Impasse-Driven Reasoning in Proof Planning

Andreas Meier and Erica Melis

German Research Center for Artificial Intelligence (DFKI),
Saarbrücken, Germany
{ameier, melis}@dfki.de

Abstract. In a problem solving process, a step may not result in the expected progress or may not be applicable as expected. Hence, knowledge how to overcome and react to impasses and other failures is an important ingredient of successful mathematical problem solving. To employ such knowledge in a proving system requires a variety of behaviors and a flexible control. Multi-strategy proof planning is a knowledge-based theorem proving approach that provides a variety of strategies and knowledge-based guidance for search at different levels. This paper introduces reasoning about impasses as a natural ingredient of meta-reasoning at a strategic level and illustrates the use of knowledge about failure handling in the proof planner MULTI.

1 Introduction

The typical proof search in automated theorem provers relies upon local search criteria mostly referring to syntactic features of the current goal and assumptions rather than analyzing a proof situation more globally. However, in order to find a mathematical proof, more often than not the global context has to be observed, e.g., the theory, the proof history, and different proof strategies.

Humans are able to employ such information in their theorem proving. When an expected progress does not occur or when the proof process gets stuck, then an intelligent problem solver analyzes the failure and attempts a new strategy. As Schoenfeld suggests in his book on mathematical problem solving [15] "*monitoring the state of a solution as it evolves and taking appropriate action in the light of new information*" is a key skill for succeeding. This means, intelligent humans do not rely upon pre-determined control only to guide their problem solving. Instead, they draw upon a repertoire of heuristic knowledge how to deal with different situations and to dynamically guide the solution construction.

Similarly, an automated theorem proving system can monitor the solution process and can employ heuristic knowledge to reason about failed proof attempts. This requires the system to have several problem solving strategies and a flexible control, which can be guided by meta-reasoning about the overall problem solving situation.

The multi-strategy proof planner MULTI offers such a knowledge-based meta-reasoning and handles failed proof attempts as we will show in this paper. We

M. Kohlhase (Ed.): MKM 2005, LNAI 3863, pp. 143–158, 2006.

describe several examples of meta-reasoning rules that analyze and exploit failures to guide proof plan manipulations and refinements. Conceptually, the failure reasoning is supported by the architecture of MULTI, which clearly separates reasoning at different levels and provides refinement and modification strategies that can be flexibly combined. In MULTI, failure reasoning is a natural ingredient of control reasoning at a strategic level.

The paper is organized as follows. First, we briefly describe proof planning with multiple strategies and its realization in the MULTI system. Afterwards, we motivate the research on failure reasoning with an example. Section 4 introduces failure reasoning captured in general meta-reasoning rules. The role and use of failure reasoning in MULTI is illustrated for ϵ-δ-proofs in section 5. Although we use ϵ-δ-proofs for illustration throughout the paper and explain the failure reasoning with ϵ-δ-proofs, this meta-reasoning is general and 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 Background: Proof Planning with Multiple Strategies

Proof planning [4] is a theorem proving technique, which plans a proof at the abstract level of *methods*, i.e., tactics enriched by explicit pre- and postconditions. Methods result from the analysis of the common structure or common procedures of a family of proofs. They can encode not only general proof steps but also steps particular to a mathematical domain. Mathematically motivated heuristics how to proceed are encoded in the control knowledge needed to search for the sequence or hierarchy of methods that results in a solution plan. Knowledge-based proof planning [14] declaratively represents control knowledge as control rules. They are evaluated at choice points in the planning process (choice of method, choice of goal, etc.).

Knowledge-based proof planning also allows to integrate (theory-specific) constraint solving [16] and other (theory-specific) external solvers, among others for the construction/instantiation of mathematical objects (e.g., the construction of a real number that satisfies certain restrictions). For instance, proof planning for ϵ-δ-proofs delegates simple equations and inequalities containing variables to the constraint solver $\mathcal{CoSIE}$, which checks the (in)consistency of the constraints and collects consistent constraints. Thereby, the variables act as place holders for still unknown terms, and $\mathcal{CoSIE}$ can compute instantiations for the variables that satisfy the collected constraints. Such place holder variables are marked with a superscript ^{p} throughout the paper.

Simple proof planning searches at the level of methods, i.e., it searches for applicable methods and introduces the instantiated methods in the proof plan under construction until all goals are closed. Typically, the planning is monolithic in the sense that functionalities such as backtracking and instantiation of variables are part of the planning and their control is hard-coded:

> Backtrack one step in the plan, if and only if no method is applicable.
> Instantiate variables only at the end, when all goals are closed.

Multi-Strategy proof planning [13] extends proof planning by the additional hierarchical level of strategies as well as by strategic control. It allows to flexibly combine refinement and modification algorithms. The instantiation of the algorithms' parameters produces strategies, which can realize different behaviors of the algorithm. Typical strategies are those running the algorithms for method introduction, variable instantiation, and backtracking, i.e., decoupled and parameterized functionalities of the simple proof planning, realizing different kinds of method introduction, backtracking and variable instantiation.

The parameters of the algorithm for method introduction include a set of methods and a set of control rules. When such a strategy is executed, then the algorithm introduces only steps that use the methods specified in the strategy. The method-level control belongs to the strategy. That is, its choices are guided by the control rules specified in the strategy. A parameter of the instantiation algorithm is the function that determines how the instantiation for a variable is computed. A parameter of the backtrack algorithm is the function that computes a set of refinement steps that will be deleted from the partial proof plan.

Let us explain some strategies frequently occurring in ϵ-δ-proofs, i.e., in proofs of conjectures about the limit or the continuity of a function f at a point a.

The standard definitions of limit, continuity, and the derivative of a function postulate the existence of a real number δ, which may depend on an arbitrarily chosen real number ϵ:

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

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

A mathematical ϵ-δ-proof of such a problem constructs a real number δ depending on ϵ that satisfies certain (in)equalities.[1] A systematic procedure for discovering a suitable δ is the incremental restriction of its range. This includes the reduction of complex (in)equalities to less complex ones, the simplest of which can be propagated to range restrictions, and the determination of terms which satisfy all the restrictions.

The mathematical strategies are mirrored for proof planning. The method-introduction strategy SolveInequality corresponds to the reduction of complex inequalities to simple ones. It successively produces simpler (in)equalities until it reaches (in)equalities that are accepted by the constraint solver $\mathcal{CoSIE}$. Its methods ComplexEstimate, FactorialEstimate, and Solve* reduce (in)equality goals of different kinds. The connection to $\mathcal{CoSIE}$ is established by the method TellCS, which closes inequalities and passes them to $\mathcal{CoSIE}$.

[1] The construction of a δ 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+ (see next section) as a challenge problem for automated theorem proving [3]. The simplest versions of this problem (problem 1 and 2 in [3]) 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 (see next section) cannot be proved by traditional provers.

ComplexEstimate reduces inequality goals of the form $|b| < e$. It exploits the triangle inequality by rewriting $b = k * a + l$ for an a for which an assumption of the form $|a| < e'$ is in the proof context and postulates for the existence of a real number v^p serving as an auxiliary variable. The resulting simpler goals are $|l| < \frac{e}{2}$, $e' < \frac{e}{2*v^p}$, $|k| \leq v^p$, and $0 < v^p$.

The method FactorialEstimate reduces inequality goals of the form $|\frac{t}{t'}| < e$. It also postulates for the existence of a real number v^p serving as an auxiliary variable and creates three simpler goals: $0 < v^p$, $v^p < |t'|$, and $|t| < e * v^p$.

Applications of Solve* exploit the transitivity of $<, >, \leq, \geq$ and reduce a goal of the form $a_1 < b_1$ to a new goal $b_2\sigma \leq b_1\sigma$ in case an assumption of the form $a_2 < b_2$ can be used and a_1, a_2 can be unified by the substitution σ.

Two other domain-independent method introduction strategies prepare composed conjectures and proof assumptions to make relevant subformulas available for the methods of SolveInequality.

The instantiation strategy ComputeInstFromCS corresponds to the actual construction of mathematical objects (real numbers). It instantiates variables that occur in the constraints collected by the constraint solver $\mathcal{CoSIE}$. ComputeInstFromCS is applicable, when $\mathcal{CoSIE}$ can compute an instantiation for a variable that is consistent with the constraints collected so far.

The domain-independent backtrack strategy BackTrackStepToGoal realizes a simple goal-triggered backtracking. When applied wrt. to a goal it removes the step that introduced this goal.

The Implementation in Multi

The implementation of proof planning with multiple strategies in Multi [11] works with two blackboards, a (object-level) proof plan blackboard and a control blackboard. During their execution, the strategies change the proof plan blackboard content and a Meta-Reasoner changes the control blackboard to guide the selection of strategies. To do so, it evaluates the strategic control knowledge, which is declaratively represented by strategic control rules. Hence, there is no need to hard-code the sequence of strategies. In a nutshell, Multi operates according to the following cycle:

Job Offers. Applicable strategies post their applicability (for the current partial plan) as 'job offers' onto the control blackboard.
Guidance. Strategic control rules are evaluated to rank the job offers in the light of situation information.
Invocation. The strategy with the highest ranked job offer is invoked.
Execution. The strategy is executed and works on the proof plan blackboard.

3 Motivating Examples

We contrast the proof planning for two ϵ-δ-proofs: for LIM+ (without impasse) and for Cont-If-Deriv, which encounters an impasse.

LIM+ states that the limit of the sum of two functions f and g equals the sum of their limits:

$$\text{if } \lim_{x \to a} f(x) = l_1 \text{ and } \lim_{x \to a} g(x) = l_2, \text{ then } \lim_{x \to a}(f(x) + g(x)) = l_1 + l_2.$$

When the definition of *lim* is expanded, the proof planning problem consists of two assumptions

$$\forall \epsilon_1 (0 < \epsilon_1 \Rightarrow \exists \delta_1 (0 < \delta_1 \wedge \forall x_1 (|x_1 - a| > 0 \wedge |x_1 - a| < \delta_1 \Rightarrow |f(x_1) - l_1| < \epsilon_1)))$$

and

$$\forall \epsilon_2 (0 < \epsilon_2 \Rightarrow \exists \delta_2 (0 < \delta_2 \wedge \forall x_2 (|x_2 - a| > 0 \wedge |x_2 - a| < \delta_2 \Rightarrow |g(x_2) - l_2| < \epsilon_2)))$$

and the conjecture

$$\forall \epsilon (0 < \epsilon \Rightarrow \exists \delta (0 < \delta \wedge \forall x (|x - a| > 0 \wedge |x - a| < \delta \Rightarrow |(f(x) + g(x)) - (l_1 + l_2)| < \epsilon))).$$

Proof planning LIM+ first decomposes the conjecture and assumptions. This results, among others, in the two new assumptions $|f({x_1}^p) - l_1| < {\epsilon_1}^p$ and $|g({x_2}^p) - l_2| < {\epsilon_2}^p$ and the new goals $0 < \delta^p$ and $|(f(c_x) + g(c_x)) - (l_1 + l_2)| < c_\epsilon$, where c_x and c_ϵ are constants that replace x and ϵ, respectively.[2] ϵ_1, x_1, ϵ_2, x_2, and δ become place holder variables (labeled with the ^{p} superscript) for which $\mathcal{CoSIE}$ can collect constraints. Both goals are tackled by the SolveInequality strategy. The goal $0 < \delta^p$ is closed by TellCS and passed to $\mathcal{CoSIE}$. The second goal $|(f(c_x) + g(c_x)) - (l_1 + l_2)| < c_\epsilon$ requires further decomposition by ComplexEstimate, which employs the new assumption $|f({x_1}^p) - l_1| < {\epsilon_1}^p$ and yields five new goals:

$$\epsilon_1 < \frac{c_\epsilon}{2 * v^p} \quad (1) \qquad |g(c_x) - l_2| < \frac{c_\epsilon}{2} \quad (4)$$

$$|1| \leq v^p \quad (2) \qquad {x_1}^p = c_x. \quad (5)$$

$$0 < v^p \quad (3)$$

(1), (2), (3), and (5) can be closed by TellCS and are passed to $\mathcal{CoSIE}$. Goal (4) is reduced by Solve* wrt. the assumption $|g({x_2}^p) - l_2| < {\epsilon_2}^p$ to the goals ${\epsilon_2}^p \leq \frac{c_\epsilon}{2}$ and ${x_2}^p = c_x$, which can both be closed by TellCS. The decomposition of the assumptions on f and g results in some further goals, which are all solved by SolveInequality.

When all goals are closed, the constraint solver $\mathcal{CoSIE}$ computes instantiations for the variables that are consistent with the collected constraints.

[2] During the decomposition of the assumptions further goals are created and during the decomposition of the conjecture further assumptions are derived. However, in order to illustrate the basic proof planning approach we ignore these additional goals and assumptions.

Cont-If-Deriv states that, if a function f has a derivative f' at point a, then f is continuous at a:

if $\lim_{x \to a} \frac{f(x)-f(a)}{x-a} = f'$, then $cont(f, a)$.

When the definitions of *lim* and *cont* are expanded, the proof planning problem consists of the assumption

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

and the conjecture

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

Proof planning for Cont-If-Deriv fails because of a typical exception. More detailed, the proof planning goes as follows: As for LIM+ the initial conjecture and assumption are decomposed. The main resulting goal is

$$|f(c_x) - f(a)| < c_\epsilon \tag{6}$$

and a new assumption is

$$|\frac{f({x_1}^p) - f(a)}{{x_1}^p - a} - f'| < {\epsilon_1}^p. \tag{7}$$

Using this assumption the goal (6) can be proved in several steps.[3] However, when decomposing the initial assumption, the goal $|c_x - a| > 0$ was created as a side goal and cannot be proved. This gives rise to an impasse because the goal-triggered backtracking with strategy BackTrackStepToGoal does not lead to a solution. This impasse is not a problem of missing methods. Rather, since the condition $|c_x - a| > 0$ is not always true, mathematically it is necessary to consider the cases $|c_x - a| > 0$ and $|c_x - a| \leq 0$. That is, $|f(c_x) - f(a)| < c_\epsilon$ has to be proved twice, once under the condition $|c_x - a| > 0$ and once under the condition $|c_x - a| \leq 0$. Because the impasse information is surfaced only at a later stage of the proof planning, its analysis should lead to a conclusion on how to modify (i.e., not just refine) the overall proof plan in order to circumvent the exception.

4 Meta-reasoning About Failed Proof Attempts

This and many other examples show that proof planning may encounter impasses, i.e., situations in which an open goal cannot be closed because there are no applicable methods or strategies or in which no instantiation for a variable can be found.

Impasses in multi-strategy proof planning may occur at two levels, inside strategies or at strategy choice points. When an impasse occurs during the

[3] That is, ComplexEstimate is applied to (6) with assumption (7) and all resulting goals can again be solved by proof planning.

execution of a strategy, the strategy interrupts and the failure is recorded. When the next strategy has to be selected for execution, the strategic control rules can reason about the failures. When an impasse occurs at strategy choice, then the failing applicability of a strategy is recorded together with the failure reasons and the strategic control rules can reason upon.

Most systems have a default behavior that is called, when such errors occur. So has MULTI. For instance, the default behavior to deal with failures in the method introduction algorithm is the goal-triggered backtracking with the strategy BackTrackStepToGoal (i.e., to backtrack the step that introduced the goal for which no applicable method can be found). However, the default reaction is only one of a variety of possible reactions to failures that is evaluated in MULTI. An evaluation of a repertoire of alternative reactions to failures is useful for many reasons, two of which are:

(1) Theorem proving may require refinements, modifications, or additional information, which are hard to predict from common proof patterns since they are exceptions. Some heuristics can exploit failures since in some cases these failures hold information that is necessary in order to discover a solution plan at all.
(2) A (knowledge-based) search and backtracking procedure that is generally suitable for different mathematical domains including domains that require higher-order formalizations is difficult to devise. Rather, knowledge of suitable failure handling and backtrack points in different domain-dependent proof situations can be used to organize the search.

Hence, in MULTI failure handling is not hard-coded once and forever. Rather, the analysis of frequent failures and possible reactions results in general (and informal) meta-reasoning rules, which can be formalized in control rules that MULTI can use. These control rules analyze information about the failure situation, the current proof plan, the history, etc. and suggest suitable proof plan modifications and refinements. In the remainder of this section, we shall introduce three general meta-reasoning rules. In the subsequent section, we shall illustrate their encoding in control rules in MULTI as well as their application to ϵ-δ-proofs.

4.1 Case Split Introduction

Case split is well-known in mathematics. More often than not, it is not obvious in advance, when it is useful to apply a case split and which cases to consider. The following rule describes the need for a case split at an abstract level.

A main goal can be solved by some methods which introduces side goals, called 'conditions'. In a situation where one of these conditions cannot be proved (while the main goal is solved), an impasse is reached. The impasse can be removed by introducing a case split on the failing condition and its negation earlier in the plan. The modification requires to prove the main goal in each of the cases.

This description is represented by the general meta-reasoning rule

Case-Split Introduction:
IF failing condition C while some methods solve main goal
THEN introduce case split $C \vee \neg C$ before application of methods

In section 5, we shall explain how this meta-reasoning rule is encoded into control rules and how the main goal and the side goals are determined.

4.2 Analysis of Variable Dependencies

Goals sharing the same variable but belonging to different proof plan branches are dependent. The instantiations and constraints of those variables may cause failures. Take, e.g., two goals g and g' that both contain a variable v^p. Lets assume that a partial proof plan for g is created, which binds v^p in such a way that g' cannot be proved anymore. The default reaction – standard goal-triggered backtracking – would remove g'. However, if the problem for the failure is not g' but the selection of an appropriate instantiation for v^p, then this backtracking will not lead to a solution proof plan. Rather, part of the subplan for g has to be removed to introduce another subplan that constrains v^p differently.

This heuristic gives rise to the general meta-reasoning rule:

Analyze VarDependencies:
IF failure on goal caused by variable instantiation/constraints
THEN backtrack variable instantiation/constraints

Again, this rule can be represented by declarative control rules. Its application to ϵ-δ-proofs is illustrated in section 5.

4.3 Unblock Desirable Steps

Often classes of proofs exhibit a common proof pattern, which suggests a particular hierarchy or combination of proof steps. This implies that during the solution process particular proof steps become 'desirable', i.e., the proof pattern suggests to apply these steps next. If such a desirable step is blocked, then meta-reasoning can analyze how to unblock the application of the desirable step. In its most general form, this meta-reasoning rule can be formulated as

Unblock Desirable Steps:
IF step S is desirable but blocked
THEN perform other steps to enable S

In section 5, we shall discuss two instances of this general rule, which both rely on the common proof pattern for ϵ-δ-proofs introduced in section 2 (i.e., their determination of desirable steps is wrt. to the proof pattern for ϵ-δ-proofs).

The first instance analyzes a blockage of the instantiation strategy ComputeInstFromCS caused by insufficiently constrained variables. To overcome this impasse the meta-reasoning suggests actions to enable the collection of further constraints. The second instance analyzes failing method applications and suggests the speculation of a lemma that would make a desirable method applicable.

5 Examples from the Domain of ϵ-δ-Proofs

To illustrate the failure reasoning we detail some of it for ϵ-δ-proofs – a domain of mathematics for which up to now the proofs are not trivially automated. As the empirical results in section 6 show, the same meta-reasoning is also applicable to other domains.

5.1 Guiding the Introduction of Case Splits

The proof of Cont-If-Deriv is an example in which the introduction of a case split is necessary. As described in section 3 proof planning fails to prove the condition $|c_x - a| > 0$ that was created as a side goal when decomposing the initial assumption. At this point the meta-reasoning rule *Case Split Introduction* suggests the introduction of a case split earlier in the proof plan.

Technically, the meta-reasoning rule *Case Split Introduction* is formalized in two control rules in MULTI that 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 a strategic control rule triggers the backtracking of all steps following the introduction of the failing condition. Afterwards, another control rule introduces the case split with the failing condition and its negation.

When proof planning encounters the impasse for Cont-If-Deriv (see section 3), the case split is introduced before the main goal $|f(c_x) - f(a)| < c_\epsilon$ is tackled. Then, SolveInequality continues and has to prove $|f(c_x) - f(a)| < c_\epsilon$ once enriching the context with $|c_x - a| > 0$ and once enriching the context with $\neg(|c_x - a| > 0)$. In the first case the failing condition $|c_x - a| > 0$ follows from the context of the case. The second case is proved 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 the context.

Other ϵ-δ-proofs also require this kind of failure reasoning and the same failure handling can also be used to introduce case splits in other mathematical domains (see section 6).

5.2 Analyzing Variable Dependencies

The meta-reasoning rule *Analyze VarDependencies* analyzes dependencies of goals that share some variables – either directly in their formulas or in the assumptions in their contexts. The meta-reasoning rule is encoded in the strategic control rule `analyze-varfailure`. When a goal with shared variables cannot be closed,

`analyze-varfailure` analyses whether sub-plans for other goals with these variables introduce constraints on the variables. If this is the case, it guides the backtracking of those steps that introduced such constraints rather than to employ the standard goal-triggered backtracking, which would backtrack the goal that cannot be solved. Afterwards, re-opened goals can be solved differently.

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

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

The decomposition of the initial goal and the initial assumption yield the goal $|f(a * c_x) - l| < c_\epsilon$ and the new assumption $|f({x_1}^p) - l| < {\epsilon_1}^p$. SolveInequality solves the goal with this assumption by an application of the method Solve*.

The decomposition of the initial assumption also results in the two goals

$$|a * c_x| > 0 \tag{8}$$

$$|a * c_x| < c_{\delta_1} \tag{9}$$

These two goals can be solved with two assumptions from their context

$$|c_x| > 0 \tag{10}$$

$$|c_x| < \delta^p \tag{11}$$

which were created during the decomposition of the initial theorem.

When tackling these two goals SolveInequality first proceeds as follows: It applies Solve* to goal (8) wrt. the assumption (11). This is possible since $|c_x| < \delta^p$ equals $\delta^p > |c_x|$ and δ^p can be trivially unified with $|a * c_x|$. The application of Solve*, however, introduces the constraint $\delta^p \mapsto |a * c_x|$ on δ^p, which affects the assumption (11) in the context of goal (9). Next, SolveInequality tackles goal (9) but fails, since with the constraint on δ^p no solution is possible (i.e., assumption (11) cannot be used as necessary, see below).

Guided by `analyze-varfailure` Multi first backtracks the application of Solve* to goal (8). Afterwards, SolveInequality solves this goal differently: It applies the method ComplexEstimate with the assumption (10) to the goal and passes the resulting inequality goals with TellCS to $\mathcal{CoSIE}$. With this solution proof plan for goal (8) SolveInequality can also solve goal (9) by applying the method ComplexEstimate with assumption (11) and passing the resulting inequality goals with TellCS to $\mathcal{CoSIE}$.

More ϵ-δ-proofs as well as problems from other domains require this failure reasoning to analyze and overcome variable dependencies (see section 6).

5.3 Meta-reasoning for Insufficiently Determined Constraints

Remember the *Unblock Desirable Steps* rule from section 4.3. One of its instances unblocks the instantiation strategy ComputeInstFromCS.

To illustrate this failure reasoning we detail the proof planning process for the problem Lim-Div, which states that the limit of the function $\frac{1}{x}$ at point $c \neq 0$ is $\frac{1}{c}$, i.e., for $c \neq 0$

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

The proof planning works as follows: Decomposition of the initial goal results in the two goals $0 < \delta^p$ and $|\frac{1}{c_x} - \frac{1}{c}| < c_\epsilon$. SolveInequality closes the goal $0 < \delta^p$ by TellCS and simplifies the second goal to $|\frac{c-c_x}{c_x * c}| < c_\epsilon$. It continues with the application of FactorialEstimate, which reduces this goal to three simpler subgoals $0 < v^p$, $|c_x * c| > v^p$, and $|c - c_x| < v^p * c_\epsilon$ with a new variable v^p. SolveInequality closes these three goals with TellCS. Since all goals are closed the strategy ComputeInstFromCS becomes a highly desirable strategy that should return instantiations for the variables δ^p and v^p computed by $\mathcal{CoSIE}$.

Now, $\mathcal{CoSIE}$ fails to determine instantiations because the constraints collected so far

$\frac{\lvert c_x - c\rvert}{c_\epsilon} < v^p$	$0 < v^p$	$v^p < \lvert c_x * c\rvert$
$0 < \delta^p$	$c \neq 0$	$0 < c_\epsilon$

are insufficient to compute a solution.[4] Hence, also the application of the desirable strategy ComputeInstFromCS is blocked.

A possibility to overcome this problem is to create further constraints by further proof planning. An instance of the meta-reasoning rule ***Unblock Desirable Steps*** suggests this "repair":

IF	a constraint solver fails to provide instantiations because of insufficient constraints while all goals are closed
THEN	consider actions to create and pass further constraints

Technically, the idea to overcome highly desirable but blocked variable instantiations by a constraint solver is encoded in the strategic control rule `unblock-constr`. When all goals are closed, but instantiation strategies are not applicable since connected constraint solvers fail to compute instantiations, then `unblock-constr` analyzes the current proof plan for possible further constraints.

One possibility to derive further constraints is the refinement of existing constraints closed by applications of TellCS. If `unblock-constr` detects such constraints that likely can be refined to simpler constraints, it triggers the backtracking of the corresponding TellCS applications (only these selected applications).[5] Afterwards, the re-opened goals can be tackled again and can be refined. Note that this backtracking serves the applicability of methods uncovering further constraints rather than the traversal of the search space.

[4] This is a common situation in constraint solving: A set of constraints has been accepted since no inconsistency could be detected, so far. Nevertheless, the collected constraints are not sufficient to compute a solution for the constrained variables. The critical constraints here are the constraints on v^p, which state that $\frac{|c_x - c|}{c_\epsilon}$ has to be less than v^p, which has to be less than $|c_x * c|$. These constraints are not inconsistent, but a solution for v^p exists only, if $\frac{|c_x - c|}{c_\epsilon} < |c_x * c|$ holds. This, however, does not follow from the constraints collected so far.

[5] Currently, the critical constraints are chosen by heuristics encoded in the control rule. We are enhancing $\mathcal{CoSIE}$ to return the critical constraints directly.

In proof planning of Lim-Div, `unblock-constr` (successively) triggers the backtracking of the applications of TellCS that close $|c - c_x| < v^p * c_\epsilon$ and $|c_x * c| > v^p$. SolveInequality reduces the re-opened goals with applications of the method ComplexEstimate and passes the resulting constraints by applications of TellCS to $\mathcal{CoSIE}$. This leads to the following constraint store (the variables ${v_1}^p$ and ${v_2}^p$ are introduced by the applications of ComplexEstimate):

$c_\epsilon > 0$	$c \neq 0$	$v^p \geq {v_1}^p * \delta^p$	${v_1}^p > c$
$v^p > 0$	${v_2}^p > 1$	$\frac{c_\epsilon * v^p}{2} > 0$	$\delta^p > 0$
$\delta^p \leq \frac{c_\epsilon * v^p}{2 * {v_2}^p}$	$v^p * 2 \leq c^2$		

Now the following instantiations consistent with these constraints can be computed: ${v_2}^p \rightarrow 2$, ${v_1}^p \rightarrow c + 1$, $v^p \rightarrow \frac{c^2}{2}$, and $\delta^p \rightarrow min(\frac{c_\epsilon * c^2}{8}, \frac{c^2}{2*(c+1)})$.

This example is not an isolated one. More ϵ-δ-proofs require this failure reasoning (see section 6). Moreover, the meta-reasoning rule is generally applicable also to other domains in which constraints solvers are used.

5.4 Lemma Speculation

Another typical instance of the *Unblock Desirable Steps* rule in section 4.3 unblocks method applications whose matching with the proof situation requires additional information. It reads as follows

IF	the application of a desirable method fails because of a unification residuum and the residuum is likely to be provable in the current context
THEN	speculate residuum as lemma and apply it to unblock the desirable method

Since lemma speculation may open a Pandora's box, the restriction *the residuum is likely to be provable in the current context* needs to be defined. For instance, in proof planning that uses a constraint solver the constraint solver can be exploited to decide whether a residuum is a promising lemma. For ϵ-δ-proofs, the meta-reasoning queries $\mathcal{CoSIE}$ whether it accepts the residuum and only then meta-reasoning suggests the speculation of the lemma. This way, we combine the domain-independent unification and matching with the domain knowledge in $\mathcal{CoSIE}$.[6] Technically, this lemma speculation is encoded by the control rule `unblock-method`.

To illustrate this failure reasoning we detail planning for the problem

$$\text{If } \lim_{x \to 0} f(x + c) = l, \text{ then } \lim_{x_1 \to c} f(x_1) = l.$$

The decomposition of the initial goal results, among others, in the goal $|f(c_{x_1}) - l| < c_{\epsilon_1}$. Decomposition of the initial assumption yields the new assumption $|f(x^p + c) - l| < \epsilon^p$. SolveInequality should apply the method Solve*

[6] Alternatively, theory unification incorporates domain-specific axioms and theorems into the unification procedures. However, the decidability of theory unification is difficult to determine and depends on the concrete set of domain equations (e.g., see [2]). Undecidable unification and matching, however, could block the complete proof planning process.

to tackle the new goal with the new assumption. However, this fails since the application-conditions of SOLVE* request the unification of $|f(x^p + c) - l|$ and $|f(c_{x_1}) - l|$, which fails. Since no other applicable method or assumption are available, MULTI's default control would backtrack, which would not lead to a solution proof plan.

When SolveInequality fails to tackle $|f(c_{x_1}) - l| < c_{\epsilon_1}$ with the assumption $|f(x^p + c) - l| < \epsilon^p$, then the analysis of the failure by `unblock-method` yields the residuum $x^p + c = c_{x_1}$, which is accepted by $\mathcal{CoSIE}$. Hence, the control rule fires and introduces $x^p + c = c_{x_1}$ as lemma and guides the rewriting of the goal $|f(c_{x_1}) - l| < c_{\epsilon_1}$ with this equation. This results in the goal $|f(x^p + c) - l| < c_{\epsilon_1}$. The application of SOLVE* to this goal and the assumption $|f(x^p + c) - l| < \epsilon^p$ is now possible, and SolveInequality can solve all resulting goals.

For other ϵ-δ-proofs the same meta-reasoning can overcome blocked unifications and matchings. For the application in other domains the only prerequisite is that a means exists that can decide whether a lemma is promising.

6 Empirical Results

The meta-reasoning rules in section 4 describe general situations in mathematical proof processes. 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 three domains, the ϵ-δ-proofs from the analysis textbook [1], the residue class domain, and inductive proofs. Table 1 gives sample problems from all three domains and the failure-reasoning they require. The numbered colons denote (i) case split introduction, (ii) unblock constraint solving, (iii) unblock by lemma speculation, (iv) analyze variable dependencies. Note that $x \to a^-$ and $x \to a^+$ denote the left-hand limit and the right-hand limit, respectively.

The relevance of failure reasoning is not only demonstrated by Table 1. 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 25 out of 70 ϵ-δ-proofs constructed by MULTI from the systematically explored testbed [1] involve failure reasoning evidences the crucial role of failure reasoning.

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 VarDependencies* rule is crucial since MULTI has to deal with nested existential quantifiers, which result in 'nested' variables shared by several goals. Hence, dependencies among the variables and the goals have to be analyzed.

Table 1. Sample proofs whose solution requires meta-reasoning about failures

Conjecture	(i)	(ii)	(iii)	(iv)
ϵ-δ-Proofs				
$\lim_{x\to 0}(f(a+x)-f(a))=0 \Rightarrow cont(f,a)$	x		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			
$\lim_{x\to a^-} f(x)=f(a) \wedge \lim_{x\to a^+} f(x)=f(a) \Rightarrow cont(f,a)$	x			
$\lim_{x\to 2} \frac{1}{1-x}=-1$		x		
$\lim_{x\to a} f(x)=l_f \wedge \lim_{x\to a} g(x)=l_g \wedge \forall x\ g(x)\neq 0 \Rightarrow \lim_{x\to a} \frac{f(x)}{g(x)}=\frac{l_f}{l_g}$		x		
$\lim_{x\to\infty} f(x)=l \Rightarrow \lim_{x\to\infty} \frac{f(x)}{x}=0$		x		
$\lim_{x\to 0} f(x+a)=l \Rightarrow \lim_{x\to a} f(x)=l$			x	
$\lim_{x\to 0^+} f(\frac{1}{x})=l \Rightarrow \lim_{x\to\infty} f(x)=l$			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
$\neg\exists e{:}_{\mathbb{Z}_9} unit(\mathbb{Z}_9,\bar{-})$				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		x	
$\forall x{:}_{item}\forall y{:}_{list}\ x\in isort(y) \Rightarrow x\in y$	x		x	
$\forall x{:}_{item}\forall y{:}_{list}\ count(x,isort(y))=count(x,y)$	x		x	
$\forall y{:}_{list}\ reverse(reverse(y))=y$			x	
$\forall x{:}_{nat}\ even(x+x)$			x	

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 the detailed description of the experiments). Some representative examples occur in Table 1.

Inductive Proofs. So far, we did not apply MULTI to inductive proofs. The inductive theorems in Table 1 are taken from [9], 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 rules for case-split introduction and lemma speculation are applicable for inductive proofs as well. For a more complete list of inductive proofs that require failure reasoning see [9].

7 Conclusions and Related Work

We presented a novel conceptual and architectural contribution to knowledge-based automated theorem proving. We demonstrated how MULTI's novel theorem proving architecture supports failure reasoning and the automatic discovery of heureka steps such as introduction of case splits and speculation of lemmas. As evidenced by empirical results, the discussed failure reasoning is generally applicable rather than overly specific.

Proof planning may fail because of an exception from proof structures and procedures captured by proof planning methods, strategies and heuristic control. Such failures can be analyzed automatically and this analysis gives rise to applying proof planning strategies, which can revise the proof plan or invent new knowledge that is needed to complete a proof, e.g., case splits or new conjectures. That is, often failures hold the key to the construction of a solution proof plan.

The failure reasoning and the subsequent proof plan modifications are possible in MULTI, since MULTI's architecture does not enforce a pre-defined backtracking or other pre-defined control. Rather, when a failure occurs, then strategic control rules, which declaratively encode failure handling heuristics, can analyze the failure and dynamically guide promising refinements and modifications of the proof plan. Further meta-reasoning that contributes to this flexible control in MULTI is discussed in [10].

Related Work

Unblocking desirable steps in MULTI is related to the control reasoning in elaborate blackboard systems, see [5] and [6]. 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.

The speculation of residuum lemmas is related to constrained resolution [7], which intertwines resolution with unification. We also intertwine unification with the main proof process by speculating unification residues as lemmas. As opposed to constrained unification, our meta-reasoning controls the speculation of lemmas since it suggests only lemmas that are directly accepted by $\mathcal{CoSIE}$.

In their interesting work described in [8, 9] Andrew Ireland and Alan Bundy extended proof planning by so-called critics as a means to patch failed proof attempts in proof planning inductive proofs. Their proof planner CLAM is specialized for proving theorems by mathematical induction and employs the so-called rippling technique that is mainly encoded in the `wave` method. The critics in CLAM are associated with the `wave` method and capture patchable exceptions to the application of this method. A critic can, e.g., introduce a case split directly preceding the wave method, in order to make a conditional wave rule applicable, in case the `wave` method is blocked because of this condition.

The failure reasoning in MULTI considerably differs from the critics mechanism in CLAM in its conceptual design. Critics are method-like entities that are bound to failing preconditions of a particular method. Moreover, the critic's patch is a special procedure that changes the proof plan. In contrast, failure reasoning in MULTI is represented by declarative control rules. These control rules

are not associated with a particular method but can reason about the current proof plan and about other information such as the proof planning 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.

To summarize, because of its flexible proof construction at the strategy level, MULTI can realize a more general failure reasoning approach not bound to particular methods.

References

1. R.G. Bartle and D.R. Sherbert. *Introduction to Real Analysis*. John Wiley& Sons, New York, 1982.
2. K.H. Bläsius and H.J. Bürckert, editors. *Deduktionssysteme*. Oldenbourg, 1992.
3. W.W. Bledsoe. Challenge Problems in Elementary Analysis. *Journal of Automated Reasoning*, 6:341–359, 1990.
4. A. Bundy. The Use of Explicit Plans to Guide Inductive Proofs. In *Proceedings of CADE–9*, volume 310 of *LNCS*, pages 111–120. Springer, 1988.
5. D.D. Corkill, V.R. Lesser, and E. Hudlicka. Unifying Data-Directed and Goal-Directed Control. In *Proceedings of AAAI-82*, pages 143 – 147. AAAI Press, 1982.
6. E.H. Durfee and V.R. Lesser. Incremental Planning to Control a Blackboard-Based Problem Solver. In *Proceedings of AAAI-86*, pages 58 – 64. AAAI Press, 1986.
7. G.P. Huet. *Constrained Resolution: A Complete Method for Higher Order Logic*. PhD thesis, Case Western Reverse University, 1972.
8. A. Ireland. The Use of Planning Critics in Mechanizing Inductive Proofs. In *Proceedings of LPAR'92*, volume 624 of *LNAI*, pages 178–189. Springer, 1992.
9. A. Ireland and A. Bundy. Productive Use of Failure in Inductive Proof. *Journal of Automated Reasoning*, 16(1-2):79–111, 1996.
10. A. Meier. MULTI – *Proof Planning with Multiple Strategies*. PhD thesis, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, 2004.
11. A. Meier and E. Melis. MULTI: A Multi-Strategy Proof Planner. In *Proceedings CADE–20*, Springer, 2005.
12. A. Meier, M. Pollet, and V. Sorge. Comparing Approaches to Explore the Domain of Residue Classes. *Journal of Symbolic Computation*, 34(4):287–306, 2002.
13. E. Melis and A. Meier. Proof Planning with Multiple Strategies. In *Proceedings CL-2000*, volume 1861 of *LNAI*, pages 644–659. Springer, 2000.
14. E. Melis and J. Siekmann. Knowledge-Based Proof Planning. *Artificial Intelligence*, 115(1):65–105, 1999.
15. A.H. Schoenfeld. *Mathematical Problem Solving*. Academic Press, New York, 1985.
16. J. Zimmer and E. Melis. Constraint solving for proof planning. *Journal of Automated Reasoning*, 2004. accepted.