Available online at www.sciencedirect.com
SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science 125 (2005) 67–90

Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Failure Reasoning in Multiple-Strategy Proof Planning

Andreas Meier[1] Erica Melis[2]

German Research Center for Artificial Intelligence (DFKI)
Saarbrücken, Germany

Abstract

Monitoring a solution process and applying the right action at the right moment are at the heart of intelligent problem solving by humans. This includes the analysis of failure events and the development of "recommendations" to overcome typical failures.
In this article, we present how meta-reasoning on failures is used in multiple-strategy proof planning with the MULTI system. MULTI allows for a flexible traversal of the search space and a flexible construction of the proof plan guided by mathematically motivated heuristics. Because of the flexible control in MULTI failures can be exploited to guide subsequent proof plan manipulations and refinements. The failure reasoning cannot only ease the derivation of a solution proof plan but is required for some problems to find a solution at all.

Keywords: Proof Planning, Meta-Reasoning

1 Introduction

In a problem solving process, a step may not result in the expected progress or may not be applicable as expected. Hence, it is part of intelligent problem solving to analyze a failure event and to develop "recommendations" to handle typical failures, i.e., to guide the subsequent solution process. This also holds for mathematical theorem proving for which "*monitoring the state of a solution as it evolves and taking appropriate action in the light of new information*" is a key skill as Schoenfeld points out in his book on mathematical problem solving [16].

[1] Email: `ameier@dfki.de`
[2] Email: `melis@dfki.de`

doi:10.1016/j.entcs.2005.01.004

Monitoring the solution process and using "recommendations" requires a flexible control approach and reasoning about the problem solving situation. Intelligent humans do not rely upon pre-determined control to guide their problem solving. Instead, they draw upon a repertoire of heuristics for dynamic solution construction. As opposed to human problem solving, search-based theorem proving systems often employ restricted control components. Typically, e.g., in provers based on the resolution principle, the control is based on "local" general-purpose heuristics that reason on the current proof facts and the possible inferences. As result, the control decisions are restricted to the selection of the inference rule and the premises for the next step in the proof derivation. Other decisions are often hard-coded into the system and not subject to reasoning about control such that monitoring and overseeing the entire problem-solving process is hardly possible.

Proof planning is an approach to mathematical theorem proving in which the proof of a theorem is planned at the abstract level of so-called methods. The knowledge-based proof planning developed in the ΩMEGA group employs declarative meta-reasoning to guide the search [14]. Mathematically motivated heuristics cannot only reason about the current goals and assumptions but also about the proof planning history and the planning context. Moreover, in ΩMEGA's automated multiple-strategy proof planner MULTI the choice points that are subject to heuristic guidance are not restricted to the next goal and the next method. Rather, also the decisions on which strategy to choose can be guided, where strategies are independent proof plan operations, and different strategies can realize, for instance, different kinds of backtracking and different kinds of variable instantiation.

In extensive experiments we applied MULTI to several mathematical domains. The analysis of MULTI's proof attempts revealed typical failure situations as well as meta-reasoning patterns on how to deal with these failure situations. It turned out that during the automated proof construction with MULTI such failure reasoning patterns do not only guide "clever" steps that ease the derivation of a solution proof plan but are often necessary to find a solution at all.

In this paper, we shall describe the realization of meta-reasoning on failures in the multiple-strategy proof planner MULTI. First, we introduce the basics of proof planning with multiple strategies and point out how MULTI's flexible control approach allows for a flexible handling of impasses in the proof planning process. Afterwards, section 3, we describe several failure reasoning patterns that analyze and exploit failures to guide proof plan manipulations and refinements. The subsequent sections, section 4 – 5, exemplify the realization and application of the failure reasoning patterns to proof plan ϵ-δ-problems. Although we illustrate the failure reasoning patterns with ϵ-δ-proofs, this meta-reasoning is applicable to other domains as well as our empirical results in section 6 evidence. Section 7 concludes the paper with a discussion of related work.

2 Proof Planning with Multiple Strategies

Proof planning was originally conceived as an extension of tactical theorem proving to implement automated theorem proving at the abstract level of tactics. Bundy's key idea in [3] is to augment individual tactics with pre- and postconditions. This results in planning operators, so-called *methods*. In the ΩMEGA [17] system proof planning is enriched by incorporating knowledge into the planning process [14] and the introduction of the additional hierarchical level of strategies [13].

Domain-specific knowledge can be encoded in methods and *control rules*. Methods can encode not only general proving steps but also steps particular to a mathematical domain. Heuristics guiding the search can be encoded in control rules. The control rules are evaluated at choice points in the planning process and can express meta-level reasoning about the current proof planning state as well as about the entire history of the proof planning process and the proof context. Further domain-specific knowledge can be contained in *external systems* that are incorporated into the proof planning process. For proof planning ϵ-δ-problems, which we shall discuss, in particular, the constraint solver $\mathcal{C}o\mathcal{SIE}$ for equations and inequalities over the reals is important.

Proof construction may require to construct mathematical objects, i.e., to instantiate existentially quantified variables by witness terms. In proof planning, *meta-variables* are used as place holders for witness terms. When proof planning ϵ-δ-problems, equations and inequalities with meta-variables are passed to $\mathcal{C}o\mathcal{SIE}$. $\mathcal{C}o\mathcal{SIE}$ checks the (in)consistency of the constraints and collects consistent constraints in a constraint store. Later, it tries to compute instantiations for the meta-variables that satisfy the collected constraints [18].

The simplest version of proof planning, realized in the previous proof planner of ΩMEGA, searches at the level of methods only, i.e., as long as there are goals it searches for applicable methods and applies the instantiated methods, which are called actions. The final sequence of actions forms a solution plan. The application of the operations backtracking and meta-variable instantiation is hard-wired with the action introduction: The proof planner backtracks, if and only if there is a goal to which no method is applicable. If this is the case, the planner backtracks the action that introduced this goal. Moreover, it instantiates meta-variables at the end only, when all goals are closed.

Case-studies revealed that this proof planning approach is somewhat inflexible and fails for a number of problems (e.g., see [13]). In particular, the previous proof planner of ΩMEGA fails for problems whose solution requires reasoning on failures. Because of its hard-coded control and its restricted functionalities, it neither provides choice points to reason on failures nor suitable alternatives (e.g., different kinds of backtracking) to choose from. For instance, consider the following two impasses: (1) If no method is applicable, then the application of a particular backtracking operation is hard-coded in the planner; reasoning on the failure and choosing between different kinds of reactions is not possible. (2) The meta-variables are always instantiated at the end by incorporated constraint solvers. When the constraint solvers fail, then reasoning on this failure and choosing alternative actions to overcome it are also not possible.

These observations as well as the observation of further drawbacks of the previous proof planning approach (see [13]) motivated the development of proof planning with multiple strategies. Proof planning with multiple strategies decomposes the previously monolithic proof planning process and replaces it by separate *strategies*, which are instances of parameterized algorithms for different proof plan refinements and modifications.

We implemented proof planning with multiple strategies in the proof planner MULTI [13]. Among others, MULTI employs general algorithms for action introduction, meta-variable instantiation, and backtracking. The algorithm for action introduction has parameters for a set of methods and a set of control rules. When MULTI executes a strategy of this algorithm, then the algorithm introduces only actions that use the methods specified in the strategy. The choices during the action computation and selection are guided by the control rules specified by the strategy. The single parameter of the instantiation algorithm is a function that determines how the instantiation for a meta-variable is computed. If MULTI applies an instantiation strategy wrt. a meta-variable mv and if the computation function of the strategy yields a term t for mv, then the instantiation algorithm substitutes mv by t in the proof plan. The single parameter of the backtrack algorithm is a function that computes a set of refinement steps of other algorithms that have to be deleted. When MULTI applies a backtrack strategy, the algorithm removes all refinement steps that are computed by the function parameter of the strategy as well as all steps that depend from these steps. Sample strategies of all three algorithms are discussed in section 4 and section 5.

In MULTI, no sequence of strategies is pre-defined or hard-coded in a control cycle. Rather, MULTI's blackboard architecture enables the flexible cooperation of independent strategies guided by meta-reasoning in *strategic control rules*.

In a nutshell, Multi operates according to the following cycle:

Job Offers Applicable strategies post their applicability in form of so-called job offers onto the blackboard.

Guidance Strategic control rules are evaluated to order the job offers.

Invocation The strategy with the highest ranked job offer is invoked.

Execution The algorithm of the invoked strategy is executed with respect to the parameter instantiation specified by the strategy.

Note that the execution of an action introduction strategy can be interrupted (i.e., interruption is a choice point in the action introduction algorithm). In this case, Multi can first apply some other strategies and then re-invoke the interrupted strategy execution. A detailed, technical description of the Multi system can be found in [10].

Failures in the action introduction algorithm, i.e., a goal for which no method is applicable, result in an interruption of the action introduction algorithm. Multi continues to work at the strategy level by collecting job offers and evaluating strategic control rules to rank the job offers. Similar to the previous proof planner, Multi's default approach to deal with such a failure is to backtrack the action that introduced this goal. However, this backtrack approach is not hard-coded into Multi's algorithm. Rather, it is realized by an according strategy of the backtrack algorithm and a strategic control rule that guides the application of this backtrack strategy after such a failure. Further backtrack strategies can be encoded that delete different sets of actions. Moreover, further strategic control rules overwriting this default behavior can be specified that reason on the failure and guide suitable subsequent proof plan refinements and modifications. Also reasoning on failing meta-variable instantiations is possible, since in Multi the instantiation of meta-variables is not pre-defined to be the final proof plan refinement.

3 Failure Reasoning

For many situations, the default handling of failures in Multi (see previous section) is not sufficient. Rather, different handling of failures is necessary. The reasons for this are twofold:
(1) Theorem proving often requires steps whose necessity is difficult to predict. Reasoning about a situation in which a failure occurred can suggest certain recovery or solution steps. Hence, the failures and their productive use can hold the key to discover a solution proof plan.
(2) Goals and applications of methods and strategies can be intertwined in complex ways. In particular, the incorporation of constraint solving into proof

planning causes dependencies that make a "standard" handling of failures difficult. Rather, dependencies have to be analyzed in order to guide suitable reactions.

In the following, we shall discuss several domain-independent and general meta-reasoning patterns on typical failures. The meta-reasoning patterns are declaratively encoded into corresponding control rules. This encoding and the concrete application to ϵ-δ-problems are discussed in section 5.

Guiding Case Splits

Case-split is a well-known technique in mathematics. But when is it useful to apply it and which cases should be considered? The following general pattern describes the need for a case-split: there is a main goal, which can be solved by methods introducing some side goals. These side goals are called conditions. If one of the conditions cannot be solved, then a partial success, i.e., the solution of the main goal, gives rise to consider patching the proof attempt by a case-split on the failing condition. Then, the main goal has to be proved for each case. This approach corresponds to the meta-reasoning pattern:

Case-Split Introduction:	
IF	failing condition while main goal is solved
THEN	introduce case-split on failing condition

In the concrete application of this meta-reasoning pattern to ϵ-δ-problems, see section 5, we shall explain how the main goal and the side goals are determined in this domain.

Unblock Meta-Variable Instantiation

During the proof planning process the application of certain key steps can become particularly "desirable". If such a desirable step should be applied but is blocked, then the application of other steps should be considered, which will unblock the desirable step. As example consider strategies of the instantiation algorithm that employ constraint solvers. When all goals are closed, the application of these strategies becomes highly desirable to instantiate the meta-variables. However, when the constraint solvers fail because the constraints collected so far are not sufficient, then the application of these strategies fails. A possibility to overcome this problem is to refine the existing constraints in order to obtain an extended set of refined constraints for which a solution exists. This approach corresponds to the meta-reasoning pattern:

Unblock Meta-Variable Instantiation:	
IF	constraint solver fails to provide instantiations because of insufficient constraints
THEN	create and pass further constraints

In the concrete application of this meta-reasoning pattern to ϵ-δ-problems, see section 5, we shall explain how certain steps are selected for backtracking in order to enable the creation of further constraints in this domain.

Analysis of Meta-Variable Dependencies

The instantiations of meta-variables and constraints on the meta-variables cause dependencies among goals that share these meta-variables. Take, e.g., two goals G and G' that both contain a meta-variable mv. Now assume that Multi first creates a partial proof plan for G and binds mv in such a way that G' cannot be proved anymore. The default backtracking in Multi would remove G'. However, the actual problem is not G' but the selection of an appropriate instantiation for mv. That is, part of the subplan for G has to be removed to introduce another subplan that instantiates mv differently. This approach corresponds to the general meta-reasoning pattern:

Analyze MV-Dependencies:	
IF	failure on goal caused by meta-variable instantiation/constraints
THEN	backtrack meta-variable instantiation/constraints

In the concrete application of this meta-reasoning pattern to ϵ-δ-problems, see section 5, we shall explain how the causal connection of a failure with the instantiation of a meta-variable or constraints on meta-variables is determined in this domain.

4 Proof Planning ϵ-δ-problems

We shall elaborate the usage of these meta-reasoning patterns for ϵ-δ-problems, which prove statements about the limit, the continuity, or the derivative of a function f at a point a. The standard definitions of limit, continuity, and derivative comprise a dependency of a δ from an ϵ. For instance, the definitions of limit and continuity are:

$$\lim_{x\to a} f = l \equiv$$
$$\forall\epsilon. (0 < \epsilon \Rightarrow \exists\delta. (0 < \delta \land \forall x. (|x-a| > 0 \land |x-a| < \delta \Rightarrow |f(x) - l| < \epsilon)))$$

$$cont(f,a) \equiv \forall \epsilon. (0 < \epsilon \Rightarrow \exists \delta. (0 < \delta \wedge \forall x. (|x-a| < \delta \Rightarrow |f(x) - f(a)| < \epsilon)))$$

An example theorem is the Cont-If-Deriv problem that states that, if a function f has a derivative f' at point a[3], then f is continuous at a. When the definitions of limit and continuity are expanded, then the problem's assumption is

$$\forall \epsilon_1. (0 < \epsilon_1 \Rightarrow \\ \exists \delta_1. (0 < \delta_1 \wedge \forall x_1. (|x_1 - a| < \delta_1 \wedge |x_1 - a| > 0 \Rightarrow |\tfrac{f(x_1)-f(a)}{x_1-a} - f'| < \epsilon_1)))$$

and the problem's theorem is

$$\forall \epsilon. (0 < \epsilon \Rightarrow \exists \delta. (0 < \delta \wedge \forall x. (|x-a| < \delta \Rightarrow |f(x) - f(a)| < \epsilon))).$$

An ϵ-δ-proof of this problem as well as of similar theorems constructs a real number δ depending on ϵ that satisfies certain inequalities.[4] The usual procedure for discovering a suitable δ is the incremental restriction of the range of values. Proof planning adopts this approach by replacing unknown witness terms such as δ by meta-variables and by cooperating with the constraint solver *CoSIE*, which collects constraints on the meta-variables.

In the remainder of this section, we describe the strategies employed by MULTI to accomplish ϵ-δ-proofs. A more detailed description of this application of MULTI is given in [10].

Strategies

Central for accomplishing ϵ-δ-proofs with MULTI is the action introduction strategy SolveInequality, see Table 1. It is applicable to goals whose formulas are inequalities. SolveInequality mainly comprises methods that deal with inequalities such as FACTORIALESTIMATE, COMPLEXESTIMATE, SOLVE*-B, TELLCS, and ASKCS. The control rule `prove-inequality` is in the list of control rules of SolveInequality.

When faced with an inequality goal, SolveInequality first tries to apply the methods TELLCS and ASKCS, which both interface *CoSIE*. TELLCS passes the goal as constraint to *CoSIE* (provided it is consistent with the constraints collected by *CoSIE* so far), whereas ASKCS asks *CoSIE* whether the goal is entailed by its current constraints. If an inequality is too complex to be handled by *CoSIE*, then SolveInequality tries to apply methods such as

[3] That is, if $\lim_{x_1 \to a} \frac{f(x_1)-f(a)}{x_1-a} = f'$.

[4] The construction of a δ is is a non-trivial task for students as well as for traditional, resolution-based automated theorem provers. Bledsoe proposed several versions of the problem Lim+ as a challenge problem for automated theorem proving [2]. The simplest versions of this problem (problem 1 and 2 in [2]) are at the edge of the capabilities of traditional automated theorem provers but the harder versions are beyond their capabilities. More difficult problems such as Cont-If-Deriv cannot be proved by traditional provers.

Strategy: SolveInequality		
Condition	*inequality-goal*	
Action	Algorithm	Action-Introduction
	Methods	ComplexEstimate, TellCS, Simplify, AskCS, Solve*-B, FactorialEstimate ...
	C-Rules	`prove-inequality`, ...

Table 1
The SolveInequality strategy.

Simplify, Solve*-B, ComplexEstimate, and FactorialEstimate that reduce an inequality to simpler inequalities. For instance, applications of the ComplexEstimate method exploit the Triangle Inequality and reduce a goal with formula $|b| < e$ to simpler inequalities in case there is an assumption $|a| < e'$ and $b = k{*}a{+}l$ holds for suitable terms k and l.[5] The resulting simpler goals are $|l| < \frac{e}{2}$, $e' < \frac{e}{2*mv}$, $|k| \leq mv$, and $0 < mv$, where mv is a new meta-variable. The method FactorialEstimate deals with fractions in inequalities. It reduces a goal of the form $|\frac{t}{t'}| < t''$ to the three subgoals $0 < mv_F$, $mv_F < |t'|$, and $|t| < t'' * mv_F$, where mv_F is a new meta-variable. Applications of Solve*-B exploit transitivity of $<, >, \leq, \geq$ and reduce a goal with formula $a_1 < b_1$ to a new goal with formula $b_2\sigma \leq b_1\sigma$ in case an assumption $a_2 < b_2$ exists and a_1, a_2 can be unified by the substitution σ. The method Simplify employs computer algebra systems to perform arithmetic simplifications of terms. When applied to a goal with arithmetic subterms that can be simplified by the computer algebra systems, it reduces the goal to a new goal with simplified subterms.

So, SolveInequality successively produces simpler inequalities until inequalities are reached that are accepted by $\mathcal{CoSIE}$. This approach – handle with $\mathcal{CoSIE}$ or simplify – is guided by the control rule `prove-inequality`. This rule first checks whether the current goal is an inequality. If this is the case, it prefers the methods of SolveInequality in the desired order: TellCS, AskCS, Simplify, Solve*-B, ComplexEstimate, FactorialEstimate etc.

To derive ϵ-δ-proofs Multi also employs the domain-independent action introduction strategies NormalizeGoal and UnwrapAss. Both strategies contain general methods for the decomposition of logic connectives and quantifiers. Whereas applications of NormalizeGoal decompose goals, applications of Un-

[5] Part of the application of the method is the computation of k, l for given b and a. This is done by the incorporated computer algebra system Maple.

wrapAss decompose assumptions.

In order to instantiate meta-variables that occur in constraints collected by $\mathcal{CoSIE}$, Multi employs the two instantiation strategies InstIfDetermined and ComputeInstFromCS. The first is applicable only, if $\mathcal{CoSIE}$ states that a meta-variable is already determined by the constraints collected so far. Then, the computation function connects to $\mathcal{CoSIE}$ and receives this instantiation for the meta-variable. ComputeInstFromCS is applicable to all meta-variables for which constraints are stored in $\mathcal{CoSIE}$. The computation function of this strategy requests from $\mathcal{CoSIE}$ to compute an instantiation for a meta-variable that is consistent with all constraints collected so far.

Application and Cooperation of the Strategies

For proof planning an ϵ-δ-problem Multi typically proceeds as follows: First, it applies NormalizeGoal to decompose the initial goal. Afterwards, it applies SolveInequality to the resulting inequality goals. Some methods of the strategy SolveInequality can only be applied when suitable assumptions are available (e.g., ComplexEstimate and Solve*-B). In case SolveInequality detects promising subformulas of assumptions, it interrupts (guided by one of its control rules) such that Multi can apply UnwrapAss to unwrap the promising subformula. Afterwards, SolveInequality can proceed and use the new assumption.

The invocation of ComputeInstFromCS is delayed by a strategic control rule until all goals are closed. This delay of the computation of instantiations for meta-variables is sensible since the instantiations should not be computed before all constraints are collected, i.e., only after all goals are closed. However, if the current constraints already determine a meta-variable, then a further delay of the corresponding instantiation is not necessary. Rather, immediate instantiations of determined meta-variables can simplify a problem [13]. To allow for the flexible instantiation of determined meta-variables SolveInequality can interrupt and cooperate with the strategy InstIfDetermined.

5 How Failure Reasoning Works (Examples)

There are default application and cooperation of strategies to accomplish ϵ-δ-proofs (see previous section). In addition, since Multi does not pre-define an order or combination of strategies, control rules can be added, which override the default behavior and implement failure reasoning patterns.

5.1 Guiding the Introduction of Case-Splits

The Cont-If-Deriv problem is an example for an ϵ-δ-problem that needs the introduction of a case-split. When tackling this problem, Multi starts as usual for ϵ-δ-proofs. It decomposes the theorem with the strategy NormalizeGoal and derives the inequality goals $0 < mv_\delta$ and $|f(c_x) - f(a)| < c_\epsilon$ (where mv_δ is a new meta-variable and c_x and c_ϵ are new constants) to which it applies SolveInequality. SolveInequality passes the first goal with an application of the method TellCS as constraint to $\mathcal{C}o\mathcal{SIE}$ but fails to reduce the second goal. Since in the initial assumption it detects $|\frac{f(x_1)-f(a)}{x_1-a} - f'| < \epsilon_1$ as a subformula, which could be used, it interrupts. Multi applies the strategy UnwrapAss whose application yields the new assumption

$$|\frac{f(mv_{x_1})-f(a)}{mv_{x_1}-a} - f'| < mv_{\epsilon_1}$$

and the three new goals $0 < mv_{\epsilon_1}$, $|mv_{x_1} - a| < c_{\delta_1}$, and $|mv_{x_1} - a| > 0$ (where mv_{x_1} and mv_{ϵ_1} are new meta-variables and c_{δ_1} is a new constant).

With the new assumption, the strategy SolveInequality closes the main goal $|f(c_x) - f(a)| < c_\epsilon$ in several steps. In between, SolveInequality interrupts once and switches to InstIfDetermined, which introduces the binding $mv_{x_1} \rightarrow c_x$. Then, it tackles the new goals from the application of UnwrapAss. It succeeds to solve $0 < mv_{\epsilon_1}$ and $|mv_{x_1} - a| < c_{\delta_1}$ but fails to solve $|mv_{x_1} - a| > 0$, which meanwhile became $|c_x - a| > 0$ wrt. the introduced binding $mv_{x_1} \rightarrow c_x$. Thus, in this situation, Multi can solve the main goal $|f(c_x) - f(a)| < c_\epsilon$ with an assumption that has some conditions. When Multi uses the assumption, then it introduces the conditions as new goals. Later, it fails to prove one of these conditions, $|c_x - a| > 0$.

The meta-reasoning pattern *Case-Split Introduction* analyzes the failure and suggests its "repair". Technically, the pattern is realized in Multi by one additional backtrack strategy and two control rules, one strategic control rule and one control rule in SolveInequality, which guide suitable backtracking and the introduction of the case-split. This works as follows: If SolveInequality fails to prove a condition of an assumption that was used to prove the main goal, then the strategic control rule triggers the additional backtrack strategy, which deletes all actions following the introduction of the failing condition.

In our example, the application of UnwrapAss and all actions that depend on it are backtracked such that $|f(c_x) - f(a)| < c_\epsilon$ becomes a goal again. When Multi re-invokes SolveInequality after this backtracking, then the control rule in SolveInequality fires and suggests the application of the method CaseSplit for the failing condition and its negation. Afterwards, SolveInequality has to prove $|f(c_x) - f(a)| < c_\epsilon$ twice: once under hypothesis $|c_x - a| > 0$ and once under hypothesis $\neg(|c_x - a| > 0)$.

$$\dfrac{\begin{array}{cc} \text{(Case 1)} & \text{(Case 2)} \\ [|c_x - a| > 0] & [\neg(|c_x - a| > 0)] \\ \vdots & \vdots \\ |f(c_x) - f(a)| < c_\epsilon & |f(c_x) - f(a)| < c_\epsilon \end{array}}{|f(c_x) - f(a)| < c_\epsilon}\ \textsc{CaseSplit}$$

For the first case it proceeds as described above. The failing condition $|c_x - a| > 0$ now follows from the hypothesis of the case. The second case is solved differently by SolveInequality. First, it simplifies the hypothesis $\neg(|c_x - a| > 0)$ to $c_x = a$. Afterwards, it uses this equation to simplify the goal $|f(c_x) - f(a)| < c_\epsilon$ to $0 < c_\epsilon$, which follows from an introduced hypothesis.

Other ϵ-δ-problems also require this kind of failure reasoning (see section 6). In other mathematical domains the same pattern occurs and leads to a case-split introduction (see discussion of related work in section 7). Whereas the failure reasoning pattern is domain independent, the actual case-split may depend on the mathematical domain. Examples for possible domain dependent case-splits are:

$$a > 0, a < 0, a = 0$$
$$n = 1, n > 1$$
$$x \in S, x \notin S$$

So far, however, we employ a general case-split into the cases *cond* and $\neg$*cond* only.

5.2 Meta-Reasoning for Repair of Constraint Handling

The problem Lim-Div is an example problem for which backtracking is guided by meta-reasoning on a highly desirable but blocked strategy. To the knowledge of the authors this is a problem that has not been proved by any other system. It states that the limit of the function $\frac{1}{x}$ at point c is $\frac{1}{c}$:

$$\forall \epsilon. (0 < \epsilon \Rightarrow \exists \delta. (0 < \delta \wedge \forall x. (|x - c| < \delta \wedge |x - c| > 0 \Rightarrow |\tfrac{1}{x} - \tfrac{1}{c}| < \epsilon)))$$

The decomposition of the initial complex goal by NormalizeGoal results in the two goals $0 < mv_\delta$ and $|\frac{1}{c_x} - \frac{1}{c}| < c_\epsilon$ (where mv_δ is a new meta-variable and c_x and c_ϵ are new constants). SolveInequality closes the first goal by an application of TellCS whereas it simplifies the second goal to $|\frac{c - c_x}{c_x * c}| < c_\epsilon$. An application of FactorialEstimate to this goal results in the three new goals $0 < mv_f$, $|c_x * c| > mv_f$, and $|c - c_x| < mv_f * c_\epsilon$ (with the new meta-variable

mv_f). SolveInequality closes these three goals with TellCS. This results in the proof plan tree for $|\frac{1}{c_x} - \frac{1}{c}| < c_\epsilon$ in figure 1, where $[|c_x - c| < mv_\delta]$ is an assumption that is created during the application of NormalizeGoal but is not used so far.

$$[|c_x - c| < mv_\delta]$$

$$\dfrac{\dfrac{\dfrac{}{0 < mv_f}\,\textsc{TellCS} \qquad \dfrac{}{|c_x * c| > mv_f}\,\textsc{TellCS} \qquad \dfrac{}{|c - c_x| < mv_f * c_\epsilon}\,\textsc{TellCS}}{|\frac{c-c_x}{c_x * c}| < c_\epsilon}\,\textsc{Factor.Est.}}{|\frac{1}{c_x} - \frac{1}{c}| < c_\epsilon}\,\textsc{Simplify}$$

Fig. 1. Initial proof plan tree for $|\frac{1}{c_x} - \frac{1}{c}| < c_\epsilon$.

Now all goals are closed and in the default behavior $\mathcal{CoSIE}$ is supposed to provide instantiations for the meta-variables mv_δ and mv_f. That is, the strategy ComputeInstFromCS, which asks $\mathcal{CoSIE}$ to compute the instantiations, becomes a highly desirable strategy. However, $\mathcal{CoSIE}$ fails to compute instantiations here and ComputeInstFromCS does not succeed. What is the problem? So far, $\mathcal{CoSIE}$ collected the constraints

$$\frac{|c_x - c|}{c_\epsilon} < mv_f,\ 0 < mv_f,\ mv_f < |c_x * c|,\ 0 < mv_\delta,\ 0 < c, \text{ and } 0 < c_\epsilon.$$

These constraints are consistent but a solution for mv_f exists only, if $\frac{|c_x - c|}{c_\epsilon} < |c_x * c|$ holds. This, however, does not follow from the collected constraints. In particular, the constraints collected so far are not sufficient for an ϵ-δ-proof since they do not establish a connection between c_ϵ and mv_δ. A possibility to overcome this problem is to refine the existing constraints in order to obtain an extended set of refined constraints for which a solution exists. That is, selected applications of TellCS (and only these selected applications) have to be backtracked in order to enable further refinement of some constraints.

The meta-reasoning pattern *Unblock Meta-Variable Instantiation* analyzes the failure and suggests its "repair". Technically, the pattern is realized in Multi by the strategic control rule `backtrack-to-unblock-cosie`. When all goals are closed, but the strategy ComputeInstFromCS is not applicable since the constraint solver fails to compute instantiations, then this control rule analyzes the constraints passed by applications of TellCS. It triggers the backtracking of actions of TellCS that pass inequalities to $\mathcal{CoSIE}$ that can be refined to simpler inequalities by applications of methods such as ComplexEstimate.[6]

[6] Currently, the critical constraints are chosen by heuristics encoded in `backtrack-to-unblock-cosie`. It would be more convenient, if $\mathcal{CoSIE}$ would directly point out what the critical constraints are. However, this kind of information is not provided by the $\mathcal{CoSIE}$

Then, these simpler inequality goals are may passed to the constraint solver.

In our example, `backtrack-to-unblock-cosie` triggers Multi to backtrack the application of TellCS that closes $|c - c_x| < mv_f * c_\epsilon$. Then, SolveInequality applies the method ComplexEstimate to the re-opened goal. This action uses the assumption $|c_x - c| < mv_\delta$ and reduces $|c - c_x| < mv_f * c_\epsilon$ to the new goals $|0| < \frac{c_\epsilon * mv_f}{2}$, $mv_\delta \leq \frac{c_\epsilon * mv_f}{2 * mv}$, $|-1| \leq mv$, and $0 < mv$ (where mv is a new meta-variable). Afterwards, TellCS passes the new inequality goals to $\mathcal{CoSIE}$. The resulting refined proof plan tree for $|c - c_x| < mv_f * c_\epsilon$ is given in figure 2.

$$\dfrac{[|c_x - c| < mv_\delta] \quad \dfrac{}{|0| < \frac{c_\epsilon * mv_f}{2}}\text{TellCS} \quad \dfrac{}{mv_\delta \leq \frac{c_\epsilon * mv_f}{2*mv}}\text{TellCS} \quad \dfrac{}{|-1| \leq mv}\text{TellCS} \quad \dfrac{}{0 < mv}\text{TellCS}}{|c - c_x| < mv_f * c_\epsilon}\text{ComplexEst.}$$

Fig. 2. Refined proof plan tree for $|c - c_x| < mv_f * c_\epsilon$.

Since $\mathcal{CoSIE}$ also fails on this extended constraint set the strategic control rule `backtrack-to-unblock-cosie` guides the backtracking of the application of TellCS that closes $|c_x * c| > mv_f$. Again, SolveInequality reduces the re-opened goal with ComplexEstimate. This action makes again use of the assumption $|c_x - c| < mv_\delta$ and reduces $|c_x * c| > mv_f$ to the new goals $|c * c| \geq mv_f * 2$, $mv_\delta \leq \frac{mv_f}{mv'}$, $|c| < mv'$, and $0 < mv'$ (where mv' is a new meta-variable). Afterwards, SolveInequality passes again the new inequality goals to $\mathcal{CoSIE}$ by applications of TellCS. Figure 3 depicts the resulting refined proof plan tree for $|c - c_x| < mv_f * c_\epsilon$.

$$\dfrac{[|c_x - c| < mv_\delta] \quad \dfrac{}{|c * c| \geq mv_f * 2}\text{TellCS} \quad \dfrac{}{mv_\delta \leq \frac{mv_f}{mv'}}\text{TellCS} \quad \dfrac{}{|c| < mv'}\text{TellCS} \quad \dfrac{}{0 < mv'}\text{TellCS}}{|c_x * c| > mv_f}\text{ComplexEst.}$$

Fig. 3. Refined proof plan tree for $|c_x * c| > mv_f$.

This results (after some $\mathcal{CoSIE}$-internal simplifications) in the following constraint store:

$c_\epsilon > 0$	$c > 0$	$mv_f \geq mv' * mv_\delta$	$mv' > c$
$mv_f > 0$	$mv > 1$	$\frac{c_\epsilon * mv_f}{2} > 0$	$mv_\delta > 0$
$mv_\delta \leq \frac{c_\epsilon * mv_f}{2*mv}$	$mv_f * 2 \leq c^2$		

Now the following bindings consistent with these constraints can be computed: $mv \rightarrow 2$, $mv' \rightarrow c + 1$, $mv_f \rightarrow \frac{c^2}{2}$, and $mv_\delta \rightarrow min(\frac{c_\epsilon * c^2}{8}, \frac{c^2}{2*(c+1)})$.

system yet.

All ϵ-δ-problems in which subgoals with fractions occur need to repair the constraint reasoning (see section 6). In other domains the same meta-reasoning to overcome blocked instantiations of constraint solvers is applicable.

5.3 Analyzing Meta-Variable Dependencies

As example for an ϵ-δ-problem that needs the analysis of meta-variable dependencies consider the following problem:

$$\lim_{x \to 0} f(a * x) = l \text{ follows from } \lim_{x_1 \to 0} f(x_1) = l \text{ and } a > 0.$$

Unfolding of the occurrences of limit and normalization result in the goal $|f(a * c_x) - l| < c_\epsilon$. Unwrapping the initial assumption yields the new assumption $|f(mv_{x_1}) - l| < mv_{\epsilon_1}$, which can be used to close the goal. Thereby, mv_{x_1} is instantiated by $a * c_x$. The unwrapping of the initial assumption also yields two goals, which become $|a * c_x| > 0$ and $|a * c_x| < c_{\delta_1}$ wrt. the instantiation $mv_{x_1} \mapsto a * c_x$. These two goals can be closed with two assumptions from the normalization of the initial theorem: $|c_x| > 0$ and $|c_x| < mv_\delta$. This works as follows: apply ComplexEstimate with the first assumption to the first goal and pass the resulting inequality goals to $\mathcal{CoSIE}$ and apply ComplexEstimate with the second assumption to the second goal and pass the resulting inequality goals to $\mathcal{CoSIE}$. Figure 4 sketches the resulting solution proof plan tree for the two goals $|a * c_x| > 0$ and $|a * c_x| < c_{\delta_1}$.

$$\dfrac{\dfrac{\ldots \qquad [|c_x| > 0]}{|a * c_x| > 0}\ \textsc{ComplexEst.} \qquad \dfrac{\ldots \qquad [|c_x| < mv_\delta]}{|a * c_x| < c_{\delta_1}}\ \textsc{ComplexEst.}}{|a * c_x| > 0 \land |a * c_x| < c_{\delta_1}}$$

Fig. 4. Sketch of solution proof with ComplexEstimate.

Since the control rule `prove-inequality` suggests the method Solve*-B before the method ComplexEstimate, Multi does not find this solution directly. Rather, Multi applies Solve*-B to the first goal $|a * c_x| > 0$ wrt. the second assumption $|c_x| < mv_\delta$. This is possible since $|c_x| < mv_\delta$ equals $mv_\delta > |c_x|$ and mv_δ can be trivially unified with $|a * c_x|$. This results in the instantiation $mv_\delta \mapsto |a * c_x|$ and the new goal $|c_x| > 0$, which equals the first assumption. Next, Multi tackles the second goal $|a * c_x| < c_{\delta_1}$ but fails. With the introduced instantiation of mv_δ the assumption $|c_x| < mv_\delta$ becomes $|c_x| < |a * c_x|$ and a solution of the goal $|a * c_x| < c_{\delta_1}$ with this assumption is not possible anymore. However, not the second goal is problematic in the end, but the instantiation of mv_δ introduced during the solution of the first goal. Figure 5 depicts the failing proof plan tree for the two goals $|a * c_x| > 0$ and $|a * c_x| < c_{\delta_1}$.

$$\cfrac{\cfrac{[|c_x| < mv_\delta] \qquad \cfrac{[|c_x| > 0]}{|c_x| > 0}\,\text{Same}}{|a * c_x| > 0}\,\text{Solve*-B} \qquad \cfrac{[|c_x| < |a * c_x|]}{|a * c_x| < c_{\delta_1}}\,???}{|a * c_x| > 0 \wedge |a * c_x| < c_{\delta_1}}$$

Fig. 5. Failing proof attempt with Solve*-B.

The meta-reasoning pattern *Analyze MV-Dependencies* analyzes the failure and suggests its "repair". Technically, the pattern is realized in Multi by the strategic control rule `prefer-constraints-deletion`, which guides the backtracking of steps that introduce instantiations or constraints for meta-variables instead of the default backtracking. In this case, the strategic control rule analyzes that the instantiation $mv_\delta \mapsto |a * c_x|$ is very unlikely to be part of a solution of an ϵ-δ-problem since the meta-variable for δ is supposed to be constrained during the proof planning process but not to be instantiated by different means than $\mathcal{C}o\mathcal{SIE}$. Hence, the control rule guides the backtracking of the Solve*-B step that closed the first goal. As result, Multi has to tackle the first goal differently, which finally results in the solution of both goals sketched above.

Note that the control rule `prefer-constraints-deletion` can also guide the successive trial and error of meta-variable instantiations. When Multi fails to solve a goal under a particular instantiation, then the instantiation of the meta-variable has to be backtracked (in order to try the next instantiation), rather than the goal for which Multi actually fails. The suitable backtracking is guided by `prefer-constraints-deletion`, which overwrites the default backtracking in Multi in this case. A domain, where `prefer-constraints-deletion` is used for such a trial and error of meta-variable instantiations are residue class problems (see [12,9]).

6 Empirical Results

Although our contribution is fundamentally conceptual and architectural, we had to show whether it is empirically relevant as well. Therefore, we tested the benefit in two domains, the ϵ-δ-proofs from the analysis textbook [1] and the residue class domain. Table 2 gives sample problems from the two domains and the failure-reasoning they require. Moreover, we included in Table 2 inductive proofs produced by the proof planner CLAM that also require failure reasoning.

ϵ-δ-proofs

The relevance of failure reasoning in this domain is not only demonstrated by Table 2. Its figures alone are underestimating because many similar problems can be formulated. Moreover, the relative frequency of failure reasoning is also important. Therefore, the fact that 22 out of 65 ϵ-δ-proofs constructed by Multi from the systematically explored testbed [1] involve failure reasoning evidences the crucial role of failure reasoning. In the appendix we give a complete list of all ϵ-δ-proofs constructed by Multi that require failure reasoning.

Conjecture	(i)	(ii)	(iii)
ϵ-δ-Proofs			
$\lim_{x\to 0}(f(a+x)-f(a))=0 \Rightarrow cont(f,a)$	x		x
$\lim_{x\to a^-} f(x)=l \wedge \lim_{x\to a^+} f(x)=l \Rightarrow \lim_{x\to a} f(x)=l$	x		
$\lim_{x\to a} f(x)=l_f \wedge \lim_{x\to a} g(x)=l_g \wedge \forall x_\blacksquare\, g(x)\neq 0 \Rightarrow \lim_{x\to a}\frac{f(x)}{g(x)}=\frac{l_f}{l_g}$		x	
$\lim_{x\to\infty} f(x)=l \Rightarrow \lim_{x\to\infty}\frac{f(x)}{x}=0$		x	
$\lim_{x\to 0} f(x)=l \wedge a>0 \Rightarrow \lim_{x\to 0} f(a*x)=l$			x
$\lim_{x\to a} f(x)=l \Rightarrow \lim_{x\to 0} f(x+a)=l$			x
Residue Class Problems			
$closed(\mathbb{Z}_3\backslash\{\bar{0}_3\},\bar{*})$			x
$\neg closed(\mathbb{Z}_3\backslash\{\bar{0}_3\},\bar{+})$			x
$\neg\exists e{:}\mathbb{Z}_{9\blacksquare}\, unit(\mathbb{Z}_9,\bar{-})$			x
$\neg inverses(\mathbb{Z}_6,\bar{*},\bar{1}_6)$			x
$\neg divisors(\mathbb{Z}_6,\bar{*})$			x
$\neg commutative(\mathbb{Z}_8,\bar{-})$			x
$\neg distibutive(\mathbb{Z}_4,\bar{-},\bar{-})$			x
Inductive Proofs			
$\forall x{:}_{item}\forall y,z{:}_{list}\; x\in y \Rightarrow x\in concatenate(y,z)$	x		
$\forall x{:}_{item}\forall y,z{:}_{list}\; (x\in y \vee x\in z) \Rightarrow x\in concatenate(y,z)$	x		
$\forall x{:}_{item}\forall y{:}_{list}\; x\in insert(x,y)$	x		
$\forall y{:}_{list}\; length(y)=length(isort(y))$	x		
$\forall x{:}_{item}\forall y{:}_{list}\; x\in isort(y)\Rightarrow x\in y$	x		
$\forall x{:}_{item}\forall y{:}_{list}\; count(x,isort(y))=count(x,y)$	x		

Table 2
Sample proofs whose solution requires meta-reasoning about failures. The numbered colons denote (i) case split introduction, (ii) unblock meta-variable instantiation, (iii) analyze meta-variable dependencies. Note that $x\to a^-$ and $x\to a^+$ denote the left-hand limit and the right-hand limit.

Residue Class Problems

The residue class conjectures classify given residue class structures wrt. their algebraic category. An example theorem is “the residue class structure

$(\mathbb{Z}_5, \bar{+})$ is associative". Other problems from this domain concern the isomorphy of two algebraic structures. An example is "the residue class structures $(\mathbb{Z}_5, \bar{+})$ and $(\mathbb{Z}_5, (x\bar{+}y)\bar{+}\bar{1}_5)$ are isomorphic".

To tackle residue class problems we developed several techniques encoded in four different method-introduction strategies in Multi. In one of these strategies, the TryAndError strategy (see [12]), the *Analyze MV-Dependencies* pattern is crucial since Multi has to deal with nested existential quantifiers, which result in 'nested' meta-variables shared by several goals. Hence, dependencies among the meta-variables and the goals have to be analyzed.

We proved about 19.000 residue class conjectures with Multi. About half of these theorems, in particular, theorems refutating a property, could be proved with the TryAndError strategy only (see [12] for detailed description of the experiments). Some representative examples occur in Table 2.

Inductive Proofs

So far, we did not apply Multi to inductive proofs. The inductive theorems in Table 2 are taken from [8], which describes failure reasoning by so-called critics in the proof planner CLAM. Since the critics employed in CLAM are a special case bound to a particular method (see related work in section 7), our general failure reasoning pattern for case-split introduction is applicable for inductive proofs as well. For a more complete list of inductive proofs that require failure reasoning see [8].

Discussion

Failure reasoning – as any control reasoning in Multi – can change the search space traversed by Multi. The actual effect of the change can vary: it can delete alternatives and prune the search space or it can introduce new alternatives that extend the search space. The three failure reasoning patterns described in this article all introduce some alternatives.

For instance, case-split introduction guided by failure reasoning introduces the new alternative case-split where otherwise Multi would perform backtracking. Thus, it extends the potential search space and can produce "overhead" as opposed to a run of Multi without the failure reasoning. Actually, this does not happen for the ϵ-δ-proofs currently solvable by Multi[7] but it happens for the (failing) application of Multi to non-theorems. As example non-theorem consider $\lim_{x\to a^-} f(x) = l \Rightarrow \lim_{x\to a} f(x) = l$, which is similar to the second problem

[7] That is, for none of the ϵ-δ-proofs currently solvable by Multi without failure reasoning, the search space traversed by Multi is extended by the failure reasoning.

in Table 2. When guided by failure reasoning, MULTI introduces for this non-theorem a case-split similar to the case-split for the second problem in Table 2 (see [11] for details). This considerably extends the search space traversed by MULTI for the non-theorem.

Since the knowledge engineering for proof planning is pretty difficult, the number of mathematical domains and problems successfully tackled by proof planning so far is growing only slowly. However, if not quantitatively then at least qualitatively, there is striking evidence for the need to meta-reason about failures in mathematics since the identified meta-reasoning patterns rely upon common techniques in mathematics. As evidence for this statement consider that failure reasoning in the proof planner CLAM (see related work in section 7) also exploits failures to guide the introduction of case-splits in a similar way but in a completely different mathematical domain, i.e., proving theorems by mathematical induction.

7 Conclusion and Related Work

We described three meta-reasoning patterns by which the multiple-strategy proof planner MULTI productively exploits failures to guide the subsequent proof planning process. They represent heuristics suggesting how to handle a failure that occurs in conjunction with a pattern of partially successful steps. The meta-reasoning patterns do not only circumvent failures, they hold the key to the construction of a solution proof plan.

The described failure reasoning and the repair modifications are possible since MULTI does not enforce a pre-defined systematic backtracking. Rather, when a failure occurs, then strategic control rules in which our heuristics are declaratively encoded can analyze the failure and can dynamically guide promising refinements and modifications of the proof plan. All the meta-reasoning patterns are generally applicable rather than over-specific as shown in the experiments (see section 6). Further meta-reasoning that exploits the flexible control in MULTI is discussed in [9].

Related Work

Failure Reasoning in CLAM

Failure reasoning in the proof planner CLAM [15] is closely related to the introduction of case-splits and lemmas in MULTI. In [7] and [8], Ireland and Bundy describe critics as a means to patch failed proof attempts in CLAM by exploiting information on failures. The motivation for the introduction of critics is similar to our motivation for failure reasoning: failures in the proof planning process often hold the key to discover a solution proof plan.

Critics in CLAM extend the hierarchy of inference rules, tactics, and methods. A critic is associated with one method – mostly with the wave method – and captures patchable exceptions to the application of this method. Critics are expressed in terms of preconditions and patches. The preconditions analyze the reasons why the method has failed to apply. The patch suggests a change to the proof plan.

The situations that trigger case-split introduction and lemma speculation in CLAM and Multi are very similar: unprovable premises of conditional facts from the context trigger case-split introduction, whereas missing premises in the current context trigger lemma speculation. However, the critics mechanism in CLAM and failure reasoning in Multi considerably differ not only in minor technical issues but also in their conceptual design. Critics are a method-like entity directly bound to failing preconditions of a particular method. Moreover, part of a critic is a patch of the failure, which is a special procedure that changes the proof plan. In contrast, failure reasoning in Multi is conducted by declarative and separate control rules. These control rules are not associated with a particular method but rather test for particular situations that can occur during the proof planning process (independent of the strategy or method that caused the situation). The control rules can reason about the current proof plan and about other information such as the history. The patch of a failure is not implemented into special procedures but is carried out by methods and strategies whose application is suggested by the control rules.

Dynamic Backtracking

Typically, backtracking methods return to prior points in the spanned search tree and thereby often erase meaningful progress towards a solution. As opposed thereto, Multi enables the backtracking of selected steps (and all steps that explicitly depend on them). This can result in a new proof plan not in the search tree traversed so far. In [6] Ginsberg describes a backtracking approach for the solution of constraint satisfaction problems that is similar to Multi's. His approach also enables the deletion of selected steps without removing all steps introduced after these steps (provided that these other steps do not explicitly depend on the steps selected for deletion). He uses the term "dynamic" backtracking because of the dynamic way in which the search is structured.

Meta-Reasoning in Blackboard Systems

Related to the unblocking of desirable steps in Multi is the control reasoning in elaborate blackboard systems, e.g., see [4] and [5]. When a highly desirable knowledge source is not applicable, then reasoning on the failure can suggest the invocation of knowledge sources that unblock the desired knowledge source.

References

[1] Bartle, R. and D. Sherbert, "Introduction to Real Analysis," John Wiley& Sons, New York, 1982.

[2] Bledsoe, W., *Challenge Problems in Elementary Analysis*, Journal of Automated Reasoning **6** (1990), pp. 341–359.

[3] Bundy, A., *The Use of Explicit Plans to Guide Inductive Proofs*, in: *Proceedings of CADE–9*, LNCS **310** (1988), pp. 111–120.

[4] Corkill, D., V. Lesser and E. Hudlicka, *Unifying Data-Directed and Goal-Directed Control*, in: *Proceedings of AAAI-82* (1982), pp. 143 – 147.

[5] Durfee, E. and V. Lesser, *Incremental Planning to Control a Blackboard-Based Problem Solver*, in: *Proceedings of AAAI-86* (1986), pp. 58 – 64.

[6] Ginsberg, M., *Dynamic Backtracking*, Journal of Artificial Intelligence Research **1** (1993), pp. 25—46.

[7] Ireland, A., *The Use of Planning Critics in Mechanizing Inductive Proofs*, in: *Proceedings of LPAR'92*, LNAI **624** (1992), pp. 178–189.

[8] Ireland, A. and A. Bundy, *Productive Use of Failure in Inductive Proof*, Journal of Automated Reasoning **16** (1996), pp. 79–111.

[9] Meier, A., "Multi – Proof Planning with Multiple Strategies," Ph.D. thesis, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken (2004).

[10] Meier, A., *The Proof Planners of* ΩMEGA*: A Technical Description*, Seki Report SR-04-03, FR Informatik, Saarland University, Saarbrücken, Germany (2004).

[11] Meier, A. and E. Melis, *Proof Planning Limit Problems with Multiple Strategies*, Seki Report SR-04-04, FR Informatik, Saarland University, Saarbrücken, Germany (2004).

[12] Meier, A., M. Pollet and V. Sorge, *Comparing Approaches to Explore the Domain of Residue Classes*, Journal of Symbolic Computation **34** (2002), pp. 287–306.

[13] Melis, E. and A. Meier, *Proof Planning with Multiple Strategies*, in: *Proceedings CL-2000*, LNAI **1861** (2000), pp. 644–659.

[14] Melis, E. and J. Siekmann, *Knowledge-Based Proof Planning*, Artificial Intelligence **115** (1999), pp. 65–105.

[15] Richardson, J., A. Smaill and I. Green, *System description: Proof planning in higher-order logic with λClam*, in: *Proceedings of the 15th International Conference on Automated Deduction (CADE–15)*, LNAI **1421** (1998), pp. 129–133.

[16] Schoenfeld, A., "Mathematical Problem Solving," Academic Press, New York, 1985.

[17] Siekmann, J., C. Benzmüller, V. Brezhnev, L. Cheikhrouhou, A. Fiedler, A. Franke, H. Horacek, M. Kohlhase, A. Meier, E. Melis, M. Moschner, I. Normann, M. Pollet, V. Sorge, C. Ullrich, C. Wirth and J. Zimmer, *Proof Development with OMEGA*, in: *Proceedings of CADE–18*, number 2392 in LNAI (2002), pp. 144–149.

[18] Zimmer, J. and E. Melis, *Constraint solving for proof planning*, Journal of Automated Reasoning (2004), accepted.

Appendix: ϵ-δ-Proofs Requiring Failure Reasoning

Case-Split Introduction:

(i) If function f has the limit $f(a)$ at a, then f is continuous at a:
$\lim_{x\to a} f(x) = f(a) \Rightarrow \mathit{cont}(f,a)$.

(ii) (Cont-If-Deriv: Theorem 6.1.2 in [1])
If function f has a derivative at a, then f is continuous at a:
$\mathit{deriv}(f,a) = f' \Rightarrow \mathit{cont}(f,a)$.

(iii) (Theorem 4.3.3, second part, in [1])
If function f has the left-hand limit l and the right-hand limit l at a, then f has the limit l at a:
$\lim_{x\to a^-} f(x) = l \wedge \lim_{x\to a^+} f(x) = l \Rightarrow \lim_{x\to a} f(x) = l$

(iv) If function f has the left-hand limit $f(a)$ and the right-hand limit $f(a)$ at a, then f is continuous at a:
$\lim_{x\to a^-} f(x) = f(a) \wedge \lim_{x\to a^+} f(x) = f(a) \Rightarrow \mathit{cont}(f,a)$

(v) If function f has the left-hand limit l_l and the right-hand limit l_r at a, then f is bounded in a neighborhood of a:
$\lim_{x\to a^-} f(x) = l_l \wedge \lim_{x\to a^+} f(x) = l_r \Rightarrow$
$(\exists \delta_{\bullet} \exists M_{\bullet}\ 0 < \delta \wedge 0 < M \wedge (\forall x_{\bullet} (|x_1 - a| > 0 \wedge |x_1 - a| < \delta_1) \Rightarrow |f(x)| \leq M))$

(vi) If function $f(a+x) - f(a)$ has limit 0 at 0, then f is continuous at a:
$\lim_{x\to 0}(f(a+x) - f(a)) = 0 \Rightarrow \mathit{cont}(f,a)$

Unblock Meta-Variable Instantiation:

(i) (Theorem 3.2.3.b in [1])
If sequence $X = (x_n)$ has the limit l_x and sequence $Y = (y_n)$ has the limit $l_y \neq 0$ and $y_n \neq 0$ for all n, then sequence $\frac{X}{Y} = (\frac{x_n}{y_n})$ has the limit $\frac{l_x}{l_y}$:
$\mathit{limseq}\ X = l_x \wedge \mathit{limseq}\ Y = l_y \wedge \forall n_{\bullet} y_n \neq 0 \Rightarrow \mathit{limseq}\ \frac{X}{Y} = \frac{l_x}{l_y}$

(ii) (Example 4.1.7.d in [1])
The function $f(x) = \frac{1}{x}$ has the limit $\frac{1}{a}$ at a, if $a > 0$:
$a > 0 \Rightarrow \lim_{x\to a} \frac{1}{x} = \frac{1}{a}$

(iii) (Example 4.1.7.e in [1])
$\lim_{x\to 2} \frac{x^3-4}{x^2+1} = \frac{4}{5}$

(iv) (Exercise 4.1.10.a in [1])
$\lim_{x\to 2} \frac{1}{1-x} = -1$

(v) (Exercise 4.1.10.b in [1])
$\lim_{x\to 1} \frac{x}{1+x} = \frac{1}{2}$

(vi) (Exercise 4.1.10.c in [1])
$\lim_{x\to 0} \frac{x^2}{|x|} = 0$

(vii) (Exercise 4.1.10.d in [1])
$\lim_{x\to 1} \frac{x^2-x+1}{x+1} = \frac{1}{2}$

(viii) (Theorem 4.2.4.b in [1])
If f has limit l_f at a and g has limit $l_g \neq 0$ at a and $g(x) \neq 0$ for all x, then $\frac{f}{g}$ has limit $\frac{l_f}{l_g}$ at a:
$\lim_{x\to a} f(x) = l_f \wedge \lim_{x\to a} g(x) = l_g \wedge \forall x\text{.}\; g(x) \neq 0 \Rightarrow \lim_{x\to a} \frac{f(x)}{g(x)} = \frac{l_f}{l_g}$

(ix) (Theorem 5.2.1.b in [1])
If f is continuous at a and g is continuous at a and $g(x) \neq 0$ for all x, then $\frac{f}{g}$ is continuous at a:
$\mathit{cont}(f,a) \wedge \mathit{cont}(g,a) \wedge \forall x\text{.}\, g(x) \neq 0 \Rightarrow \mathit{cont}(\frac{f}{g}, a)$

(x) (Theorem 6.1.3.a in [1])
If f has the derivative f' at a, then $a * f$ has the derivative $a * f'$ at a:
$\mathit{deriv}(f,a) = f' \Rightarrow \mathit{deriv}(a * f, a) = a * f'$

(xi) (Theorem 6.1.3.b in [1])
If f has the derivative f' at a and g has the derivative g' at a, then $f+g$ has the derivative $f'+g'$ at a:
$\mathit{deriv}(f,a) = f' \wedge \mathit{deriv}(g,a) = g' \Rightarrow \mathit{deriv}(f+g,a) = f'+g'$

(xii) (Theorem 6.1.3.c in [1])
If f has the derivative f' at a and g has the derivative g' at a, then $f * g$ has the derivative $f' * g(a) + f(a) * g'$ at a:
$\mathit{deriv}(f,a) = f' \wedge \mathit{deriv}(g,a) = g' \Rightarrow \mathit{deriv}(f * g, a) = f' * g(a) + f(a) * g'$

(xiii) (Cont-If-Deriv: Theorem 6.1.2 in [1])
If f has a derivative at a, then f is continuous at a:
$\mathit{deriv}(f,a) = f' \Rightarrow \mathit{cont}(f,a)$

(xiv) If function $f(x)$ has limit l as $x \to \infty$, then $\frac{f(x)}{x}$ has limit 0 as $x \to \infty$:
$\lim_{x\to\infty} f(x) = l \Rightarrow \lim_{x\to\infty} \frac{f(x)}{x} = 0$

Analyse Meta-Variable Dependencies:

(i) (Exercise 4.1.12 in [1])
If $f(x)$ has limit l at 0 and $a > 0$, then $f(a * x)$ has the limit l at 0:
$\lim_{x\to 0} f(x) = l \wedge a > 0 \Rightarrow \lim_{x\to 0} f(a * x) = l$

(ii) (Exercise 4.1.3 first part in [1])

If the function $f(x)$ has the limit l at a, then the function $f(x+a)$ has the limit l at 0:
$\lim\limits_{x \to a} f(x) = l \Rightarrow \lim\limits_{x \to 0} f(x+a) = l$

(iii) If function $f(a+x) - f(a)$ has limit 0 at 0, then f is continuous at a:
$\lim\limits_{x \to 0}(f(a+x) - f(a)) = 0 \Rightarrow cont(f, a)$