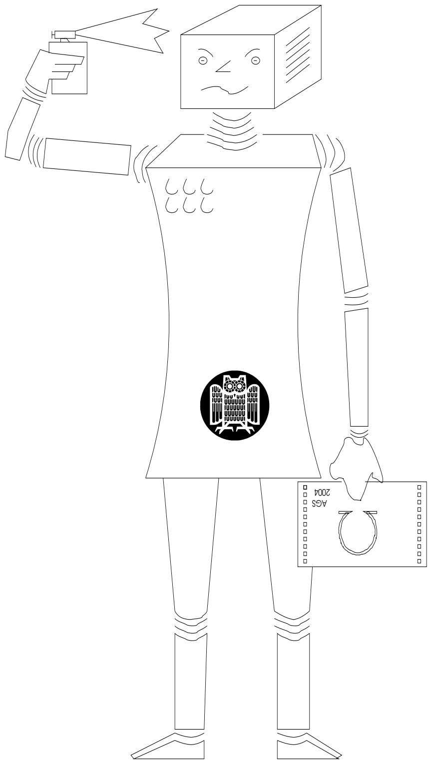
UNIVERSITÄT DES SAARLANDES
FACHRICHTUNG INFORMATIK
D–66123 SAARBRÜCKEN
GERMANY
WWW: http://www.ags.uni-sb.de/

SEKI-REPORT ISSN 1437-4447



Proof Planning Limit Problems with Multiple Strategies

Andreas Meier and Erica Melis
DFKI GmbH, 66041 Saarbrücken, Germany
E-Mail: {ameier|melis}@dfki.de
WWW:
http://www.ags.uni-sb.de/{~ameier|~melis}

SEKI Report SR–2004–04

This SEKI Report was internally reviewed by:
Martin Pollet
FR Informatik, Universität des Saarlandes, D–66123 Saarbrücken, Germany
E-mail: `pollet@ags.uni-sb.de`
WWW: `http://www.ags.uni-sb.de/~pollet`

Editor of SEKI series:
Claus-Peter Wirth
FR Informatik, Universität des Saarlandes, D–66123 Saarbrücken, Germany
E-mail: `cp@ags.uni-sb.de`
WWW: `http://www.ags.uni-sb.de/~cp/welcome.html`

Proof Planning Limit Problems with Multiple Strategies

Andreas Meier and Erica Melis
DFKI GmbH, 66041 Saarbrücken, Germany
E-Mail: `{ameier|melis}@dfki.de`
WWW: `http://www.ags.uni-sb.de/{~ameier|~melis}`

September 18, 2004

Abstract

The development of proof planning in the ΩMEGA system was and is strongly influenced by the still ongoing case study on limit problems. In this report we describe the application of ΩMEGA's recent proof planning approach, which is called proof planning with multiple strategies, to problems from the limit domain. In particular, we point out how drawbacks we encountered with the previous proof planner of ΩMEGA when applied to limit problems motivated and influenced the development of proof planning with multiple strategies.

Contents

1 Introduction

Proof planning is an application of Artificial Intelligence planning techniques for automated theorem proving at the abstract level of tactics. Proof planning was introduced by Bundy [10], in order to avoid super-exponential search when proving theorems by induction. This proof planning for inductive proofs is implemented in the proof planners CLAM [14] and λCLAM [39] in Edinburgh.

The research group developing the ΩMEGA proof development system in Saarbrücken extended Bundy's proof planning approach such that proof planning can use domain knowledge in order to plan proofs in different mathematical domains. The resulting proof planning approach is called knowledge-based proof planning [36]. The scientific research in the ΩMEGA group was and is driven by case studies. These case studies are used to evaluate former approaches and their implementation, and the analysis of the case studies gives insights for the extension of an approach and the development of new approaches. In addition, more often than not new applications give rise to new requirements which have to be integrated into the extended model.

The most influential case study for the development of proof planning in ΩMEGA is the case study on proof planning limit problems. Theorems of the limit domain make statements about the limit of functions and sequences as well as about continuity and the derivative of functions. Early results about proof planning in the limit domain are reported in [31]. Following publications point out the achieved progress in proof planning limit problems as well as the development of ΩMEGA's proof planning approach and their mutual influences and dependencies, for instance, see [33, 32, 27, 36, 37, 35].

The recent proof planning approach in ΩMEGA is proof planning with multiple strategies [34]. The development of proof planning with multiple strategies was – among others – motivated and strongly influenced by drawbacks we encountered with ΩMEGA's previous proof planner, when systematically tackling the limit problems from the analysis textbook [2]. In this report, we describe the application of proof planning with multiple strategies to problems from the limit domain. In particular, we discuss limit problems that can be solved by the new multiple-strategy proof planner MULTI, while the previous proof planner of ΩMEGA fails to solve them, and point out the reasons for MULTI's success.

The structure of the report is as follows: We first describe the basics of proof planning in ΩMEGA in section 2. This section contains brief descriptions of MULTI and ΩMEGA's previous proof planner. In the subsequent section, we introduce the limit domain and explain how ΩMEGA's previous proof planner solves problems from the limit domain. The next two sections then discuss the application of MULTI to limit problems: whereas section 4 gives a general account, section 5 is devoted to the realization of failure reasoning in MULTI to solve limit problems. The report concludes in section 6 with the discussion of the results and related work.

2 Proof Planning

Proof planning was originally conceived as an extension of tactical theorem proving to enable automated theorem proving at the abstract level of tactics. BUNDY's key idea in

[10] is to augment individual tactics with pre- and postconditions. This results in planning operators, so-called *methods*. In the ΩMEGA [42] system the traditional proof planning approach is enriched by incorporating mathematical knowledge into the planning process (see [36]) and the introduction of strategies (see [34]).

Domain-specific knowledge can be encoded in *methods*, in *control rules*, and in *external systems* such as computer algebra systems or constraint solvers. Methods can encode not only general proving steps but also steps particular to a mathematical domain. Control rules enable meta-level reasoning about the current proof planning state as well as about the entire history of the proof planning process in order to guide the search. The recent development of proof planning with multiple-strategies introduces strategies and their heuristic control as another hierarchical level, which provides the possibility to encode (mathematical) domain knowledge.

In the following, we briefly introduce the basics of proof planning in ΩMEGA and sketch the two planners of the ΩMEGA system, the simple planner PLAN and the multiple-strategy planner MULTI. A detailed description of the planners is given in [30].

2.1 Basics of Proof Planning

Proof planning in ΩMEGA considers mathematical theorems as planning problems.[1] The *initial state* of a *proof planning problem* consists of the proof *assumptions* and the *goal description* consists of the *theorem*. Methods are the operators of proof planning, where methods are tactics known from tactical theorem proving augmented with pre- and postconditions in order to derive operators for AI-planning. Proof planning searches for a solution plan, i.e., a sequence of instantiated methods that transforms the initial state into a state in which the theorem holds. In order to find a solution plan, it searches for applicable methods and applies the instantiated methods. Similar to AI-planning we call the instantiation of a method (i.e., the instantiation of a proof planning operator) an *action*.

The effects and the preconditions of actions as well as the initial proof planning problem in ΩMEGA consist of proof lines as used in [1]. A proof line is of the form $L.\ \Delta \vdash F\ (\mathcal{R})$, where L is a unique label, $\Delta \vdash F$ a sequent denoting that the formula F can be derived from the set of hypotheses Δ, and $(\mathcal{R})$ is a justification expressing how the line was derived. Lines that are not yet derived from other lines are called open lines and have an open justification. A line that is not open is called a *closed line*. During the proof planning process all constructed proof lines are stored in a data-structure called $\mathcal{PDS}$ [15]. For instance, the initial $\mathcal{PDS}$ for the proof problem with theorem Thm and assumptions $Ass_1, \ldots, Ass_n$ is:

$L_{Ass_1}.$	L_{Ass_1}	$\vdash Ass_1$	(Hyp)
		$\vdots$	
$L_{Ass_n}.$	L_{Ass_n}	$\vdash Ass_n$	(Hyp)
$L_{Thm}.$	$L_{Ass_1}, \ldots, L_{Ass_n}$	$\vdash Thm$	$(Open)$

When a new action is added, then the new lines derived by this action are added into the $\mathcal{PDS}$. Moreover, all effect lines of the action are justified by an application of the method

[1] See [48, 40] for introductions to AI-planning.

of the action to the premises of the action. For instance, if an action of method M has the premise lines L_1 and L_2 and the effect line L_3, then L_3 becomes justified in the $\mathcal{PDS}$ by $(M\ L_1\ L_2)$. Since an action justifies its effect lines, a closed $\mathcal{PDS}$, i.e., a $\mathcal{PDS}$ without open lines, also forms a tactic-level proof. Expansions of the actions, which correspond to tactic applications, can result in a proof in ΩMEGA's underlying higher-order natural deduction (ND) calculus [19]. However, in the remainder of the report we will focus on the creation of abstract proof objects at the planning level and not on the expansion to the ND calculus.

Central during the proof planning process are so-called tasks, which express which proof lines (closed and open) can be used to construct a subplan for an open line. A *task* is a pair $(L_{open}, \mathcal{SUPPS}_{L_{open}})$ where L_{open} is an open line and $\mathcal{SUPPS}_{L_{open}}$ is a set of lines. The first element of a task is called the *task line* or the *goal of the task* and the second element is called the *support lines* or *supports*. The formula of the goal is also called *task formula*. A task with goal L_{open} and supports $\mathcal{SUPPS}_{L_{open}}$ is written as $L_{open} \blacktriangleleft \mathcal{SUPPS}_{L_{open}}$. During the planning process a list of all current tasks is stored in a so-called *agenda*. For a problem with theorem Thm and assumptions $Ass_1, \ldots, Ass_n$ the *initial agenda* consists of the task $L_{Thm} \blacktriangleleft \{L_{Ass_1}, \ldots, L_{Ass_n}\}$.

2.2 Methods and Control Rules

Methods

Methods encode the knowledge of the relevant proof steps of mathematical domains. Technically, a method in ΩMEGA is a frame data structure with the slots **premises**, **conclusions**, **application conditions**, and **proof schema**.

The *premises and conclusions of a method* specify the preconditions and the effects of the method. The conclusions should be logically inferable from the premises. The union of conclusions and premises is called the *outline* of a method. Declarative descriptions of the formulas of the outline can be given in the proof schema, which also provides the schematic or procedural expansion information.

Premises and conclusions may be annotated with $\oplus$ and $\ominus$. The annotations are needed to indicate whether a method is used for forward or backward search. As opposed to AI-planning, where operators typically can be applied for both forward search and backward search, a method in ΩMEGA is either used in forward search or in backward search. This is because methods typically comprise complex computations that are reasonable either in one direction or in the other direction.

Backward and forward methods are specified as follows: A *backward method* has $\ominus$ *conclusions* and $\oplus$ *premises* as well as $\ominus$ *premises* and *blank premises*. To compute an action of the method, one of the $\ominus$ conclusions is matched with the goal of a given task and both, the $\ominus$ premises and the blank premises, are matched with supports of the task. When the resulting action is introduced into the proof plan, then the goal is closed in the $\mathcal{PDS}$ and the $\oplus$ premises are added to the $\mathcal{PDS}$ and become goals of new tasks. These new tasks inherit the supports of the initial task except that the $\ominus$ premises are removed. The blank premises are not affected. A *forward method* has $\oplus$ *conclusions* as well as $\ominus$ premises and blank premises. To compute an action of the method, the $\ominus$ premises and the blank premises are matched with the support lines of a given task. When the resulting action is introduced into the proof plan, then the $\oplus$ conclusions are added to the $\mathcal{PDS}$ and become new support lines of the task. Moreover, the $\ominus$ premises are removed from the supports of the task. Again, the blank premises are not affected.

Method: $=$SUBST-B				
premises	$\oplus L_2$, L_1			
conclusions	$\ominus L_3$			
appl. conds.	(1) *valid-position-p*(f,pos) (2) [*term-at-position*$(f,pos) = t \vee$ *term-at-position*$(f,pos) = t'$]			
proof schema	$L_1.$	Δ	$\vdash t \doteq t'$	$()$
	$L_2.$	Δ	$\vdash f'$	$(Open)$
	$L_4.$	Δ	$\vdash \forall P_{\alpha o}. P(tf') \Rightarrow P(tf)$	$(\equiv_E \ \doteq)$
	$L_5.$	Δ	$\vdash (\lambda f)(tf') \Rightarrow (\lambda f)(tf)$	$(\forall_E \ L_4 \ \lambda f)$
	$L_6.$	Δ	$\vdash f[tf'] \Rightarrow f[tf]$	$(\lambda\leftrightarrow \ L_5)$
	$L_3.$	Δ	$\vdash f$	$(\Rightarrow_E \ L_2 \ L_6)$

Figure 1: The $=$SUBST-B method.

Consider the method $=$SUBST-B, given in Figure 1, which can be used in all domains that employ the equality $\doteq$. Essentially, the method performs an equality substitution. It has two preconditions L_1 and L_2, where the proof schema determines L_1 to be an equation. The only conclusion is L_3. $=$SUBST-B is a backward method. The introduction of an action of $=$SUBST-B closes a task line whose formula matches with the formula of L_3 and introduces a new task whose goal is the instantiation of L_2. That is, the formula of the new goal results from the formula of the initial goal by substitution with the equation, which is the formula of a support of the initial task that matched with L_1. For instance, $=$SUBST-B applied to the task $even(a+1) \blacktriangleleft \{a = 1, \ldots\}$[2] introduces the new goal $even(1+1)$.

The *application conditions of a method* are meta-level descriptions that restrict the applicability of a method. The application conditions can consist of arbitrary LISP functions. The method $=$SUBST-B has two application conditions: (1) the position *pos* has to be a valid position in the formula f and (2) the subterm in f at the position *pos* is t or t'. Note that application conditions reason only about whether the application of a method is valid in a certain situation; they do not reason about whether the application is useful.

The *proof schema of a method* is a declarative description of the outline of a method. Moreover, it describes the expansion of actions of the method, which corresponds to both tactic expansions and expansions of HTN-planning [45]. When an action of a method is expanded, then for each conclusion a new subproof is introduced into the $\mathcal{PDS}$. For instance, the proof schema of $=$SUBST-B specifies that the defined concept $\doteq$ in the premise is replaced by its definition (i.e., the Leibniz-Equality definition). Then, some calculus rules $\forall_E$, $\lambda\leftrightarrow$, and $\Rightarrow_E$ are applied to derive the conclusion of the method.

Generally, proof construction may require to construct mathematical objects, for instance, if a method has to instantiate existentially quantified variables by witness terms.

[2]To simplify this example, we just write the formulas of the goal and the support line instead of the whole proof lines.

A witness term has to be a concrete term. However, if the method is applied at an early stage of the proof, the planner generally has no knowledge of the witness term. Therefore, the actual instantiation of witnesses can be postponed; rather, methods can introduce so-called *meta-variables* as temporary substitutes for the actual witness terms, which will be determined at a later point in the planning process and subsequently instantiated.

Further methods relevant for proof planning limit problems are discussed in section 3.2.

Notation In this report, we write mv for meta-variables. If several meta-variables occur, we attach subscripts to mv in order to distinguish the meta-variables. We either use the variable for whose instantiation the meta-variable is a substitute as subscript (e.g., we write mv_x if mv is a substitute for the instantiation of the variable x) or we use numbers. If the decomposition of a quantified formula results in the introduction of a constant, then we write c for this constant. Similar to the notation for meta-variables, we use either the initial variable or numbers as subscripts to distinguish several occurring constants.

Notation Methods are written in TEXT SMALL CAPITAL FONT (e.g., =SUBST-B). The name of backward methods ends with -B and the name of forward methods ends with -F.

Control Rules

Control rules provide guidance of the proof planning process by declaratively representing heuristical knowledge that corresponds to mathematical intuition about how to prove a goal in a certain situation. In particular, these rules provide the basis for meta-level reasoning and a global guidance since they can express conditions for a decision that depends on all available knowledge about the proof planning process so far. The control rules used in ΩMEGA's proof planning were adopted from the control rule approach of the AI-planner PRODIGY [47],

In the planning process control rules guide decisions at choice points, e.g., which task to tackle next or which method to apply next. They achieve this by reasoning about the heuristic utility of different alternatives[3] in order to promote the alternatives that seem to suit best in the current situation, where 'situation' comprises all available information on the current status such as the current tasks, their supports, the planning history, failed attempts, etc. To manipulate an alternative list control rules can remove elements, prefer certain elements, or add new elements. This way, the ranking of alternatives is dynamically changed. This can help to prune the search space or to promote certain promising search paths.

Technically, control rules consist of an IF- and a THEN-part. The IF-part is a predicate on the current proof planning 'situation', whereas in the THEN-part modifications of alternative lists are stated. Moreover, each control rules specifies its kind, i.e., the choice point in the proof planning process it guides.

Figure 2 gives as example the control rule `prove-inequality`, which is evaluated during the selection of the next method to apply. In its IF-part `prove-inequality` checks whether the current goal is an inequality. If this is the case, it prefers the methods TELLCS-B, TELLCS-F, ASKCS-B, SIMPLIFY-B, SIMPLIFY-F, SOLVE*-B, COMPLEXESTIMATE-B, FACTORIALESTIMATE-B, and SETFOCUS-B in this order (these methods are explained in

[3]As opposed to application conditions of methods, which reason about the legal feasibility of applications of methods (see last section).

```
(control-rule prove-inequality
              (kind methods)
              (IF (and (goal-matches (REL A B))
                       (in REL {<,>,≤,≥})))
              (THEN (prefer (TellCS-B TellCS-F AskCS-B Simplify-B
                             Simplify-F Solve*-B ComplexEstimate-B
                             FactorialEstimate-B SetFocus-B))))
```

Figure 2: The control rule `prove-inequality`.

section 3.2). The *prefer* states that the methods specified in the control rule are preferred before all other methods, i.e., the specified methods are ordered in front of the resulting alternative list. Other possible modifications of alternative lists are *select*, *reject*, *defer*, and *order-in-front*. *select* states that all other methods except those specified in the control rule are eliminated from the list of alternative methods. *reject* removes all alternatives specified in the control rule from a given alternative list, the latter two manipulations reorder the alternative list. *defer* orders all specified alternatives at the end of the alternative list, and *order-in-front* orders specified alternatives in front of other specified alternatives. Finally, there is the *insert* modification. It allows to introduce new elements in an alternative list. A typical situation for using an *insert* control rule is when a general control rule – which is applied first – removes some elements from the alternative list, which are needed in a particular situation. Then a more specific *insert* control rule, which is applied later on, can introduce the needed elements again.

Notation Control rules are denoted in `typewriter font` (e.g., `prove-inequality`). Technically, control rules are frame data structures. Since they are considerably simpler as, for instance, methods, we do not present them in the data structure fashion (as we do with methods) rather we give their LISP encoding. That is, the content of Figure 2 is the specification of the control rule `prove-inequality` as it is in ΩMEGA's data base.

2.3 From PLAN to MULTI

PLAN is ΩMEGA's simple proof planner. It proceeds by successively computing and introducing actions into a proof plan under construction. Table 1 shows the outline of PLAN's algorithm. First, PLAN selects a task to work on. Then, it computes actions for this task and selects one action, which it introduces into the proof plan under construction. This results in new tasks on which PLAN continues. If PLAN fails to compute an action for a selected task, then it performs backtracking. Although actions can perform both, forward reasoning and backward reasoning, an action is always chosen with respect to a task in order to close or to reduce the gap between the goal and the supports of the task.

Some decisions in PLAN can be guided by control rules, for instance, the selection of the next task and the selection of the next action. Other decisions, however, are hard-coded into the system. For instance, PLAN employs backtracking if and only if it tackles a task, for which it fails to compute an action. Moreover, it employs external constraint solvers to obtain instantiations for meta-variables if and only if the agenda is empty and the $\mathcal{PDS}$ is closed.

1. When the current agenda is empty and the current $\mathcal{PDS}$ is closed, then apply external constraint solvers to compute variable instantiations consistent with the collected constraints and terminate.
2. Select a task T from the agenda.
3. Compute and select an action A with respect to T.
4. If an action A could be computed for T, then introduce A. Goto step 1.
5. If no action A could be computed for T, then backtrack the action whose introduction created the task T. Goto step 1.

Table 1: Cycle of PLAN.

When we extended the exploration of the limit domain and when we explored further domains we encountered problems of the simple proof planning approach realized in PLAN (see section 3.4 for the discussion of several problems) that caused us to reconsider ΩMEGA's proof planning approach and gave rise to the development of multi-strategy proof planning, which we realized in the MULTI system.

Proof planning with multiple strategies decomposes the previous monolithic proof planning process and replaces it by separated parameterized algorithms as well as different instances of these algorithms, so-called strategies. The strategies, which specify different behaviors of the algorithms, are the basic elements for proof construction in multiple-strategy proof planning. That is, the goal of multiple-strategy proof planning is to compute a sequence of strategy applications that derives a given theorem from a given set of assumptions. The decision on when to apply a strategy is not encoded once and forever into the system but rather is determined by meta-level reasoning using heuristic control knowledge of strategies and their combination.

Algorithms

MULTI enables the incorporation of heterogeneous, parameterized algorithms for different kinds of proof plan refinements and modifications. Currently, MULTI employs the following algorithms:

PPLANNER refines a proof plan by introducing new actions.

INSTMETA refines a proof plan by instantiating meta-variables.

BACKTRACK modifies a proof plan by removing refinements of other algorithms.

EXP refines a proof plan by expanding complex steps.

ATP refines a proof plan by solving subproblems with traditional machine-oriented automated theorem provers.

CPLANNER refines a proof plan by transferring steps from a source proof plan or fragment.

The decomposition of the previous monolithic proof planner of ΩMEGA allows to extend and generalize the functionalities of its subcomponents. This results in the independent and parameterized algorithms **PPlanner**, **InstMeta**, and **BackTrack** for action introduction, meta-variable instantiation, and backtracking. **Exp**, **ATP**, and **CPlanner** integrate new refinements of the proof plan.

Strategies

Instances of these algorithms can be specified in different strategies. Technically, a *strategy* is a condition-action pair. The condition part states when the strategy is applicable. The action part consists of a modification or refinement algorithm and an instantiation of its parameters. Similar to the knowledge of the applicability of methods we separate the legal and heuristic knowledge of the applicability of strategies. The condition part of a strategy states the legal conditions that have to be satisfied in order for the strategy to be applicable, whereas *strategic control rules* reason about the heuristic utility of the application of strategies.

To execute or to apply a strategy means to apply its algorithm to the current proof planning state with respect to the parameter instantiation specified by the strategy. For instance, the parameters of **PPlanner** are a set of methods, a list of control rules, and a termination condition. When Multi executes a **PPlanner** strategy, the **PPlanner** algorithm introduces only actions that use the methods specified in the strategy. **PPlanner** evaluates the control rules specified by the strategy during the computation and selection of actions. The application of the strategy terminates, when its termination condition is satisfied. Hence, different strategies of **PPlanner** provide a means to structure the method and control rule knowledge. Both algorithms, **InstMeta** and **BackTrack**, have one parameter. The parameter of **InstMeta** is a function that determines how the instantiation for a meta-variable is computed. If Multi applies a **InstMeta** strategy with respect to a meta-variable mv, and if the computation function of the strategy yields a term t for mv, then **InstMeta** substitutes mv by t in the proof plan. The parameter of **BackTrack** is a function that computes a set of refinement steps of other algorithms that have to be deleted. When Multi applies a **BackTrack** strategy, then **BackTrack** removes all refinement steps that are computed by the function of the strategy as well as all steps that depend from these steps. Examples of strategies are introduced and discussed in section 4.1.

Notation Strategies are denoted in the sans serif font (e.g., NormalizeLineTask, UnwrapHyp).

Tasks

Multi extends the task concept of PLAN. Since Multi employs further kinds of tasks, the tasks used in PLAN (i.e., a pair consisting of an open line and its supports) are called *line-tasks* in Multi. Another kind of tasks are *instantiation-tasks*. The introduction of a meta-variable into the plan results in an instantiation-task, that is, the task to instantiate this meta-variable. The instantiation task for meta-variable mv is written as $mv|^{Inst}$.

Different tasks can be tackled by different algorithms and strategies. For instance, since strategies of **InstMeta** introduce instantiations for meta-variables they can tackle instantiation-tasks, whereas strategies of **PPlanner** or **ATP** can tackle line-tasks. A strategy checks in its condition part whether it is applicable to a particular task. That is, the condition of a strategy is a predicate on tasks. To *apply a strategy to a task* means to execute the strategy with respect to the task.

MULTI

When we designed proof planning with multiple strategies, we aimed at a system that allows for the flexible cooperation of independent components for proof plan refinement and modification, guided by meta-reasoning. For the implementation we decided to use a blackboard architecture because this is an established means to organize the cooperation of independent components for solving a complex problem.

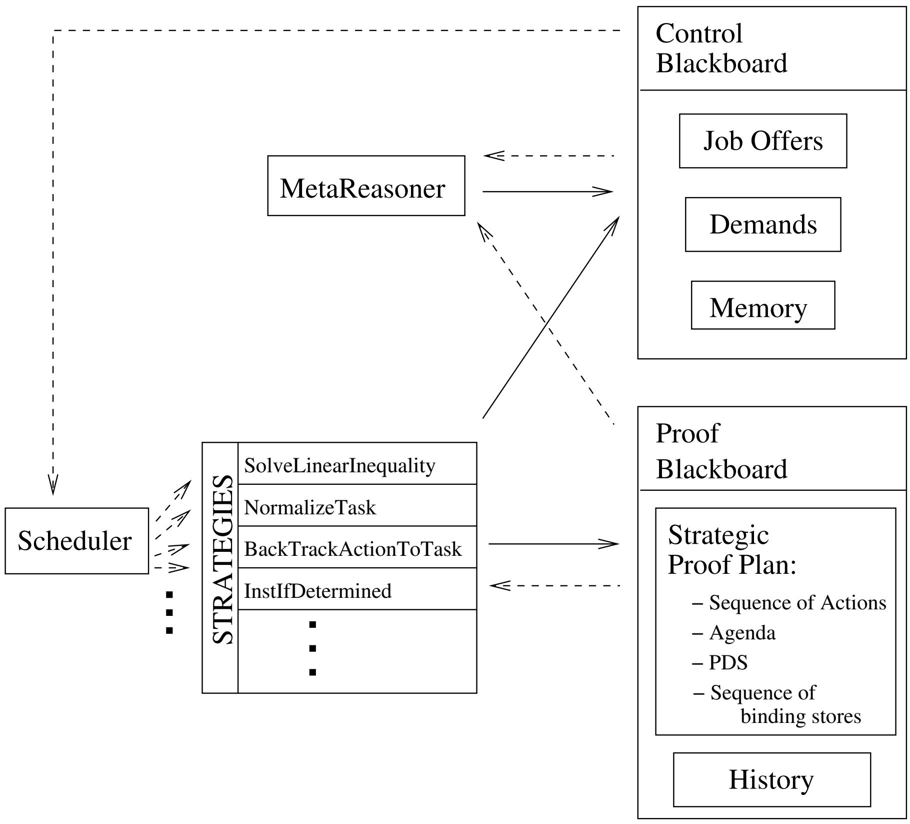


Figure 3: MULTI's blackboard architecture.

MULTI's architecture is displayed in Figure 3. In this figure dashed arrows indicate information flow whereas solid arrows indicate that a knowledge source changes the content of the respective blackboard. MULTI's architecture is similar to the HEARSAY-III [18] and the BB1 [21] blackboard systems in that it employs two blackboards, the so-called *proof blackboard* and the *control blackboard*.

We decided for a two-blackboard architecture to emphasize the importance of both the solution of the proof planning problem whose status is stored on the proof blackboard and the solution of the control problem, that is, which possible strategy should the system perform next. The proof blackboard contains the current strategic proof plan, which consists of a sequence of actions, an agenda, a $\mathcal{PDS}$, and a sequence of binding stores, which store the collected instantiations of meta-variables, as well as the strategic history. The control blackboard contains three repositories to store information relevant for the control problem: job offers, demands, and a memory.

Corresponding to the two blackboards, there are also two sets of knowledge sources shown in Figure 3 that work on these blackboards. The strategies are the knowledge sources that work on the proof blackboard. A strategy can change the proof blackboard by refining or modifying the agenda, the $\mathcal{PDS}$, and bindings of the meta-variables. Moreover, each strategy application is recorded in the history of strategies. The strategy component contains all the strategies that can be used. If a strategy's condition part is satisfied with respect to a certain task in the agenda, then the strategy posts its applicability with respect to this task as a job offer onto the control blackboard. Technically, a *job offer* is a pair (S, T) with a strategy S and a task T, which signs that T satisfies the condition of S. That is, in the terminology of blackboard systems, a task that satisfies the condition of a strategy is the event that triggers the strategy. The MetaReasoner is the knowledge source working on the control blackboard. It evaluates strategic control knowledge represented by strategic control rules in order to rank the job offers. The architecture contains a *scheduler* that checks the control blackboard, for its highest ranked job offer. Then, it executes the strategy of the job offer with respect to the task specified in the job offer. In a nutshell, MULTI operates according to the cycle in Figure 4, which passes the following steps:

Job Offer Strategies whose condition is true put a job offer onto the control blackboard.

Guidance The MetaReasoner evaluates the strategic control rules to order the job offers on the control blackboard.

Invocation A scheduler invokes the strategy who posed the highest ranked job offer.

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

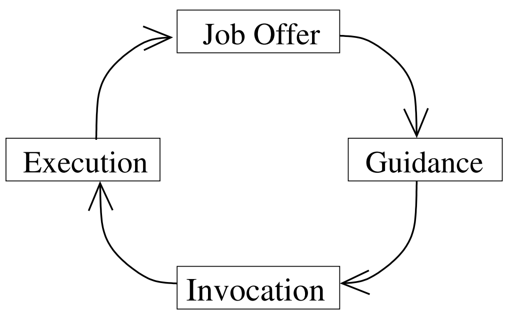


Figure 4: Cycle of MULTI.

The choice of a job offer can depend on particular demand information issued by strategies onto the control blackboard and the content of the memory. An executed strategy can reason on whether it should interrupt. This can be sensible if the strategy is stuck or if it turns out that it should not proceed before another strategy is executed. Then, the execution of a strategy interrupts itself, places demands for other strategies onto the control blackboard, and stores a pair consisting of its execution status and the demands it posed in the memory. Interrupted executions of a strategy stored in the memory place job offers for their re-invocation onto the control blackboard. A job offer from the memory consists just of a pointer to the memory entry that posed this job offer. If such a job offer is scheduled, the interrupted strategy execution is re-invoked from the memory.

By posing demands and interrupting strategies particularly desired cooperations between strategies can be realized. For instance, in order to enable a flexible instantiation of meta-variables during the proof planning process (as opposed to PLAN's approach) **PPlanner** strategies and **InstMeta** strategies have to cooperate (see section 4.2). This cooperation works as follows: The **PPlanner** strategy contains some control rules, which check whether instantiations of meta-variables should be introduced before the execution of the **PPlanner** strategy continues. If this is the case, such a control rule causes the interruption of the **PPlanner** strategy and poses the demand that an **InstMeta** strategy should be applied with respect to the instantiation-task of the meta-variable. This is possible, since interruption is an explicit choice point in the **PPlanner** algorithm. The status of the interrupted **PPlanner** strategy is stored in the memory from where it can be reinvoked as soon as the posed demand is satisfied by an application of an **InstMeta** strategy.

Multi allows to reason on existing meta-variables and possible instantiations for them. An equation of the form $mv_\alpha :=^b t_\alpha$ where mv_α is a meta-variable and t_α is a term of the same type α is called a *binding*. t is called the *instantiation* of the binding for mv. During the strategic proof planning process the current set of bindings is stored in a so-called *binding store*. Multi constructs a sequence of binding stores in order to keep track of the dependencies between the changing bindings and the introduced actions. The introduction of a new binding creates a new binding store in the sequence. All following steps are performed with respect to this current binding store. See [30] for the technical details how bindings are backtracked and how the backtracking of bindings changes the strategic proof plan under construction.

Reasoning at the Strategy-Level

In the Multi system, no order or combination of refinements or modifications on the proof blackboard is pre-defined. The choice of strategy applications results from meta-reasoning at the strategy-level that is conducted by the MetaReasoner, which evaluates the strategic control rules on the job offers on the control blackboard. Strategic control rules are formulated in the same control rule language as control rules on tasks, methods, supports and parameters, and actions (see section 2.2). They can reason about all information stored on the control blackboard and the proof blackboard (i.e., about the proof plan constructed so far and the plan process history) as well as about the mathematical domain of the proof planning problem.

The advantage of this knowledge-based control approach is that the control of Multi can be easily extended and changed by modifying the strategic control rules. In contrast, when the combination of integrated components of a system is hard-coded into a control procedure, then each extension or change requires re-implementation of parts of the main control procedure.

The backbone of the strategic control in Multi are the strategic control rules `prefer-demand-satisfying-offers`, `prefer-memory-offers`, `defer-memory-offers`, `prefer-backtrack-if-failure`, and `reject-applied-offers`. The former three rules realize the use of demands and the memory in Multi for the goal-directed cooperation of strategies.

`prefer-demand-satisfying-offers` states that, if a job offer on the control blackboard satisfies a demand on the control blackboard, then this job offer is preferred. Similarly, `prefer-memory-offers` states that, if there is a job offer from an interrupted strategy execution in the memory and all demands of this strategy execution are already satisfied, then this job offer should be preferred. `defer-memory-offers` defers job offers from interrupted strategy executions, if they have still unsatisfied demands.

The rules `prefer-backtrack-if-failure` and `reject-applied-offers` realize a basic failure reasoning and the rejection of already applied strategies. The purpose of `prefer-backtrack-if-failure` is to integrate backtracking with strategies of **PPlanner**. When a **PPlanner** strategy runs into a failure, that is, it encounters a line-task for which it finds no applicable action, then it interrupts and stores the status of its execution in the memory. `prefer-backtrack-if-failure` causes backtracking by preferring a job offer of the a **BackTrack** strategy with the line-task on which the execution of the **PPlanner** strategy failed. Afterwards, the interrupted strategy execution can be re-invoked on the changed proof blackboard. The idea behind `reject-applied-offers` is that a strategy that failed on a task should not be tried again on this task (although it is still applicable to the task, and, thus, it places a job offer onto the control blackboard). `reject-applied-offers` checks whether a job offer corresponds to a strategy execution that has already been tried but was backtracked later on. In this case, `reject-applied-offers` rejects the job offer.

The priority[4] of these control rules increases in the following order: `prefer-demand-satisfying-offers`, `prefer-memory-offers`, `defer-memory-offers`, `reject-applied-offers`, `prefer-backtrack-if-failure`. Although these control rules are the backbone of Multi's control, they realize only a default behavior and can be excluded by the user of Multi or can be overridden by other strategic control rules with higher priority. For instance, in section 5.1 we shall see how more specific control rules enable an elaborate failure reasoning.

3 The Limit Domain

3.1 Introduction

Theorems of the *limit domain* make statements about the limit $\lim_{x \to a} f(x)$ of a function f at a point a, about the limit $limseq\, X$ of a sequence X, about the continuity of a function f at a point a, and about the derivative of a function f at a point a. Since the standard definitions of limit, continuity, and derivative are

$$
\begin{array}{lcl}
lim & \equiv & \lambda f.\lambda a.\lambda l.\forall \epsilon.(0 < \epsilon \Rightarrow \\
 & & \qquad \exists \delta.(0 < \delta \wedge \forall x.(|x-a| > 0 \wedge |x-a| < \delta \Rightarrow |f(x) - l| < \epsilon))) \\
limseq & \equiv & \lambda X.\lambda l.\forall \epsilon.(0 < \epsilon \Rightarrow \exists k.(k \in \mathbb{N} \wedge \forall n.(n \in \mathbb{N} \wedge n > k \Rightarrow |(X\ n) - l| < \epsilon))) \\
cont & \equiv & \lambda f.\lambda a.\forall \epsilon.(0 < \epsilon \Rightarrow \exists \delta.(0 < \delta \wedge \forall x.(|x-a| < \delta \Rightarrow |f(x) - f(a)| < \epsilon))) \\
deriv & \equiv & \lambda f.\lambda a.\lambda f'.\lim_{x \to a} \frac{f(x)-f(c)}{x-c}.
\end{array}
$$

[4] The MetaReasoner evaluates first the strategic control rules with lower priority. Since they are evaluated later on, the strategic control rules with higher priority cause the final changes of the alternative list of job offers.

The proofs of these theorems are so-called *ϵ-δ-proofs*, i.e., proofs that postulate the existence of a δ such that a conjecture of the form $\ldots |X| < \epsilon$ is proved under assumptions of the form $\ldots |Y| < \delta$.

Notation Instead of the formula $lim(f_{\nu\nu}, a_\nu, l_\nu)$ we henceforth write the more common equation expression $\lim_{x \to a} f(x) = l$. Analogously, we write $limseq\, X = l$ instead of $limseq(X_{\nu\nu}, l_\nu)$ and $deriv(f, a) = f'$ instead of $deriv(f_{\nu\nu}, a_\nu, f'_\nu)$.

An example theorem from the limit domain is LIM+ that states that the limit of the sum of two functions f and g equals the sum of their limits; that is, if $\lim_{x \to a} f(x) = l_1$ and $\lim_{x \to a} g(x) = l_2$ then $\lim_{x \to a}(f(x) + g(x)) = l_1 + l_2$. When the definition of $\lim_{x \to a}$ is expanded, the corresponding 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 theorem becomes

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

Similar theorems in this class are LIM- and LIM* for the difference and the product of limits of functions. Moreover, there are corresponding theorems about continuity. Continuous+ states that the sum of two continuous functions is continuous, and Continuous- and Continuous* make similar statements for the difference and product of continuous functions. We shall introduce some further examples from the limit domain in the remainder of the report.

When proving a limit theorem like LIM+, a δ has to be constructed that depends on an ϵ such that certain estimations hold. This is a non-trivial task for students as well as for traditional automated theorem provers.[5] The typical way a mathematician discovers a suitable δ is by incrementally restricting the possible values of δ. When proof planning limit theorems, PLAN adapts this approach by cooperating with the constraint solver $\mathcal{CoSIE}$ [37], a constraint solver for inequalities and equations over the field of real numbers: (in)equality tasks that are simple enough for $\mathcal{CoSIE}$ (i.e., tasks that are in the input language for $\mathcal{CoSIE}$) are passed to $\mathcal{CoSIE}$ and $\mathcal{CoSIE}$ provides suitable instantiations for δ, when solutions for meta-variables are computed and inserted into the final proof plan.

3.2 Methods to Prove Limit Problems

For finding ϵ-δ-proofs, among others, the general methods $\exists$I-B, $\exists$E-F, $\forall$I-B, $\forall$E-F, $\wedge$I-B, $\wedge$E-F, $\Rightarrow$I-B, $\Rightarrow$E-F, SETFOCUS-B, and =SUBST-B and the domain-specific methods

[5]BLEDSOE proposed in 1990 several versions of LIM+ as a challenge problem for automated theorem proving [6]. The simplest versions of LIM+ (problem 1 and 2 in [6]) are at the edge of the capabilities of traditional automated theorem provers but LIM* is certainly beyond their capabilities.

TellCS-B, TellCS-F, AskCS-B, Solve*-B, FactorialEstimate-B, Simplify-B, Simplify-F, and ComplexEstimate-B are required. We start with a brief explanation of the general methods and then discuss the domain-specific methods with more details. =Subst-B is explained already in section 2.2.

Actions of the methods ∀I-B, ∃E-F, ∃I-B, ∀E-F, ∧I-B, ∧E-F, ⇒I-B, and ⇒E-F apply certain natural deduction rules. Actions of ∀I-B perform backward applications of the ND-rule $\forall_I$ by reducing a goal with formula $\forall x. P[x]$ to a new goal with formula $P[c]$, where the variable x is replaced by a constant c. Similarly, actions of ∀E-F perform a forward $\forall_E$ step and derive a new support $P[mv]$ with a new meta-variable mv from a given support $\forall x. P[x]$. Actions of ∃I-B perform a backward $\exists_I$ step. They close a goal with formula $\exists x. P[x]$ and introduce a goal with the formula $P[mv]$ in which x is replaced by a new meta-variable mv. Actions of the ∃E-F method perform a forward step with the $\exists_E$ rule. They introduce a new hypothesis $P[c]$ for a support $\exists x. P[x]$, where the variable x is replaced by a constant c, and thereby also reduce a goal to a new goal with the new hypothesis. Actions of ∧I-B perform a backward $\wedge_I$ step and reduce a task whose goal has the formula $A_1 \wedge A_2$ to new tasks whose goals have the formulas A_1 and A_2. Actions of ∧E-F perform the corresponding forward $\wedge_E$ decompositions on conjunctive support lines. Actions of ⇒I-B perform a backward $\Rightarrow_I$ step and reduce a task with goal $A \Rightarrow B$ to a new task whose goal has the formula B and A as additional hypothesis. Moreover, A becomes the formula of a new support for this task. Actions of ⇒E-F perform an $\Rightarrow_E$ step. When applied to a task with goal C and an support with formula $A \Rightarrow B$ they introduce two new tasks: a task with goal C, which contains also a new support with B as formula, and a task with goal A. Actions of SetFocus-B highlight a subformula in a support.

Figure 5 and Figure 6 show the two methods ComplexEstimate-B and TellCS-B whose application conditions comprise calls to external systems, respectively. Both methods are central for planning limit problems.

Method: ComplexEstimate-B				
premises	$L_1, \oplus L_2, \oplus L_4, \oplus L_5, \oplus L_6, \oplus L_7$			
conclusions	$\ominus L_9$			
appl. conds.	*linearextract*(a, b, l, k, σ)			
proof schema	$L_1.$	$\triangle$	$\vdash \lvert a\rvert < \epsilon'$	$()$
	$L_2.$	$\triangle$	$\vdash \epsilon'\sigma < \frac{\epsilon\sigma}{2*mv}$	$(Open)$
	$L_3.$	$\triangle$	$\vdash \lvert a\sigma\rvert < \frac{\epsilon\sigma}{2*mv}$	$(< trans\ L_1\ L_2)$
	$L_4.$	$\triangle$	$\vdash \lvert k\sigma\rvert \leq mv$	$(Open)$
	$L_5.$	$\triangle$	$\vdash \lvert l\sigma\rvert < \frac{\epsilon\sigma}{2}$	$(Open)$
	$L_6.$	$\triangle$	$\vdash 0 < mv$	$(Open)$
	$L_7.$	$\triangle$	$\vdash conjunct$	$(Open)$
	$L_8.$	$\triangle$	$\vdash b\sigma \doteq k\sigma * a\sigma + l\sigma$	(CAS)
	$L_9.$	$\triangle$	$\vdash \lvert b\rvert < \epsilon$	$(fix\ L_3\ L_4\ L_5\ L_6\ L_7\ L_8)$

Figure 5: The ComplexEstimate-B method.

ComplexEstimate-B is a method for estimating the magnitude of the absolute value of complex terms.[6] ComplexEstimate-B is applicable to tasks whose goal has the formula

[6]ComplexEstimate-B essentially is a reconstruction (see [32]) of Bledsoe's limit heuristic that was

$|b| < \epsilon$ (corresponding to line L_9 in Figure 5) and that have supports with formula $|a| < \epsilon'$ (corresponding to line L_1 in Figure 5). In its application conditions ComplexEstimate-B uses the function *linearextract*. When applied to a and b *linearextract* employs the computer algebra system Maple [38] to compute suitable terms k and l such that $b = k * a + l$ holds. *linearextract* also computes a substitution σ such that $b\sigma = k\sigma * a\sigma + l\sigma$ holds (where $b\sigma, k\sigma, l\sigma$ result from b, k, l by the application of the substitution σ, respectively). Thereby, the substitution σ maps meta-variables in a, b to terms. ComplexEstimate-B is applicable only, if Maple provides k and l such that *linearextract* evaluates to true. If this is the case, the application of a corresponding action of the method reduces the original task to five tasks whose goals correspond to the lines L_2, L_4, L_5, L_6, L_7 in Figure 5. L_7 has the formula $conjunct$. This formula is the conjunction of the mappings of the substitution σ. That is, if σ maps the meta-variables $mv_1, \ldots, mv_n$ to the terms $t_1, \ldots, t_n$, respectively, then $conjunct$ has the form $mv_1 \doteq t1 \wedge \ldots \wedge mv_n \doteq t_n$. If σ is empty, then $conjunct$ is simply $True$, the primitive truth. The justification fix for L_9 in the proof schema is only an abbreviation that stands for a sequence of about 20 tactic steps that comprises, in particular, an application of the triangle inequality. The application of Maple is reflected in line L_8 of the proof schema, which is justified by the tactic CAS. When this tactic is expanded, it employs the sapper [43] system to obtain a formal proof of the statement $b\sigma = k\sigma * a\sigma + l\sigma$ suggested by Maple.

For instance, when applied to a task with formula $|(f(c_x) - g(c_x)) - (l_1 - l_2)| < \epsilon$ and a support with formula $|f(mv_x) - l_1| < \epsilon'$ with a meta-variable mv_x, then *linearextract* succeeds and provides $k = 1$, $l = g(c_x) - l_2$, and a substitution σ that maps mv_x to c_x. The application of a corresponding action of ComplexEstimate-B reduces the given task to new tasks whose goals are $|1| \leq mv$, $\epsilon' < \frac{\epsilon}{2*mv}$, $|g(c_x) - L_2| < \frac{\epsilon}{2}$, $0 < mv$, and $mv_x \doteq c_x$.

Method: TellCS-B	
premises	
conclusions	$\ominus L_1$
appl. conds.	(1) *metavar-in*$(a) \vee$ *metavar-in*(b) (2) *test-CS*$(\mathcal{CoSIE}, a\ rel\ b)$
proof schema	$L_1.\quad \triangle \quad \vdash rel_{o\nu\nu}(a_\nu, b_\nu) \quad (ProveCS)$

Figure 6: The TellCS-B method.

The method TellCS-B realizes an interface to $\mathcal{CoSIE}$. TellCS-B is applicable to tasks with formulas $rel(a, b)$ where rel is a binary predicate. Examples of matching predicates are, for instance, $<, \leq$. In its application conditions TellCS-B first tests whether a or b contain some meta-variables. If this is the case, $rel(a, b)$ is interpreted as a constraint on these meta-variables. TellCS-B applies then the function *test-CS* that connects to $\mathcal{CoSIE}$ to test (1) whether $rel(a, b)$ is a syntactically valid constraint for $\mathcal{CoSIE}$ (in particular, rel has to be $<, \leq, >, \geq, \doteq$, or $\neq$) and (2) whether $rel(a, b)$ is not inconsistent with the current constraint store of $\mathcal{CoSIE}$. If this is the case, TellCS-B is applicable. The introduction of the corresponding action of TellCS-B closes the goal without producing further subtasks and passes $rel(a, b)$ as new constraint to $\mathcal{CoSIE}$.

used in a special-purpose program [7].

$\mathcal{CoSIE}$ can provide instantiations of the constrained meta-variables that are consistent with the collected constraints. For instance, suppose during the proof planning process there are three tasks whose goals have the formulas $0 < mv_D$, $mv_D \leq \delta_1$, $mv_D \leq \delta_2$, which all contain the meta-variable mv_D. All three goals are closed by actions of TELLCS-B. Moreover, suppose there are also two supports with formulas $0 < \delta_1$ and $0 < \delta_2$, which are passed to $\mathcal{CoSIE}$ by actions of the method TELLCS-F, which is the analogous of TELLCS-B to pass constraints in supports to $\mathcal{CoSIE}$.[7] From the resulting constraint store, $\mathcal{CoSIE}$ can compute $min(\delta_1, \delta_2)$ as suitable instantiation for mv_D. Moreover, $\mathcal{CoSIE}$ provides traces of its computations, which can be used to expand the applications of the actions of TELLCS-B.

Another method that establishes a connection to $\mathcal{CoSIE}$ is ASKCS-B. Similar to TELLCS-B, this method is applicable to tasks whose goal formulas are of the form $rel(a, b)$. But whereas TELLCS-B demands that a or b contain some meta-variables, ASKCS-B covers the case that a and b contain no meta-variables. An application condition of ASKCS-B passes the formula to $\mathcal{CoSIE}$ and asks $\mathcal{CoSIE}$ whether the formula holds with respect to the constraints collected so far. If this is the case, then ASKCS-B closes the goal. Since $\mathcal{CoSIE}$ can also handle formulas on concrete real numbers, for instance, $1 < 2$ or $0 \leq 0$, ASKCS-B can also close goals whose formulas are expressions on concrete real numbers.

Note that besides TELLCS-B and TELLCS-F also the methods ∀I-B and ∃E-F pass constraints to $\mathcal{CoSIE}$. Actions of ∀I-B perform backward applications of the ND-rule $\forall_I$ by reducing a task with task formula $\forall x. P[x]$ to a new task with task formula $P[c]$, where the variable x is replaced by a constant c. For each meta-variable mv in $P[c]$ an action of ∀I-B also passes the *Eigenvariable constraint* $c \notin mv$ to $\mathcal{CoSIE}$ that states that the instantiation for mv is not allowed to contain c. This constraint guarantees the adherence with the Eigenvariable conditions of the $\forall_I$ rule of the ND-calculus. Actions of the ∃E-F method perform a forward step with the $\exists_E$ rule. Similar to action of ∀I-B they pass Eigenvariable constraints to $\mathcal{CoSIE}$ that demand the adherence of the Eigenvariable conditions of the $\exists_E$ rule.

Applications of the SOLVE*-B method exploit transitivity of $<, >, \leq, \geq$ and reduce a goal with formula $a_1 < b_1$ to a new task with formula $b_2\sigma \leq b_1\sigma$ in case a support $a_2 < b_2$ exists and a_1, a_2 can be unified by the substitution σ. Then, also a further new task is created whose formula is the conjunction of all mappings of the substitution σ (compare description of method COMPLEXESTIMATE-B).

SIMPLIFY-B passes the formula of a given goal to the computer algebra system MAPLE and asks MAPLE to simplify it. If MAPLE succeeds, then the given goal is reduced to a new goal with the simplified formula. The analogous method SIMPLIFY-F derives a support with a simpler formula from a given support by calling MAPLE. The method FACTORIALESTIMATE-B 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.

[7]The functionality of TELLCS-F is slightly extended as opposed to TELLCS-B. TELLCS-F also passes constraints to $\mathcal{CoSIE}$ that contain no meta-variables, whereas TELLCS-B handles only constraints with meta-variables.

3.3 Proof Planning Limit Problems with PLAN

When applied to an ϵ–δ–problem, PLAN first decomposes the initial task with a complex formula into subtasks whose formulas are (in)equalities. This is done by actions that decompose formulas in tasks, e.g., actions of the methods $\wedge$I-B, $\forall$I-B, $\exists$I-B etc.

When faced with an inequality goal, PLAN first tries to apply the methods TELLCS-B and ASKCS-B, which both employ $\mathcal{C}o\mathcal{SIE}$. TELLCS-B passes the goal to $\mathcal{C}o\mathcal{SIE}$, whereas ASKCS-B asks $\mathcal{C}o\mathcal{SIE}$ whether the goal is entailed by its current constraints. If an inequality is too complex to be handled by $\mathcal{C}o\mathcal{SIE}$, then PLAN tries to apply methods that reduce an inequality to simpler inequalities. So, PLAN successively produces simpler inequalities, until it reaches inequalities that are accepted by $\mathcal{C}o\mathcal{SIE}$. This approach – handle with $\mathcal{C}o\mathcal{SIE}$ or simplify – is guided by the control rule `prove-inequality` given in Figure 2 in section 2.2, which is the central control rule to accomplish ϵ-δ-proofs with PLAN. In its IF-part `prove-inequality` checks whether the current goal is an inequality. If this is the case, it prefers the methods TELLCS-B, TELLCS-F, ASKCS-B, SIMPLIFY-B, SIMPLIFY-F, SOLVE*-B, COMPLEXESTIMATE-B, FACTORIALESTIMATE-B, and SETFOCUS-B in this order.

In order to apply methods such as COMPLEXESTIMATE-B and SOLVE*-B unwrapping of (in)equality supports from the initial assumptions is necessary. This is realized as follows: First, PLAN applies SETFOCUS-B to highlight a promising subformula in a support (the application of SETFOCUS-B is suggested by `prove-inequality` if no other method is applicable, promising subformulas are chosen by another control rule guiding the supports and parameters choice point). Next, the highlighted subformula is unwrapped by actions that decompose supports, e.g., actions of the methods $\wedge$E-F, $\forall$E-F, $\exists$E-F etc.

Finally, when no task is left and PLAN invokes the function *employ-CS*, $\mathcal{C}o\mathcal{SIE}$ computes instantiations for the meta-variables that are consistent with the collected constraints.

Next, we briefly discuss the application of PLAN to the LIM+ problem.[8] PLAN first decomposes the initial theorem to tasks with the formulas $0 < mv_\delta$ and $|(f(c_x) + g(c_x)) - (l_1 + l_2)| < c_\epsilon$ where mv_δ is a meta-variable introduced for δ and c_x and c_ϵ are constants that replace x and ϵ, respectively. Moreover, the assumptions $0 < c_\epsilon$, $|c_x - a| > 0$, and $|c_x - a| < mv_\delta$ are created during the decomposition of the initial theorem and become supports of the new tasks. $0 < mv_\delta$ can be passed directly to $\mathcal{C}o\mathcal{SIE}$ by an action of TELLCS-B. $|(f(c_x)+g(c_x))-(l_1+l_2)| < c_\epsilon$ cannot be passed to $\mathcal{C}o\mathcal{SIE}$ directly. This triggers the decomposition of one of the two initial assumptions. If the initial assumption on f is decomposed, then PLAN obtains as new supports $0 < c_{\delta_1}$ and $|f(mv_{x_1}) - l_1| < mv_{\epsilon_1}$. Now PLAN can compute and introduce an action of COMPLEXESTIMATE-B using the latter new support line. During the evaluation of the application conditions of COMPLEXESTIMATE-B the substitution $mv_{x_1} \mapsto c_x$ is created and the computer algebra system MAPLE computes a decomposition $(f(c_x)+g(c_x))-(l_1+l_2) = 1*(f(c_x)-l_1)+(g(c_x)+l_2)$ (that is, the variables k and l of COMPLEXESTIMATE-B are bound to 1 and $g(c_x) - l_2$, respectively). Thus, the action of COMPLEXESTIMATE-B introduces new tasks with formulas $mv_{\epsilon_1} < \frac{c_\epsilon}{2*mv}$, $|1| \leq mv$, $0 < mv$, $|g(c_x) - l_2| < \frac{c_\epsilon}{2}$, and $mv_{x_1} \doteq c_x$. The formulas of the first three (new) tasks and of the last one can all be passed directly to $\mathcal{C}o\mathcal{SIE}$ by actions of TELLCS-B. To deal with the remaining task with formula $|g(c_x) - l_2| < \frac{c_\epsilon}{2}$ PLAN decomposes the

[8]A detailed description on how MULTI solves this problem is given in section 4.

second initial assumption (on g) and derives new support lines with formulas $0 < c_{\delta_2}$ and $|g(mv_{x_2}) - l_2| < mv_{\epsilon_2}$. An action of SOLVE*-B reduces the goal with respect to the second new support to two new tasks with formulas $mv_{\epsilon_2} \leq \frac{c_\epsilon}{2}$ and $mv_{x_2} \doteq c_x$. Both tasks are closed by actions of TELLCS-B and their formulas are passed to $\mathcal{CoSIE}$.

The decomposition of the initial assumptions results not only in the used support lines but also in tasks with the formulas $0 < mv_{\epsilon_1}$, $|mv_{x_1} - a| > 0$, $|mv_{x_1} - a| < c_{\delta_1}$ from the assumption on f and the analogue tasks from the assumption on g. The task $0 < mv_{\epsilon_1}$ is closed by the introduction of an action of TELLCS-B, which passes the formula to $\mathcal{CoSIE}$. To close the other tasks PLAN introduces actions of the method SOLVE*-B that use the supports with formulas $|c_x - a| < mv_\delta$ and $|c_x - a| > 0$ (from the decomposition of the initial goal). The application of SOLVE*-B to the task $|mv_{x_1} - a| < c_{\delta_1}$ and the support $|c_x - a| < mv_\delta$ results in two new tasks with formulas $mv_\delta \leq c_{\delta_1}$ and $mv_{x_1} \doteq c_x$. The application of SOLVE*-B to the task $|mv_{x_1} - a| > 0$ and the support $|c_x - a| > 0$ results also in two new tasks with formulas $0 \leq 0$ and $mv_{x_1} \doteq c_x$. Whereas $0 \leq 0$ is closed by an actions of ASKCS-B the other three tasks are closed by actions of TELLCS-B, which pass their formulas to $\mathcal{CoSIE}$. The corresponding tasks from the assumption on g are handled in the same way. Thereby the constraints $mv_\delta \leq c_{\delta_2}$, $mv_{x_2} \doteq c_x$, and $mv_{x_2} \doteq c_x$ are passed to $\mathcal{CoSIE}$. Moreover, some actions of the TELLCS-F method during the planning process pass constraints in support lines to $\mathcal{CoSIE}$: $0 < c_{\delta_1}$, $0 < c_{\delta_2}$, $0 < c_\epsilon$.

After propagating constraints, $\mathcal{CoSIE}$ has the final constraint store in Figure 7. When asked for suitable instantiations for the meta-variables, $\mathcal{CoSIE}$ provides the bindings $mv_{x_1} \mapsto c_x, mv_{x_2} \mapsto c_x, mv \mapsto 1$, $mv_{\epsilon_1} \mapsto \frac{c_\epsilon}{2}$, $mv_{\epsilon_2} \mapsto \frac{c_\epsilon}{2}$, and $mv_\delta \mapsto min(c_{\delta_1}, c_{\delta_2})$. These instantiations computed by $\mathcal{CoSIE}$ are exactly the solutions that standard textbooks use for δ, ϵ_1, and ϵ_2 for LIM+.

$$
\begin{array}{llll}
mv_{x_1} = c_x & & & \\
mv_{x_2} = c_x & & & \\
0 & < & c_{\delta_1} & < & +\infty \\
0 & < & c_{\delta_2} & < & +\infty \\
0 & < & c_\epsilon & < & +\infty \\
0 & < & mv_{\epsilon_1} & \leq & \frac{c_\epsilon}{2}, \frac{c_\epsilon}{2*mv} \\
0 & < & mv_{\epsilon_2} & \leq & \frac{c_\epsilon}{2} \\
0 & < & mv_\delta & \leq & c_{\delta_1}, c_{\delta_2} \\
1 & \leq & mv & \leq & \frac{c_\epsilon}{2*mv_{\epsilon_1}}
\end{array}
$$

Figure 7: The final constraint store of $\mathcal{CoSIE}$ for LIM+.

PLAN can successfully plan all the challenge problems of BLEDSOE [6], i.e., the limit theorems LIM+, LIM-, LIM*, the theorems Continuous+, Continuous−, Continuous*, $\lim_{x \to a} x = a$, $\lim_{x \to a} c = c$, and the theorem that the composition of continuous functions is again continuous. Moreover, we tried to apply PLAN to tackle systematically the limit problems recorded in the textbook of BARTLE and SHERBERT "Introduction to Real Analysis" [2]. A summary of these experiments can be found in the master thesis of Jürgen Zimmer [50].

3.4 Drawbacks of PLAN

It turned out that PLAN failed to plan several theorems from [2]. This is not due to missing or inappropriate methods but due to PLAN's inadequate algorithm. This observation (among others) motivated the development of the MULTI system. We illustrate the drawbacks of PLAN with the discussion of two examples.

3.4.1 Flexible Meta-Variable Instantiation

PLAN instantiates meta-variables only if all tasks are closed. This restriction causes that PLAN fails on some problems since it cannot flexibly instantiate meta-variables during the planning process whenever needed or beneficial (even if there are still tasks) guided by meta-reasoning.

For instance, consider exercise 4.1.3 in the analysis textbook [2].

Exercise 4.1.3 Let $f : \mathbb{R} \to \mathbb{R}$ and let $c \in \mathbb{R}$. Show that $\lim_{x_1 \to c} f(x_1) = l$ if and only if $\lim_{x \to 0} f(x+c) = l$.

Two implications have to be proof planned for solving this exercise:

$$\lim_{x_1 \to c} f(x_1) = l \quad \Rightarrow \quad \lim_{x \to 0} f(x+c) = l \tag{1}$$

and

$$\lim_{x \to 0} f(x+c) = l \quad \Rightarrow \quad \lim_{x_1 \to c} f(x_1) = l \tag{2}$$

With respect to the definition of limit given in section 3.1 for (1) we need to show that

$$\forall \epsilon. (0 < \epsilon \Rightarrow \exists \delta. (0 < \delta \wedge \forall x. (|x - 0| > 0 \wedge |x - 0| < \delta \Rightarrow |f(x+c) - l| < \epsilon)))$$

holds under the assumption that

$$\forall \epsilon_1. (0 < \epsilon_1 \Rightarrow \exists \delta_1. (0 < \delta_1 \wedge \forall x_1. (|x_1 - c| > 0 \wedge |x_1 - c| < \delta_1 \Rightarrow |f(x_1) - l| < \epsilon_1))).$$

PLAN first decomposes the task formula. This results in new tasks with formulas $0 < mv_\delta$ and $|f(c_x + c) - l| < c_\epsilon$ and new supports with formulas $|c_x - 0| < mv_\delta$ and $|c_x - 0| > 0$ where mv_δ is a meta-variable and c_x and c_ϵ are constants. The new task with formula $0 < mv_\delta$ can be directly closed with an action of TELLCS-B. The formula $|f(c_x + c) - l| < c_\epsilon$ of the other task is too complex to be sent to $\mathcal{CoSIE}$ directly. Hence PLAN unwraps the assumption which results in a new support with formula $|f(mv_{x_1}) - l| < mv_{\epsilon_1}$ as well as two new tasks with formulas $|mv_{x_1} - c| < c_\delta$ and $|mv_{x_1} - c| > 0$. Now the task with formula $|f(c_x + c) - l| < c_\epsilon$ can be closed by an action of SOLVE*-B that uses the new support. This action yields new tasks with the formulas $mv_{\epsilon_1} \leq c_\epsilon$ and $mv_{x_1} \doteq c_x + c$, which both can be closed and passed to $\mathcal{CoSIE}$ by actions of TELLCS-B.

The tasks with formulas $|mv_{x_1} - c| < c_{\delta_1}$ and $|mv_{x_1} - c| > 0$ should be closed by the method SOLVE*-B using the supports $|c_x - 0| < mv_\delta$ and $|c_x - 0| > 0$. However, SOLVE*-B is not applicable and hence proof planning is blocked because $(mv_{x_1} - c)$ and $(c_x - 0)$ cannot be unified. If PLAN could use the information that $c_x + c$ is the (only) suitable instantiation for mv_{x_1} available in the constraint store, then an eager instantiation of mv_{x_1} by $c_x + c$ would unblock the planning because the formulas of the task would be instantiated to $|c_x + c - c| < c_{\delta_1}$ and $|c_x + c - c| > 0$. Then, the tasks could be reduced to tasks with the simplified formulas $|c_x| < c_{\delta_1}$ and $|c_x| > 0$ to which SOLVE*-B would be applicable using the simplified supports $|c_x| < mv_\delta$ and $|c_x| > 0$ that are implied by $|c_x - 0| < mv_\delta$ and $|c_x - 0| > 0$.

3.4.2 Flexible Backtracking and Reasoning on Failures

If a task occurs for which PLAN fails to compute an applicable action (we call this situation a *failure*), then PLAN's only remedy is dependency directed backtracking by deleting the action that introduced this task. Moreover, failures are the only events that trigger backtracking in PLAN. These restrictions cause that PLAN fails on some limit problems and that it cannot make use of knowledge of how to deal and productively make use of failures.

For instance, IRELAND and BUNDY describe in [24, 25] how to patch failed proof attempts of the proof planner CLAM by exploiting information on failures. We encountered situations in the limit domain where failures can be productively used. The *Cont-If-Deriv* theorem states that a function f is continuous at point a if it has a derivative f' at point a. In the proof planning process the definition of continuous and derivative in both, the task and the assumption, is replaced first by its ϵ–δ–definition. Further decomposition of the task formula results in a task with formula $|f(c_x) - f(a)| < c_\epsilon$ where c_ϵ and c_x are constants. The decomposition of the assumption results in a new support with formula $|\frac{f(mv_{x'})-f(a)}{mv_{x'}-a} - f'| < mv_{\epsilon'}$ where $mv_{x'}$ and $mv_{\epsilon'}$ are new meta-variables. Indeed, the task can be proved under this assumption. This results — among others — in a task with the formula $mv_{x'} \doteq c_x$, which is closed by an action of the method TELLCS-B that passes the formula to $\mathcal{C}o\mathcal{SIE}$. Unfortunately, another task with formula $|mv_{x'} - a| > 0$ is also created during the decomposition of the assumption. This task can be reduced to a task with the formula $mv_{x'} \neq a$. Suppose, we use the information $mv_{x'} \doteq c_x$ by eager instantiation of meta-variables such that this tasks results in $c_x \neq a$. Nevertheless, proof planning reaches a dead end at this task since there is no support available to close it. How can we deal with this failure? The analysis of this and similar situations indicates that a case-split is needed on $c_x \neq a \vee c_x \doteq a$, which has to be introduced before the task $|f(c_x) - f(a)| < c_\epsilon$ is tackled. Then, this task has to be proved for two cases: In the first case, $c_x \neq a$ is assumed and the task $|f(c_x) - f(a)| < c_\epsilon$ can be proved from the assumption as described above. Obviously the problematic subtask $c_x \neq a$ can now be closed directly by the assumption $c_x \neq a$ of the case-split. In the second case, $c_x \doteq a$ is assumed and the task follows since $|f(c_x) - f(a)| < c_\epsilon$ can be simplified to $|f(a) - f(a)| = 0 < c_\epsilon$ by an action of =SUBST-B. The resulting task is satisfied by a support with the same formula that resulted from the decomposition of the original task. When should the case-split be introduced? By mathematical intuition it should be introduced when the task $c_x \neq a$ is created and cannot be closed. This demands reasoning about this failure, to backtrack to a certain point in the search space, and to introduce the case-split. An *a priori* introduction of a case-split is not possible since neither the need for a case-split nor the elements for the cases are given.

Another situation where we could make use of failures in a productive way arises in examples like exercise 4.1.3 (see last section). We have to show that

$$\forall \epsilon_1 . (0 < \epsilon_1 \Rightarrow \exists \delta_1 . (0 < \delta_1 \wedge \forall x_1 . (|x_1 - c| > 0 \wedge |x_1 - c| < \delta_1 \Rightarrow |f(x_1) - l| < \epsilon_1)))$$

holds under the assumption that

$$\forall \epsilon . (0 < \epsilon \Rightarrow \exists \delta . (0 < \delta \wedge \forall x . (|x - 0| > 0 \wedge |x - 0| < \delta \Rightarrow |f(x + c) - l| < \epsilon))).$$

The decomposition of the task formula results — among others — in a task with formula $|f(c_{x_1}) - l| < c_{\epsilon_1}$. Unwrapping the assumption yields a new support line with formula $|f(mv_x + c) - l| < mv_\epsilon$. Actually, SOLVE*-B should be applied to this task. However the computation of a corresponding action of this method fails since c_{x_1} and $mv_x + c$ cannot be unified. How can we deal with this failure? We analyzed this situation and similar ones and found that the application of methods is sometimes blocked because unifications of terms do not succeed but have a residue $t_1 = t_2$. For some examples this residue $t_1 = t_2$ can be collected by $\mathcal{CoSIE}$ since it is not inconsistent with $\mathcal{CoSIE}$'s current constraint store. The analysis of these examples indicates that, if (1) a method application is blocked because of a failed unification with a residue $t_1 = t_2$ and (2) $\mathcal{CoSIE}$ states that this residue $t_1 = t_2$ is not inconsistent with its current constraint store, then we can speculate the lemma $t_1 = t_2$ as new open task and rewrite the task on which the planner failed with this equation. Afterwards the speculated lemma can be closed by an action of TELLCS-B and the rewritten task can be solved since the unification becomes unblocked.[9] In our example we would speculate the lemma $mv_x + c \doteq c_{x_1}$ and would reduce the task with respect to this equation to a new task with formula $|f(c_{x_1}) - l| < mv_\epsilon$. Then, SOLVE*-B is applicable with respect to the rewritten task and the support $|f(c_{x_1}) - l| < c_{\epsilon_1}$. Similar to the introduction of a case-split, the lemma $t_1 = t_2$ cannot be speculated *a priori*. First, the application of methods such as SOLVE*-B has to fail. Then, the analysis of this failure can provide suitable t_1 and t_2 such that $t_1 = t_2$ can be speculated.

4 ϵ-δ-Proof Plans with MULTI

In this section, we describe the general approach to tackle limit problems with MULTI. First, we introduce the employed strategies and their cooperation in section 4.1 and section 4.2. Then, we discuss the application of MULTI to the LIM+ problem in section 4.3. Finally, we explain how MULTI solves problems such as exercise 4.1.3 (on which PLAN fails, see section 3.4) by enabling eager instantiation. When illustrating the application of MULTI with examples, we try to avoid the tedious details. In particular, we skip the technical details of the constructed strategic proof plans. Rather, we use the $\mathcal{PDS}$ as a means to display and discuss the constructed proof plans.

[9] In general, the introduction of unification residues as new tasks opens a Pandora's box: whenever we deal with a residue we introduce some new residues, which in turn must be dealt with. How we restrict the introduction of residues in tasks in order to avoid this problem is described in section 5.2.

4.1 The Strategies

PPLANNER Strategies

The methods and control rules for ϵ-δ-proofs are structured into the three **PPLANNER** strategies NormalizeLineTask, UnwrapHyp, and SolveInequality.

The strategy SolveInequality, see Table 2, is central for accomplishing ϵ-δ-proofs with MULTI. It is applicable to prove line-tasks whose formulas are inequalities or whose formulas can be reduced to inequalities. A formula is reducible to inequalities if it contains defined terms whose unfolding will result in inequalities, for instance, *lim*, *limseq*, *cont*, and *deriv*. SolveInequality mainly comprises methods that deal with inequalities such as COMPLEXESTIMATE-B, TELLCS-B, TELLCS-F, ASKCS-B, and SOLVE*-B. To unfold occurrences of defined concepts it employs the methods DEFNUNFOLD-B and DEFNUNFOLD-F. DEFNUNFOLD-B is the method for unfolding defined concepts in goals, whereas DEFNUNFOLD-F unfolds defined concepts in supports. The list of control rules of SolveInequality contains the rules `prove-inequality` and `eager-instantiate`. The strategy terminates, when there are no further line-tasks whose formulas are inequalities or whose formulas can be reduced to inequalities.

Strategy: SolveInequality		
Condition	*inequality-task*	
Action	Algorithm	**PPLANNER**
	Methods	COMPLEXESTIMATE-B, TELLCS-B, TELLCS-F, SOLVE*-B, ASKCS-B, DEFNUNFOLD-F, DEFNUNFOLD-B ...
	C-Rules	`prove-inequality`, `eager-instantiate`, ...
	Termination	*no-inequalities*

Table 2: The SolveInequality strategy.

The central idea of SolveInequality to tackle inequality goals is similar to the approach of PLAN when accomplishing ϵ-δ-proofs (see section 3.3): pass to $\mathcal{CoSIE}$ or simplify. Hence, also similar to PLAN's approach, the control rule `prove-inequality` given in Figure 2 in section 2.2 is central in SolveInequality.

SolveInequality comprises the knowledge of how to deal with inequalities and with problems that can be reduced to inequalities. As opposed thereto, the strategies NormalizeLineTask and UnwrapHyp comprise the domain-independent, general knowledge of how to decompose complex formulas with logical connectives and quantifiers.

NormalizeLineTask (see Table 3) is used to decompose line-tasks whose goals are complex formulas with logical connectives and quantifiers. Typical methods in NormalizeLineTask are $\wedge$I-B and $\forall$I-B (see section 3.2). NormalizeLineTask terminates, when all complex line-tasks are decomposed to literal line-tasks.

The aim of UnwrapHyp (see Table 4) is to unwrap a focused subformula of an assumption in order to make it available for proving a line-task. The list of its methods includes, for instance, $\forall$E-F and $\wedge$E-F. The control rule `tackle-focus` determines that, if UnwrapHyp is applied, then the actions of the available methods can be used only if they

Strategy: NormalizeLineTask		
Condition	*complex-line-task*	
Action	Algorithm	**PPLANNER**
	Methods	$\forall$I-B, $\exists$I-B, $\wedge$I-B, ...
	C-Rules	
	Termination	*literal-line-tasks-only*

Table 3: The NormalizeLineTask strategy.

use a support in their premises that carries a focus and when their conclusions do not tackle the focused subformula. For instance, if a line-task has the supports $B_1 \wedge B_2$ and $A_1 \wedge (A_2 \wedge focus(A_3 \wedge A_4))$, then only actions of $\wedge$E-F that use the second support with the focus are allowed. The introduction of two actions of $\wedge$E-F derive the new support $focus(A_3 \wedge A_4)$ to which no further action of $\wedge$E-F can be applied since it would decompose the focused subformula. Similar to NormalizeLineTask and SolveInequality, UnwrapHyp terminates as soon as all focused formulas are unwrapped.

Strategy: UnwrapHyp		
Condition	*focus-in-subformula*	
Action	Algorithm	**PPLANNER**
	Methods	$\forall$E-F, $\exists$E-F, $\wedge$E-F, ...
	C-Rules	`tackle-focus`
	Termination	*focus-at-top*

Table 4: The UnwrapHyp strategy.

INSTMETA Strategies

In order to instantiate meta-variables that occur in constraints collected by $\mathcal{CoSIE}$, we implemented the two **INSTMETA** strategies InstIfDetermined and ComputeInstFromCS (see Table 5). InstIfDetermined 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 unique 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.

BACKTRACK Strategies

Simple backtracking of one action as in PLAN is realized by the **BACKTRACK** strategy BackTrackActionToTask (see Table 6). BackTrackActionToTask instantiates the **BACKTRACK** algorithm with the function *step-to-line-task*, which computes the action that introduced a line-task. The application of BackTrackActionToTask then deletes this actions as well as all actions that may depend on this action. BackTrackActionToTask is applicable to each line-task. We will introduce further **BACKTRACK** strategies as we go along.

Strategy: InstIfDetermined		
Condition	*determined-in-cs*	
Action	Algorithm	**InstMeta**
	Function	*get-determined-instantiation*

Strategy: ComputeInstFromCS		
Condition	*mv-in-cs*	
Action	Algorithm	**InstMeta**
	Function	*compute-consistent-instantiation*

Table 5: The **InstMeta** strategies InstIfDetermined and ComputeInstFromCS.

Strategy: BackTrackActionToTask		
Condition	*line-task*	
Action	Algorithm	**BackTrack**
	Function	*step-to-line-task*

Table 6: The BackTrackActionToTask strategy.

4.2 Cooperation of the Strategies

As stated above, SolveInequality is the central strategy to accomplish ϵ-δ-proofs. NormalizeLineTask or UnwrapHyp are employed when complex formulas have to be decomposed. Technically, the cooperation between SolveInequality and NormalizeLineTask and UnwrapHyp works as follows. For line-tasks whose goals are complex formulas that contain inequality subformulas (e.g., goals that arise from unfolding *lim*, *limseq*, *cont*, or *deriv*) SolveInequality interrupts and places a demand for the strategy NormalizeLineTask on the control blackboard. Guided by this demand, Multi invokes NormalizeLineTask, which decomposes the complex goal. When re-invoked by Multi, SolveInequality can tackle the inequalities in the resulting goals. The switch from SolveInequality to UnwrapHyp is driven by missing support inequalities, which are needed for the application of the methods ComplexEstimate-B and Solve*-B. If the other methods preferred by `prove-inequality` fail, then the application of SetFocus-B highlights a subformula in an existing support. Afterwards, SolveInequality interrupts and places a demand for the invocation of UnwrapHyp to unwrap the highlighted subformula. When the subformula is unwrapped, SolveInequality can continue with a new support that may enable further steps. The application of SetFocus-B (i.e., the selection of the support and the subformula to highlight) is guided by the control rule `choose-unwrap-support` for the supports and parameters choice point. `choose-unwrap-support` analyzes the supports of the task on which the other methods are not applicable. It searches for inequality subformulas in the supports that are similar to the goal of the task. The idea is that similar formulas are likely to unify with the goal such that ComplexEstimate-B and Solve*-B become applicable.

The invocation of ComputeInstFromCS is delayed by the strategic control rule `delay-ComputeInstCosie` until all line-tasks 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, that is, not before all line-tasks are closed (see discussion in section 3.4.1). However, when 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 as we shall see in section 4.4 (see also the discussion of PLAN's drawbacks in section 3.4.1).

To enable the flexible instantiation of determined meta-variables SolveInequality cooperates with the strategy InstIfDetermined. Technically, this works as follows. When $\mathcal{CoSIE}$ signals that a meta-variable is determined, then the control rule `eager-instantiate` in SolveInequality fires. It interrupts SolveInequality and places a demand for InstIfDetermined with respect to the determined meta-variable. After the introduction of a binding for the meta-variable by InstIfDetermined MULTI re-invokes SolveInequality.

The cooperation with the **BACKTRACK** strategy BackTrackActionToTask is guided by the general strategic control rule `prefer-backtrack-if-failure` (see section 2.3). Further **BACKTRACK** strategies and their guidance by failure reasoning are explained in section 5.

4.3 The LIM+ Example

In this section, we shall discuss the application of MULTI to the LIM+ problem with the strategies described in the previous section. The LIM+ problem states that the limit of the sum of two functions f and g equals the sum of their limits. That is, the problem states that

$$\begin{array}{rl} & LIM{+}\colon \lim\limits_{x\to a}(f(x)+g(x)) = l_f + l_g \\ \text{follows from} & Lim_f\colon \lim\limits_{x\to a} f(x) = l_f \\ \text{and} & Lim_g\colon \lim\limits_{x\to a} g(x) = l_g. \end{array}$$

Figure 8 and Figure 9 show the interesting parts, i.e., the parts created by SolveInequality, of the resulting $\mathcal{PDS}$. We indicate the contributions of NormalizeLineTask and UnwrapHyp by justifications in the $\mathcal{PDS}$ such as (UnwrapHyp L_3) in line L_{49}, which abbreviate the proof segments created by these strategies. The complete $\mathcal{PDS}$ is given in appendix A. Note that we describe the proof planning process in progress. Hence, we introduce meta-variables, when they arise. When there is a binding for a meta-variable during the proof planning process, then the proof lines created after the introduction of the binding use the instantiation of the meta-variable in order to clarify the following computations.

The proof planning process starts with the invocation of SolveInequality on the initial task $LIM{+} \blacktriangleleft \{Lim_f, Lim_g\}$. SolveInequality first unfolds the occurrences of lim. Afterwards, it switches to NormalizeLineTask, which decomposes the resulting complex goal in line L_1 into the goals $|(f(c_x)+g(c_x)) - (l_f + l_g)| < c_\epsilon$ in L_{12} and $0 < mv_\delta$ in L_8 where c_ϵ and c_x are constants introduced for the universally quantified variables ϵ and x in L_1 and mv_δ is a meta-variable introduced for the existentially quantified variable δ.

Both new goals are inequalities and SolveInequality tackles them guided by the control rule `prove-inequality`. It closes $0 < mv_\delta$ directly by an application of TELLCS-B, which passes the formula to $\mathcal{CoSIE}$. $|(f(c_x)+g(c_x)) - (l_f + l_g)| < c_\epsilon$ is not accepted by $\mathcal{CoSIE}$ and therefore TELLCS-B is not applicable. SolveInequality simplifies this goal to $|((f(c_x)+g(c_x)) - l_f) - l_g| < c_\epsilon$ in line L_{16} but then fails to solve this goal with the given supports. `choose-unwrap-support` detects the subformula $|f(x_1) - l_f| < \epsilon_1$ of L_2

Lim_f.	Lim_f	$\vdash \lim\limits_{x\to a} f(x) = l_f$	(Hyp)
Lim_g.	Lim_g	$\vdash \lim\limits_{x\to a} g(x) = l_g$	(Hyp)
L_2.	Lim_f	$\vdash \forall \epsilon_1. (0 < \epsilon_1 \Rightarrow \exists \delta_1. (0 < \delta_1 \wedge \forall x_1. (\|x_1 - a\| < \delta_1 \wedge \|x_1 - a\| > 0 \Rightarrow \|f(x_1) - l_f\| < \epsilon_1)))$	(DefnUnfold-F Lim_f)
L_3.	Lim_g	$\vdash \forall \epsilon_2. (0 < \epsilon_2 \Rightarrow \exists \delta_2. (0 < \delta_2 \wedge \forall x_2. (\|x_2 - a\| < \delta_2 \wedge \|x_2 - a\| > 0 \Rightarrow \|g(x_2) - l_g\| < \epsilon_2)))$	(DefnUnfold-F Lim_g)
L_{21}.	L_{21}	$\vdash 0 < c_{\delta_1} \wedge \forall x_1. (\|x_1 - a\| < c_{\delta_1} \wedge \|x_1 - a\| > 0 \Rightarrow \|f(x_1) - l_f\| < mv_{\epsilon_1})$	(Hyp)
L_{42}.	L_{42}	$\vdash 0 < c_{\delta_2} \wedge \forall x_2. (\|x_2 - a\| < c_{\delta_2} \wedge \|x_2 - a\| > 0 \Rightarrow \|g(x_2) - l_g\| < mv_{\epsilon_2})$	(Hyp)
L_{11}.	L_{11}	$\vdash \|c_x - a\| > 0 \wedge \|c_x - a\| < mv_\delta$	(Hyp)
L_5.	L_5	$\vdash 0 < c_\epsilon$	(Hyp)
L_{52}.	$\mathcal{H}_2$	$\vdash mv_{x_2} \doteq c_x$	(TellCS-B)
L_{53}.	$\mathcal{H}_2$	$\vdash mv_{\epsilon_2} \leq \frac{1}{2} * c_\epsilon$	(TellCS-B)
L_{49}.	$\mathcal{H}_2$	$\vdash \|g(mv_{x_2}) - l_g\| < mv_{\epsilon_2}$	(UnwrapHyp L_3)
L_{48}.	$\mathcal{H}_2$	$\vdash \|g(c_x) - l_g\| < \frac{1}{2} * c_\epsilon$	(Solve*-B L_{49} L_{52} L_{53})
L_{37}.	$\mathcal{H}_1$	$\vdash \|g(c_x) - l_g\| < \frac{1}{2} * c_\epsilon$	(UnwrapHyp L_3 L_{48} L_{39} L_{50} L_{51})
L_{31}.	$\mathcal{H}_1$	$\vdash \|1\| \leq mv$	(TellCS-B)
L_{32}.	$\mathcal{H}_1$	$\vdash mv_{\epsilon_1} \leq \frac{c_\epsilon}{2 * mv}$	(TellCS-B)
L_{33}.	$\mathcal{H}_1$	$\vdash \|g(c_x) - l_g\| < \frac{c_\epsilon}{2}$	(Simplify-B L_{37})
L_{34}.	$\mathcal{H}_1$	$\vdash 0 < mv$	(TellCS-B)
L_{35}.	$\mathcal{H}_1$	$\vdash mv_{x_1} \doteq c_x$	(TellCS-B)
L_{28}.	$\mathcal{H}_1$	$\vdash \|f(mv_{x_1}) - l_f\| < mv_{\epsilon_1}$	(UnwrapHyp L_2)
L_{27}.	$\mathcal{H}_1$	$\vdash \|((f(c_x) + g(c_x)) - l_f) - l_g\| < c_\epsilon$	(ComplexEstimate-B L_{28} L_{31} L_{32} L_{33} L_{34} L_{35})
L_{16}.	$\mathcal{H}_3$	$\vdash \|((f(c_x) + g(c_x)) - l_f) - l_g\| < c_\epsilon$	(UnwrapHyp L_2 L_{27} L_{18} L_{29} L_{30})
L_{12}.	$\mathcal{H}_3$	$\vdash \|(f(c_x) + g(c_x)) - (l_f + l_g)\| < c_\epsilon$	(Simplify-B L_{16})
L_8.	$\mathcal{H}_4$	$\vdash 0 < mv_\delta$	(TellCS-B)
L_1.	Lim_f, Lim_g	$\vdash \forall \epsilon. (0 < \epsilon \Rightarrow \exists \delta. (0 < \delta \wedge \forall x. (\|x - a\| < \delta \wedge \|x - a\| > 0 \Rightarrow \|(f(x) + g(x)) - (l_f + l_g)\| < \epsilon)))$	(NormalizeLineTask L_8 L_{12})
LIM+.	Lim_f, Lim_g	$\vdash \lim\limits_{x\to a} (f(x) + g(x)) = l_f + l_g$	(DefnUnfold-B L_1)

$\mathcal{H}_1 = \{Lim_f, Lim_g, L_5, L_{11}, L_{21}\}$, $\mathcal{H}_2 = \{Lim_f, Lim_g, L_5, L_{11}, L_{21}, L_{42}\}$
$\mathcal{H}_3 = \{Lim_f, Lim_g, L_5, L_{11}\}$, $\mathcal{H}_4 = \{Lim_f, Lim_g, L_5\}$

Figure 8: ϵ-δ-proof for LIM+ (part I).

as a promising support and guides the application of the method SetFocus-B to highlight the subformula.[10] This triggers the interruption of SolveInequality and the invocation of UnwrapHyp for this subformula. The application of UnwrapHyp yields the new support $|f(mv_{x_1}) - l_f| < mv_{\epsilon_1}$ in line L_{28}, but also the three new goals $0 < mv_{\epsilon_1}$ in line L_{18}, $|mv_{x_1} - a| < c_{\delta_1}$ in L_{29}, and $|mv_{x_1} - a| > 0$ in L_{30}. Here UnwrapHyp introduces the constant c_{δ_1} for the existentially quantified variable δ_1 and the meta-variables mv_{ϵ_1} and mv_{x_1} for the

[10] Note that SetFocus-B and TellCS-F are special methods: they do not produce new proof lines but only highlight subformulas and handle the communication with $\mathcal{CoSIE}$, respectively. Since they do not produce new proof lines they are not visible in the $\mathcal{PDS}$.

universally quantified variables ϵ_1 and x_1 in L_2.

When SolveInequality is re-invoked, it can apply ComplexEstimate-B to the goal $|((f(c_x)+g(c_x))-l_f)-l_g|<c_\epsilon$ and the new support $|f(mv_{x_1})-l_f|<mv_{\epsilon_1}$. This results in the five new goals $|1|\leq mv$ in L_{31}, $mv_{\epsilon_1}\leq\frac{c_\epsilon}{2*mv}$ in L_{32}, $|g(c_x)-l_g|<\frac{c_\epsilon}{2}$ in L_{33}, $0<mv$ in L_{34}, and $mv_{x_1}\doteq c_x$ in L_{35}. Except L_{33} all goals are closed by applications of TellCS-B, which pass the respective formulas as constraints to $\mathcal{CoSIE}$. Since $mv_{x_1}\doteq c_x$ determines mv_{x_1} in $\mathcal{CoSIE}$ the control rule `eager-instantiate` fires and interrupts SolveInequality. Its demand causes Multi to invoke InstIfDetermined on the instantiation-task of mv_{x_1}. InstIfDetermined introduces the binding $mv_{x_1}{:=}^b c_x$ into the strategic proof plan.

The re-invoked SolveInequality simplifies $|g(c_x)-l_g|<\frac{c_\epsilon}{2}$ to $|g(c_x)-l_g|<\frac{1}{2}*c_\epsilon$ in L_{37} but then fails on this goal with the existing supports. `choose-unwrap-support` detects the subformula $|g(x_2)-l_g|<\epsilon_2$ of L_3 as a promising support and guides the corresponding application of the method SetFocus-B to highlight this subformula. Afterwards, SolveInequality interrupts and Multi switches to UnwrapHyp, which unwraps the subformula and yields the new support $|g(mv_{x_2})-l_g|<mv_{\epsilon_2}$ in line L_{49}. The unwrapping yields also the three new goals $0<mv_{\epsilon_2}$ in line L_{39}, $|mv_{x_2}-a|<c_{\delta_2}$ in L_{50}, and $|mv_{x_2}-a|>0$ in L_{51}. UnwrapHyp introduces the constant c_{δ_2} for the existentially quantified variable δ_2 and the meta-variables mv_{ϵ_2} and mv_{x_2} for the universally quantified variables ϵ_2 and x_2 in L_3.

When re-invoked, SolveInequality applies Solve*-B to the goal $|g(c_x)-l_g|<\frac{1}{2}*c_\epsilon$ and the new support $|g(mv_{x_2})-l_g|<mv_{\epsilon_2}$. This results in the new goals $mv_{x_2}\doteq c_x$ in L_{52} and $mv_{\epsilon_2}\leq\frac{1}{2}*c_\epsilon$ in L_{53}, which SolveInequality closes by TellCS-B. $mv_{x_2}\doteq c_x$ determines the meta-variable mv_{x_2} in $\mathcal{CoSIE}$. Thus, the control rule `eager-instantiate` suggests a switch from SolveInequality to InstIfDetermined, which introduces the binding $mv_{x_2}{:=}^b c_x$ into the strategic proof plan.

L_{18}.	$\mathcal{H}_3$	$\vdash 0<mv_{\epsilon_1}$	(TellCS-B)
L_{39}.	$\mathcal{H}_3$	$\vdash 0<mv_{\epsilon_2}$	(TellCS-B)
L_{11}.	L_{11}	$\vdash |c_x-a|>0\wedge|c_x-a|<mv_\delta$	(*Hyp*)
L_{14}.	L_{11}	$\vdash |c_x-a|>0$	($\wedge$E-F L_{11})
L_{13}.	L_{11}	$\vdash |c_x-a|<mv_\delta$	($\wedge$E-F L_{11})
L_{61}.	$\mathcal{H}_1$	$\vdash 0\leq 0$	(AskCS-B)
L_{59}.	$\mathcal{H}_1$	$\vdash mv_\delta\leq c_{\delta_1}$	(TellCS-B)
L_{57}.	$\mathcal{H}_2$	$\vdash 0\leq 0$	(AskCS-B)
L_{55}.	$\mathcal{H}_2$	$\vdash mv_\delta\leq c_{\delta_2}$	(TellCS-B)
L_{29}.	$\mathcal{H}_1$	$\vdash |mv_{x_1}-a|<c_{\delta_1}$	(Solve*-B L_{13} L_{59})
L_{30}.	$\mathcal{H}_1$	$\vdash |mv_{x_1}-a|>0$	(Solve*-B L_{14} L_{61})
L_{50}.	$\mathcal{H}_2$	$\vdash |mv_{x_2}-a|<c_{\delta_2}$	(Solve*-B L_{13} L_{55})
L_{51}.	$\mathcal{H}_2$	$\vdash |mv_{x_2}-a|>0$	(Solve*-B L_{14} L_{57})

$\mathcal{H}_1=\{Lim_f,Lim_g,L_5,L_{11},L_{21}\}$, $\mathcal{H}_2=\{Lim_f,Lim_g,L_5,L_{11},L_{21},L_{42}\}$
$\mathcal{H}_3=\{Lim_f,Lim_g,L_5,L_{11}\}$, $\mathcal{H}_4=\{Lim_f,Lim_g,L_5\}$

Figure 9: ϵ-δ-proof for LIM+ (part II).

Afterwards, SolveInequality has to deal with the remaining goals L_{18}, L_{29}, L_{30}, and L_{39}, L_{50}, L_{51}, which resulted from the applications of the UnwrapHyp strategy. Figure 9 gives the $\mathcal{PDS}$ segment created by SolveInequality for these goals. It closes L_{18} and L_{39} directly by TellCS-B. The inequalities in the other goals cannot be passed to $\mathcal{CoSIE}$ directly because TellCS-B is not applicable to them. Instead, SolveInequality applies

SOLVE*-B to these goals with supports that stem from the decomposition of the initial goal by NormalizeLineTask. The applications of SOLVE*-B result in inequality goals, which SolveInequality closes either with TELLCS-B or ASKCS-B.

After closing all line-tasks, SolveInequality terminates. Next, MULTI invokes ComputeInstFromCS on the instantiation-tasks and $\mathcal{CoSIE}$ provides instantiations for the meta-variables that are consistent with the collected constraints (see Figure 7 in section 3.1). ComputeInstFromCS inserts these instantiations as the bindings

$$mv{:=}^b 1,\ mv_{\epsilon_1}{:=}^b \tfrac{c_\epsilon}{2},\ mv_{\epsilon_2}{:=}^b \tfrac{c_\epsilon}{2},\ \text{and}\ mv_\delta{:=}^b min(c_{\delta_1}, c_{\delta_2})$$

into the strategic proof plan.

4.4 Eager Instantiation

We discussed already in section 3.4.1 that PLAN fails to solve some limit problems that require the eager instantiation of meta-variables. In the following, we shall see how MULTI solves those problems since it performs eager instantiation guided by the control rule `eager-instantiate`.

We illustrate MULTI's eager meta-variable instantiation with the first part of exercise 4.1.3 in the analysis textbook [2], which states that

$$Thm\colon \lim_{x \to 0} f(x+c) = l \text{ follows from } Ass\colon \lim_{x_1 \to c} f(x_1) = l,$$

Figure 10 and Figure 11 show the $\mathcal{PDS}$ segments created by SolveInequality for this problem. As in the previous section, we indicate and abbreviate the proof parts generated by NormalizeLineTask and UnwrapHyp by justifications in the $\mathcal{PDS}$.

When invoked on the initial task $Thm \blacktriangleleft \{Ass\}$, SolveInequality unfolds the occurrences of lim in the goal and the supports and then switches to NormalizeLineTask, which decomposes the resulting complex goal. This results in the two goals $0 < mv_\delta$ in L_7 and $|f(c_x+c)-l| < c_\epsilon$ in L_{11} where c_ϵ and c_x are constants introduced for the universally quantified variables ϵ and x in L_1 and mv_δ is a meta-variable introduced for the existentially quantified variable δ.

SolveInequality closes $0 < mv_\delta$ by TELLCS-B but fails to tackle $|f(c_x + c) - l| < c_\epsilon$ with the current supports. A promising support is the subformula $|f(x_1) - l| < \epsilon_1$ of L_2. Thus, after highlighting the subformula with SETFOCUS-B, SolveInequality switches to UnwrapHyp. The application of UnwrapHyp yields the new support $|f(mv_{x_1}) - l| < mv_{\epsilon_1}$ in L_{26} and the new goals $0 < mv_{\epsilon_1}$ in L_{16}, $|mv_{x_1} - c| < c_{\delta_1}$ in L_{27}, and $|mv_{x_1} - c| > 0$ in L_{28}. UnwrapHyp introduces the constant c_{δ_1} for the existentially quantified variable δ_1 and the meta-variables mv_{ϵ_1} and mv_{x_1} for the universally quantified variables ϵ_1 and x_1 in L_2.

When re-invoked, SolveInequality applies SOLVE*-B to $|f(c_x + c) - l| < c_\epsilon$ and the new support $|f(mv_{x_1})-l| < mv_{\epsilon_1}$. This results in the new goals $mv_{x_1} \doteq c_x + c$ in L_{29} and $mv_{\epsilon_1} \leq c_\epsilon$ in L_{30}, which SolveInequality both closes by TELLCS-B. Since $mv_{x_1} \doteq c_x + c$ determines the meta-variable mv_{x_1} in $\mathcal{CoSIE}$, SolveInequality switches to InstIfDetermined, which introduces the binding $mv_{x_1}{:=}^b c_x + c$ into the strategic proof plan.

$Ass.$	Ass	$\vdash \lim_{x_1 \to c} f(x_1) = l$	(Hyp)
$L_2.$	Ass	$\vdash \forall \epsilon_1 \text{.} (0 < \epsilon_1 \Rightarrow \exists \delta_1 \text{.} (0 < \delta_1 \wedge \forall x_1 \text{.} (\lvert x_1 - c\rvert < \delta_1 \wedge \lvert x_1 - c\rvert > 0 \Rightarrow \lvert f(x_1) - l\rvert < \epsilon_1)))$	(DEFNUNFOLD-F Ass)
$L_{19}.$	L_{19}	$\vdash 0 < c_{\delta_1} \wedge \forall x_1 \text{.} (\lvert x_1 - c\rvert < c_{\delta_1} \wedge \lvert x_1 - c\rvert > 0 \Rightarrow \lvert f(x_1) - l\rvert < mv_{\epsilon_1})$	(Hyp)
$L_4.$	L_4	$\vdash 0 < c_\epsilon$	(Hyp)
$L_{29}.$	$\mathcal{H}_1$	$\vdash mv_{x_1} \doteq c_x + c$	(TELLCS-B)
$L_{30}.$	$\mathcal{H}_1$	$\vdash mv_{\epsilon_1} \leq c_\epsilon$	(TELLCS-B)
$L_{26}.$	$\mathcal{H}_1$	$\vdash \lvert f(mv_{x_1}) - l\rvert < mv_{\epsilon_1}$	(UnwrapHyp L_2)
$L_{25}.$	$\mathcal{H}_1$	$\vdash \lvert f(c_x + c) - l\rvert < c_\epsilon$	(SOLVE*-B L_{26} L_{29} L_{30})
$L_{11}.$	$\mathcal{H}_2$	$\vdash \lvert f(c_x + c) - l\rvert < c_\epsilon$	(UnwrapHyp L_2 L_{25} L_{16} L_{27} L_{28})
$L_7.$	Ass, L_4	$\vdash 0 < mv_\delta$	(TELLCS-B)
$L_1.$	Ass	$\vdash \forall \epsilon \text{.} (0 < \epsilon \Rightarrow \exists \delta \text{.} (0 < \delta \wedge \forall x \text{.} (\lvert x - 0\rvert < \delta \wedge \lvert x - 0\rvert > 0 \Rightarrow \lvert f(x + c) - l\rvert < \epsilon)))$	(NormalizeLineTask L_7 L_{11})
$Thm.$	Ass	$\vdash \lim_{x \to 0} f(x + c) = l$	(DEFNUNFOLD-B L_1)
$\mathcal{H}_1 = \{Ass, L_4, L_{10}, L_{19}\},\ \mathcal{H}_2 = \{Ass, L_4, L_{10}\}$			

Figure 10: ϵ-δ-proof for first part of exercise 4.1.3 (part I).

$L_{10}.$	L_{10}	$\vdash \lvert c_x - 0\rvert > 0 \wedge \lvert c_x - 0\rvert < mv_\delta$	(Hyp)
$L_{13}.$	L_{10}	$\vdash \lvert c_x - 0\rvert > 0$	($\wedge$E-F L_{10})
$L_{12}.$	L_{10}	$\vdash \lvert c_x - 0\rvert < mv_\delta$	($\wedge$E-F L_{10})
$L_{36}.$	L_{10}	$\vdash \lvert c_x\rvert > 0$	(SIMPLIFY-F L_{13})
$L_{32}.$	L_{10}	$\vdash \lvert c_x\rvert < mv_\delta$	(SIMPLIFY-F L_{12})
$L_{34}.$	$\mathcal{H}_1$	$\vdash mv_\delta \leq c_{\delta_1}$	(TELLCS-B)
$L_{31}.$	$\mathcal{H}_1$	$\vdash \lvert c_x\rvert < c_{\delta_1}$	(SOLVE*-B L_{32} L_{34})
$L_{35}.$	$\mathcal{H}_1$	$\vdash \lvert c_x\rvert > 0$	(WEAKEN-B L_{36})
$L_{27}.$	$\mathcal{H}_1$	$\vdash \lvert mv_{x_1} - c\rvert < c_{\delta_1}$	(SIMPLIFY-B L_{31})
$L_{28}.$	$\mathcal{H}_1$	$\vdash \lvert mv_{x_1} - c\rvert > 0$	(SIMPLIFY-B L_{35})
$L_{16}.$	$\mathcal{H}_2$	$\vdash 0 < mv_{\epsilon_1}$	(TELLCS-B)
$\mathcal{H}_1 = \{Ass, L_4, L_{10}, L_{19}\},\ \mathcal{H}_2 = \{Ass, L_4, L_{10}\}$			

Figure 11: ϵ-δ-proof for first part of exercise 4.1.3 (part II).

Afterwards, SolveInequality has to deal with the remaining goals L_{16}, L_{27}, and L_{28}, which resulted from the application of UnwrapHyp. Figure 11 gives the $\mathcal{PDS}$ segment created by SolveInequality for these goals. It closes L_{16} by TELLCS-B. The goals in L_{27} and L_{28} become $\lvert(c_x + c) - c\rvert < c_{\delta_1}$ and $\lvert(c_x + c) - c\rvert > 0$ with respect to the binding $mv_{x_1} :=^b c_x + c$ in the strategic proof plan. Applications of SIMPLIFY-B reduce these two goals to the $\lvert c_x\rvert < c_{\delta_1}$ in L_{31} and $\lvert c_x\rvert > 0$ in L_{35}. SolveInequality closes these new goals with the supports $\lvert c_x\rvert > 0$ and $\lvert c_x\rvert < mv_\delta$ that are derived from L_{10}, which was introduced during the application of NormalizeLineTask.

$\mathcal{C}o\mathcal{SIE}$ has the final constraint store depicted in Figure 12. It computes instantiations for the meta-variables that are consistent with these constraints. ComputeInstFromCS inserts these instantiations as the bindings $mv_\delta :=^b c_{\delta_1}$ and $mv_{\epsilon_1} :=^b c_\epsilon$ into the strategic proof plan.

Responsible for the success of SolveInequality on L_{27} and L_{28} is the eager introduction

$$\begin{array}{lllll} mv_{x_1} = c_x + c & & & & \\ 0 & < & c_{\delta_1} & < & +\infty \\ 0 & < & c_\epsilon & < & +\infty \\ 0 & < & mv_{\epsilon_1} & \leq & c_\epsilon \\ 0 & < & mv_\delta & \leq & c_{\delta_1} \end{array}$$

Figure 12: The final constraint store of $\mathcal{CoSIE}$ for the first part of exercise 4.1.3.

of the binding $mv_{x_1} :=^b c_x + c$. This binding changes the formulas of L_{27} and L_{28} and so SIMPLIFY-B becomes applicable.[11]

Another problem from the limit domain that requires eager meta-variable instantiation is exercise 4.1.12 in [2], which states that

$$\textit{Thm}\text{: } \lim_{x \to 0} f(a * x) = l \text{ follows from } \textit{Ass}\text{: } \lim_{x_1 \to 0} f(x_1) = l \text{ for } a > 0.$$

First, MULTI reduces the initial goal $\lim_{x \to 0} f(a * x) = l$ to $|f(a * c_x) - l| < c_\epsilon$. Then, it unwraps the support $|f(mv_{x_1}) - l| < mv_{\epsilon_1}$. The application of SOLVE*-B to this goal and this support results in the goal $mv_{x_1} \doteq a * c_x$, which is passed to $\mathcal{CoSIE}$. Since this formula determines mv_{x_1} the binding $mv_{x_1} :=^b a * c_x$ is introduced into the strategic proof plan. The remaining goals $|mv_{x_1} - 0| < c_{\delta_1}$ and $|mv_{x_1} - 0| > 0$ that result from the unwrapping of the support become $|a * c_x| < c_{\delta_1}$ and $|a * c_x| > 0$ with respect to this binding. They are then solved by applications of COMPLEXESTIMATE-B with the supports $|c_x| > 0$ and $|c_x| < mv_\delta$.[12] See also section 5.2 for further examples that require eager meta-variable instantiation.

5 Failure Reasoning in the Limit Domain

In this section, we shall discuss three types of situations we encountered when tackling limit problems whose solution requires meta-reasoning on failures. In two situations the failures can be exploited to guide the introduction of case-splits and the speculation of lemmas, two eureka steps whose necessity is difficult to spot and whose introduction is difficult to guide in general. In the third situation we guide backtracking by meta-reasoning on desirable but blocked strategies. All three types of situations have in common that failures in the proof planning process can be productively used and hold the key to discover a solution proof plan.

[11] PLAN, which does not allow for eager meta-variable instantiation, would fail on the goals L_{27} and L_{28} since it cannot close $|mv_{x_1} - c| < c_{\delta_1}$ and $|mv_{x_1} - c| > 0$ from $|c_x| < mv_\delta$ and $|c_x| > 0$ derivable from L_{10}.

[12] PLAN would fail on these goals since without eager meta-variable instantiation it cannot apply COMPLEXESTIMATE-B to solve $|mv_{x_1}| < c_{\delta_1}$ and $|mv_{x_1}| > 0$ with $|c_x| > 0$ and $|c_x| < mv_\delta$, respectively. Rather, it would apply SOLVE*-B to these goals and supports. This results in the subgoal $mv_{x_1} \doteq c_x$, which $\mathcal{CoSIE}$ rejects since it is not consistent with the already collected constraint $mv_{x_1} \doteq a * c_x$. Thus, TELLCS-B is not applicable and PLAN fails.

5.1 Guiding Case-Splits

A well-known technique from mathematics to deal with complex problems is to split the problem into cases and to solve the cases separately.[13] But how should the eureka step case-split be controlled? That is, when should MULTI decide for a case-split and which cases should it consider? We found a type of situations in which the need for a case-split and its construction can be spotted by failure reasoning.

As example consider the Cont-If-Deriv problem. This problem states that a function f is continuous at point a if it has a derivative f' at point a. That is,

$$Thm\colon\ cont(f,a) \text{ follows from } Ass\colon\ deriv(f,a)=f'.$$

We give the $\mathcal{PDS}$ segment created by SolveInequality before the failure occurs in Figure 13. As in the previous sections we abbreviate the proof parts generated by NormalizeLineTask and UnwrapHyp by strategic justifications in the $\mathcal{PDS}$.

As usual, SolveInequality unfolds the defined concepts and then switches to NormalizeLineTask for the decomposition of the complex goal. The resulting main goal is $|f(c_x)-f(a)|<c_\epsilon$.
SolveInequality fails to tackle this goal with the current supports. Since the control rule `choose-unwrap-support` detects the subformula $|\frac{f(x_1)-f(a)}{x_1-a}-f'|<\epsilon_1$ in L_3 as a promising support SolveInequality switches to UnwrapHyp whose application yields the new support $|\frac{f(mv_{x_1})-f(a)}{mv_{x_1}-x}-f'|<mv_{\epsilon_1}$ in line L_{25} and the three new goals $0<mv_{\epsilon_1}$ in L_{18}, $|mv_{x_1}-a|<c_{\delta_1}$ in L_{26}, and $|mv_{x_1}-a|>0$ in L_{27}. With the new support SolveInequality closes the main goal $|f(c_x)-f(a)|<c_\epsilon$ in several steps as described in Figure 13 (in between SolveInequality interrupts once and switches to InstIfDetermined to introduce the binding $mv_{x_1}{:=}^b c_x$). Then, it tackles the new goals from the application of UnwrapHyp (see the region between the dashed lines in Figure 13). It succeeds to solve L_{18} and L_{26} but fails to solve L_{27} whose formula becomes $|c_x-a|>0$ with respect to the binding $mv_{x_1}{:=}^b c_x$ meanwhile introduced.

MULTI succeeded to solve the goal $|f(c_x)-f(a)|<c_\epsilon$ with the derived support $|\frac{f(mv_{x_1})-f(a)}{mv_{x_1}-x}-f'|<mv_{\epsilon_1}$. However, it failed to prove $|c_x-a|>0$, one of the conditions of the support $|\frac{f(mv_{x_1})-f(a)}{mv_{x_1}-x}-f'|<mv_{\epsilon_1}$. The partial success, i.e., the solution of the initial goal, gives rise to consider to patch the proof attempt by introducing a case-split $|c_x-a|>0 \vee \neg(|c_x-a|>0)$ on the failing condition.

In general, the failure and its solution follow this pattern: there is a goal G, which MULTI can solve with a support G' that has some conditions $Conds$. When MULTI uses G', then it introduces the conditions $Conds$ as new goals. Afterwards, it fails to prove some of these new goals. We call such a goal a *failing condition*, whereas we call the initial goal G the *main goal*. The failure "failing condition while main goal is solved" can be productively used by introducing a case-split on the failing condition. Then, the main goal G has to be proved several times under different case-split hypotheses.

We shall elaborate this idea with our example. If SolveInequality fails to prove a condition of a support that was used to prove the main goal, then a strategic control rule triggers the

[13] SCHOENFELD mentions this technique as a frequently used heuristic: "Decompose the domain of the problem and work on it case by case." ([41] p. 109)

$Ass.$	Ass	$\vdash deriv(f,a) = f'$	(Hyp)
$L_2.$	Ass	$\vdash \lim\limits_{x_1 \to a} \frac{f(x_1)-f(a)}{x_1-a} = f'$	(DefnUnfold-F Ass)
$L_3.$	Ass	$\vdash \forall \epsilon_1 . (0 < \epsilon_1 \Rightarrow \exists \delta_1 . (0 < \delta_1 \wedge \forall x_1 . (\lvert x_1 - a\rvert < \delta_1 \wedge \lvert x_1 - a\rvert > 0 \Rightarrow \lvert \frac{f(x_1)-f(a)}{x_1-a} - f'\rvert < \epsilon_1)))$	(DefnUnfold-F L_2)
$L_{15}.$	L_{15}	$\vdash 0 < c_{\delta_1} \wedge \forall x_1 . (\lvert x_1 - a\rvert < c_{\delta_1} \wedge \lvert x_1 - a\rvert > 0 \Rightarrow \lvert \frac{f(x_1)-f(a)}{x_1-a} - f'\rvert < mv_{\epsilon_1})$	(Hyp)
$L_{11}.$	L_{11}	$\vdash \lvert c_x - a\rvert < mv_\delta$	(Hyp)
$L_7.$	L_7	$\vdash 0 < c_\epsilon$	(Hyp)
$L_{27}.$	$\mathcal{H}_1$	$\vdash \lvert mv_{x_1} - a\rvert > 0$	$(Open)$
$L_{44}.$	$\mathcal{H}_1$	$\vdash mv_\delta \leq c_{\delta_1}$	(TellCS-B)
$L_{26}.$	$\mathcal{H}_1$	$\vdash \lvert mv_{x_1} - a\rvert < c_{\delta_1}$	(Solve*-B L_{11} L_{44})
$L_{18}.$	$\mathcal{H}_2$	$\vdash 0 < mv_{\epsilon_1}$	(TellCS-B)
$L_{42}.$	$\mathcal{H}_1$	$\vdash 0 < \frac{\frac{c_\epsilon}{2}}{2}$	(AskCS-B)
$L_{37}.$	$\mathcal{H}_1$	$\vdash \lvert f'\rvert \leq mv'$	(TellCS-B)
$L_{38}.$	$\mathcal{H}_1$	$\vdash mv_\delta \leq \frac{\frac{c_\epsilon}{2}}{2*mv'}$	(TellCS-B)
$L_{39}.$	$\mathcal{H}_1$	$\vdash \lvert 0\rvert < \frac{\frac{c_\epsilon}{2}}{2}$	(Simplify-B L_{42})
$L_{40}.$	$\mathcal{H}_1$	$\vdash 0 < mv'$	(TellCS-B)
$L_{36}.$	$\mathcal{H}_1$	$\vdash mv_\delta \leq mv$	(TellCS-B)
$L_{28}.$	$\mathcal{H}_1$	$\vdash \lvert x - a\rvert \leq mv$	(Solve*-B L_{11} L_{36})
$L_{29}.$	$\mathcal{H}_1$	$\vdash mv_{\epsilon_1} \leq \frac{c_\epsilon}{2*mv}$	(TellCS-B)
$L_{30}.$	$\mathcal{H}_1$	$\vdash \lvert f' * c_x - f' * a\rvert < \frac{c_\epsilon}{2}$	(ComplexEstimate-B L_{11} L_{37} L_{38} L_{39} L_{40})
$L_{31}.$	$\mathcal{H}_1$	$\vdash 0 < mv$	(TellCS-B)
$L_{32}.$	$\mathcal{H}_1$	$\vdash mv_{x_1} \doteq c_x$	(TellCS-B)
$L_{25}.$	$\mathcal{H}_1$	$\vdash \lvert \frac{f(mv_{x_1})-f(a)}{mv_{x_1}-x} - f'\rvert < mv_{\epsilon_1}$	(UnwrapHyp L_3)
$L_{24}.$	$\mathcal{H}_1$	$\vdash \lvert f(c_x) - f(a)\rvert < c_\epsilon$	(ComplexEstimate-B L_{25} L_{28} L_{29} L_{30} L_{31} L_{32})
$L_{12}.$	$\mathcal{H}_2$	$\vdash \lvert f(c_x) - f(a)\rvert < c_\epsilon$	(UnwrapHyp L_3 L_{24} L_{18} L_{26} L_{27})
$L_9.$	Ass, L_7	$\vdash 0 < mv_\delta$	(TellCS-B)
$L_1.$	Ass	$\vdash \forall \epsilon . (0 < \epsilon \Rightarrow \exists \delta . (0 < \delta \wedge \forall x . (\lvert x - a\rvert < \delta \Rightarrow \lvert f(x) - f(a)\rvert < \epsilon)))$	(NormalizeLineTask L_9 L_{12})
$Thm.$	Ass	$\vdash cont(f,a)$	(DefnUnfold-B L_1)

$\mathcal{H}_1 = \{Ass, L_7, L_{11}, L_{15}\}$, $\mathcal{H}_2 = \{Ass, L_7, L_{11}\}$

Figure 13: ϵ-δ-proof for CONT-IF-DERIV (part I).

backtracking of the unwrapping and the use of the support. In our example, this control rule guides the backtracking of the application of UnwrapHyp and all actions that depend on it such that the resulting proof plan consists only of the unfolding of the defined concepts and the application of NormalizeLineTask. In particular, L_{12} becomes open again. When Multi re-invokes SolveInequality, then a control rule in SolveInequality fires that checks whether the last step was backtracking triggered by a failing condition. This control rule then suggests the application of the method CaseSplit-B on the re-opened main goal L_{12} with respect to the failing condition $\lvert c_x - a\rvert > 0$ and its negation $\neg(\lvert c_x - a\rvert > 0)$. This results in the $\mathcal{PDS}$ in Figure 14.

$Ass.$	Ass	$\vdash deriv(f,a) = f'$	(Hyp)
$L_2.$	Ass	$\vdash \lim\limits_{x_1 \to a} \frac{f(x_1)-f(a)}{x_1-a} = f'$	(DefnUnfold-F Ass)
$L_3.$	Ass	$\vdash \forall \epsilon_1 \text{.} (0 < \epsilon_1 \Rightarrow \exists \delta_1 \text{.} (0 < \delta_1 \wedge \forall x_1 \text{.} (\lvert x_1 - a\rvert < \delta_1 \wedge \lvert x_1 - a\rvert > 0 \Rightarrow \lvert \frac{f(x_1)-f(a)}{x_1-a} - f'\rvert < \epsilon_1)))$	(DefnUnfold-F L_2)
$L_{11}.$	L_{11}	$\vdash \lvert c_x - a\rvert < mv_\delta$	(Hyp)
$L_7.$	L_7	$\vdash 0 < c_\epsilon$	(Hyp)
$L_{45}.$	L_{45}	$\vdash \lvert c_x - a\rvert > 0 \vee \neg(\lvert c_x - a\rvert > 0)$	(TertiumNonDatur)
$L_{48}.$	L_{48}	$\vdash \neg(\lvert c_x - a\rvert > 0)$	(Hyp)
$L_{49}.$	$\mathcal{H}_4$	$\vdash \lvert f(c_x) - f(a)\rvert < c_\epsilon$	$(Open)$
$L_{46}.$	L_{46}	$\vdash \lvert c_x - a\rvert > 0$	(Hyp)
$L_{47}.$	$\mathcal{H}_3$	$\vdash \lvert f(c_x) - f(a)\rvert < c_\epsilon$	$(Open)$
$L_{12}.$	$\mathcal{H}_2$	$\vdash \lvert f(c_x) - f(a)\rvert < c_\epsilon$	(CaseSplit-B L_{45} L_{47} L_{49})
$L_9.$	Ass, L_7	$\vdash 0 < mv_\delta$	(TellCS-B)
$L_1.$	Ass	$\vdash \forall \epsilon \text{.} (0 < \epsilon \Rightarrow \exists \delta \text{.} (0 < \delta \wedge \forall x \text{.} (\lvert x - a\rvert < \delta \Rightarrow \lvert f(x) - f(a)\rvert < \epsilon)))$	(NormalizeLineTask L_9 L_{12})
$Thm.$	Ass	$\vdash cont(f,a)$	(DefnUnfold-B L_1)

$\mathcal{H}_3 = \{Ass, L_7, L_{11}, L_{45}, L_{46}\}$, $\mathcal{H}_2 = \{Ass, L_7, L_{11}\}$
$\mathcal{H}_4 = \{Ass, L_7, L_{11}, L_{45}, L_{46}\}$

Figure 14: ϵ-δ-proof for CONT-IF-DERIV (part II).

Afterwards, SolveInequality has to prove $|f(c_x) - f(a)| < c_\epsilon$ twice: once in L_{47} with hypothesis $|c_x - a| > 0$ and once in L_{49} with hypothesis $\neg(|c_x - a| > 0)$. To tackle L_{47} SolveInequality does not again perform proof search from the scratch. The subplan that was backtracked in order to enable the introduction of case-split at the right place is exactly a proof plan for this case; except the last missing step that derives the failing condition directly from the hypothesis of this case. Hence, triggered by a control rule, SolveInequality switches for this subproblem to the **CPlanner** strategy TaskDirectedAnalogy, which transfers the backtracked proof segment to a proof plan for L_{47}. The second case in L_{49} has to be solved differently by SolveInequality.[14] First, it simplifies the hypothesis $\neg(|c_x - a| > 0)$ to $c_x \doteq a$. Afterwards, it applies this equation with =Subst-B to $|f(c_x) - f(a)| < c_\epsilon$ in L_{49}. The resulting goal $|f(a) - f(a)| < c_\epsilon$ can be simplified with Simplify-B to $0 < c_\epsilon$, which follows from L_7.

Cont-If-Lim=f and Lim-If-Both-Sides-Lim are other problems that require this kind of failure reasoning. Cont-If-Lim=f states that a function f is continuous at point a if the limit at point a is $f(a)$. The unfolding of the definitions and the application of NormalizeLineTask result in the main goal $|f(c_x) - f(a)| < c_\epsilon$ that can be solved by unwrapping $|f(mv_{x_1}) - f(a)| < mv_{\epsilon_1}$ from the assumption. However, the subgoal $|c_x - a| > 0$ that is created by UnwrapHyp cannot be solved. This failing condition triggers the same case-split and the same solution of the resulting two cases as in the Cont-If-Deriv problem. The Lim-If-Both-Sides-Lim problem states that a function f has a limit l at point a, if both the right-hand and the left-hand limit of f at a are l.[15] Unfolding of the definitions and the application of NormalizeLineTask result in the main goal $|f(c_x) - l| < c_\epsilon$. A support to solve the main goal

[14] To transfer again the backtracked proof segment would not result in a solution of the subproblem since the hypothesis of this case is not suitable to solve the remaining failing condition.

[15] *Right-hand and left-hand limit* are defined as follows:

can be unwrapped either from the right-hand limit assumption or from the left-hand limit assumption. However, in both cases the application of UnwrapHyp yields an condition that cannot be closed. For instance, when UnwrapHyp unwraps the right-hand limit assumption, then there is the failing condition $c_x - a > 0$. This failing condition triggers the case-split into the cases $c_x - a > 0$ and $\neg(c_x - a > 0)$ for the main goal $|f(c_x) - l| < c_\epsilon$. Whereas the first case can be solved by unwrapping the right-hand limit assumption, the second case requires to unwrap the left-hand limit.

5.2 Lemma Speculation

It is common mathematical practice to speculate lemmas during a proof attempt and to prove the lemmas separately. Since technically arbitrary formulas can be introduced, lemma speculation introduces an infinite branching point into the search space that is difficult to control in automated theorem proving. We found a type of situations in which suitable (and necessary) lemmas can be speculated by failure reasoning.

As example consider the second part of exercise 4.1.3 from the analysis textbook [2]. This problem states that

$$Thm\colon \lim_{x_1 \to c} f(x_1) = l \text{ follows from } Ass\colon \lim_{x \to 0} f(x+c) = l.$$

Figure 15 depicts the $\mathcal{PDS}$ segment created by SolveInequality until the failure occurs. As in the previous section, we indicate and abbreviate the proof parts generated by NormalizeLineTask and UnwrapHyp by strategic justifications.

SolveInequality unfolds the defined concepts and then switches to NormalizeLineTask, which decomposes the complex goal. This results in the goal $|f(c_{x_1}) - l| < c_{\epsilon_1}$ in L_{11}, which SolveInequality cannot tackle with the given supports. Hence, it switches to UnwrapHyp in order to decompose the subformula $|f(x+c) - l| < \epsilon$ in L_2. The application of UnwrapHyp yields the new support $|f(mv_x + c) - l| < mv_\epsilon$ in line L_{26} and the three additional goals $0 < mv_\epsilon$ in L_{16}, $|mv_x - 0| < c_\delta$ in L_{27}, and $|mv_x - 0| > 0$ in L_{28}.

Next, SolveInequality should apply SOLVE*-B to tackle $|f(c_{x_1}) - l| < c_{\epsilon_1}$ with the new support $|f(mv_x + c) - l| < mv_\epsilon$. However, this fails since SOLVE*-B fails to unify $|f(mv_x + c) - l|$ and $|f(c_{x_1}) - l|$, which is one of the application conditions of SOLVE*-B. Since no other method is applicable and there is also no further promising subformula to unwrap, MULTI would backtrack next. The analysis that $|f(mv_x + c) - l|$ and $|f(c_{x_1}) - l|$ are quite similar and that the unification is blocked only because of the residue $mv_x + c = c_{x_1}$ give rise to consider to patch the proof attempt by speculating the residue $mv_x + c = c_{x_1}$ as lemma.

In general, the failure and its solution follow this pattern: A method tests in its application conditions for a unifier or a matching of two terms t and t'. The unification or matching of t and t' fails because of some residues. If these residues look promising to be provable in the current context, then they are speculated as lemmas. The lemmas are used

---

$$limR_{(\nu\nu)\nu\nu o} \equiv \lambda f_{\nu\nu}.\lambda a_\nu.\lambda l_\nu.\forall \epsilon_\nu.(0 < \epsilon \Rightarrow \exists \delta_\nu.(0 < \delta \wedge \forall x_\nu.(x - a > 0 \wedge x - a < \delta \Rightarrow |f(x) - l| < \epsilon)))$$

$$limL_{(\nu\nu)\nu\nu o} \equiv \lambda f_{\nu\nu}.\lambda a_\nu.\lambda l_\nu.\forall \epsilon_\nu.(0 < \epsilon \Rightarrow \exists \delta_\nu.(0 < \delta \wedge \forall x_\nu.(a - x > 0 \wedge a - x < \delta \Rightarrow |f(x) - l| < \epsilon)))$$

$Ass.$	Ass	$\vdash \lim_{x\to 0} f(x+c)=l$	(Hyp)
$L_2.$	Ass	$\vdash \forall \epsilon_\centerdot (0<\epsilon \Rightarrow \exists \delta_\centerdot (0<\delta \wedge \forall x_\centerdot (\|x-0\|<\delta \wedge \|x-0\|>0 \Rightarrow \|f(x+c)-l\|<\epsilon)))$	(DefnUnfold-F Ass)
$L_{19}.$	L_{19}	$\vdash 0<c_\delta \wedge \forall x_\centerdot (\|x-0\|<c_\delta \wedge \|x-0\|>0 \Rightarrow \|f(x+c)-l\|<mv_\epsilon)$	(Hyp)
$L_4.$	L_4	$\vdash 0<c_{\epsilon_1}$	(Hyp)
$L_{10}.$	L_{10}	$\vdash \|c_{x_1}-c\|>0 \wedge \|c_{x_1}-c\|<mv_\delta$	(Hyp)
$L_{27}.$	$\mathcal{H}_1$	$\vdash \|mv_x-c\|<c_{\delta_1}$	($Open$)
$L_{28}.$	$\mathcal{H}_1$	$\vdash \|mv_x-c\|>0$	($Open$)
$L_{16}.$	$\mathcal{H}_2$	$\vdash 0<mv_\epsilon$	($Open$)
$L_{26}.$	$\mathcal{H}_1$	$\vdash \|f(mv_x+c)-l\|<mv_\epsilon$	(UnwrapHyp L_2)
$L_{25}.$	$\mathcal{H}_1$	$\vdash \|f(c_{x_1})-l\|<c_{\epsilon_1}$	($Open$)
$L_{11}.$	$\mathcal{H}_2$	$\vdash \|f(c_{x_1})-l\|<c_{\epsilon_1}$	(UnwrapHyp L_2 L_{25} L_{16} L_{27} L_{28})
$L_7.$	Ass, L_4	$\vdash 0<mv_{\delta_1}$	(TellCS-B)
$L_1.$	Ass	$\vdash \forall \epsilon_{1\centerdot} (0<\epsilon_1 \Rightarrow \exists \delta_{1\centerdot} (0<\delta_1 \wedge \forall x_{1\centerdot} (\|x_1-c\|<\delta_1 \wedge \|x_1-c\|>0 \Rightarrow \|f(x_1)-l\|<\epsilon_1)))$	(NormalizeLineTask L_7 L_{11})
$Thm.$	Ass	$\vdash \lim_{x_1\to c} f(x_1)=l$	(DefnUnfold-B L_1)

$\mathcal{H}_1=\{Ass, L_4, L_{10}, L_{19}\}$, $\mathcal{H}_2=\{Ass, L_4, L_{10}\}$

Figure 15: ϵ-δ-proof for second part of exercise 4.1.3 (part I).

to rewrite the initial terms such that afterwards the unification or matching succeeds and the method becomes applicable.

The question is, when is a residue promising to be provable in the current context? In the limit domain, we exploit the constraint solver $\mathcal{CoSIE}$ to decide whether residues are promising lemmas. Whereas the employed unification and matching are decidable procedures that depend on no domain-specific knowledge, $\mathcal{CoSIE}$ employs domain knowledge of inequalities and equations over the field of real numbers. To exploit this domain knowledge as well as the context information passed to $\mathcal{CoSIE}$ so far we query $\mathcal{CoSIE}$ whether it accepts the residues before we speculate them as lemmas. In this way, we combine the domain-independent unification and matching with the domain knowledge contained in $\mathcal{CoSIE}$.[16]

Technically, the described productive use of failing unifications and matchings for lemma speculation is encoded in the control rule `choose-equation-residues` in SolveInequality. This control rule analyzes the residues of blocked unifications and matchings and queries $\mathcal{CoSIE}$ whether it accepts the residues. If this is the case, `choose-equation-residues` fires and suggests the application of the method =Subst*-B. This method is similar to the method =Subst-B. However, it rewrites a goal by simultaneously applying a set of equations, whereas =Subst-B applies only one equation. Moreover, the equations are given as parameters to =Subst*-B and become new goals, i.e., are speculated as lemmas,

[16] An alternative to this combination is theory unification, which incorporates domain-specific equations 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 [5]). We prefer decidable unification and matching procedure in order to avoid undecidable application conditions whose evaluation can block the complete proof planning process.

whereas the application of $=$Subst-B requires one equation as derived assumption.

We shall elaborate this approach with our example. When SolveInequality fails to tackle $|f(c_{x_1}) - l| < c_{\epsilon_1}$ with the new support $|f(mv_x + c) - l| < mv_\epsilon$, then Multi creates the failure record

$$applcondfailure(unify(|f(mv_x + c) - l|, |f(c_{x_1}) - l|), \text{Solve*-B}, A')$$

for the method Solve*-B. This failure record states that the unification of the two terms $|f(mv_x + c) - l|$ and $|f(c_{x_1})$. The analysis of the failure record by the control rule `choose-equation-residues` yields the residue $mv_x + c = c_{x_1}$, which is accepted by $\mathcal{C}o\mathcal{SIE}$. Hence, the control rule `choose-equation-residues` fires and guides the application of $=$Subst*-B with $mv_x + c \doteq c_{x_1}$ as new lemma.

L_{30}.	$\mathcal{H}_1$	$\vdash mv_x + c \doteq c_{x_1}$	(TellCS-B)
L_{31}.	$\mathcal{H}_1$	$\vdash mv_\epsilon \leq c_{\epsilon_1}$	(TellCS-B)
L_{26}.	$\mathcal{H}_1$	$\vdash \|f(mv_x + c) - l\| < mv_\epsilon$	(UnwrapHyp L_2)
L_{29}.	$\mathcal{H}_1$	$\vdash \|f(mv_x + c) - l\| < c_{\epsilon_1}$	(Solve*-B L_{26} L_{31})
L_{25}.	$\mathcal{H}_1$	$\vdash \|f(c_{x_1}) - l\| < c_{\epsilon_1}$	($=$Subst*-B L_{29} L_{30})

Figure 16: ϵ-δ-proof for second part of exercise 4.1.3 (part II).

Figure 16 displays the application of $=$Subst*-B and the following $\mathcal{PDS}$ segment computed by SolveInequality for our example. The application of $=$Subst*-B to the goal $|f(c_{x_1}) - l| < c_{\epsilon_1}$ in L_{25} results in the new goals $|f(mv_x + c) - l| < c_{\epsilon_1}$ in L_{29} and $mv_x + c \doteq c_{x_1}$ in L_{30}. SolveInequality closes $mv_x + c \doteq c_{x_1}$ with TellCS-B, which passes the constraint to $\mathcal{C}o\mathcal{SIE}$. $|f(mv_x + c) - l| < c_{\epsilon_1}$ is closed by Solve*-B with respect to the support $|f(mv_x + c) - l| < mv_\epsilon$ in L_{26}. This is now possible since the unification became unblocked. The resulting goal in L_{31} is closed by TellCS-B.

$\mathcal{C}o\mathcal{SIE}$ derives $mv_x \doteq c_{x_1} - c$ from the given formula $mv_x + c \doteq c_{x_1}$. This determines mv_x, so that SolveInequality switches to InstIfDetermined, which introduces the binding $mv_x :=^b c_{x_1} - c$ into the strategic proof plan. With respect to this binding the remaining goals in L_{27} and L_{28} become $|(c_{x_1} - c) - 0| < c_\delta$ and $|(c_{x-1} - c) - 0| > 0$. Applications of Simplify-B reduce these goals to $|c_{x_1} - c| < c_\delta$ and $|c_{x-1} - c| > 0$, which SolveInequality closes with supports derived from line L_{10}.

Another problem from the limit domain, which requires a similar speculation of lemmas is the reverse of exercise 4.1.12 from [2], which states that

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

Unfolding of lim and normalization result in the goal $|f(c_{x_1}) - l| < c_{\epsilon_1}$. Unwrapping of the assumption yields $|f(a * mv_x) - l| < mv_\epsilon$. The application of Solve*-B with respect to these two terms is blocked since the unification has the residue $a * mv_x = c_{x_1}$. Since $\mathcal{C}o\mathcal{SIE}$ accepts the constraint $a * mv_x \doteq c_{x_1}$ SolveInequality can unblock the unification and can apply Solve*-B. $\mathcal{C}o\mathcal{SIE}$ yields $\frac{c_{x_1}}{a}$ as instantiation for mv_x.[17]

[17] This is another example that needs eager meta-variable instantiation. Since $a * mv_x \doteq c_{x_1}$ determines

5.3 Goal-Directed Backtracking

Goal-directed reasoning selects and applies steps in order to achieve some given goals. That is, a step is either chosen since it directly achieves some of the current goals or since its effects enable some other desirable steps that are likely to help to achieve the given goals. Typically, in search procedures backtracking is not a goal-directed operation in its own right but only a necessary operation to traverse the search space. MULTI provides the freedom to backtrack any actions in the proof plan under construction. This allows for *goal-directed backtracking*, that is, backtracking that is not just part of the traversal of the search space but that aims to work towards the current goals by enabling desirable steps. In this section, we shall discuss a type of situation in which goal-directed backtracking is suggested by meta-reasoning on a highly desirable but blocked strategy.

As example problem consider the problem LIM-DIV-1-X, which states that

$$Thm\colon \lim_{x\to c}\frac{1}{x}=\frac{1}{c} \text{ for } c>0.$$

Figure 17 depicts the $\mathcal{PDS}$ that is created for this problem before the highly desirable but blocked strategy occurs.

$Ass.$	Ass	$\vdash 0<c$	(Hyp)
$L_8.$	L_8	$\vdash \lvert c_x-c\rvert<mv_\delta \wedge \lvert c_x-c\rvert>0$	(Hyp)
$L_4.$	L_7	$\vdash 0<c_\epsilon$	(Hyp)
$L_{10}.$	L_8	$\vdash \lvert c_x-c\rvert<mv_\delta$	($\wedge$E-F L_8)
$L_{11}.$	L_8	$\vdash \lvert c_x-c\rvert>0$	($\wedge$E-F L_8)
$L_{13}.$	$\mathcal{H}_1$	$\vdash 0<mv_f$	(TELLCS-B)
$L_{14}.$	$\mathcal{H}_1$	$\vdash \lvert c_x*c\rvert>mv_f$	(TELLCS-B)
$L_{15}.$	$\mathcal{H}_1$	$\vdash \lvert c-c_x\rvert<mv_f*c_\epsilon$	(TELLCS-B)
$L_{12}.$	$\mathcal{H}_1$	$\vdash \lvert\frac{c-c_x}{c_x*c}\rvert<c_\epsilon$	(FACTORIALESTIMATE-B L_{13} L_{14} L_{15})
$L_9.$	$\mathcal{H}_1$	$\vdash \lvert\frac{1}{c_x}-\frac{1}{c}\rvert<c_\epsilon$	(SIMPLIFY-B L_{12})
$L_6.$	Ass, L_7	$\vdash 0<mv_\delta$	(TELLCS-B)
$L_1.$	Ass	$\vdash \forall\epsilon.(0<\epsilon\Rightarrow\exists\delta.(0<\delta\wedge \forall x.(\lvert x-c\rvert<\delta\wedge\lvert x-c\rvert>0 \Rightarrow \lvert\frac{1}{x}-\frac{1}{c}\rvert<\epsilon)))$	(NormalizeLineTask L_6 L_9)
$Thm.$	Ass	$\vdash \lim_{x\to c}\frac{1}{x}=\frac{1}{c}$	(DEFNUNFOLD-B L_1)
$\mathcal{H}_1=\{Ass,L_4,L_8\}$			

Figure 17: ϵ-δ-proof for LIM-DIV-1-X before failure.

The unfolding of the defined symbol *lim* and the normalization of the resulting complex goal results in the two goals $0<mv_\delta$ in L_6 and $\lvert\frac{1}{c_x}-\frac{1}{c}\rvert<c_\epsilon$ in L_9. SolveInequality closes the first goal by an application of TELLCS-B whereas it simplifies the second goal to $\lvert\frac{c-c_x}{c_x*c}\rvert<c_\epsilon$ in L_{12}. An application of FACTORIALESTIMATE-B to this goal results in the three goals $0<mv_f$ in L_{13}, $\lvert c_x*c\rvert>mv_f$ in L_{14}, and $\lvert c-c_x\rvert<mv_f*c_\epsilon$ in L_{15}. SolveInequality closes these three goals with TELLCS-B.

mv_x, the binding $mv_x{:=}^b\,\frac{c_{x_1}}{a}$ is introduced into the proof plan. The unwrapping of the support also yields the two goals $\lvert mv_x-0\rvert<c_\delta$ and $\lvert mv_x-0\rvert>0$, which are simplified with respect to the binding to $\lvert\frac{c_{x_1}}{a}\rvert<c_\delta$ and $\lvert\frac{c_{x_1}}{a}\rvert>0$. Whereas MULTI can solve these two goals from the supports $\lvert c_{x_1}\rvert>0\wedge\lvert c_{x_1}\rvert<mv_\delta$ by applications of COMPLEXESTIMATE-B, PLAN fails to prove the goals without the eager instantiation.

Since then all line-tasks are closed $\mathcal{C}o\mathcal{SIE}$ is supposed to provide instantiations for the meta-variables mv_δ and mv_f that are consistent with the collected constraints. That is, the strategy ComputeInstFromCS, which asks $\mathcal{C}o\mathcal{SIE}$ to compute the instantiations, becomes a highly desirable strategy. However, $\mathcal{C}o\mathcal{SIE}$ fails to compute instantiations in this situation and ComputeInstFromCS does not succeed.

What is the problem? So far, $\mathcal{C}o\mathcal{SIE}$ did collect the constraints

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

The critical constraints are the constraints on mv_f that state that $\frac{|c_x - c|}{c_\epsilon}$ has to be less than mv_f, which has to be less than $|c_x * c|$. So far, these constraints are not inconsistent. However, these constraints are consistent and 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 constraints collected so far.[18] In particular, the constraints collected so far are not sufficient for an ϵ-δ-proof since they do not establish a connection between the ϵ and the δ.

A possibility to overcome this problem is to refine and extend the set of constraints in order to obtain a set of constraints for which a solution can be computed. To do so, applications of TELLCS-B can be backtracked in a goal-directed manner in order to enable the applications of other methods that further refine the selected constraints.

We encoded the described idea in the strategic control rule `backtrack-to-unblock-cosie`. When all line-tasks are closed, but ComputeInstFromCS is not applicable since $\mathcal{C}o\mathcal{SIE}$ fails to compute instantiations, then this control rule analyzes the constraints passed to $\mathcal{C}o\mathcal{SIE}$ by TELLCS-B. It triggers the backtracking of actions of TELLCS-B that pass complex inequalities to $\mathcal{C}o\mathcal{SIE}$ that can be further refined.[19] When SolveInequality tackles the re-opened proof lines, it cannot close them again with TELLCS-B but has to refine them. Afterwards, it can pass the refined goals to $\mathcal{C}o\mathcal{SIE}$.

We shall elaborate this idea with our example. Triggered by the strategic control rule `backtrack-to-unblock-cosie` MULTI backtracks the application of TELLCS-B that closes L_{15}. SolveInequality reduces the re-opened goal L_{15} with COMPLEXESTIMATE-B. Afterwards, it passes the resulting inequality goals by applications of TELLCS-B to $\mathcal{C}o\mathcal{SIE}$. Since $\mathcal{C}o\mathcal{SIE}$ also fails on this extended constraint set MULTI backtracks the application of TELLCS-B that closes L_{14}. Again, SolveInequality reduces the re-opened goal with COMPLEXESTIMATE-B and passes the resulting inequalities to $\mathcal{C}o\mathcal{SIE}$. The new $\mathcal{PDS}$ segments for L_{14} and L_{15} are shown in Figure 18.

This results in the following constraint store:

[18] Note that the application condition *test-CS* of TELLCS-B does not ask $\mathcal{C}o\mathcal{SIE}$ to establish consistency for a new constraint, i.e., to check the existence of a solution binding of the meta-variables. Rather, $\mathcal{C}o\mathcal{SIE}$ checks only whether it derives an inconsistency. In this case a new constraint is rejected. If no inconsistency is detected, then $\mathcal{C}o\mathcal{SIE}$ collects the constraint. Hence, it can happen – a quite common situation in constraint solving – that the available information is not sufficient to compute a solution binding of the constraint variables, although inconsistency is not detected.

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

L_{10}.	L_8	$\vdash \lvert c_x - c \rvert < mv_\delta$	($\wedge$E-F L_8)
L_{11}.	L_8	$\vdash \lvert c_x - c \rvert > 0$	($\wedge$E-F L_8)
L_{22}.	$\mathcal{H}_1$	$\vdash 0 < mv'$	(TellCS-B)
L_{23}.	$\mathcal{H}_1$	$\vdash \lvert c \rvert < mv'$	(TellCS-B)
L_{24}.	$\mathcal{H}_1$	$\vdash \lvert c * c \rvert \geq mv_f * 2$	(TellCS-B)
L_{25}.	$\mathcal{H}_1$	$\vdash mv_\delta \leq \frac{mv_f}{mv'}$	(TellCS-B)
L_{14}.	$\mathcal{H}_1$	$\vdash \lvert c_x * c \rvert > mv_f$	(ComplexEstimate-B L_{10} L_{22} L_{23} L_{24} L_{25})
L_{17}.	$\mathcal{H}_1$	$\vdash \lvert -1 \rvert \leq mv$	(TellCS-B)
L_{18}.	$\mathcal{H}_1$	$\vdash mv_\delta \leq \frac{c_\epsilon * mv_f}{2 * mv}$	(TellCS-B)
L_{19}.	$\mathcal{H}_1$	$\vdash \lvert 0 \rvert < \frac{c_\epsilon * mv_f}{2}$	(TellCS-B)
L_{20}.	$\mathcal{H}_1$	$\vdash 0 < mv$	(TellCS-B)
L_{15}.	$\mathcal{H}_1$	$\vdash \lvert c - c_x \rvert < mv_f * c_\epsilon$	(ComplexEstimate-B L_{10} L_{17} L_{18} L_{19} L_{20})

Figure 18: Extended ϵ-δ-proof for LIM-DIV-1-X.

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

Bindings that are consistent with these constraints are: $mv {:=}^b 2$, $mv' {:=}^b c+1$, $mv_f {:=}^b \frac{c^2}{2}$, and $mv_\delta {:=}^b min(\frac{c_\epsilon * c^2}{8}, \frac{c^2}{2*(c+1)})$. Unfortunately, the solution of the above constraint system is not in the scope of the current $\mathcal{C}o\mathcal{SIE}$ system. That is, $\mathcal{C}o\mathcal{SIE}$ fails to provide instantiations although a solution that is consistent with all constraints exists and establishes a connection between the ϵ and the δ of our ϵ-δ-proof.[20] Since `backtrack-to-unblock-cosie` detects no further inequality goals that probably can be further refined Multi terminates without bindings for the meta-variables. Despite the successful failure analysis that triggered goal-directed backtracking, the problem cannot be solved completely because of drawbacks of the current $\mathcal{C}o\mathcal{SIE}$ system.

All problems of the limit domain that result in absolute values of fractions that are tackled with FactorialEstimate-B need the described failure reasoning. For instance, exercises 4.1.10$(a)-(d)$ in [2]:

$$\lim_{x\to 2} \frac{1}{1-x} = -1, \lim_{x\to 1} \frac{x}{x+1} = \frac{1}{2}, \lim_{x\to 0} \frac{x^2}{|x|} = 0, \lim_{x\to 1} \frac{x^2-x+1}{x+1} = \frac{1}{2},$$

and problems on the derivative of functions such as theorem 6.1.3(a) and (b) in [2]:

$$deriv(f,a) = f' \Rightarrow deriv(\alpha * f, a) = \alpha * f',$$
$$deriv(f,a) = f' \wedge deriv(g,a) = g' \Rightarrow deriv(f+g,a) = f' + g'.$$

Note that the current $\mathcal{C}o\mathcal{SIE}$ system fails for all these problems to compute suitable instantiations.

[20] The reason for $\mathcal{C}o\mathcal{SIE}$ failing to find this solution is the mutual dependency of the variables mv_f and mv_δ. mv_f occurs in an upper bound of mv_δ, and in turn mv_δ occurs in a lower bound of mv_f. The search procedure of the current $\mathcal{C}o\mathcal{SIE}$ system is not complete in a sense that it can not resolve all dependencies of this kind.

6 Results and Discussion

This report presents the application of MULTI to the limit domain. MULTI can solve all problems that PLAN can solve[21] and it successfully plans various problems that are beyond the capabilities of PLAN. In particular, MULTI can solve problems that require eager meta-variable instantiations as well as problems that require meta-reasoning on failures to introduce case-splits, to speculate lemmas, and to guide goal-directed backtracking.

The discussed speculation of lemmas is not possible in PLAN since it does not create and maintain suitable information on failures such as the failure records of MULTI. All other problems are beyond the capabilities of PLAN since it cannot flexibly combine planning, backtracking, and meta-variable instantiation based on meta-reasoning.

We conclude the report with a discussion of related work and an evaluation of the realized proof planning approach.

6.1 Related Work

Related Work on Proving Limit Theorems

Some of the knowledge encoded in the methods of the SolveInequality strategy is similar to ideas implemented in the theorem prover IMPLY [7] developed by BLEDSOE. For instance, COMPLEXESTIMATE-B is inspired by BLEDSOE's limit heuristic. BLEDSOE and HINES developed a resolution-based prover for inequalities [9], which can prove, for instance, the Continuous+ problem. BEESON worked on ϵ-δ-proofs automatically created by the systems MATHPERT and WEIERSTRASS [3]. All these systems rely on special-purpose routines that are implemented into the systems. As opposed thereto, only the strategies, methods, and control rules are domain-specific in ΩMEGA's knowledge-based proof planning, the representational techniques and reasoning procedures are general-purpose.

With a particular control setting the automated theorem prover OTTER [29] can solve a simple version of LIM+ (simplified by the provision of additional lemmas that "encode" necessary computations etc.). However, this setting is tailored to LIM+ and does not work for LIM* or other limit theorems. In auto-mode OTTER is not able to prove the simple version of LIM+. In contrast, our strategies, methods, and control rules cover the mathematical knowledge in a form that is general enough to solve all limit problems in Appendix B. More theorems could be formulated.

The LIM+ problem was also proved in CLAM [46] with a special heuristic called *colored rippling*. But LIM* and other theorems of the limit domain turned out to be too difficult for CLAM.

Related Work on Failure Reasoning

Failure reasoning in the proof planner CLAM is closely related to the lemma speculation and the introduction of case-splits in MULTI. Since a detailed comparison of the failure

[21] In particular, all challenge problems that BLEDSOE proposed in 1990 [6], among them the limit theorems LIM+, LIM-, LIM*, the theorems Continuous+, Continuous-, Continuous*, $\lim_{x\to a} x = a$, and $\lim_{x\to a} c = c$ (see [36]).

reasonings requires some technical details of CLAM we shall discuss it in the subsequent section 6.2.

The speculation of residue lemmas has something in common with HUETS constrained resolution [22]. Since unification is undecidable in higher-order logics constrained resolution intertwines resolution steps with unification. Instead of solving the unification problem $t = t'$ as a precondition of a resolution step, the resolution step is performed and $t = t'$ becomes part of the resolution problem. This process is difficult to control since the introduced unification residue $t = t'$ can be as difficult to solve as the rest of the proof. We also intertwine unification with the main proof process by speculating unification residues as lemmas. But, as opposed to constrained unification, we strictly control the speculation of the lemmas since we allow only for such lemmas that are directly accepted by $\mathcal{C}o\mathcal{SIE}$.

Related to goal-directed backtracking in MULTI is the goal-directed reasoning in elaborate blackboard systems such as HEARSAY-III and BB1 (e.g., see [16, 26]). One approach to integrate goal-directed reasoning in blackboard systems is the construction (and modification) of meta-plans of highly desirable knowledge source applications that guide the following solution process [17]. 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. When performing goal-directed backtracking, we do not construct meta-plans of strategy applications but we also exploit knowledge of when the application of particular strategies is highly desirable and how to unblock a highly desirable but blocked strategy.

6.2 Failure Reasoning in CLAM

In the following, we shall first describe the use of critics in CLAM and then compare failure reasoning with critics with our failure reasoning encoded in control rules.

Critics in CLAM

BUNDY and IRELAND propose critics as a means to patch failed proof attempts by exploiting information on failures in [24] and [25]. The motivation for the introduction of critics is similar to our motivation for failure reasoning: failures in the proof planning process, in particular, failures occurring after partially successful operations, often hold the key to discover a solution proof plan.

Critics in CLAM extend the hierarchy of inference rules, tactics, and methods. They are introduced in order to complement proof methods. A critic is associated with one method and captures patchable exceptions to the application of the method. Since the application of a method can fail in various ways, each method may be associated with a number of critics. Critics are expressed in terms of preconditions and patches. The preconditions analyze the reasons why the method has failed to apply. The proposed patch suggests a change to the proof plan. This change can be a manipulation of the whole proof plan or the change can be a local manipulation of goals.

To describe the failure reasoning in CLAM we have to consider the construction of inductive proofs in CLAM in some detail. Proof construction in CLAM relies on the domain-independent rippling heuristic [13, 23]. The rippling heuristic is based upon the observation that the induction hypothesis is syntactically similar to the induction conclusion. In order to derive the induction conclusion from the induction hypothesis the `ripple` method tries to rewrite the induction conclusion, such that the induction hypothesis can be used. The `ripple` method iterates over the `wave` method, which applies conditional rewrite rules of the form $Conds \rightarrow (LHS \Rightarrow RHS)$, where LHS is the left hand side, RHS is the right hand side, and $Conds$ are the conditions of the rewrite rule. When $Hyps$ and $Conc$ denote the current hypotheses and the conclusion, respectively, then the preconditions of the `wave` method are:[22]

1. There is a subterm Sub of the conclusion $Conc$, which should be rewritten.

2. There is a conditional rewrite rule $Conds \rightarrow (LHS \Rightarrow RHS)$ such that LHS matches with Sub.

3. The conditions $Conds$ are satisfied by the hypotheses $Hyps$ (i.e., $Hyps \vdash Conds$ is a tautology).

The application of the `wave` method fails, when one of its preconditions is not satisfied. BUNDY and IRELAND realized two patches for the method, which are implemented as critics associated with the method:

1. A failure of precondition 2, i.e., there is no rewrite rule that can be applied, triggers the `lemma-discovery` critic. The preconditions for the application of this critic are: (1) precondition 1 of the `wave` method holds and (2) preconditions 2 and 3 fail. The patch of the critic involves the speculation and proof of a rewrite rule to unblock this situation. This process may involve backtracking, when a speculated rewrite rule cannot be proved.

2. A failure of precondition 3, i.e., the condition of a matching rewrite rule is not satisfied in the current context, triggers the `missing-condition` critic. The preconditions for the application of this critic are: (1) precondition 1 of the `wave` method holds, (2) precondition 2 of the `wave` method holds with respect to a rewrite rule $Conds \rightarrow (LHS \Rightarrow RHS)$, and (3) precondition 3 fails for $Conds$. The patch of the critic is to perform a case analysis based upon the unprovable conditions $Conds$.

These two critics are tailored to the possible failures of the application of the `wave` method. The general ideas behind the critics are:

Lemma Speculation: When no methods are applicable with respect to the current context, the controlled speculation (and the proof) of new lemmas can unblock the proof planning process.

Case Analysis: Splitting a problem into different cases can unblock the proof planning process, when no methods are applicable.

[22]Actually, there are different `wave` methods for different kinds of rippling (e.g., longitudinal-rippling and transverse-rippling), which have some more preconditions that differ slightly among the different `wave` methods, see [13, 25] for details. For the sake of simplicity we discuss here only the relevant preconditions.

Bundy and Ireland describe also critics of other methods that patch the selection of the induction schemata and generalize conjectures in order for an inductive proof to succeed (see [25]).

Comparison with Failure Reasoning in MULTI

The situations that trigger lemma speculation and case-splits in CLAM and MULTI are very similar: missing premises in the current context (i.e., missing rewrite rules in CLAM or missing supports in MULTI) trigger lemma speculation; unprovable premises of conditional facts from the context (i.e., conditional rewrite rules in CLAM or conditional supports in MULTI) cause case-splits. 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 in CLAM are an extra concept introduced for failure reasoning. A critic reasons on failures of the one method it is directly associated with, i.e., it reasons on failing preconditions of the method. Part of a critic is a patch of the failure. Technically, this patch is a special procedure that can change the complete proof plan.

In contrast, failure reasoning in MULTI is conducted by control rules. The control rules are not associated with a particular method but rather test for particular situations that can occur during the proof planning process (independent from which strategy or method caused the situation). The control rules reason on the current proof plan and on all other available 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.

The advantage of MULTI's approach is that control rules allow for method- and strategy-independent reasoning on failures. For instance, the rule `choose-equation-residues`, which guides the lemma speculation can deal with failing unifications and matchings in the application conditions of any employed method. It is domain-independent since it could be employed in cooperation with other constraint solvers similar to the cooperation with $\mathcal{CoSIE}$ described in section 5.2.

We decided to realize patches in MULTI by control rules that guide the application of existing strategies and methods since procedural patches are difficult to maintain. Both the introduction and the deletion of a patch for a desired manipulation requires the implementation of special procedures. For complex proof plan manipulations the cooperation of several methods and strategies can be necessary and has to be guided by several control rules. For instance, when performing case analysis, MULTI has to backtrack the application of the conditional support. Afterwards, it has to introduce the case-split and finally it has to replay the backtracked parts again (in order to avoid to prove again from the scratch). The necessary failure reasoning and the knowledge of how to patch this failure is distributed among three control rules: one strategic control rule that guides the backtracking, one control rule that guides the case split, and one control rule that guides the replay of the backtracked parts. Although the failure reasoning is distributed we see the three involved control rules as one meta-reasoning entity that is distributed for technical reasons.

6.3 Evaluation of the Proof Planning Approach

Knowledge-based proof planning relies on the acquisition, formalization, and use of domain-specific knowledge in methods, control rules, and strategies. However, there is the constant danger to acquire over-specific knowledge as Bundy points out:

> *A new method or critic may originally be inspired by only a handful of examples. There is a constant danger of producing methods and critics that are too find tuned to these initial examples. This can arise both from a lack of imagination in generalizing from the specific situation and from the temptation to get quick results in automation. Such over-specificity leads to a proliferation of methods and critics with limited applicability.*
>
> Bundy, [12]

Bundy suggests in [12] and [11] the criteria *generality* and *parsimony* to evaluate the appropriateness of proof planning methods and critics. Generality means that each method or critic should apply successfully in a wide range of situations, whereas parsimony means that a few methods should generate a large number of proofs.

These criteria of Bundy do not consider mathematical content, which is an important issue in knowledge-based proof planning. The methods, control rules, and strategies in knowledge-based proof planning should be rich in mathematical content. Thus, the art of knowledge-based proof planning is to acquire domain knowledge that, on the one hand, comprises meaningful mathematical techniques and powerful heuristic guidance, and, on the other hand, is general enough to tackle a broad class of problems.

In the following, we shall evaluate proof planning limit theorems with Multi. We discuss the amount of mathematical and domain-specific knowledge in strategies, methods, and control rules and discuss how general they are. We discuss generality not only in the sense of Bundy, that is, to how many problem classes a concrete strategy, method, or control rule applies. Rather, we discuss also how general the encoded principle is and how it can be transfered to other domains.

SolveInequality

The approach to tackle inequality problems with the SolveInequality strategy fits into a much more general heuristic strategy described by Schoenfeld:

> *In a problem 'to find' or 'to construct', it may be useful to assume that you have the solution to the given problem. With the solution (hypothetically) in hand, determine the properties it must have. Once you know what those properties are, you can find the object you seek.*
>
> Schoenfeld, [41] p. 23

When tackling inequality problems, SolveInequality assumes that solutions for existentially quantified variables exist (e.g., for the δ in ϵ-δ-proofs) and substitutes the existentially quantified variables by meta-variables. Afterwards, it collects constraints on the introduced meta-variables in $\mathcal{CoSIE}$, which at the end computes instantiations for the meta-variables.

Now that we know that SolveInequality fits into the general strategy "assume, collect properties, then compute", could we encode a general version of this strategy that can tackle various domains and subsumes SolveInequality? Probably not, since, as SCHOENFELD points out, such a general heuristic strategy alone provides no adequate information on how to use this strategy in a concrete case.

> [...] *that a typical heuristic strategy is very broadly defined — too broadly, in fact, for the description of the strategy to serve as a useful guide to its implementation.*
>
> Schoenfeld,[41] pp. 70 and 72

Rather, such general strategies have to be filled with domain-specific knowledge such that the general strategy is only a summary label for a class of substrategies for different domains:

> [...] *the successful implementation of heuristic strategies in any particular domain often depends heavily on the possession of specific subject matter knowledge.*
> [...] *More often than not, a capsule description of a strategy is a summary label that includes under it a class of more precise substrategies that may be only superficially related.*
>
> Schoenfeld,[41] pp. 92 and 95

Thus, in the sense of SCHOENFELD, SolveInequality is a substrategy of the general strategy "assume, collect properties, then compute". It instantiates this general principle with the specific knowledge on how to apply it to inequalities over the reals.

The main control rule of SolveInequality, `prove-inequality`, encodes the essential idea of how SolveInequality implements the general principle for inequalities over the reals: reduce complex inequalities to simple inequalities and pass simple inequalities to the connected constraint solver. To tackle complex inequalities `prove-inequality` suggests domain-specific methods such as SIMPLIFY-B, COMPLEXESTIMATE-B, and FACTORIALESTIMATE-B. The methods encode mathematical knowledge of inequalities, real numbers, and the operations $+, -, *, /$ on real numbers. This knowledge is partially contained in the computer algebra system MAPLE that is employed within COMPLEXESTIMATE-B and SIMPLIFY-B. Moreover, `prove-inequality` suggests the methods TELLCS-B, TELLCS-F, and ASKCS-B that interface the constraint solver $\mathcal{CoSIE}$. These methods do not contain domain-specific mathematical knowledge but provide a domain-independent interface to constraint solvers.

The domain-specific methods of SolveInequality are hardly reusable in another substrategy of "assume, collect properties, then compute" for other domains. However, they could be useful for other problem classes dealing with inequalities over the reals. Currently, the methods TELLCS-B, TELLCS-F, and ASKCS-B interface only $\mathcal{CoSIE}$. However, they provide general functionalities, namely adding constraints and asking whether a constraint is entailed, that are independent of a concrete constraint solver. Thus, they can be used also in other domains with other constraint solvers (e.g., problems on sets with a constraint solver on sets).

The essence of the control rule `prove-inequality` could be reused in other substrategies of the "assume, collect properties, then compute" strategy for other domains with constraint solvers. In such a domain, the adaption of `prove-inequality` would suggest domain-specific methods to tackle complex expressions of this domain until TELLCS-B, TELLCS-F, and ASKCS-B involve a constraint solver of the domain to handle the simple expressions.

SolveInequality also contains some logic-level methods, for instance, CONTRA-B to perform indirect proofs and DEFNUNFOLD-B and DEFNUNFOLD-F for unfolding of defined concepts. These methods are domain-independent and contain no particular mathematical knowledge. The decision when to perform an indirect proof and which definitions to unfold and which not are difficult problems in theorem proving in general (e.g., see [8, 49, 20] for discussions on unfolding of defined concepts). Their application within SolveInequality is guided by control rules that encode mathematical heuristics. For instance, since the purpose of SolveInequality is to tackle inequalities it only unfolds defined concepts that result in inequalities. This knowledge is encoded in the control rule `select-unfold-defined-concept`, which guides the application of DEFNUNFOLD-B and DEFNUNFOLD-F. The meta-reasoning to guide indirect proofs in the limit domain is discussed in [35].

SolveInequality employs some further control rules that do not encode mathematically meaningful heuristics but deal with technical peculiarities that occur during the search process. As example for such a control rule consider `block-simplify`, which restricts applications of the methods SIMPLIFY-F and SIMPLIFY-B. Both methods employ MAPLE to simplify arithmetic terms. Unfortunately, it turned out that the normal form simplifications provided by MAPLE sometimes result in terms that are not suitable for further proof planning with them (from the proof plan perspective they are not simplified). To avoid this `block-simplify` rejects all applications of SIMPLIFY-F and SIMPLIFY-B that are supporting the proof planning process.

Altogether, SolveInequality is not restricted to limit problems. Rather, its approach is general enough to tackle also other inequality problems over the reals. However, since we did focus on limit problems so far, the methods of SolveInequality are focused on inequalities with absolute values. To extend the solvability horizon of the strategy some methods are needed that tackle complex inequalities without absolute values, for instance, methods similar to COMPLEXESTIMATE-B or methods that isolate subterms in complex inequalities (isolating x in $(c-x)+a<\epsilon$ results in $x>(c+a)-\epsilon$).[23]

NormalizeLineTask and UnwrapHyp

The **PPLANNER** strategies NormalizeLineTask and UnwrapHyp contain only logic-level methods to decompose complex formulas in goals and supports. Thus, they are very general in the sense of BUNDY, but they do not encode any specific mathematical knowledge. However, they implement operations that are important in mathematical problem solving in general since the decomposition of complex goals and the unwrapping of subformulas of complex assumptions is necessary in all mathematical domains where complex statements are composed from primitive ones by logical connectives and quantifiers.

[23] An example theorem that requires the handling of complex inequalities without absolute values is the Squeeze-Theorem. Although we employ this theorem when proving problems with the ReduceToSpecial strategy it currently cannot be proved by MULTI.

InstMeta Strategies

Similar to the methods TellCS-B, TellCS-F, and AskCS-B the **InstMeta** strategies InstIfDetermined and ComputeInstFromCS encode no particular mathematical knowledge but provide interface functions to constraint solvers. Although, currently they interface only $\mathcal{CoSIE}$, they provide functionalities, namely retrieving particular entailed constraints and computation of instantiations, that are independent of a concrete constraint solver. Thus, they could be employed also in other domains.

Failure Reasoning

The described mathematical knowledge to speculate lemmas and to introduce case-splits are general meta-reasoning patterns, promising also for other domains. As evidence for this statement consider that the corresponding critics in CLAM exploit very similar failures in a completely different domain to guide similar proof modifications.

The domain-specific part of the lemma speculation described in section 5.2 is the decision of which lemmas are promising and which not. To avoid the speculation of arbitrary lemmas that cannot be proved in the current context, SolveInequality asks $\mathcal{CoSIE}$ whether it accepts a potential lemma. This exploits the domain-specific information encoded in $\mathcal{CoSIE}$ as well as the context information passed to $\mathcal{CoSIE}$ so far. The same approach could be performed in other domains with constraint solvers that contain particular domain knowledge. Other domains may need different kinds of guidance to decide whether lemmas are promising.

The domain-specific part of the case-split introduction discussed in section 5.1 is the decision of which cases to consider. In the limit domain, the general case-split $C \vee \neg C$ was sufficient so far to deal with a failing condition C. The case-split $C \vee \neg C$ is domain-independent since it relies only on the *tertium-non-datur* axiom of ΩMEGA's underlying logic. However, it can be necessary to construct domain-specific case-splits. For instance, when C equals $a < b$, then the case-split $a < b \vee a = b \vee a > b$ could be considered. Different domains maybe provide different kinds of domain-specific case-splits.

The goal-directed backtracking discussed in section 5.3 is just one particular example of goal-directed reasoning on failures. More generally stated the principle works as follows: Suppose there is a meta-plan (either explicitly constructed somewhere or implicitly encoded in control rules) of the desired solution process, and suppose that a step S of this meta-plan fails. Then, the failure can be analyzed and further steps can be considered in order to unblock S. The concrete pattern (unblock ComputeInstFromCS if there are no further goals) is restricted to the limit domain (and maybe some other domains with constraint solvers). The general principle, however, is a domain-independent, promising meta-reasoning pattern for any domain for which a kind of meta-plan of the desired solution process exists.

Summary

Typical questions of referees of our papers on proof planning are, for instance:

- How many new methods are typically needed when a new chapter in a book is considered?

- How many of the methods can typically be reused, when a new chapter in a book is considered?

A general answer to those questions is not possible. When extending the domain of proof planning, the crucial question is whether the knowledge acquired so far is sufficient to tackle the new problems.

To illustrate this subtle point consider the following experiences in the limit domain. We started to develop proof planning in the limit domain with examples from chapter 4 and chapter 5 in [2] on the limit of functions and the continuity of functions. On the one hand, we found that the acquired knowledge was not sufficient to deal with several problems in chapter 4 and chapter 5. These problems need additional knowledge about particular functions involved. Currently, MULTI cannot solve, for instance, problems involving the square-root function since the methods and theorems do not contain appropriate knowledge of this function. On the other hand, we found that with the knowledge acquired for chapter 4 and chapter 5 MULTI can solve problems on the derivative of functions without any extensions in form of further methods, control rules, or theorems although this is a new chapter (chapter 6) in [2].

These experiences demonstrate the success and the limitation of the current proof planning for limit problems realized in MULTI:

1. The implemented methods, control rules, and strategies are not too fine tuned to our initial examples. In particular, the control rules contain the necessary control knowledge in a form that is general enough to deal also with new problems for which the domain knowledge in the methods and strategies is sufficient.

2. The implemented methods, control rules, and strategies are not sufficient to deal with any limit problems. They are mainly restricted to terms composed of $+, -, *, /, ||$. To deal with further expressions such as square-root requires further specific knowledge.

A Lim+ Example

Lim_f.	Lim_f	$\vdash \lim_{x \to a} f(x) = l_f$	(Hyp)
Lim_g.	Lim_g	$\vdash \lim_{x \to a} g(x) = l_g$	(Hyp)
L_2.	Lim_f	$\vdash \forall \epsilon_1 \centerdot (0 < \epsilon_1 \Rightarrow \exists \delta_1 \centerdot (0 < \delta_1 \wedge \forall x_1 \centerdot (\|x_1 - a\| < \delta_1 \wedge \|x_1 - a\| > 0 \Rightarrow \|f(x_1) - l_f\| < \epsilon_1)))$	(DefnUnfold-F Lim_f)
L_3.	Lim_g	$\vdash \forall \epsilon_2 \centerdot (0 < \epsilon_2 \Rightarrow \exists \delta_2 \centerdot (0 < \delta_2 \wedge \forall x_2 \centerdot (\|x_2 - a\| < \delta_2 \wedge \|x_2 - a\| > 0 \Rightarrow \|g(x_2) - l_g\| < \epsilon_2)))$	(DefnUnfold-F Lim_g)
L_{17}.	Lim_f	$\vdash 0 < mv_{\epsilon_1} \Rightarrow \exists \delta_1 \centerdot (0 < \delta_1 \wedge \forall x_1 \centerdot (\|x_1 - a\| < \delta_1 \wedge \|x_1 - a\| > 0 \Rightarrow \|f(x_1) - l_f\| < mv_{\epsilon_1}))$	($\forall$E-F L_2)
L_{18}.	$\mathcal{H}_3$	$\vdash 0 < mv_{\epsilon_1}$	(TellCS-B)
L_{20}.	$\mathcal{H}_3$	$\vdash \exists \delta_1 \centerdot (0 < \delta_1 \wedge \forall x_1 \centerdot (\|x_1 - a\| < \delta_1 \wedge \|x_1 - a\| > 0 \Rightarrow \|f(x_1) - l_f\| < mv_{\epsilon_1}))$	($\Rightarrow_E$ L_{18} L_{17})
L_{21}.	L_{21}	$\vdash 0 < c_{\delta_1} \wedge \forall x_1 \centerdot (\|x_1 - a\| < c_{\delta_1} \wedge \|x_1 - a\| > 0 \Rightarrow \|f(x_1) - l_f\| < mv_{\epsilon_1})$	(Hyp)
L_{23}.	L_{21}	$\vdash 0 < c_{\delta_1}$	($\wedge$E-F L_{21})
L_{24}.	L_{21}	$\vdash \forall x_1 \centerdot (\|x_1 - a\| < c_{\delta_1} \wedge \|x_1 - a\| > 0 \Rightarrow \|f(x_1) - l_f\| < mv_{\epsilon_1})$	($\wedge$E-F L_{21})
L_{25}.	L_{21}	$\vdash \|mv_{x_1} - a\| < c_{\delta_1} \wedge \|mv_{x_1} - a\| > 0 \Rightarrow \|f(mv_{x_1}) - l_f\| < mv_{\epsilon_1})$	($\forall$E-F L_{24})
L_{38}.	Lim_g	$\vdash 0 < mv_{\epsilon_2} \Rightarrow \exists \delta_2 \centerdot (0 < \delta_2 \wedge \forall x_2 \centerdot (\|x_2 - a\| < \delta_2 \wedge \|x_2 - a\| > 0 \Rightarrow \|g(x_2) - l_g\| < mv_{\epsilon_2}))$	($\forall$E-F L_3)
L_{39}.	$\mathcal{H}_3$	$\vdash 0 < mv_{\epsilon_2}$	(TellCS-B)
L_{41}.	$\mathcal{H}_3$	$\vdash \exists \delta_2 \centerdot (0 < \delta_2 \wedge \forall x_2 \centerdot (\|x_2 - a\| < \delta_2 \wedge \|x_2 - a\| > 0 \Rightarrow \|g(x_2) - l_g\| < mv_{\epsilon_2}))$	($\Rightarrow_E$ L_{39} L_{38})
L_{42}.	L_{42}	$\vdash 0 < c_{\delta_2} \wedge \forall x_2 \centerdot (\|x_2 - a\| < c_{\delta_2} \wedge \|x_2 - a\| > 0 \Rightarrow \|g(x_2) - l_g\| < mv_{\epsilon_2})$	(Hyp)
L_{44}.	L_{42}	$\vdash 0 < c_{\delta_2}$	($\wedge$E-F L_{42})
L_{45}.	L_{42}	$\vdash \forall x_2 \centerdot (\|x_2 - a\| < c_{\delta_2} \wedge \|x_2 - a\| > 0 \Rightarrow \|g(x_2) - l_g\| < mv_{\epsilon_2})$	($\wedge$E-F L_{42})
L_{46}.	L_{42}	$\vdash \|mv_{x_2} - a\| < c_{\delta_2} \wedge \|mv_{x_2} - a\| > 0 \Rightarrow \|g(mv_{x_2}) - l_g\| < mv_{\epsilon_2}))$	($\forall$E-F L_{45})
L_{11}.	L_{11}	$\vdash \|c_x - a\| > 0 \wedge \|c_x - a\| < mv_\delta$	(Hyp)
L_{14}.	L_{11}	$\vdash \|c_x - a\| > 0$	($\wedge$E-F L_{11})
L_{13}.	L_{11}	$\vdash \|c_x - a\| < mv_\delta$	($\wedge$E-F L_{11})
L_5.	L_5	$\vdash 0 < c_\epsilon$	(Hyp)
L_{61}.	$\mathcal{H}_1$	$\vdash 0 \leq 0$	(AskCS-B)
L_{59}.	$\mathcal{H}_1$	$\vdash mv_\delta \leq c_{\delta_1}$	(TellCS-B)
L_{57}.	$\mathcal{H}_2$	$\vdash 0 \leq 0$	(AskCS-B)
L_{55}.	$\mathcal{H}_2$	$\vdash mv_\delta \leq c_{\delta_2}$	(TellCS-B)
L_{52}.	$\mathcal{H}_2$	$\vdash mv_{x_2} = c_x$	(TellCS-B)

L_{53}.	$\mathcal{H}_2$	$\vdash mv_{\epsilon_2} \leq \frac{1}{2} * c_\epsilon$	(TellCS-B)
L_{50}.	$\mathcal{H}_2$	$\vdash \lvert mv_{x_2} - a\rvert < c_{\delta_2}$	(Solve*-B L_{13} L_{55})
L_{51}.	$\mathcal{H}_2$	$\vdash \lvert mv_{x_2} - a\rvert > 0$	(Solve*-B L_{14} L_{57})
L_{47}.	$\mathcal{H}_2$	$\vdash \lvert mv_{x_2} - a\rvert < c_{\delta_2} \wedge \lvert mv_{x_2} - a\rvert > 0$	($\wedge$I-B L_{50} L_{51})
L_{49}.	$\mathcal{H}_2$	$\vdash \lvert g(mv_{x_2}) - l_g\rvert < mv_{\epsilon_2}$	($\Rightarrow_E$ L_{47} L_{46})
L_{48}.	$\mathcal{H}_2$	$\vdash \lvert g(c_x) - l_g\rvert < \frac{1}{2} * c_\epsilon$	(Solve*-B L_{49} L_{52} L_{53})
L_{43}.	$\mathcal{H}_2$	$\vdash \lvert g(c_x) - l_g\rvert < \frac{1}{2} * c_\epsilon$	($\Rightarrow$E-F L_{47} L_{46} L_{48})
L_{40}.	$\mathcal{H}_1$	$\vdash \lvert g(c_x) - l_g\rvert < \frac{1}{2} * c_\epsilon$	($\exists$E-F L_{41} L_{43})
L_{37}.	$\mathcal{H}_1$	$\vdash \lvert g(c_x) - l_g\rvert < \frac{1}{2} * c_\epsilon$	($\Rightarrow$E-F L_{39} L_{38} L_{40})
L_{31}.	$\mathcal{H}_1$	$\vdash \lvert 1\rvert \leq mv$	(TellCS-B)
L_{32}.	$\mathcal{H}_1$	$\vdash mv_{\epsilon_1} \leq \frac{c_\epsilon}{2*mv}$	(TellCS-B)
L_{33}.	$\mathcal{H}_1$	$\vdash \lvert g(c_x) - l_g\rvert < \frac{c_\epsilon}{2}$	(Simplify-B L_{37})
L_{34}.	$\mathcal{H}_1$	$\vdash 0 < mv$	(TellCS-B)
L_{35}.	$\mathcal{H}_1$	$\vdash mv_{x_1} = c_x$	(TellCS-B)
L_{29}.	$\mathcal{H}_1$	$\vdash \lvert mv_{x_1} - a\rvert < c_{\delta_1}$	(Solve*-B L_{13} L_{59})
L_{30}.	$\mathcal{H}_1$	$\vdash \lvert mv_{x_1} - a\rvert > 0$	(Solve*-B L_{14} L_{61})
L_{26}.	$\mathcal{H}_1$	$\vdash \lvert mv_{x_1} - a\rvert < c_{\delta_1} \wedge \lvert mv_{x_1} - a\rvert > 0$	($\wedge$I-B L_{29} L_{30})
L_{28}.	$\mathcal{H}_1$	$\vdash \lvert f(mv_{x_1}) - l_f\rvert < mv_{\epsilon_1}$	($\Rightarrow_E$ L_{26} L_{25})
L_{27}.	$\mathcal{H}_1$	$\vdash \lvert((f(c_x) + g(c_x)) - l_f) - l_g\rvert < c_\epsilon$	(ComplexEstimate-B L_{28} L_{31} L_{32} L_{33} L_{34} L_{35})
L_{22}.	$\mathcal{H}_1$	$\vdash \lvert((f(c_x) + g(c_x)) - l_f) - l_g\rvert < c_\epsilon$	($\Rightarrow$E-F L_{26} L_{25} L_{27})
L_{19}.	$\mathcal{H}_3$	$\vdash \lvert((f(c_x) + g(c_x)) - l_f) - l_g\rvert < c_\epsilon$	($\exists$E-F L_{20} L_{21})
L_{16}.	$\mathcal{H}_3$	$\vdash \lvert((f(c_x) + g(c_x)) - l_f) - l_g\rvert < c_\epsilon$	($\Rightarrow$E-F L_{18} L_{17} L_{19})
L_{12}.	$\mathcal{H}_3$	$\vdash \lvert(f(c_x) + g(c_x)) - (l_f + l_g)\rvert < c_\epsilon$	(Simplify-B L_{16})
L_{10}.	$\mathcal{H}_4$	$\vdash \lvert c_x - a\rvert < mv_\delta \wedge \lvert c_x - a\rvert > 0 \Rightarrow \lvert(f(c_x) + g(c_x)) - (l_f + l_g)\rvert < c_\epsilon$	($\Rightarrow$I-B L_{12})
L_9.	$\mathcal{H}_4$	$\vdash \forall x\_(\lvert x - a\rvert < mv_\delta \wedge \lvert x - a\rvert > 0 \Rightarrow \lvert(f(x) + g(x)) - (l_f + l_g)\rvert < c_\epsilon)$	($\forall$I-B L_{10})
L_8.	$\mathcal{H}_4$	$\vdash 0 < mv_\delta$	(TellCS-B)
L_7.	$\mathcal{H}_4$	$\vdash 0 < mv_\delta \wedge \forall x\_(\lvert x-a\rvert < mv_\delta \wedge \lvert x-a\rvert > 0 \Rightarrow \lvert(f(x) + g(x)) - (l_f + l_g)\rvert < c_\epsilon)$	($\wedge$I-B L_8 L_9)
L_6.	$\mathcal{H}_4$	$\vdash \exists \delta\_(0 < \delta \wedge \forall x\_(\lvert x-a\rvert < \delta \wedge \lvert x-a\rvert > 0 \Rightarrow \lvert(f(x) + g(x)) - (l_f + l_g)\rvert < c_\epsilon))$	($\exists$I-B L_7)
L_4.	Lim_f, Lim_g	$\vdash 0 < c_\epsilon \Rightarrow \exists \delta\_(0 < \delta \wedge \forall x\_(\lvert x - a\rvert < \delta \wedge \lvert x - a\rvert > 0 \Rightarrow \lvert(f(x) + g(x)) - (l_f + l_g)\rvert < c_\epsilon))$	($\Rightarrow$I-B L_6)
L_1.	Lim_f, Lim_g	$\vdash \forall \epsilon\_(0 < \epsilon \Rightarrow \exists \delta\_(0 < \delta \wedge \forall x\_(\lvert x - a\rvert < \delta \wedge \lvert x - a\rvert > 0 \Rightarrow \lvert(f(x) + g(x)) - (l_f + l_g)\rvert < \epsilon)))$	($\forall$I-B L_4)
LIM+.	Lim_f, Lim_g	$\vdash \lim_{x \to a}(f(x) + g(x)) = l_f + l_g$	(DefnUnfold-B L_1)

$\mathcal{H}_1 = \{Lim_f, Lim_g, L_5, L_{11}, L_{21}\}$, $\mathcal{H}_2 = \{Lim_f, Lim_g, L_5, L_{11}, L_{21}, L_{42}\}$

$\mathcal{H}_3 = \{Lim_f, Lim_g, L_5, L_{11}\}$, $\mathcal{H}_4 = \{Lim_f, Lim_g, L_5\}$

B Limit Theorems

The following theorems from the limit domain can be proved by MULTI so far. We tested mainly conjectures from [2]. Many similar theorems could be formulated. In the following, X, Y denote sequences over the reals, f and g denote functions over the reals, and a, b denote arbitrary but fix reals. For problems marked with $(*)$ $\mathcal{CoSIE}$ fails to compute instantiations for meta-variables for the reasons discussed in section 5.3.

Limits of sequences

1. (Exercise 3.1.7 first part in [2])
   If the sequence $|X| = |(x_n)|$ has the limit 0, then the sequence $X = (x_n)$ has also the limit 0:
   $\mathit{limseq}\ |X| = 0 \Rightarrow \mathit{limseq}\ X = 0$

2. (Theorem 3.2.2 in [2])
   If the sequence $X = (x_n)$ has an limit l, then the sequence X is bounded:
   $\mathit{limseq}\ X = l \Rightarrow \exists m.\, 0 < m \wedge \forall n.\, |x_n| < m$

3. (Theorem 3.2.3.a first part in [2])
   If the sequence $X = (x_n)$ has the limit l_x and the sequence $Y = (y_n)$ has the limit l_y, then the sequence $X + Y = (x_n + y_n)$ has the limit $l_x + l_y$:
   $\mathit{limseq}\ X = l_x \wedge \mathit{limseq}\ Y = l_y \Rightarrow \mathit{limseq}\ X + Y = l_x + l_y$

4. (Theorem 3.2.3.a second part in [2])
   If the sequence $X = (x_n)$ has the limit l_x and the sequence $Y = (y_n)$ has the limit l_y, then the sequence $X - Y = (x_n - y_n)$ has the limit $l_x - l_y$:
   $\mathit{limseq}\ X = l_x \wedge \mathit{limseq}\ Y = l_y \Rightarrow \mathit{limseq}\ X - Y = l_x - l_y$

5. (Theorem 3.2.3.a third part in [2])
   If the sequence $X = (x_n)$ has the limit l_x and the sequence $Y = (y_n)$ has the limit l_y, then the sequence $X * Y = (x_n * y_n)$ has the limit $l_x * l_y$:
   $\mathit{limseq}\ X = l_x \wedge \mathit{limseq}\ Y = l_y \Rightarrow \mathit{limseq}\ X * Y = l_x * l_y$

6. (Theorem 3.2.3.a fourth part in [2])
   If the sequence $X = (x_n)$ has the limit l_x, then the sequence $a * X = (a * x_n)$ has the limit $a * l_x$:
   $\mathit{limseq}\ X = l_x \Rightarrow \mathit{limseq}\ a * X = a * l_x$

7. $(*)$(Theorem 3.2.3.b in [2])
   If the sequence $X = (x_n)$ has the limit l_x and the sequence $Y = (y_n)$ has the limit $l_y \neq 0$ and $y_n \neq 0$ for all n, then the sequence $\frac{X}{Y} = (\frac{x_n}{y_n})$ has the limit $\frac{l_x}{l_y}$:
   $\mathit{limseq}\ X = l_x \wedge \mathit{limseq}\ Y = l_y \wedge \forall n.\, y_n \neq 0 \Rightarrow \mathit{limseq}\ \frac{X}{Y} = \frac{l_x}{l_y}$

8. (Theorem 3.2.4 in [2])
   If the sequence $X = (x_n)$ has a limit l and $x_n \geq 0$ for all n, then $l \geq 0$:
   $\mathit{limseq}\ X = l \wedge \forall n.\, x_n \geq 0 \Rightarrow l \geq 0$

9. (Theorem 3.2.5 in [2])
If the sequence $X = (x_n)$ has a limit l_x and the sequence $Y = (y_n)$ has a limit l_y and $x_n \leq y_n$ for all n, then $l_x \leq l_y$:
$\mathit{limseq}\ X = l_x \wedge \mathit{limseq}\ Y = l_y \wedge \forall n.\, x_n \leq y_n \Rightarrow l_x \leq l_y$

10. (Theorem 3.2.6 in [2])
If the sequence $X = (x_n)$ has a limit l and $a \leq x_n \leq b$ for all n, then $a \leq l \leq b$:
$\mathit{limseq}\ X = l \wedge \forall n.\, a \leq x_n \leq b \Rightarrow a \leq l \leq b$

Limits of functions

1. (LIMC: Example 4.1.7.a in [2])
The function $f(x) = b$ has the limit b at a:
$\lim\limits_{x \to a} b = b$

2. (LIMV: Example 4.1.7.b in [2])
The function $f(x) = x$ has the limit a at a:
$\lim\limits_{x \to a} x = a$

3. (Example 4.1.7.c in [2])
The function $f(x) = x^2$ has the limit a^2 at a:
$\lim\limits_{x \to a} x^2 = a^2$

4. (*) (LIM-DIV-1-X: Example 4.1.7.d in [2])
The function $f(x) = \frac{1}{x}$ has the limit $\frac{1}{a}$ at a, if $a > 0$:
$a > 0 \Rightarrow \lim\limits_{x \to a} \frac{1}{x} = \frac{1}{a}$

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

6. (Exercise 4.1.2 first part in [2])
If f has limit l at a, then the function $|f(x) - l|$ has the limit 0 at a:
$\lim\limits_{x \to a} f(x) = l \Rightarrow \lim\limits_{x \to a} |f(x) - l| = 0$

7. (Exercise 4.1.2 second part in [2])
If the function $|f(x) - l|$ has the limit 0 at a, then f has the limit l at a:
$\lim\limits_{x \to a} |f(x) - l| = 0 \Rightarrow \lim\limits_{x \to a} f(x) = l$

8. (Exercise 4.1.3 first part in [2])
If the function $f(x)$ has the limit l at a, then the function $f(x+a)$ has the limit l at 0:
$\lim\limits_{x \to a} f(x) = l \Rightarrow \lim\limits_{x \to 0} f(x+a) = l$

9. (Exercise 4.1.3 second part in [2])
If the function $f(x+a)$ has the limit l at 0, then the function $f(x)$ has the limit l at a:
$\lim\limits_{x \to 0} f(x+a) = l \Rightarrow \lim\limits_{x \to a} f(x) = l$

10. (Exercise 4.1.7 in [2])
If $k > 0$ and $|f(x) - l| \leq k * |x - a|$ for all x, then f has the limit l at a:
$k > 0 \wedge \forall x. |f(x) - l| \leq k * |x - a| \Rightarrow \lim\limits_{x \to a} f(x) = l$

11. (Exercise 4.1.8 in [2])
$\lim\limits_{x \to a} x^3 = a^3$

12. (∗) (Exercise 4.1.10.a in [2])
$\lim\limits_{x \to 2} \frac{1}{1-x} = -1$

13. (∗) (Exercise 4.1.10.b in [2])
$\lim\limits_{x \to 1} \frac{x}{1+x} = \frac{1}{2}$

14. (∗) (Exercise 4.1.10.c in [2])
$\lim\limits_{x \to 0} \frac{x^2}{|x|} = 0$

15. (∗) (Exercise 4.1.10.d in [2])
$\lim\limits_{x \to 1} \frac{x^2-x+1}{x+1} = \frac{1}{2}$

16. (Exercise 4.1.12 in [2])
If $f(x)$ has limit l at 0 and $a > 0$, then $f(a * x)$ has the limit l at 0:
$\lim\limits_{x \to 0} f(x) = l \wedge a > 0 \Rightarrow \lim\limits_{x \to 0} f(a * x) = l$

17. (Reverse of exercise 4.1.12)
If $f(a * x)$ has the limit l at 0 and $a > 0$, then $f(x)$ has limit l at 0:
$\lim\limits_{x \to 0} f(a * x) = l \wedge a > 0 \Rightarrow \lim\limits_{x \to 0} f(x) = l$

18. (Theorem 4.2.2 in [2])
If f has a limit at a, then f is bounded in a neighborhood of a:
$$\lim_{x \to a} f(x) = l \Rightarrow \exists m, \delta. m > 0 \wedge \delta > 0 \wedge \forall x. (|x - a| < \delta \wedge |x - a| > 0) \Rightarrow |f(x)| < m$$

19. (LIM+: Theorem 4.2.4.a first part in [2])
If f has limit l_f at a and g has limit l_g at a, then $f + g$ has limit $l_f + l_g$ at a:
$\lim\limits_{x \to a} f(x) = l_f \wedge \lim\limits_{x \to a} g(x) = l_g \Rightarrow \lim\limits_{x \to a} f(x) + g(x) = l_f + l_g$

20. (LIM-: Theorem 4.2.4.a second part in [2])
If f has limit l_f at a and g has limit l_g at a, then $f - g$ has limit $l_f - l_g$ at a:
$\lim\limits_{x \to a} f(x) = l_f \wedge \lim\limits_{x \to a} g(x) = l_g \Rightarrow \lim\limits_{x \to a} f(x) - g(x) = l_f - l_g$

21. (LIM*: Theorem 4.2.4.a third part in [2])
If f has limit l_f at a and g has limit l_g at a, then $f * g$ has limit $l_f * l_g$ at a:
$\lim\limits_{x \to a} f(x) = l_f \wedge \lim\limits_{x \to a} g(x) = l_g \Rightarrow \lim\limits_{x \to a} f(x) * g(x) = l_f * l_g$

22. (Theorem 4.2.4.a fourth part in [2])
If f has limit l_f at a, then $a * f$ has limit $a * l_f$ at a:
$\lim\limits_{x \to a} f(x) = l_f \Rightarrow \lim\limits_{x \to a} a * f(x) = a * l_f$

23. (∗) (Theorem 4.2.4.b in [2])
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:
$\lim_{x\to a} f(x) = l_f \wedge \lim_{x\to a} g(x) = l_g \wedge \forall x. g(x) \neq 0 \Rightarrow \lim_{x\to a} \frac{f(x)}{g(x)} = \frac{l_f}{l_g}$

24. (Example 4.2.5.b in [2])
$\lim_{x\to 2}(x^2+1) * (x^3-4) = 20$

25. (Example 4.2.8.b in [2])
$\lim_{x\to 0} sin(x) = 0$

26. (Example 4.2.8.c in [2])
$\lim_{x\to 0} cos(x) = 1$

27. (Example 4.2.8.f in [2])
$\lim_{x\to 0} x * sin(\frac{1}{x}) = 0$

28. (Exercise 4.2.1 in [2])
$\lim_{x\to 1}(x+1) * (2 * x + 3) = 10$

29. (Theorem 4.3.3 first part in [2])
If f has limit l at a, then f has the left-hand limit l at a:
$\lim_{x\to a} f(x) = l \Rightarrow limL_{x\to a} f(x) = l$

30. (Theorem 4.3.3 second part in [2])
If f has limit l at a, then f has the right-hand limit l at a:
$\lim_{x\to a} f(x) = l \Rightarrow limR_{x\to a} f(x) = l$

31. (Lim-If-Both-Sides-Lim: Theorem 4.3.3 third part in [2])
If f has the left-hand limit l and the right-hand limit l at a, then f has the limit l at a:
$limL_{x\to a} f(x) = l \wedge limR_{x\to a} f(x) = l \Rightarrow \lim_{x\to a} f(x) = l$

Continuity of functions

1. (Example 5.1.5.a in [2])
The function $f(x) = b$ is continuous at a:
$cont(b, a)$

2. (Example 5.1.5.b in [2])
The function $f(x) = x$ is continuous at a:
$cont(x, a)$

3. (Example 5.1.5.b in [2])
The function $f(x) = x^2$ is continuous at a:
$cont(x^2, a)$

4. (Exercise 5.1.6 in [2])
   If f is continuous at a, then for any $\epsilon > 0$ there exists a δ-neighborhood of a such that if x, y in this δ-neighborhood then $|f(x) - f(y)| < \epsilon$:
   $$cont(f,a) \Rightarrow \forall \epsilon.\,(\epsilon > 0 \Rightarrow \exists \delta.\,(\delta > 0 \wedge \forall x, y.\,(|x-a| < \delta \wedge |y-a| < \delta \Rightarrow |f(x) - f(y)| < \epsilon)))$$

5. (Exercise 5.1.11 in [2])
   If $k > 0$ and $|f(x) - f(y)| \leq k * |x - y|$ for all x,y, then f is continuous at a:
   $k > 0 \wedge \forall x, y.\,|f(x) - f(y)| \leq k * |x - y| \Rightarrow cont(f,a)$

6. (Continuous+: Theorem 5.2.1.a first part in [2])
   If f is continuous at a and g is continuous at a, then $f + g$ is continuous at a:
   $cont(f,a) \wedge cont(g,a) \Rightarrow cont(f+g,a)$

7. (Continuous-: Theorem 5.2.1.a second part in [2])
   If f is continuous at a and g is continuous at a, then $f - g$ is continuous at a:
   $cont(f,a) \wedge cont(g,a) \Rightarrow cont(f-g,a)$

8. (Continuous*: Theorem 5.2.1.a third part in [2])
   If f is continuous at a and g is continuous at a, then $f * g$ is continuous at a:
   $cont(f,a) \wedge cont(g,a) \Rightarrow cont(f*g,a)$

9. (Theorem 5.2.1.a fourth part in [2])
   If f is continuous at a, then $a * f$ is continuous at a:
   $cont(f,a) \Rightarrow cont(a*f,a)$

10. ($*$) (Theorem 5.2.1.b in [2])
    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:
    $cont(f,a) \wedge cont(g,a) \wedge \forall x.\,g(x) \neq 0 \Rightarrow cont(\frac{f}{g},a)$

11. (Theorem 5.2.7 in [2])
    If f is continuous at a and g is continuous at $f(a)$, then the composition $g \circ f$ is continuous at a:
    $cont(f,a) \wedge cont(g,f(a)) \Rightarrow cont(g \circ f,a)$

12. (Exercise 5.2.6 in [2])
    If f has the limit l at a and g is continuous at l, then the composition $g \circ f$ has the limit $g(l)$ at a:
    $\lim_{x \to a} f(x) = l \wedge cont(g,l) \Rightarrow \lim_{x \to a} g(f(x)) = g(l)$

13. (Cont-If-Lim=f)
    If f has the limit $f(a)$ at a, then f is continuous at a:
    $\lim_{x \to a} f(x) = f(a) \Rightarrow cont(f,a)$

Derivatives of functions

1. $(*)$ (Theorem 6.1.3.a in [2])
   If f has the derivative f' at a, then $a * f$ has the derivative $a * f'$ at a:
   $deriv(f, a) = f' \Rightarrow deriv(a * f, a) = a * f'$

2. $(*)$ (Theorem 6.1.3.b in [2])
   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:
   $deriv(f, a) = f' \wedge deriv(g, a) = g' \Rightarrow deriv(f + g, a) = f' + g'$

3. $(*)$ (Theorem 6.1.3.c in [2])
   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:
   $deriv(f, a) = f' \wedge deriv(g, a) = g' \Rightarrow deriv(f * g, a) = f' * g(a) + f(a) * g'$

4. $(*)$ (Cont-If-Deriv: Theorem 6.1.2 in [2])
   If f has a derivative at a, then f is continuous at a:
   $deriv(f, a) = f' \Rightarrow cont(f, a)$

References

[1] P.B. Andrews. Transforming Matings into Natural Deduction Proofs. In Bibel and Kowalski [4], pages 281–292.

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

[3] M. Beeson. Automatic generation of epsilon-delta proofs of continuity. In J. Calment and J. Plaza, editors, *Artificial Intelligence and Symbolic Computation*, pages 67–83. Springer VerlagGermany, 1998.

[4] W. Bibel and R.A. Kowalski, editors. *Proceedings of the 5th Conference on Automated Deduction (CADE–5)*, volume 87 of *LNCS*, Les ArcsFrance, June7–9 1980. Springer VerlagGermany.

[5] K.H. Bläsius and H.J. Bürckert, editors. *Deduktionssysteme.* Oldenbourg, 1992.

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

[7] W.W. Bledsoe, R.S. Boyer, and W.H. Henneman. Computer Proofs of Limit Theorems. *Artificial Intelligence*, 3(1):27–60, 1972.

[8] W.W. Bledsoe and P. Bruell. A Man-Machine Theorem Proving System. *Artificial Intelligence*, 5(1):51–72, 1974.

[9] W.W. Bledsoe and L. Hines. Variable Elimination and Chaining in a Resolution-Based Prover for Inequalities. In Bibel and Kowalski [4], pages 70 – 87.

[10] A. Bundy. The Use of Explicit Plans to Guide Inductive Proofs. In E.L. Lusk and R.A. Overbeek, editors, *Proceedings of the 9th International Conference on Automated Deduction (CADE–9)*, volume 310 of *LNCS*, pages 111–120, Argonne, Illinois, USA, 1988. Springer VerlagGermany.

[11] A. Bundy. A science of reasoning. In *Computational Logic: Essays in Honor of Alan Robinson.* 1991.

[12] A. Bundy. A Critique of Proof Planning. In *Festschrift in Honour of Robert Kowalski.* 2002.

[13] A. Bundy, A. Stevens, F. van Hermelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, 62:185–253, 1993.

[14] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam System. In Stickel [44], pages 647–648.

[15] L. Cheikhrouhou and V. Sorge. $\mathcal{PDS}$ — A Three-Dimensional Data Structure for Proof Plans. In *Proceedings of the International Conference on Artificial and Computational Intelligence for Decision, Control and Automation in Engineering and Industrial Applications (ACIDCA'2000)*, Monastir, Tunisia, March22–24 2000.

[16] D.D. Corkill, V.R. Lesser, and E. Hudlicka. Unifying Data-Directed and Goal-Directed Control. In D. Waltz, editor, *Proceedings of the Second National Conference on Artificial Intelligence (AAAI-82)*, pages 143 – 147, Carnegie-Mellon University / University of Pittsburgh, Pittsburgh, Pennsylvania, USA, August18–20 1982. AAAI Press, Menlo Park, CA, USA.

[17] E.H. Durfee and V.R. Lesser. Incremental Planning to Control a Blackboard-Based Problem Solver. In T. Kehler and S. Rosenschein, editors, *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 58 – 64, Philadelphia, Pennsylvania, USA, August11–15 1986. AAAI Press, Menlo Park, CA, USA.

[18] L.D. Erman, P. London, and S. Fickas. The Design and an Example Use of HEARSAY-III. In B. Buchanan, editor, *Proceedings of the 6th International Joint Conference on Artificial Intelligence (ICJAI)*, pages 409–415, Tokyo, Japan, August20–23 1979. Morgan Kaufmann.

[19] G. Gentzen. Untersuchungen über das Logische Schließen I und II. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935.

[20] F. Giunchiglia and T. Walsh. Theorem Proving with Definition. In *Proceedings of AISB 89, Society for the Study of Artificial Intelligence and Simulation of Behaviour*, 1989.

[21] B. Hayes-Roth. A Blackboard Architecture for Control. *Artificial Intelligence*, 25:251–321, 1985.

[22] G.P. Huet. *Constrained Resolution: A Complete Method for Higher Order Logic.* PhD thesis, Case Western Reverse University, 1972.

[23] D. Hutter. Guiding inductive proofs. In Stickel [44].

[24] A. Ireland. The Use of Planning Critics in Mechanizing Inductive Proofs. In A. Voronkov, editor, *Proceedings of the 3rd International Conference on Logic Programming and Automated Reasoning (LPAR'92)*, volume 624 of *LNAI*, pages 178 – 189, St. Petersburg, Russia, July1992. Springer VerlagGermany.

[25] A. Ireland and A. Bundy. Productive Use of Failure in Inductive Proof. *Journal of Automated Reasoning*, 16(1-2):79–111, 1996.

[26] M.V. Johnson Jr. and B. Hayes-Roth. Integrating Diverse Reasoning Methods in the BB1 Blackboard Control Architecture. In K. Forbus and H. Shrobe, editors, *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 30 – 35, Seattle, Washington, USA, July13–17 1987. AAAI Press, Menlo Park, CA, USA.

[27] M. Kerber, M. Kohlhase, and V. Sorge. Integrating Computer Algebra Into Proof Planning. *Journal of Automated Reasoning*, 21(3):327–355, 1998.

[28] H. Kirchner and C. Ringeissen, editors. *Proceedings of Third International Workshop on Frontiers of Combining Systems (FROCOS 2000)*, volume 1794 of *LNCS*, NancyFrance, March22–24 2000. Springer VerlagGermany.

[29] W. McCune. Otter 3.0 Reference Manual and Guide. Technical Report ANL-94-6, Argonne National Laboratory, Argonne, Illinois 60439, USA, 1994.

[30] A. Meier. The proof planners of ΩMEGA: A technical description. Seki Report SR-2004-03, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 2004.

[31] E. Melis. Progress in proof planning: Planning limit theorems automatically. Seki Report SR-97-08, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 1997.

[32] E. Melis. AI-Techniques in Proof Planning. In H. Prade, editor, *Proceedings of of the 13th European Conference on Artifical Intelligence*, pages 494–498, Brighton, UK, August23–28 1998. John Wiley & Sons, Chichester, UK.

[33] E. Melis. The “Limit” Domain. In R. Simmons, M. Veloso, and S. Smith, editors, *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, pages 199–206, Pittsburgh, PEN, USA, June7–10 1998. AAAI Press, Menlo Park, CA, USA.

[34] E. Melis and A. Meier. Proof Planning with Multiple Strategies. In J. Loyd, V. Dahl, U. Furbach, M. Kerber, K. Lau, C. Palamidessi, L.M. Pereira, and Y. Sagivand P. Stuckey, editors, *First International Conference on Computational Logic (CL-2000)*, volume 1861 of *LNAI*, pages 644–659, London, UK, 2000. Springer-Verlag.

[35] E. Melis and M. Pollet. Domain Knowledge for Search Heuristics in Proof Planning. In *Proceedings of AIPS-2000 Workshop: Analyzing and Exploiting Domain Knowledge*, pages 12–15, 2000.

[36] E. Melis and J. Siekmann. Knowledge-Based Proof Planning. *Artificial Intelligence*, 115(1):65–105, 1999.

[37] E. Melis, J. Zimmer, and T. Müller. Integrating Constraint Solving into Proof Planning. In Kirchner and Ringeissen [28], pages 32–46.

[38] D. Redfern. *The Maple Handbook: Maple V Release 5*. Springer VerlagGermany, 1999.

[39] J.D.C. Richardson, A. Smaill, and I.M. Green. System description: Proof planning in higher-order logic with λ*Clam*. In C. Kirchner and H. Kirchner, editors, *Proceedings of the 15th International Conference on Automated Deduction (CADE–15)*, volume 1421 of *LNAI*, pages 129–133, LindauGermany, July5–10 1998. Springer VerlagGermany.

[40] S. Russell and P. Norvig. *Artificial Intelligence - A Modern Approach*. Prentice Hall, Englewood Cliffs, 1995.

[41] A.H. Schoenfeld. *Mathematical Problem Solving*. Academic Press, New York, 1985.

[42] J. Siekmann, 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.P. Wirth, and J. Zimmer. Proof Development with OMEGA. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated*

Deduction (CADE–18), number 2392 in LNAI, pages 144–149, Kopenhagen, Denmark, 2002. Springer VerlagGermany.

[43] V. Sorge. Non-Trivial Symbolic Computations in Proof Planning. In Kirchner and Ringeissen [28], pages 121–135.

[44] M. Stickel, editor. *Proceedings of the 10th International Conference on Automated Deduction (CADE–10)*, volume 449 of *LNAI*, KaiserslauternGermany, 1990.

[45] A. Tate. Generating Project Networks. In R. Reddy, editor, *Proceedings of the 5th International Joint Conference on Artificial Intelligence (ICJAI)*, pages 888–893, Cambridge, MA, USA, August22–25 1977. Morgan Kaufmann, San Mateo, CA, USA.

[46] Y. Tetsuya, A. Bundy, I. Green, T. Walsh, and D. Basin. Coloured rippling: An extension of a theorem proving heuristic. In A.G. Cohn, editor, *Proceedings of of the 11th European Conference on Artifical Intelligence*, pages 85 – 89. John Wiley & Sons, Chichester, UK, 1994.

[47] M.M. Veloso, J. Carbonell, M.A. Perez, D. Borrajo, E. Fink, and J. Blythe. Integrating Planning and Learning: The Prodigy Architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1):81–120, 1995.

[48] D.S. Weld. An Introduction to Least Commitment Planning. *AI Magazine*, 15(4):27–61, 1994.

[49] L. Wos. The Problem of Definition Expansion and Contraction. *Journal of Automated Reasoning*, 3:433–435, 1987.

[50] J. Zimmer. Constraintlösen für Beweisplanung. Master's thesis, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, 2000.