle}{\langle \hat{M}M, \hat{E}E, F, P, \mathrm{action}\ M\rangle} \quad \text{if } M \text{ is a meta-rule set whose parameters are bound according to } E \subseteq F. \tag{4}$$

Lemma 2. *Let* $P[\mathrm{cur}]$ *be a proof-tree with leaf* cur, $\hat{M}$, $\hat{M}'$ *stacks of sets of meta-rules,* $\hat{E}$, $\hat{E}'$ *stacks of bindings,* F *a binding, and* action *a sequence of tactics and calls to meta-rule sets. If* $\langle \hat{M}, \hat{E}, F, P[\mathrm{cur}], \emptyset\rangle \vdash_a \langle \hat{M}', \hat{E}', F, P[\mathrm{cur}\circ P'], \mathrm{action}\rangle$, *then no subaction of* action *failed, each continuation condition and the persistence condition are satisfied, and the proof-tree is expanded by the subtree* P'.

By an induction argument we deduce that the leaves of $P[\mathrm{cur}\circ P']$ are labeled by subgoals sufficient for the formula labeling the root node. Thus the conjecture is verified if a proof-tree is computed, such that each[3] leaf is labeled true. This corollary justifies the definition of *solved* stated in §2.1.

Meta-Rules: The interaction of the calculi $\vdash_c$, $\vdash_a$, and the meta-rule sets is defined by (5) and (6). The inference rules operate on triples consisting of a stack of meta-rule sets, a stack of meta-variable bindings, and the proof-tree. The inference rules treat a meta-rule set M as unordered set. The meta-strategy deciding which meta-rule to apply as well as the backtracking behavior (cf. §2.3 and [10]) is not defined by the inference system. For the implementation of our framework we decided to arrange the meta-rules as totally ordered sequence $M_<$, i.e. applicable meta-rules occurring earlier in $M_<$ are preferred[4].

$$\frac{\begin{array}{c}\langle \hat{M}M, \hat{E}E, P[\mathrm{cur}]\rangle \\ \mathrm{cond} \Rightarrow \mathrm{action}\ \in M \\ \langle P[\mathrm{cur}], E, \emptyset\rangle \vdash_c \langle P[\mathrm{cur}], E\cup F, \mathrm{cond}\rangle \\ \langle M, E, E\cup F, P[\mathrm{cur}], \emptyset\rangle \vdash_a \langle \hat{M}', \hat{E}', E\cup F, P[\mathrm{cur}\circ G], \mathrm{action}\rangle \\ \langle P[\mathrm{cur}], E, \emptyset\rangle \not\vdash_c \langle P[\mathrm{cur}], F', \mathsf{tc}(M)\rangle\end{array}}{\langle \hat{M}\hat{M}', \hat{E}\hat{E}', P[\mathrm{cur}\circ G]\rangle} \tag{5}$$

$$\frac{\begin{array}{c}\langle \hat{M}M, \hat{E}E, P[\mathrm{cur}]\rangle \\ \langle P[\mathrm{cur}], E, \emptyset\rangle \vdash_c \langle P[\mathrm{cur}], E', \mathsf{tc}(M)\rangle\end{array}}{\langle \hat{M}, \hat{E}, P[\mathrm{cur}]\rangle} \tag{6}$$

[3] In the presence of or-nodes in the proof-tree the condition needs to be refined.

[4] Most other strategies, e.g. using the meta-rule with the most specific precondition, can be simulated by ordering the meta-rules appropriately. But our decision allows for maximal flexibility.

The successful application of a meta-rule is described by (5). The optional persistence condition is treated as part of the action. For a meta-rule set M a call to M has the form META-RULE-SET M UNTIL $\mathsf{tc}(M)$, $\mathsf{tc}(M)$ being the optional *termination condition* for this call. The planner returns to the previous meta-rule set, if $\mathsf{tc}(M)$ is satisfied for the active meta-rule set M. Inference rule (6) is applicable in this case: M and E are popped from the stack and planning proceeds using the top of $\hat{M}$ as active meta-rule set.

A proof resp. proof plan for the conjecture ϕ is constructed by computing a derivation $\langle I, \emptyset, P_\phi \rangle \vdash \langle \hat{M}, \hat{E}, P \rangle$ using (5) and (6), where I is the initial meta-rule set, P_ϕ is the proof-tree consisting of a single node labeled by ϕ, and P is proof-tree, s.t. each leaf is labeled true[3].

2.3 Structured Proof Planning

Sequencing: Consider an action $T_1, \ldots, T_n$: If T_{i-1} yields $k > 1$ goals, only the proof-tree node corresponding to one of them is expanded by the $T_i, \ldots, T_n$ according to $\vdash_a$. Hence this case has been ruled out for the presentation of $\vdash_a$ to ease understanding. To generalize $\vdash_a$, the proof-tree component with the current goal as designated node $P[\mathrm{cur}]$ has to be replaced by a proof-tree with k designated nodes $P[\mathrm{cur}_1, \ldots, \mathrm{cur}_k]$, each of which is to be passed to the next tactic T_i. Similarly $P[\mathrm{cur}{\circ}G]$ in the result needs to be replaced by $P[\mathrm{cur}_1{\circ}G_1, \ldots, \mathrm{cur}_k{\circ}G_k]$ where T_i applied to cur_j yields the goals G_j. Thus the remainder of an action is applied to each leaf of the proof-tree generated as goal in the previous step.

Recovery: The calculus presented in §2.2 defines a successful derivation for a conjecture, i.e. a proof plan and a proof. It does not specify the reaction to a failing subaction or an unsatisfiable continuation condition or persistence condition. The action is retracted in this case, i.e. all effects of subactions are taken back. The planner backtracks over alternative bindings of the retracted meta-rule's precondition and further applicable meta-rules.

Only if the termination condition $\mathsf{tc}(M)$ for the active meta-rule set M is satisfied, the planner returns to the previous meta-rule set. Then the call to M (considering the subaction) is completed and the previous environment is restored, cf. (6). In this case the proof-tree is expanded as if each action, executed while M is the active meta-rule set, were part of the action which contains the call to M. If at least one meta-rule has been applied, no meta-rule is applicable anymore, and the termination condition is not satisfied, the planner backtracks within the meta-rule set M. The call to M fails if no meta-rule is applicable when control enters M. In latter case the proof tree is not expanded.

Ordered (cf. §2.2) meta-rule sets can simulate conditionals. To express an action IF c THEN a_1 ELSE a_2 we use a call to the meta-rule set (7), while specifying the termination condition to apply only one meta-rule: if UNTIL $\mathsf{one\text{-}step}(M)$. For the remainder of this paper we will use IF c THEN a_1 ELSE a_2 as an abbreviation for a call to a meta-rule set analogous to (7).

$$\begin{array}{lrl} \text{META-RULE-SET } \mathsf{if} & & \\ \mathsf{if\text{-}true}: & c & \Rightarrow a_1 \\ \mathsf{else}: & \mathsf{true} & \Rightarrow a_2 \end{array} \qquad (7)$$

3 Modeling Strategies

We will present some strategies for automated induction theorem proving, utilizing the additional information available within our framework of *structured incremental proof planning*, thus exemplifying the power of our approach.

3.1 Case-Analysis Using Structured Meta-Rule Sets

Case analysis and induction are competing strategies. Hence it is hard to decide a priori which technique is more appropriate. We view induction as a special case analysis and the decision, whether the case-split is an induction, is postponed. By revising the partial proof we decide, if an induction hypothesis can be applied and which induction hypothesis has to be chosen. Fig. 1 presents the meta-rule set eval, which is the initial meta-rule set for the strategy outlined above. We iterate over eval and apply special techniques for nested cases, symbolic evaluation, and case analysis. This strategy is similar to *lazy induction* [11].

META-RULE-SET eval
nested-case: nested-case(Hyp) $\Rightarrow$ META-RULE-SET manipulate-hypo(Hyp)
UNTIL solved(Goal) $\vee$ sym-eval-p(Term)
sym-eval: sym-eval-p(Term) $\Rightarrow$ sym-eval(current,Term)
csp: need-case-split(Term) $\Rightarrow$ case-split(F,current),
IF base-case(Term) THEN META-RULE-SET eval UNTIL solved(Goal)
ELSE META-RULE-SET eval-step UNTIL no-meta-rule-applicable(S)

Fig. 1. Meta-rule set eval used as initial meta-rule set.

The meta-rule nested-case is applicable within a nested case-split, which is tested by the meta-predicate nested-case(Hyp). Hyp is bound to the premises of the case corresponding to the current goal. The call manipulate-hypo(Hyp) succeeds, if it is possible to simplify or disprove the hypotheses, or to derive a new condition which enables further symbolic evaluation. This goal is reflected by the termination condition solved(Goal) $\vee$ sym-eval-p(Term). The parameter Hyp of the meta-rule set manipulate-hypo propagates knowledge from the outer meta-rule set eval into the new meta-rule set. If the known conditions Hyp could be disproved (disprove, fig. 2), then the current subgoal has been solved and the termination condition is satisfied. Hence the planer returns to the surrounding meta-rule set eval and resumes after selecting another unsolved branch of the proof-tree. If a new condition is derived from the hypothesis Hyp (cond-required) or if the known conditions are simplified (simplify), symbolic evaluation should be possible and the planner proceeds with the meta-rule set eval.

The meta-rule sym-eval (fig. 1) checks, if symbolic evaluation is possible and the tactic sym-eval exhaustively applies rewrite rules until no rewrite rule is applicable anymore. If no further symbolic evaluation is possible and a case-split enables further symbolic evaluation, a case-split is performed (csp).

When analyzing proofs by induction one usually concentrates on the step-

META-RULE-SET manipulate-hypo(Hyp)
disprove: hyp-disprovable(Hyp) $\Rightarrow$ disprove(Hyp,current)
cond-required: extra-cond-required(Hyp,C) $\Rightarrow$
deduce-cond(Hyp,C,Current) $\hookleftarrow$ sym-eval-p(Hyp,C,Term)
simplify: simplify-p(Hyp) $\Rightarrow$ simplify(Hyp,current)

Fig. 2. Meta-rule set manipulate-hypo.

cases and applies strategies more cautiously there. The base-cases[5] often can be solved straight forward by applying case analysis and symbolic evaluation. Hence we want to distinguish between strategies applied for the base-cases and strategies applied for the step-cases. This is described by the meta-rule csp. Note that we do not attach a heuristic termination condition, which examines the current goal, to the call of the meta-rule set eval-step (fig. 3). The meta-predicate no-meta-rule-applicable represents a non-heuristic termination condition, because it is independent of the current goal. Using this termination condition, the planner iterates over the given set of meta-rules until no more meta-rule is applicable. If the application of the meta-rule set eval-step does not complete the proof of the step-case a new iteration over the meta-rule set eval is started.

META-RULE-SET eval-step
nested-case-step: nested-case(Hyp) $\Rightarrow$ META-RULE-SET manipulate-hypo(Hyp)
UNTIL solved(G) $\vee$ sym-eval-p(F)
ind: no-sym-eval-p(Term) $\wedge$ induction-split(Split) $\wedge$
induction-hyp(Hyp,Split) $\wedge$ hyp-applicable(Hyp,C) $\wedge$
ind-base-case-solved(F) $\Rightarrow$ fertilize(Hyp,current)
eval-step: sym-eval-p(Subterm) $\Rightarrow$ sym-eval-step(current,Subterm)

Fig. 3. Meta-rule set eval-step.

ind captures the central idea of lazy induction: Inspecting the partial proof and deciding, whether an adequate induction hypothesis is applicable. The meta-predicate induction-split(Split) inspects the case-splits on the path from the current goal to the root of the proof-tree if there exists a split which satisfies the induction criterion. The induction criterion is satisfied, if a formula ϕ' is an induction hypothesis for a conjecture ϕ, and if ϕ' is smaller than ϕ wrt. some well-founded relation. If the previous split does not satisfy the induction criterion, the meta-predicate looks for another case-split. If no such case-split satisfies the induction criterion the meta-predicate is not satisfiable. Thus the meta-rule ind is not applicable. The variable Split is bound to the formula which led to the case-split. The meta-predicate induction-hyp(Hyp,Split) binds Hyp to the induction hypothesis for the formula Split, and hyp-applicable(Hyp,C) checks, if the induction hypothesis is applicable for the current goal. If the hypothesis does not match the current goal, Split is bound to another candidate as described above. The meta-predicate ind-base-case-solved(F) ensures that the base-case is verified. By revising case-splits and exploiting the knowledge of the partial proof we are

[5] When referring to "base-cases" resp. "step-cases" we also include non-recursive resp. recursive cases of a case-split wrt. a recursively defined function.

able to decide, if an earlier case-split satisfies the induction criterion and if the induction hypothesis is applicable. If symbolic evaluation is possible, the meta-rule eval-step is applied. In contrast to sym-eval the tactic sym-eval-step performs only a single rewrite. Otherwise an applicable induction hypothesis may be lost.

Possible outcomes of a call to the meta-rule set eval-step are: (i) The step-case is solved by the meta-rules eval-step and nested-case-step. Then the goal ψ, which has been the current goal when eval-step was called, is proved. (ii) The meta-rule ind is applied successfully and the goal is solved by eval-step after fertilization. Hence ψ is proved by induction. (iii) The conjecture is evaluated by eval-step, but no earlier case-split satisfies the induction criterion or a possible induction hypothesis is not applicable. (iv) An earlier case-split is revised and an induction hypothesis is applicable, but symbolic evaluation does not suffice to solve the conjecture. For (iii) and (iv) the goal is returned (via the expanded proof-tree) to the meta-rule set eval for further manipulation (case-split).

3.2 Proving Theorems by Revising Case-Splits

To illustrate the planning procedure outlined above, consider the following definitions for ord and ins and the theorem (8)[6]. ord checks whether l is le-ordered, while ins inserts x into l respecting the le-ordering. The proof starts with an application of csp from the meta-rule set eval, which is illustrated in fig. 4.

$$\forall x : \mathsf{number}, l : \mathsf{list}\ \mathsf{ord}(l) = \mathsf{ord}(\mathsf{ins}(x, l)) \tag{8}$$

$$\begin{array}{ll}
\multicolumn{2}{l}{\textit{function}\ \mathsf{ord}(l : \mathsf{list}) : \mathsf{bool} \Leftarrow} \\
\quad \textit{if}\ l = \mathsf{nil} & \textit{then}\ \mathsf{true} \\
\quad \textit{if}\ l = \mathsf{add}(\mathsf{hd}(l), \mathsf{nil}) & \textit{then}\ \mathsf{true} \\
\quad \textit{if}\ l = \mathsf{add}(\mathsf{hd}(l), \mathsf{tl}(l)) \wedge \mathsf{not}(\mathsf{le}(\mathsf{hd}(l), \mathsf{hd}(\mathsf{tl}(l)))) & \textit{then}\ \mathsf{false} \\
\quad \textit{if}\ l = \mathsf{add}(\mathsf{hd}(l), \mathsf{tl}(l))) \wedge \mathsf{le}(\mathsf{hd}(l), \mathsf{hd}(\mathsf{tl}(l))) & \textit{then}\ \mathsf{ord}(\mathsf{tl}(l))
\end{array} \tag{9}$$

$$\begin{array}{ll}
\multicolumn{2}{l}{\textit{function}\ \mathsf{ins}(x : \mathsf{number}, l : \mathsf{list}) : \mathsf{list} \Leftarrow} \\
\quad \textit{if}\ l = \mathsf{nil} & \textit{then}\ \mathsf{add}(x, \mathsf{nil}) \\
\quad \textit{if}\ l = \mathsf{add}(\mathsf{hd}(l), \mathsf{tl}(l)) \wedge \mathsf{le}(x, \mathsf{hd}(l)) & \textit{then}\ \mathsf{add}(x, l) \\
\quad \textit{if}\ l = \mathsf{add}(\mathsf{hd}(l), \mathsf{tl}(l)) \wedge \mathsf{not}(\mathsf{le}(x, \mathsf{hd}(l))) & \textit{then}\ \mathsf{add}(\mathsf{hd}(l), \mathsf{ins}(x, \mathsf{tl}(l)))
\end{array} \tag{10}$$

The base-case $l = \mathsf{nil}$ resp. the step-case $\mathsf{le}(x, \mathsf{hd}(l))$ is solved by applying the meta-rule sym-eval from the meta-rule set eval resp. eval-step from the meta-rule set eval-step. The second step-case $\mathsf{not}(\mathsf{le}(x, \mathsf{hd}(l)))$ is evaluated within the meta-rule set eval-step by the meta-rule eval-step. Now the precondition of ind is tested. The first meta-predicate induction-split(Split) is satisfied, as the case-split wrt. the definition of ins also satisfies the induction criterion. But the potential

[6] The data structure bool is freely generated by the constructors true and false. list is generated by nil and add, hd and tl are destructor functions for add. We omit the definitions for le ($\leq$) and not. We also omit the quantifier prefix henceforth.

[7] Note that we omitted the literal $l = \mathsf{add}(\mathsf{hd}(l), \mathsf{tl}(l))$ from the condition for lack of space. Obvious structural premises will also be omitted from the remaining figures. The edge labels are premises guarding the cases.

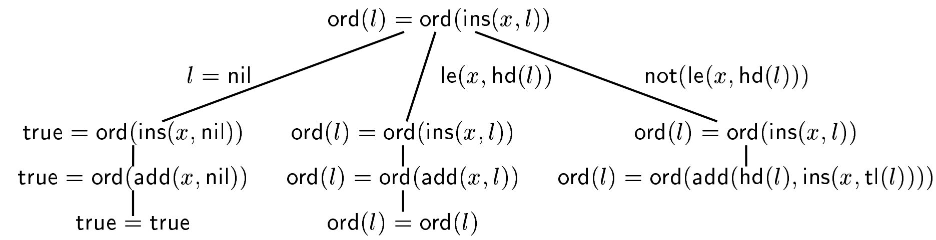

Fig. 4. A case-split of (8) wrt. the definition of ins (10)[7].

induction hypothesis $\mathsf{ord}(\mathsf{tl}(l)) = \mathsf{ord}(\mathsf{ins}(x, \mathsf{tl}(l)))$ is not applicable. Thus the precondition of the meta-rule ind is not satisfiable and ind is not applicable. Hence the subgoal is returned to the surrounding meta-rule set eval.

To prove the conjecture in the step-case another case-split wrt. (10) is performed. The case $\mathsf{tl}(l) = \mathsf{nil}$ is proved by sym-eval and nested-case from the meta-rule set eval. The proof of the step-case $\mathsf{le}(x, \mathsf{hd}(\mathsf{tl}(l)))$ is performed under control of the meta-rule set eval-step. First the meta-rule eval-step and then nested-case succeed. Within the meta-rule set manipulate-hypo the meta-rule cond-required applies and infers a new condition $\mathsf{le}(\mathsf{hd}(l), \mathsf{hd}(\mathsf{tl}(l)))$, which enables further evaluation. Finally eval-step completes the proof, cf. fig. 5.

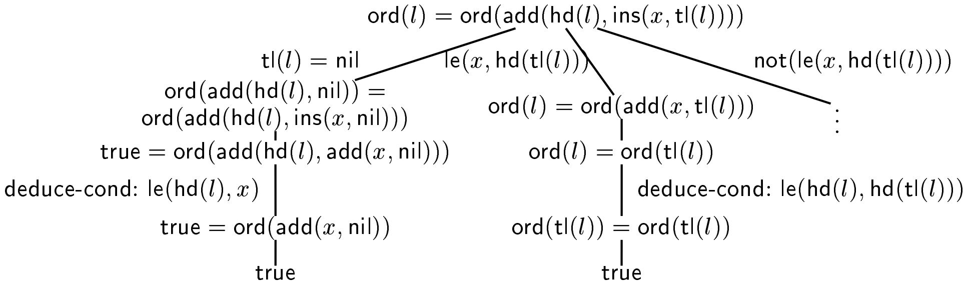

Fig. 5. Two cases of the second step-case from fig. 4.

The proof of the second step-case $\mathsf{not}(\mathsf{le}(x, \mathsf{hd}(\mathsf{tl}(l))))$ is shown in fig. 6. First the meta-rule eval-step from the meta-rule set eval-step is applicable. However, afterwards no more meta-rule of this set is applicable. Hence the subgoal $\mathsf{ord}(l) = \mathsf{ord}(\mathsf{add}(\mathsf{hd}(l), \mathsf{add}(\mathsf{hd}(\mathsf{tl}(l)), \mathsf{ins}(x, \mathsf{tl}(\mathsf{tl}(l))))))$ is returned to the meta-rule set eval again, where the meta-rule csp suggests a case-split to symbolically evaluate for ord. The base-case of this conjecture is solved by the meta-rule set eval again. The step-case $\mathsf{le}(\mathsf{hd}(l), \mathsf{hd}(\mathsf{tl}(l)))$ is proved by the meta-rule set eval-step by applying the meta-rule eval-step. The remaining subgoal is the step-case $\mathsf{not}(\mathsf{le}(\mathsf{hd}(l), \mathsf{hd}(\mathsf{tl}(l))))$. The meta-rule set eval-step applies first the meta-rule eval-step. Afterwards the meta-rule ind is tested. Now the precondition is satisfiable. The split satisfying the induction criterion and whose induction hypothesis is applicable is marked by a box in fig. 6. So the original conjecture (8) is verified. (8) cannot be proven merely by induction: A case-split followed by an induction is needed to solve the step-case of the case-split.

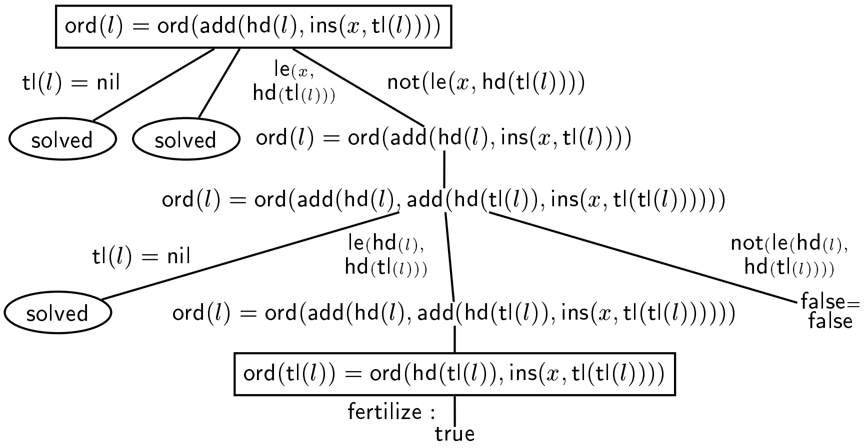

Fig. 6. The proof of the third case $\mathsf{not}(\mathsf{le}(x, \mathsf{hd}(\mathsf{tl}(l))))$ from fig. 5.

3.3 Experimental Results

The strategy specified by the meta-rule sets presented in §3.1 also works for proofs by case analysis or proofs by inductions alone. Experimental results are shown in table 1. For some examples (7,8) an induction is needed within the base-case of a case analysis. For other examples a case-split is used first, followed by an induction to verify the step-case. So we combine non-inductive case-splits and induction on demand, which is the key idea of lazy induction.

#	theorem	#	theorem
1	$\mathsf{even}(x) = \mathsf{even1}(x)$	2	$\mathsf{mem}(x, t) \to \mathsf{mem}(x, \mathsf{app}(s, t))$
3	$\mathsf{mem}(x, t) \to \mathsf{mem}(x, \mathsf{ins}(a, t))$	4	$\mathsf{subset}(x, y) \to \mathsf{subset}(x, \mathsf{ins}(a, y))$
5	$\mathsf{subset}(x, \mathsf{delete}(a, y)) \to \mathsf{subset}(x, y)$	6	$\mathsf{ord}(l) = \mathsf{ord}(\mathsf{ins}(a, l))$
7	$\mathsf{even}(\mathsf{plus}(x, y)) \vee \mathsf{odd}(\mathsf{plus}(y, x))$	8	$\mathsf{mem}(x, \mathsf{ins}(x, y))$
9	$\mathsf{even}(\mathsf{len}(\mathsf{app}(x, y))) = \mathsf{even}(\mathsf{len}(\mathsf{app}(y, x)))$	10	$\mathsf{len}(\mathsf{app}(a, b)) = \mathsf{len}(\mathsf{app}(b, a))$
11	$\mathsf{even}(\mathsf{half}(x)) \vee \mathsf{odd}(\mathsf{half}(x))$		

Table 1. Experimental Results.

Examples 7-10 are taken from [8]. We observe that we need less lemmata. In [8] exploiting the failure of an inductive proof is proposed. Lemmata are speculated or an inappropriate induction scheme is revised, in order to succeed. Utilizing the additional information of the partial proof resp. proof plan we are able to prove these theorems merely by case analysis and induction. Other systems cannot reinspect partial proof attempts. User guidance or lemmata are needed in this case to recover from the failure.

4 Conclusion, Related and Future Work

While our technique is forward chaining, a mixed forward/backward chaining technique is presented in [7]. There the tactic knowledge is split into a declarative part and a procedural part. The latter can be interpreted during proof checking. Similar to [2] (higher order) formula schemes are used to represent tactics' pre- and postconditions. A method similar to incremental proof planning has been introduced in [12], but a meta-reasoning step consists of speculating lemmata (new goals) which in turn are verified by other automated theorem provers. As with our technique meta-level and object-level reasoning are interleaved.

The use of meta-predicates on proof-trees and the possibility to reason about tactics' results is the main difference in expressiveness between our approach and similar systems. The strategy knowledge is structured in several sets of meta-rules. Switching the active meta-rule set adopts the strategy to the current situation. We demonstrated our approach with a solution to the control problem for proofs requiring case analysis and induction. Our technique is implemented in the TIGER proof planner [9, 10].

References

1. Robert S. Boyer and J Strother Moore. *A Computational Logic.* ACM Monograph Series. Academic Press, 1979.
2. Alan Bundy. A science of reasoning. In J-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 178–198. MIT press, 1991.
3. Alan Bundy, Andrew Stevens, Frank van Harmelen, Alan Smaill, and Andrew Ireland. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, 62(2):185–253, August 1993.
4. R. L. Constable, T. B. Knoblock, and J. L. Bates. Writing programs that construct proofs. *Journal of Automated Reasoning*, 1:285–326, 1985.
5. Stefan Gerberding and Axel Noltemeier. Choosing induction relations within the INKA system (extended abstract). In *Proceedings of the 19th German Annual Conference on Artificial Intelligence, Bielefeld, Germany*, pages 329–331, 1995.
6. Stefan Gerberding and Axel Noltemeier. Incremental proof planning by meta-rules. In Douglas D. Dankel II, editor, *Proceedings of the 10th FLAIRS Conference, Daytona Beach, Fa.*, pages 171–175. Florida AI Research Society, May 1997.
7. Xiaorong Huang, Manfred Kerber, Jörn Richts, and Arthur Sehn. Planning mathematical proofs with methods. SEKI Report SR-94-08, Universität Kaiserslautern, 1994.
8. Andrew Ireland and Alan Bundy. Productive use of failure in inductive proofs. *Journal of Automated Reasoning*, 16:79–111, 1996.
9. Axel Noltemeier. Inkrementelle Beweisplanung mit Metaregeln. Diploma Thesis, University of Darmstadt, 1996.
10. Brigitte Pientka. Strukturierung der Beweisplanung durch Metaregelmengen. Diploma Thesis, University of Darmstadt, 1997.
11. Martin Protzen. Lazy generation of induction hypotheses. In *Proceedings of the 12th International Conference on Automated Deduction, Nancy, France*, LNAI vol. 814, pages 42–56. Springer, 1994.
12. Inger Sonntag and Jörg Denzinger. Extending automated theorem proving by planning. SEKI Report SR-93-02, Universität Kaiserslautern, 1993.