Analogy in $C^{\lambda}\!AM$[*]

Erica Melis [†]

Abstract

CIAM is a proof planner, developed by the Dream group in Edinburgh, that mainly operates for inductive proofs. This paper addresses the question how an analogy model that I developed independently of *CIAM* can be applied to *CIAM* and it presents analogy-driven proof plan construction as a control strategy of *CIAM*. This strategy is realized as a derivational analogy that includes the reformulation of proof plans. The analogical replay checks whether the reformulated justifications of the source plan methods hold in the target as a permission to transfer the method to the target plan. Since *CIAM* has very efficient heuristic search strategies, the main purpose of the analogy is to suggest lemmas, to replay not commonly loaded methods, to suggest induction variables and induction terms, and to override control rather than to construct a target proof plan that can be built by *CIAM* itself more efficiently.

1 Introduction

Analogy is a heuristic problem solving strategy that guides the problem solving of a target problem, which is similar to a source problem, by using the source problem solving. There are at least two different paradigms for analogy: derivational analogy [Carbonell 86] and transformational analogy [Carbonell 83]. *Derivational* analogy guides the target solution by replaying decisions of the source problem solving *process*, and it uses information available during this process only, e.g., the justifications for the decisions made. *Transformational* analogy *transforms* only the final source solution to satisfy the constraints of the target problem.

The analogy-driven proof plan construction in [Melis 95] was originally developed for the Ω-MKRP environment [Huang *et al* 94a, Huang *et al* 94b] with its method

*This research was supported by HC&M grant CHBICT930806.

†Department of Artificial Intelligence, University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN; Email:melis@cs.uni-sb.de, on leave from University of Saarbrücken, Germany

data structures and planner. Now a question was whether this analogy model can be applied in another proof planner as well and which of its purposes and techniques are dependent on the actual planner.

Because of the huge search space in general (proof) planning, usually the main purpose of analogy is to guide the search for methods. *CIAM*'s search for methods, however, is quite restricted already because inductive theorem proving is a rather goal-directed type of mathematical reasoning and *CIAM* uses efficient search heuristics such as rippling. Therefore analogy might be helpful or efficiently used only if *CIAM*'s proof planning breaks down. Hence, the focus of applying analogy in *CIAM* will be different, although we shall use the model of theorem proving by analogy in [Melis 95].

This paper is organised as follows. First, I briefly review the proof planning in *CIAM* and the model of analogy-driven proof plan construction. In section 4 the purposes of analogy in *CIAM* are discussed. In section 5.1 the matchings underlying the reformulations and the reformulations themselves are described. In section 5.3 the analogical replay is introduced along with the justifications of *CIAM* methods. A comprehensive appendix contains examples referred to throughout the whole paper.

I assume the reader to be familiar with proof planning and notations of *CIAM*. For a comprehensive introduction see [Bundy *et al* 91a].

2 Proof Planning in *CIAM*

A proof plan is an abstract representation of a proof that consists of methods corresponding to tactics. A tactic executes a number of logical inferences. In the proof planning context a *method* is a (partial) specification of an available tactic [Gordon *et al* 79]. Proof planning exploits the information encoded in methods to build tactics tailored for the goal at hand. Proof planning constructs proof plans that are trees[1] of method nodes which are connected by sequents, called goals[2]. The execution of a complete plan, i.e. without open goals, yields a proof of the top goal.

Proof planning has been introduced in [Bundy 88] in order to restrict the search in theorem proving. Proof planning has been applied successfully to inductive theorem proving by the Edinburgh proof planner *CIAM* [Bundy *et al* 91b]. This required to analyze families of inductive proofs and to identify common patterns in them which provide proof plans that can guide future proofs from the same family.

Fortunately, inductive theorem proving is itself goal-directed and inductive proofs reveal a certain pattern. Furthermore, strong domain-specific heuristics have been

[1] or more generally, forests

[2] in backward planning

discovered for inductive proofs, such as rippling [Hutter 90, Bundy *et al* 93], that restrict the search for methods. Rippling systematically uses so-called *wave-rules* to eliminate differences between the induction hypothesis and the induction conclusion. To that end it considers goal expressions which are annotated by so-called *wave-fronts* (indicated by boxes), i.e. terms that appear in the induction conclusion but not in the induction hypothesis. So-called wave-holes are indicated by underlining. If, for instance, the induction hypothesis is

$$(x + y) + z = x + (y + z),$$

then the induction conclusion is

$$(\boxed{s(\underline{x})} + y) + z = \boxed{s(\underline{x})} + (y + z).$$

If we remove the structure of the not underlined parts of the boxes, then we obtain the *skeleton* which equals the induction hypothesis. Rippling systematically applies wave-rules that move or remove the wave-fronts in order to obtain an expression that matches the induction hypothesis. Wave-rules are rewrites with the same kind of annotation that encode terminating annotated rewriting.

The methods of *CIAM* have the form (see [vanHarmelen *et al* 93])

```
method(name(...Args...),          % name slot: Prolog term
       H==>G,                     % input slot: sequent,
                                    H hypotheses, G goal
       [...Preconditions...],     % precondition-slot: list of conjuncts
       [...Postconditions...],    % postcondition-slot: list of conjuncts
       [...Outputs...],           % output slot: list of sequents
       tactic(...Args...)         % tactic slot: Prolog term
```

Version3 of *CIAM*, to which I refer throughout this paper, works with the following methods:

- ELEMENTARY applies if current method is easily provable using some propositional and equational reasoning, and very elementary facts
- EQUAL checks whether there is any equality Term=Var among the hypotheses. If so, it uses the equality to rewrite all hypotheses and the goal in a definite order @¡.
- EVAL_DEF symbolically evaluates a term in the goal by applying one of its defining equations.
- FERTILIZE uses the induction hypothesis to rewrite the current goal or match the current goal with the induction hypothesis.

- GENERALISE replaces a common subterm in both halfs of an equality, or implication, or inequality by a new variable.
- NORMAL removes the implication from an implicational goal and appends the antecedens to the hypotheses.
- WAVE finds a subexpression to which a (conditional) wave-rule applies, where the condition of the rule is required to hold already.
- INDUCTION chooses induction variables and an appropriate induction scheme. Its output are the subgoals for the base case and the step case (induction conclusion).

*CLAM*3 constructs a proof plan as a set of nodes which are data structures with the following slots:

- PlanName: the name of the conjecture being planned.
- Address: the address of the node within the plan tree.
- Hypotheses: the goal hypotheses.
- Conclusion: the goal conclusion.
- Status: the status of the goal sequent, *open* or *closed.*
- Subst: list of substitutions generated for meta variables in the goal sequent.
- Method: the method chosen by the planner at that node
- Preconds: successful and failed method preconditions tested at that node.
- Count: number of subgoals.

Backward proof planning starts with the conjecture as an open goal g. It searches for a method M applicable to an open (sub)goal g and introduces M into the proof plan. The subgoals g_i produced by the application of M become the new open subgoals whereas g gets the status "closed". The planning continues to search for a method applicable to one of the open goals. *CLAM* plans backward only.

3 Analogy-Driven Proof Plan Construction

The analogy in [Melis 95] works at the proof *plan* level and employs both the derivational and the transformational paradigm by reusing control knowledge (justifications) from the source planning process and by transforming the source proof plan via reformulations.

Some proof plans can be transferred even if the formal proofs cannot be transferred. For instance, an Ω-MKRP method calling an automated theorem prover can provide different formal proofs in the source in the target when executed. Similarly, the ELMENTARY method can yield different formal proofs for source and target when executed; and WAVE or EVAL_DEF may have to remove a different number of quantifiers which causes different formal proofs when the plan is executed. Even more obvious is the advantage of analogically replaying proof *plans* if a transfer of hierarchically superior methods is possible only and the transfer of details fails.

Reformulations do more than just instantiate variables. They are mappings of a proof plan to a proof plan[3]. Reformulations encode mathematical heuristics on how a proof plan changes depending on certain changes of the problem to be proved or of the proof assumptions. In general, reformulations may insert, replace, or delete methods, may change methods, sequents, and justifications of proof plan nodes. A reformulation can be triggered by the match obtained between a source (sub-)goal with a target (sub-)goal or between source assumptions and target assumptions.

The analogy-driven proof plan construction in [Melis 95] assumes that a source proof plan is given. It follows the parametrized source proof plan[4] in order to construct a target plan. The source plan is *linearised*, i.e., ordered by the sequence in which the nodes have been added to the source plan. As in [Veloso 94], *justifications*, that encode reasons for the decisions made, annotate the source plan nodes. These justifications capture the subgoaling structure (the tree structure) of a plan and point to reasons for the decisions, such as application conditions of a method, user-given guidance, hierarchical information, or pre-programmed control knowledge.

Analogy-driven proof plan construction works as a control strategy for proof planning and replays planning decisions of a reformulated source proof plan if their justifications are satisfied in the target (see Table 1). Otherwise, the analogical replay tries to establish certain justifications or interleaves analogy with to base-level planning. Base-level planning is proof planning that is not guided by analogy.

The analogical replay skips a source method if the status of the corresponding target goal g is closed rather than open, that is, if g is an instantiation of an assumption. Then the next source goal is considered, i.e., a source method that became redundant in the target is skipped. New subgoals can be introduced by the analogical replay to establish justifications and proof assumptions.

[3]Reformulations usually, but not necessarily, preserve the verifiability of methods in a plan. In the Ω-MKRP environment a method M is called verifiable if given the preconditions of M, the method yields a correct proof of the postconditions of M in case the constraints of M are satisfied. Note that the method slot names are different from those in *CL^AM*

[4]in which instantiated parameters are replaced by a certain kind of variables

```
input: linearised source plan, (open) target goal
output: target plan
  1. while there are open target goals do
  2. if source plan is exhausted, then base-level plan for the open goals.
  3. Get next (sub)goal P from source plan.
  4. if there is a reformulation $\rho$, such that $\rho$P matches an open target goal $g_T$
     for which the justifications hold, then
       • reformulate source plan by $\rho$ and link $g_T$ to source plan.
       • if $g_T$ is an open goal, then
           • Select from the reformulated source the relevant
             method M. Apply an appropriate reformulation
             triggered by the justifications of M.
           • Check M's justifications. If they hold, link M to the
             source plan and update open goals.
           • if a justification does not hold, then choose suitable
             action:
               • Try to establish the justifications by other
                 means.
               • Or base-level plan.
```

Table 1: Simplified analogy-driven proof plan construction

4 Purposes of Analogy in *CIAM*

What can analogy buy for inductive theorem proving with *CIAM*? I briefly discuss some advantages and refer to examples (written in `typewriter` font) given in the appendix of this paper. There `A-B` denotes the example with the source `A` and the target `B`. Roughly, inductive theorem proving by analogy can:

- provide a plan for a conjecture for which base-level planning does not succeed with a plan, e.g., for `div3term` (from `div3-div3term`) and `doublehalf` (from `halfdouble-doublehalf`), and for `times2right` (from `plus2right-times2right`),
- obtain a proof plan for the target theorem that is considerably shorter than a plan built by base-level planning, e.g., for `halfplus` from `comp-halfplus2`.

In more detail, analogy in *CIAM* offers the following features:

1. The analogy suggests an induction in the target case, i.e.,

   (a) the analogy can suggest induction variables and existentially quantified variables to be instantiated (*IE-variables*) (all examples),

(b) the analogy can suggest induction terms (all examples),

2. The analogy can suggest lemmas in the target case, e.g., in `assp-assm`, `halfplus2-lenapp`,
3. The analogy can avoid the application of critics as defined in [Ireland & Bundy 94]: The analogical replay can suggest lemmas and thus avoid the application of the lemma discovery critics in the target, e.g., for `lenapp-halfplus2`.
   Since the analogy can suggest an induction term, it can avoid the application of the induction revision critic in the target, e.g., in `halfplus2-lenapp`,
4. The analogy can guide non-inductive subproofs, e.g., in `div3-div3term`.
5. The analogy may suggest plan islands for proof planning which can then be tackled by reasoning backward from the next transferred subgoal. Example: `assp-assm` in case the distributivity axiom is not given for `assm`,
6. Analogy may find generalisations via analogy, e.g., for `assapp-assplus1`, where variables are generalised apart guided by analogy mapping constraints,
7. Analogy can suggest the application of a not commonly loaded method, e.g. NORMAL, if this method was loaded for the source planning and is transferred to the target, for instance, for `div3-div3term`,
8. Analogy can guide the application of a method even if a heuristic precondition does not hold as, e.g., for `zerotimes1-zerotimes3`. Analogy is a control strategy for planning that can overwrite heuristic preconditions.

5 Analogy in CL^AM

In order to use the analogy-driven proof plan construction for CL^AM we have to consider analogical transfer for backward planning situations only. This yields the simplification of the procedure already assumed in Table 1.

For analogy, the structure of the plan nodes, as presented above, has to be augmented by a justification slot containing different kinds of justifications which are explained in detail below: l-justifications, e-justifications, p-justifications, and IF-equations.

A somewhat elaborate analogy must allow for mappings that map a function symbol at different positions in the source theorem (or in the source rewrites) to different target functions (i.e., lambda terms)[5]. For this purpose, source function symbols at different positions are differentiated by additional indices. Because of

[5]Even more elaborate mappings are conceivable that map arbitrary functions (lambda terms), to functions

the rippling heuristic in *CIAM*, the functions that belong to the skeleton of a wave-rule have the same index on both sides of the wave-rule. Dealing with indexed function symbols while running *CIAM* yields additional information relevant for the replay of methods, namely a sequence of *IF-equations* of the form $f_i = f_j$. In the source planning process these IF-equations result from matching an expression from a goal with a rewrite or with a hypothesis, or from equalities used by the method ELEMENTARY. The IF-equations can be obtained by slightly extending the matching in *CIAM*. The IF-equations belong to the justifications of the respective method and restrict the analogical mappings.

*CIAM*3 does not work with hierarchical methods and, hence, hierarchy information will not be included into the justifications throughout this paper. Potentially, however, hierarchy information has to be part of the justifications if available.

After parametrizing the source plan nodes, the analogy in *CIAM* consists of the subprocesses

1. Find the best[6] match of the source conjecture with the target conjecture and of the source rewrites with target rewrites, restricted by heuristics[7].
2. Replay the source plan which includes to apply reformulations.
3. Base-level plan for any open goal.

These subprocesses are explained in the following sections. As discussed in section 5.2, certain matchings determine reformulations of the proof plan which go beyond a pure mapping. Reformulations of the source proof plan are an important part of the analogical replay. Here, I consider reformulations that change the proof plan but do not change single methods. I shall first discuss the matchings and their restrictions, then suggest corresponding reformulations, and finally present the analogical replay in *CIAM*.

5.1 Matching

First a basic mapping m_b is constructed that matches the source and the target conjecture. The basic mapping is then expanded by an extended mapping m_e of source with target rewrites.

5.1.1 Basic Matching

The theorem tree ts is the parse tree of the source theorem. A *ripple* tree ts_r of a source theorem is the subtree of its theorem tree ts that contains the successful

[6]"Best" will consider heuristics such as as many IF-equations as possible should be satisfied.

[7]Actually, the matching and derivational replay should be interleaved. The separation is for efficiency reasons but it may turn out not to work well.

rippling paths with all the induction variables and existentially quantified variables to be instantiated, jointly denoted as *IE-variables*. The example in Figure 1 shows the theorem trees of `lenapp` and `halfplus2` with the ripple paths as bold lines. In the figure the source IE-variable is x and the target IE-variable is x.

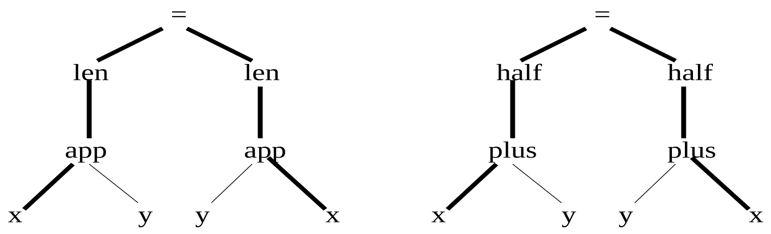

Figure 1: Parse trees of lenapp and halfplus2

A basic match establishes the basic mapping m_b in order to match the parametrized source theorem with the given target theorem. m_b has to be a second-order substitution because source function symbols can be mapped to target functions.

m_b maps the source theorem tree ts to the target theorem tree tt and thereby it maps the ripple-subtree ts_r of the source theorem tree ts to a subtree tt_r of the target theorem tree tt. In order to successfully transfer the methods of the source plan, in particular from the step case subplan, the rippling paths in the target tree should be similar to the rippling paths in the source tree. That is, tt_r should be a target ripple-subtree.

Labelled Fragments

Therefore we propose constraints for m_b that heuristically preserve the ripple paths. Such a heuristic makes the analogy dependent on the rippling used in the source proof and, thus, it is tailored for transferring the step case of a proof plan. We could buy more generality for less restriction of the mapping.

Since *labelled fragments*, introduced in [Hutter 94], heuristically determine the successful rippling paths we require m_b to preserve labelled fragments or to change labelled fragments in a controlled way that leads to similar successful rippling paths. Figure 2 displays labelled fragments of some functions. Labelled fragments provide an abstract representation of wave-rules and allow for reasoning about the whole ripple process. The labelled fragments of function/relation nodes in a theorem tree heuristically determine the ripple paths and the set of IE-variables. The ripple paths in a theorem tree abstractly encode the consecutive application of the wave-method relying on the wave-rules the labelled fragments are built from.

According to [Hutter 94], labelled fragments are rather insensitive to missing lemmas. This means that if a function f has a labelled fragment, constructed from

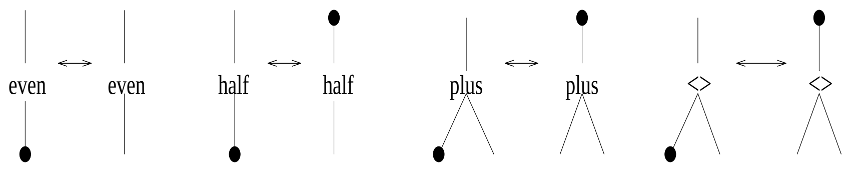

Figure 2: Labelled fragments

f's definition or other rewrites, that contributes to an *abstract* ripple path in the theorem tree, and if a particular lemma is missing for the *concrete* rippling at a node N, then a lemma can likely be proven that can be used for rippling at N of the theorem tree. This heuristic is highly desirable for analogies because analogy should work despite missing lemmas.

It is even more sensible to base m_b on labelled fragments if the INDUCTION method determines the IE-variables by reasoning about fragments (as, e.g., in INKA [Hutter 94]). Then the relevant fragments are computed anyway.

Match Constraints

If the tree structure and the labelled fragments are preserved by m_b, then the abstract reasoning about the ripple process is the same for source and target and yields corresponding IE-variables and corresponding ripple paths. Then $m_b(ts_r) = tt_r$ heuristically constitutes a ripple subtree in the target in which the images of IE-variables of the source are IE-variables of the target.

m_b is a second-order substitution of the indexed source theorem with the target theorem that maps IE-variables to target terms and functions/terms to target functions/terms. We restrict m_b by the following constraints (for controlled exceptions see section 5.2).

b1 m_b preserves the labelled fragments of the nodes of ts_r.

b2 IE-variables are mapped to target terms. Those target terms occur in the subtree tt_r of the target theorem tree only. This ensures that target rippling cannot take place outside $m_b(ts_r)$.

b3 If an image t_i of an IE-variable also occurs at position P in an image of a source term which does not belong to the IE-variables, then t_i has to be renamed at position P. This ensures that target IE-terms are images of source IE-variables only. This constraint may generalise variables apart[8] in the target as in the example `assapp-assplus1`.

[8]That is, generalise a conjecture by replacing variables with several occurrences by new variables in places.

The problem of generalizing variables apart is a difficult one (see, e.g., [Hesketh 91].). In some cases, i.e., if an appropriate source is available, the generalisation can be guided by analogy as a byproduct of b3. [9]

b4 We get into trouble if in the source a universally quantified variable x is actually used as a sink in rippling and $m_b(x)$ is an existentially quantified target variable which cannot be used as a sink. Hence we do not allow m_b to map a universally quantified variable to an existentially quantified one. Maybe this constraint can be loosened later on.

b5 In order to reduce the search for mappings, the largest terms outside ts_r are mapped to corresponding largest terms outside tt_r.

For non-inductive source proof plans these constraints do not apply. b1 - b5 restrict the analogical mappings and restrict the search space of source problems for a given target problem. More heuristics, e.g., one that prefers identical mapping, will be necessary for the basic and extended matching.

5.1.2 Extended Matching

The extended matching creates the extended mapping m_e that maps source rewrites to target rewrites. For examples see section 10. Extended mappings are applied in order to instantiate method justifications in the target, such as "A rewrite *source_def* is recorded as a definition".

Note that the mapping of source rewrites is to some extent restricted by IF-equations and by m_b. Take, for instance, `lenapp-halfplus2` with
IF-equations: $app_2 = app_3$, $len_2 = len_3$, $::_3 = ::_1$, $len_4 = len_1$,
$s_1 = s_2$, $len_2 = len_4$, $app_2 = app_4$, $app_4 = app_1$,
$\mathrm{m}_b : len_{1;2} \mapsto half$, and $\mathrm{m}_b : app_{1;2} \mapsto +$,
with the source rewrites
`app2`: $app_3((v_0 ::_2 a), b) = v_0 ::_3 (app_4(a, b))$ and
`length2`: $len_3(v_0 ::_1 a) = s_1(len_4(a))$, and with the target rewrites
`plus2`: $+(s(a), b) = s(+(a, b))$ and
`half2`: $half(s(s(a))) = s(half(a))$.
m_e maps `app2` to `plus2` because of m_b and $app_3 = app_2, app_2 = app_4$, and similarly m_e maps `len2` to `half2`. In general, however, this restriction does not uniquely determine the rewrite to be matched.

Different extended mappings have to be constructed for the target step case and for the target base cases because different sets of IF-equations belong to

[9]Other examples are source $x * (y * y) = (x * y) * y$
target $x + (x + x) = (x + x) + x$, and
source $app(rev(x), rev(y)) = rev(app(y, x))$
target $app(rev(x), rev(x)) = rev(app(x, x))$.

the step and base cases, respectively and different rewrites are used, see, e.g., `plus2right-doubleplus2right` or `lenapp-halfplus2`.

According to the paradigm of derivational analogy, the extended mappings should actually be constructed stepwise while following the source plan as mentioned in a footnote above. This may be quite inefficient because backtracking may cause an annulment of work done during the analogical replay. Hence, we propose to find the mapping first and to replay the source plan afterwards.

5.2 Reformulation

The matchings trigger the application of a reformulation that carries out the mappings and may execute some other changes of the source proof plan. Three procedures perform the reformulations dependent on the basic matching, the extended matching, and the IF-equations, respectively.

5.2.1 Basic Reformulation

First of all, the basic reformulation procedure applies the substitution m_b. Some basic matchings may require additional changes of the source proof plan as described in the following.

shrink If m_b maps $f_i \mapsto \lambda w.w$, then the WAVE method for f_i becomes redundant and can be removed from the proof plan. Example: `plussumsum-plus2right`.[10]

blow If m_b maps a source function symbol f to a function $\lambda\overline{x}.g_1(g_2(\overline{x})$ that has the same (combined) labelled fragment as f, then m_e suggests a corresponding target rewrite with $\mathrm{m}_b(f)$ in its skeleton.

An alternative not considered yet, is to introduce two WAVE methods (for g_1 and g_2) instead of one source WAVE method for f.
Example: `plus2right-doubleplus2right`
A special case of *blow* is the introduction of a function that has the same fragment as the identity function.

reduce A normalizing reformulation[11] adds a method M (or subplan) on top of the proof plan for which $output(M) \rightarrow input(M)$ holds. The first option of the *reduce* reformulation introduces a method REDUCE, for which the input is $(\forall) f(t_1) = f(t_2)$ and its output is $t_1 = t_2$. REDUCE utilizes the equality axiom scheme

[10]This is a special case of $\lambda\overline{x}f_1(f_2(\overline{x})) \mapsto \lambda\overline{w}g(\overline{w})$ which is, however, not a second-order substitution.

[11]A normalising reformulation introduces a method or subplan on top of the source proof plan. Normalising reformulations resemble the initial-segment concatenations in Carbonell's transformational analogy [Carbonell 83].

$\forall x_1 \forall x_2 (x_1 = x_2 \rightarrow f(x_1) = f(x_2))$, where f is a function or relation. The method REDUCE could eventually have other parameters (or submethods) that guide its actual work.

reduce can be applied if the source theorem is $(\forall) t_1 = t_2$ and the target theorem is $(\forall) f(x_1, \ldots, t'_1, \ldots, x_n) = f(x_1, \ldots, t'_2, \ldots, x_n)$ and $\mathrm{m}_b : t_1 \mapsto t'_1, t_2 \mapsto t'_2$. In order to trigger *reduce*, the matching procedure has to be extended slightly.

reduce can ease the analogy that otherwise might suggest difficult to prove lemmas. It can also shorten target proof plans substantially[12]. *reduce* should be preferred to other reformulations at least in cases where $t_1 = t'_1, t_2 = t'_2$ because it yields a very simple analogy. In other cases we may need a heuristic in order to decide which reformulation to prefer: *reduce* or *blow*, e.g., prefer *reduce* to *blow* if the mapping $\mathrm{m}_b : t_1 \mapsto t'_1, t_2 \mapsto t'_2$ maps function symbols to function symbols only.

symm If the second-order match of a source theorem with a target theorem permutes arguments of f for $\mathrm{m}_b(f)$, then the labelled fragment of f and $\mathrm{m}_b(f)$ have to be permuted in the same way.
For instance, in `apprev-plussumsum`
$\mathrm{m}_b : app_1(w_1, w_2) \mapsto \lambda w_1 \lambda w_2 . plus(w_2, w_1)$ requires a labelled fragment of $plus$ that is symmetric to the labelled fragment of app_1. Those symmetric labelled fragment of app and $plus$ are provided by the rewrites `assapp` and `plus2` here.

The corresponding reformulation changes, apart from the substitutions of m_b, only position parameters of methods in the source plan. For a detailed example see `apprev - plussumsum` in section 10.

condt If the source theorem is Th_s and the target theorem is $C_t \rightarrow Th_t$, and if we can find a basic mapping m_b with $m_b(Th_s) = Th_t$ that meets the constraints, then a *condt* reformulation of the proof plan replaces (weak) FERTILIZE by Kraan's VERY (WEAK) FERTILIZE [Kraan 94]. In the step case[13] *condt* introduces into the target planning the instantiated antecedents' C_{tbj} of the target base cases as an additional subgoal and introduces subplans for their proof or disproof into the plan. *condt* transfers the source base case if C_{tbj} can be proven, and it terminates a base case if C_{tbj} is disproved. [14] For a detailed example see `halfdouble-doublehalf` in section 10.

[12]E.g., this reformulation yields a plan of `halfplus` with 13 methods, as opposed to base-level planning that comes up with a plan with 57 methods and lemma calculation.

[13]VERY (WEAK) FERTILIZE produces the additional lemma $C'_t \rightarrow C_t$ to be proved, where C'_t is the antecedents of the induction conclusion and C_t is the antecedents of the induction hypothesis.

[14]Since this reformulation is a foreseeable heuristic change of the proof plan we don't have to rely on a modification during the replay that would establish preconditions which are not satisfied in the target. Such a modification could replace weak FERTILIZE by VERY WEAK FERTILIZE for those specific cases because the precondition `partial_induction_hyp_match` does not hold in

add-args If m_b maps a function f to an $m_b(f)$ and duplicates arguments, then the source subplans for treating this argument has to be duplicated. Put more technically, if $m_b(f)$ has k occurrences of a variable w_i at positions $i_1, \ldots, i_k$ in the target and the fragment of $m_b(f)$ agrees at positions $i_1, \ldots, i_k$ with the fragment of f at position i in the source and is the same everywhere else, then the source subplan for treating the ith argument of f has to be introduced k times (for each occurrence of w_i) into the target plan (see for a more general reformulation `Add-Args` [Melis 95]).

As opposed to a mapping m_b that adds a constant argument and does not cause any changes of the plan structure, for *add-args* the ith argument of f in the source may contain IE-variables.

5.2.2 IF-Reformulation

As the basic reformulaton was guided by the basic match, the IF-reformulation is guided by IF-equations. This procedure can apply the reformulations*1to2* or *2to1*:

1to2 If the mapped IF-equations require $m(f_i) = m(f_j)$, where f_j stems from a wave-rule, but $m(f_i)(m(f_i)) = m(f_j)$ actually holds in the target, then change a primitive induction to a two step induction in the target and double the steps between INDUCTION method and the occurrence of the IF-equation.

This reformulation could be extended later to composing wave-fronts more generally. The reformulation removes the IF-equation from the justification. Example: `lenapp-halfplus2`

2to1 If the mapped IF-equations require $m(f_i) = m(f_j)$, where, f_j stems from a wave-rule, and $m_e(f_j)(m_e(f_j)) = m_e(f_i)$, then the reformulation changes a two step induction to a primitive induction in the target and collapses the corresponding double steps to single steps between the INDUCTION method and the occurrence of the IF-equation. Again, this reformulation can be extended later to more general compositions of wave-fronts. The reformulation removes the IF-equation from the justification.
Example: `halfplus2-lenapp`

5.2.3 Extended Reformulation

First of all, the extended reformulation procedure applies the substitution m_e. Certain features of m_e require additional changes of the source proof plan:

abs If m_e maps a function symbol f to a lambda term with an unspecified non-constant argument o that can have different instantiations, then a replayed

the target.

source plan would treat the abstract symbol o as if it were absent. In the target, however, this amounts to an abstraction.

For each new occurrence of o a new meta-variable o_i has to be introduced. The instantiations of the o_i have to be computed by running the methods M_i that first introduce o_i into a subgoal[15]. From the target plan node of M_i the substitution of the meta-variable o_i has to be stored. Very likely a target proof plan branch will have to be completed by base-level planning because a method-justification wont be satisfied.
Example: `assp-assm` with $m_e : s \mapsto \lambda w. + (w, o)$.[16]

case Let R_T be the target rewrite that belongs to a set of conditional target rewrites for which the disjunction of all the conditions is a valid formula.

If m_e maps a source wave-rule R_S to rewrite R_T, then additionally introduce a case split into the source proof plan where R_S was applied. Introduce the remaining source subplan as the R_T branch of the plan and declare the other subgoals of the case split open goals (to be proved by base-level planning).
Example: The extended mapping maps
plus2: $plus(s(x), y) = s(plus(x, y))$ to
union4: $\neg member(x, y) \rightarrow union(h :: x, y) = h :: union(x, y)$
by $\mathrm{m}_e : plus \mapsto union$ etc.

5.3 Analogical Replay

The analogical replay is the core of the analogy-driven proof plan construction that applies the reformulations, checks the justifications, and reacts to failures of justifications in the target.

Ideally, the matching that triggers the reformulations and the analogical replay should be integrated. Currently, however, the matching is accomplished before the replay for pure efficiency of the match procedure. After the basic and extended mappings have been found, the source proof plan is replayed analogically. According to Table 1, the analogical replay follows the (linearised) source plan and applies the reformulations, triggered by the mappings, to the source plan and its nodes.

At an INDUCTION method node, the reformulation instantiates the induction hypothesis, the induction variables, and the induction term. The INDUCTION method runs with these instantiated parameters, checks whether the induction term is a proper one (terminating order), and yields the step case and base case subgoals. The target base case terms, such as 0 or nil, have to be matched with a base case

[15]latest when the instantiation is relevant to satisfy a method justification

[16]The name *abs* might be misleading because actually the source is an abstraction of the target here.

term in the source. There can be more target base cases than source base cases or vice versa.

The replay checks the justifications of the respective (reformulated) source plan node and transfers the source method to the target plan if its justifications hold in the target. If a justification does not hold in the target, a suitable action is taken. The actions to be taken depend on the actual justification that is not satisfied. They are described in section 5.3.2.

If the attempt to establish a justification fails, the source method cannot be transferred and a gap remains in the target proof plan. That is, the target plan node has an empty method slot and a conclusion that contains a special meta variable, a gap variable $?_i$. The gap variable is a place holder for the unknown subexpression of the sequent in the (current) target node that corresponds to the source subexpression that was changed by the source method which could not be transferred. See, for an example, `assp-assm` or `lenapp-halfplus2` (base case). The status of this target plan node remains open. The replay stops as soon as no subexpression other than a gap variable occurs in the target node conclusion (see the base case of `plus2right-doubleplus2right`).

The actual replay may result in a proof plan that has open goals of different origins: If the replay stopped, i.e., the gap is open-ended, the open goals have to be closed by base-level planning. For gaps between two proper plan nodes elaborate techniques have to be developed in the future that aim at closing these gaps, e.g., by forward reasoning, abduction. So far we can close gaps by base-level planning only if gap variables can be instantiated during the analogical replay. As the example `assp-assm` shows, the transfer of following source methods may force an instantiation of a gap variable or at least provide information about possible instantiations. In this case, the gap variable can be (partially) instantiated which assists a subsequent base-level planning for the gap.

5.3.1 Justifications

In *CIAM* justifications exist in a bigger variety and play a more important role than they do in Ω-MKRP, so far. The reason is that *CIAM*'s methods, in particular their preconditions, have been polished for a long time in order to restrict the search for methods. Hence, a variety of reasons for the application and applicability of a *CIAM* method exist. I analyse how justifications can be constructed for nodes of *CIAM*'s proof plans.

Some preconditions of methods yield justifications that have to be checked by the analogical replay. Preconditions of methods can be classified into

1 legal preconditions the failure of which causes a failure of the transfer of the source method,

- e legal preconditions which the analogical replay first should try to establish if they are not satisfied in the target,
- p legal preconditions that involve the proof of some formula,
- h heuristic preconditions that can be overridden, and
- i preconditions that are mere instantiation conditions.

Some current *CLAM* methods have hybrid preconditions, e.g., `trivial(H==>C)` that is classified `h,p`. The actual classification of the preconditions of *CLAM*3 methods is given in the first appendix (section 9).

The justifications of the node N consist of goal dependencies, the relevant IE-equations, and of the following justifications provided by the successful preconditions of the method applied in N:

- l-preconditions become l-justifications. In *CLAM*3, e.g., an l-precondition of the method EQUAL is that an equation occurs in an hypothesis.
- e-preconditions yield e-justifications. In *CLAM*3 those justifications state, e.g., that a wave-rule or definition is stored in the library.
- h-preconditions are ignored and do not yield any justification. In *CLAM*3, e.g., a heuristic precondition of the method EQUAL is that the replacement for a term is a free variable. This precondition could be dropped, thus allowing for another analogous replacement of a target term. Note, that dropping a h-precondition without the guidance from analogy might not be a good idea.
- h,p-preconditions are weakened to p-justifications. In *CLAM*3, e.g., a p-precondition of the method WAVE is that the condition of a conditional wave-rule is in the hypotheses already (trivially provable). This is weakened to a justification that requires that the condition is provable from the hypotheses.

5.3.2 Actions if Justifications Do not Hold

The analogical replay chooses an action or base-level plans if a justification of the method to be transferred does not hold in the target, as shown in Table 1. In the following the actions are described. Base-level planning is choosen if no transfer takes place, either because an action failed or because no action was possible at all.

Wrong Goal Dependencies

The goal dependencies are represented by the address of a plan node. The source node has to be in the same branch (base or step case) of the proof plan as the target node considered by the replay. This may be not the case if a target branch is complete (no more open goal) whereas the corresponding source node has subgoals. Then selecting the "next" node in the source cannot yield a jusitified transfer. Therefore, if the source goal dependencies do not hold for the selected open target goal, then do not transfer the method.

Violated IF-equations

If an IF-equation does not hold in the target, then produce an open target goal by replacing the subexpression that resulted from the last rewriting of the source goal by a gap variable $?_j$ in the target. Record a gap, i.e., `no method` in the target node.

In case a source rewrite corresponds to two or more different target rewrites, e.g., in `assp-assm` with given `distr`, each alternative may be considered by m_e using additional indices. This amounts to different copies of the same source rewrite. An appropriate copy of the rewrite has to be chosen by the replay in order to satisfy the IE-equations.

Violated Justifications originated in Preconditions

If a precondition-justification is not satisfied in the target, the analogical replay takes the following actions specific for particular justifications.

1. If an l-justification cannot be satisfied and does not contain a gap-meta-variable $?_j$, then the source method is not transferred. If an l-justification contains a gap-meta-variable $?_j$, then the replay suggests satisfying the justification by instantiating $?_j$ in a controlled way.
   For the justification of the method FERTILIZE(strong), e.g., that requires to match the induction hypothesis H: $lhs = rhs$ with the goal G: $\sigma(lhs) = ?$, the replay can instantiate ? by $\sigma(rhs)$.

   If the justification `induction_hyp_match(Var, H, G, SubstL)` of FERTILIZE(strong) with G: $\sigma(lhs) = ?_1$ and H: $lhs = rhs$, then the replay instantiates $?_1$ by $\sigma(rhs)$. See, for instance, `assp-assm`.

   The replay suggests the same kind of instantiation for the method ELEMENTARY by assuming G: $lhs = rhs$. (With a more elaborate suggest-procedure a gap-meta-variable in the justifications of other methods could be partially instantiated, e.g., for methods GENERALISE, NORMAL, and FERTILIZE(weak). This is, however, beyond our current scope.)

2. Base-level planning is invoked in order to establish a p-justification.
   For instance, the replay tries to establish `(H==>C)` by base-level planning (with a time limit) . If this fails, the source method is not transferred. If it succeeds, the method is transferred. Example: `div3-div3term`.

3. If an e-justification does not hold in the target, the replay suggests a target rewrite by mapping the rewrite in the justification. The suggested target rewrite may contain meta-variables for the function variables of the parametrized source that have not yet been mapped by m_b and m_e (For an example see `plus2right-doubleplus2right`.). The suggestion of rewrites should be accompanied by a disprove procedure[17]. (See, e.g., base case of example `plus2right-doubleplus2right`) which avoids a waste of effort for the analogical replay in case a false lemma is suggested. Only if the suggested rewrite is proved, the method can be transferred.

In the following, some examples are given for the effect of the analogy as a control strategy that can override the default control

- When the plan of
  `zerotimes1`: $x = 0 \rightarrow times(x, y) = 0$ is replayed for the target
  `zerotimes3`: $times(x, y) = 0 \rightarrow times(times(x, y), y) = 0$, then the h-precondition of the EQUAL method `freevarinterm(G,Var)` that holds in the source with `Var =` x does not hold for $\mathrm{m}_b(x) = times(x, y)$. Since this precondition is ignored for the justifications, the replay transfers the EQUAL method nevertheless.

  The same holds for the class of cnc_examples such as
  source: `cnc_times`: $x = y \rightarrow app(x, z) = app(y, z)$
  target: `cnc_termplus`: $*(u, x) = *(v, y) \rightarrow +(*(u, x), z) = +(*(v, y), z)$

- If we took the precondition `trivial(H==>C)` as a justification, the analogical replay of the following example would not work:
  source: `div3`:$\neg y = 0 \rightarrow div(+(y, x), y) = s(div(x, y))$
  target: `div3term`:$\neg * (y, z) = 0 \rightarrow div(+(y, x), y) = s(div(x, y))$.
  The h,p-precondition `trivial(H==>C)` of the method EVAL_DEF yields the source p-justification $(\neg y = 0 \rightarrow \neg y = 0)$ which is mapped to the target justification $(\neg * (y, z) = 0 \rightarrow \neg y = 0)$. This target justification can be established by base-level planning for $(\neg * (y, z) = 0 \rightarrow \neg y = 0)$. Since the planning succeeds, the replay transfers the method EVAL_DEF.

[17]available in CI^AM3.2 that can prove many wrong suggestions to be false

6 Related Work

Villafiorita [Villafiorita & Sebastiani 94] tries to incorporate user-supplied abstractions into analogy in inductive theorem proving. For the purpose of program optimisation Madden [Madden 91] investigated a reformulation of proof plans that replaces a course-of-values induction by a stepwise induction.

Kolbe and Walther [Kolbe & Walther 94] reuse equational proofs by second-order substitutions of a "schematic shell" including the used axioms and the conjecture. They calculate equational constraints by a separate calculus. By second-order matching the "schematic shell" with the target conjecture (induction conclusion) and target rewrites they obtain the instantiated rewrites which are relevant in the target. Similar to [Kolbe & Walther 94] we store IF-equations used during source planning. The presented reformulation *add-args* is similar to a patch in [Kolbe & Walther 95]. Their paradigm, however, is different to analogy because they generalise and re-instantiate. Further differences are:

- The analogy can suggest IE-variables and induction conclusion, they need the target induction conclusion to be given.
- The analogy may result in a proof plan with gaps and does not necessarily provide an exact replay with a full match of all axioms which is, however, required in [Kolbe & Walther 94].
- The generalization approach aims at providing the target rewrites only, while the analogy finds a target proof plan.
- The analogy in *CIAM* does not use the analogical replay if the proof planning goes through with little search.
- Both approaches can suggest lemmas from (partially) instantiated constraints and source rewrites. In addition, analogy can suggest lemmas by replaying lemma-critics results.

7 Conclusion and Future Work

The analogy-driven proof plan construction has been applied within the *CIAM* environment. Not surprisingly, the particular procedures of reformulation turned out to be dependent on the data stuctures for methods and plans. Some reformulations that encode frequently used domain specific heuristics depend on the scope of the planner. For instance, constraint b1 encapsulates a heuristic specific for inductive theorem proving and can be exploited for inductive proofs only. For each planner several types of justifications are needed. The actual justifications, however, depend on the means for search control in the planner.

The main contribution of this application of analogy-driven proof plan construction has been to identify match heuristics, reformulations, justifications, and actions to be taken if the justifications are not satisfied in the target for CL^AM. Thereby we have been able to show with some relatively simple examples what analogy has to offer even for a planning system with little average search for methods.

The power of the analogy-driven proof plan construction stems from the given source plan, from the reformulations, and from the actions taken in order to establish justifications. There is, of course, a tradeoff between the power and the costs of the matching and reformulation.

The analogical replay may construct proof plans with gaps that cannot be closed by base-level planning. Furthermore, reformulations encode heuristics that can fail, which is only natural for any analogy process.

The work on analogy in CL^AM will be continued and will examine more challenging examples that CL^AM cannot solve without analogy. More elaborate techniques for closing gaps in target proof plans have to be investigated. Hierarchical information will be included into the justifications.

8 Acknowledgement

I would like to thank the Dream group for guiding me into CL^AM and discussions on analogy in CL^AM, in particular Alan Bundy, Andrew Ireland, Alan Smaill, and Jon Whittle.

9 Appendix I

The *classification of preconditions*, as it follows for CL^AM3 methods, has to be done manually, as far as I can see.

```
elementary   h,p    elementary(H==>G,I)

equal         l     ((hyp(HName:Term=Var in T,H), Dir=left)
                    v
                    (hyp(HName:Var=Term in T,H), Dir=right)
                     ),
              l     (not freevarinterm(Term,_)
                     orelse
                    (atomic(Var), not atomic(Term))
                     orelse
                    (atomic(Var), atomic(Term), Term @< Var)
                     ),
              h     freevarinterm(G,Var)

eval_def      h     not contains_meta_variables(G),
              i     expression_at(G,Pos,Exp),
              l     not wave_fronts(_, [_|_], Exp),
              e     function_defn(Exp,Rule:C=>Exp:=>R),
             h,p    trivial(H==>C)

fertilize     l     induction_hyp_match(Var, H, G, SubstL)
                  OR
              l     partial_induction_hyp_match(Var, H, G,NG,SubstL)

generalise    i      matrix(Vs,M1,G),
              i      sinks(M,_,M1),
              l      member(M,[(L=R in _),(L=>R),geq(L,R),leq(L,R),greater(L,R),less(
              l      exp_at(L,_,Exp),
              l      not atomic(Exp),
              l      not constant(Exp,_),
              l      not oyster_type(Type, _, Exp),
              l      object_level_term(Exp),
              l      exp_at(R,_,Exp),
              i      type_of(Exp,Type),
              i      append(Vs,H,VsH),
              i      hfree([Var],VsH),
              i      replace_all(Exp,Var,G, NewG1),
                     \+ disprove([Var:Type]==>NewG1)
```

```
normal        l     matrix(Binds, Cond => Body, G),
              l     not wave_occ(Binds, Cond, _, _),
              l     not sinks(_, [_|_], G),
              l     not wave_fronts(_, [_|_], G),
              l     member(Body, [_ = _ in _, _ => _])

wave          l      wave_occ_at(G,Pos,L),
              e      wave_rule_match(Rule,long(Dir),C=>L:=>R,[]),
             h,p     trivial(H==>C)
                    OR
              l      wave_occ_at(G,Pos,L),
              e      wave_rule_match(Rule,trans(Dir),C=>L:=>R,[]),
             h,p     trivial(H==>C),
              l      sinkable(R)

induction     l      wave_occ(H, G, VarTyps, IndTerms)

casesplit     i     matrix(Vars, Matrix, G),
              i     exp_at(Matrix, MatPos, L),
              e     wave_rule(_,_,[_|_]=>LL:=>_),
              i     copy(LL,L),
              l     correct_wave_vars(LL, L),
              e     condition_set(LL, CConds, _)
```

10 Appendix II: Examples

10.1 comp-halfplus2

This is an example for the application of the reformulation *reduce*.
source theorem: $+(a,b) = +(b,a)$
target theorem: $half(+(a,b)) = half(+(b,a))$.[18]

Source plan

```
induction([s(b)],[b:pnat]) then
  [eval_def([2,1],plus1) then
     induction([s(a)],[a:pnat]) then
       [eval_def([1,1],plus1) then
          elementary(...),
```

[18]We do not index the function symbols of the source theorem because it is not necessary in this case.

```
        wave([1,1],[plus2,equ(left)]) then
          fertilize(weak(v0)) then
            elementary(...)
       ],
   wave([1,1],[lemma1,equ(left)]) then
     wave([2,1],[plus2,equ(left)]) then
       fertilize(weak(v0)) then
         elementary(...)
  ]
```

An analogical transfer using the reformulation *blow* with mapping $m_b : + \mapsto \lambda a \lambda b.half(+(a,b))$ rather than *reduction* would not succeed with a transfer of the base case and would cause a lot of effort to transfer the step case.

In terms of analogy it is a good guess, to reduce the proof of `halfplus2` to a source proof of `comp` because the latter is available already. Besides, the proof plan that results by introducing an additional method REDUCE(EQUALITY) into the plan of `comp` is much shorter than one that would result from base-level planning even if all necessary wave-rules and definitions for *plus* and *half* were given[19]. The resulting target proof plan deviates from the common pattern though. The resulting target plan is

```
reduce(equality) then
 induction([s(b)],[b:pnat]) then
   [eval_def([2,1],plus1) then
      induction([s(a)],[a:pnat]) then
        [eval_def([1,1],plus1) then
          elementary(...),
         wave([1,1],[plus2,equ(left)]) then
          fertilize(weak(v0)) then
            elementary(...)
        ],
    wave([1,1],[lemma1,equ(left)]) then
      wave([2,1],[plus2,equ(left)]) then
        fertilize(weak(v0)) then
          elementary(...)
  ]
```

[19]plan time `halfplus2` 119.116 compared with plan time 4.283 for `comp` which does not even have to be planned again

10.2 assp-assm

This is an example for the application of the reformulation *abs* and for handling gaps and gap variables.
source theorem: $+_1(x, +_2(y, z)) = +_4(+_3(x, y), z)$
target theorem: $*(x, *(y, z)) = *(*(x, y), z)$,
where the distributivity axiom is not given as a rewrite. The source proof plan is

```
induction([s1(x)],[x:pnat]) then
  [eval_def([1,1],plus1) then
     eval_def([1,2,1],plus1) then
       elementary(...),
   wave([1,1],[plus2,equ(left)]) then
     wave([1,2,1],[plus2,equ(left)]) then
       wave([2,1],[plus2,equ(left)]) then
         fertilize(weak(v0)) then
           elementary(...)
  ]
```

- The labelled fragments of $+$ and $*$ are equal and, hence, $m_b : +_{1;2;3;4} \mapsto *$. Since the source induction variable is x, the induction variable in the target is identified as $m_b(x) = x$.

- The only source wave-rule is plus2:
  $+_5(\boxed{s_1(\underline{x})}, y) \Rightarrow \boxed{s_2(\underline{+_5(x, y)})}$ and the only given target rewrite is
  $*(s(x), y) = +(*(x, y), y)$.
  The IF-equations of the source are:
  $+_1 = +_5$
  $+_3 = +_5$
  $+_4 = +_5$
  $s_1 = s_2$.
  m_e maps $s_1 \mapsto s(w_1)$ and $s_2 \mapsto +(w, o)$ for an abstract symbol o that may have different instantiations.

- The first method is INDUCTION($s_1(x)$), justified by plus2.

- The step case goal in the source is: $+_1(\boxed{s_1(\underline{x})}, +_2(y, z)) = +_4(+_3(\boxed{s_1(\underline{x})}, y), z)$ which yields a target goal
  $*(s(x), *(y, z)) = *(*(s(x), y), z)$ because of $m_e : s_1 \mapsto s$.

- The next method, WAVE(plus2), has as justifications the source wave-rule plus2 and $+_1 = +_5$ which requires $m_e : +_5 \mapsto *$.

- The next source goal is:
$\boxed{s_2(\underline{+_5(x, +_2(y,z))})} = \ldots$ and the corresponding target goal is
$+(*(x,*(y,z)),o_1) = \ldots$. The instantiation $*(y,z)$ of the meta-variable o_1 is obtained by running the wave method and it is stored.

- The next method, WAVE(plus2), has as justifications the source wave-rule plus2 and $+_3 = +_5$ which hold in the target as well.

- The next source goal is:
$\boxed{s_2(\underline{+_5(x, +_2(y,z))})} = +_4(\boxed{s_2(\underline{+_5(x,y)})}, z)$. The corresponding target goal is $+(*(x,*(y,z)),o_1) = *(+(*(x,y),o_2),z)$. The instantiation y of the meta-variable o_2 is obtained by running the wave method and it is stored.

- The next method, WAVE(plus2)(*), has as justifications the source wave-rule plus2 and $+_4 = +_5$, and $s_1 = s_2$. The mapped version of the latter does not hold in the target because $s(w) \neq +(w,o)$. Hence, the method cannot be transferred to the target. The replay yields a **gap** and produces an **island** node with no method and with the conlusion
I: $+(*(x,*(y,z)),o_1) =?$.

- The next source method, FERTILIZE(weak), has $+_5 = +_1$ as a justification which holds in the target. An l-justification of the node is
$partial\_ind\_hyp\_match(\_, [+_1(x, +_2(y,z)) = +_4(+_3(x,y),z)], [s_2(+_5(x,+_2(y,z))) = s_2(+_5(+_5(x,y)),z)], \_)$. The corresponding target justification
$partial\_ind\_hyp\_match(\_, [*(x,*(y,z)) = *(*(x,y),z)], [+(*(x,*(y,z)),o_1) =?], \_)$ holds. The method yields the subgoal $+(*(*(x,y),z),o_1) =?$.

- The next source method, ELEMENTARY, has the justifications $+_{4,3} = +_5$, which holds in the target, and `true`$(+(*(*(x,y),z),o_1) =?)$. The latter justification needs an instantiation of ? to be satisfiable. A possible instantiation is $? = +(*(*(x,y),z),o_1)$.

- The instantiation of ? together with the substitutions of o_1 and o_2 can be used to close the gap in the target proof plan that remains after the analogical replay: Closing the gap requires a target method and justification that produces $\ldots = +(*(*(x,y),z),*(y,z))$ from $\ldots = *(+(*(x,y),y),z)$. The WAVE method justified by the distributivity rewrite (`distr`) can close this gap. In fact, the justification $s_1 = s_2$ of wave(plus2)(*) is mapped to $+(\_,o) = (\_,o)$ in case m_e maps `plus2` to `distr`: $*(+(x,z),y) = +(*(x,y),*(z,y))$.

If `distr` was given as a target rewrite, the source rewrite `plus2` might have two different images under m_e, namely `times2` and `distr`. In such a situation the extended matching provides $m_e : +_{51} \mapsto *, s_{11} \mapsto s, s_{21} \mapsto +(\_,o)$ (for plus2 $\mapsto$ times2) and
$m_e : +_{52} \mapsto *, s_{12} \mapsto +(\_,o), s_{22} \mapsto +(\_,o)$ (for plus2 $\mapsto$ distr). In order to satisfy

the IF-equation $s_1 = s_2$ at plan node N, the image `distr` of `plus2` has to be chosen for the transfer of the method at N.

The source **base case** conjecture is $+_1(0, +_2(y, z)) = +_4(+_3(0, y), z)$ and
the target base case conjecture is $*(0, *(y, z)) = *(*(0, y), z)$, where 0 is introduced by the INDUCTION method.
The source rewrite is plus1: $+_7(0_1, x) = x$.
The given target rewrite is times1: $*(0, x) = 0$.
The base case IF-equations are:
$+_1 = +_7$,
$0 = 0_1$,
$+_3 = +_7$,
$+_2 = +_4$.
$\mathrm{m}_e : +_7 \mapsto *, 0_1 \mapsto 0$. This mapping suggests a target rewrite $*(0, x) = 0$ which can be disproved. Therefore the target base case has to be completed by base-level planning.

The target proof plan becomes

```
induction([s(x)],[x:pnat]) then
[                                        %BASE-LEVEL  yields
   eval_def([1,1],times1) then
     eval_def([1,2,1],times1) then
       eval_def([2,1],times1),then
         elementary(...),                %%

   wave([1,1],[times2,equ(left)]) then
     wave([1,2,1],[times2,equ(left)]) then
       wave([2,1],[distr,equ(left)]) then
         fertilize(weak(v0)) then
           elementary(...)
  ]
```

10.3 apprev-plussumsum

This is an example for the application of the reformulation *symm* and it shows that step case and base case replay have to be handled separately and depending on different m_e and sets of IF-equations.

Source theorem: $app_1(rev(y), rev(x)) = rev(app_2(x, y))$
Target theorem: $+(sum(x), sum(y)) = sum(app(x, y))$

Source rewrites:
`rev2`: $rev_2(h ::_3 l) = app_5(rev_2(l), h ::_4 nil)$
`app2`: $app_3(h ::_1 l_1, l_2) = h ::_2 app_3(l_1, l_2)$
`assapp`: $app_6(l, app_7(m, n)) = app_8(app_6(l, m), n)$.

The **IF-equations** of the source planning are:
$app_1 = app_6$
$app_2 = app_3$
$app_5 = app_7 = app_8$
$rev = rev_2$
$::_1 = ::_2 = ::_3$.

The source proof plan is

```
induction([v0::m],[m:int list]) then
  [eval_def([1,2,1],app1) then
     eval_def([2,1,1],rev1) then
        generalise(rev(l),v0:int list) then
           induction([v1::v0],[v0:int list]) then
              [eval_def([1,1],app1) then
                  elementary(...),
               wave([1,1],[app2,equ(left)]) then
                  fertilize(weak(v2)) then
                     elementary(...)
              ],
   wave([1,2,1],[app2,equ(left)]) then
     wave([2,1],[rev2,equ(left)]) then
        wave([2,1,1],[rev2,equ(left)]) then
           wave([1,1],[assapp,equ(left)]) then
              fertilize(weak(v1)) then
                 elementary(...)
  ]
```

Given target rewrites are:
`app2`: $app(h :: l_1, l_2) = h :: app(l_1, l_2)$
`sum2`: $sum(h :: l) = +(h, sum(l))$
`plus2`: $+(s(x), y) = s(+(x, y))$.

The basic mapping is
$\mathrm{m}_b : app_2(w_1, w_2) \mapsto app(w_1, w_2)$,
$app_1(w_1, w_2) \mapsto +(w_2, w_1)$
$rev(w_1) \mapsto sum(w_1)$.
$\mathrm{m}_b : app_1(w_1, w_2) \mapsto \lambda w_1 \lambda w_2 . plus(w_2, w_1)$ requires a labelled fragment of $plus$ that is symmetric to the labelled fragment of app_1 that originates in `assapp`. `plus2` provides such a symmetric labelled fragment.

The extended match yields $\mathbf{m}_e$: $::_1 \mapsto ::$, $::_2$ $(w_1, w_2) \mapsto \lambda w_1 \lambda w_2 . w_1$, $app_5(w_1, w_2) \mapsto +(w_2, w_1)$ matches `rev2` with `sum2` and `app2` with `app2`. `assapp` can not be matched consistently with `plus2` but m_b and m_e **suggest** a target lemma `assplus`: $+(+(n, m), l) = +(n, +(m, l))$.

m_e has been constructed following the sequence `app2, rev2, assapp`, in which the rewrites were used in the source planning. If the m_e was not established in the same sequence as prescribed by the sequence in which the source rewrites had been employed in the source planning process, we could have obtained a target lemma which is wrong. E.g., with the order `app2, assapp, rev2`, we obtain
$\mathrm{m}_e : app_5 \mapsto \lambda w_1 \lambda w_2 . s(w_1)$ mapping `assapp` to `plus2`. Then a target lemma
$sum(h :: l) = s(sum(l))$ corresponding to `rev2` would be suggested which is wrong since it yields $s(sum(l)) = +(h, sum(l))$.

The **source base case** conjecture is
$app_1(rev(l), rev(nil)) = rev(app_2(nil, l))$ and the target base case conjecture is
$+(sum(nil), sum(l)) = sum(app(nil, l))$.

The source rewrites in the base case are
`app1`: $app_1 0(nil_3, l) = l$
`rev1`: $rev_4(nil) = nil_2$
`sum1`: $sum(nil) = 0_2$

The IF-equations in the source base case are
$app_1 = app_3$
$app_1 = app_4$
$::_2 = ::_1$.

The base case replay of the `apprev-plussumsum` has the following steps:

- The first method, eval_def(`app1`) can be transferred to method eval_def(`app1`) because the IF-equation and preconditions are satisfied by $\mathrm{m}_e : nil \mapsto nil$.
- The next method, eval_def(`rev1`), can be transferred to method eval_def(`sum1`) because the justifications are satisfied.
- The next method, generalise, can be transferred and the term to be generalized is $sum(l)$ instead of the source term $rev(l)$.

- The resulting subgoal is $+(0, v_o) = v_0$ which is an instance of the target rewrite `plus1`. Thus the rest of the branch is skipped because of redundancy.

  (Otherwise the next method, induction justified by `app2`, could be transferred only if its justification reformulated by $m_b : app_1 \mapsto \lambda w_1 \lambda w_2. +(w_2, w_1), app_3 \mapsto \lambda w_1 \lambda w_2. +(w_2, w_1)$ to
  $+(l_2, F_1(h, l_1)) = F_1(h, +(l_2, l_1))$[20] could be instantiated to a valid lemma. Such an instantiation is $F_1(x, y) \mapsto \lambda x \lambda y.s(y)$ which is not provided by a mapping. Then the suggested lemma became $+(l_2, s(l_1)) = s(+(l_2, l_1))$. Using the suggested lemma the rest of the source proof could be transferred.)

  The resulting target proof plan is

```
induction([v0::x],[x:pnat list]) then
  [eval_def([1,2,1],app1) then
     eval_def([1,1,1],sum1) then          %POSITION CHANGED by symm
       generalise(sum(y),v0:int) then
                                          %SKIP, BASE-LEVEL yields:
         eval_def([1,1],plus1) then
           elementary(...),               %%%

   wave([1,2,1],[app2,equ(left)]) then
     wave([2,1],[sum2,equ(left)]) then
       wave([1,1,1],[sum2,equ(left)]) then  %POSITION CHANGED  by symm
         wave([1,1],[assplus,equ(left)]) then
           fertilize(weak(v1)) then
             elementary(...)
  ]
```

[20]Because of the IF-equation $::_2 = ::_1$, F_1 is the meta variable for $::_1$ and $::_2$ in the parametrized `app2` which is not yet instantiated by m_b or m_e.

10.4 `halfdouble-doublehalf`

This is an example for the application of the reformulation *condt*.

source theorem `halfdouble`: $half(double(n)) = n$
target theorem `doublehalf`: $even(n) \rightarrow double(half(n)) = n$.
The source plan is

```
induction([s(n)],[n:pnat]) then
  [eval_def([1,1,1],double1) then
     eval_def([1,1],half1) then
       elementary(...),
   wave([1,1,1],[double2,equ(left)]) then
     wave([1,1],[half3,equ(left)]) then
       fertilize(weak(v0)) then
         elementary(...)
  ]
```

The basic mapping $m_b : double \mapsto half, half \mapsto double$.
The extended mapping $m_e : half_{1,2} \mapsto double$
$double_{1,2} \mapsto half$
$s_{1,4} \mapsto \lambda x.s(s(x))$
$\lambda x.s_2(s_2(x)) \mapsto s$
$\lambda x.s_3(s_3(x)) \mapsto s$.

The reformulation *condt* replaces FERTILIZATION(weak) by VERY WEAK FERTILIZATION in the plan node with the conclusion
$even(s(s(n))) \rightarrow s(s(double(half(n)))) = s(s(n))$. VERY WEAK FERTILIZATION results in the (transferred) subgoal $s(s(n)) = s(s(n))$ and in the additional subgoal lemma1: $even(s(s(n))) \rightarrow even(n)$.

The target base case for $s(0)$ is terminated because of $even(s(0)) = false$. The target base case for $x = 0$ is transferred because of $even(0) = true$.

The resulting target proof plan is

```
induction([s(s((n))],[n:pnat]) then
  [eval_def([1],even1) then                     %ADDITIONAL EVALUATION
    eval_def([1,1,1,2],half1) then
      eval_def([1,1,2],double1) then
        elementary(...),
  [eval_def([1],even1) then                     %ADDITIONAL EVALUATION
       elementary(...),
   wave([1,1,1,2],[half2,equ(left)]) then
     wave([1,1,2],[double3,equ(left)]) then
       fertilize(very_weak(v0),lemma1) then   %REPLACED METHOD
         elementary(...)
  ]
```

10.5 `div3-div3term`

source theorem: `div3`: $\neg y = 0 \rightarrow div(plus(y,x),y) = s(div(x,y))$
target theorem: `div3term`: $\neg times(y,z) = 0 \rightarrow div(plus(y,x),y) = s(div(x,y))$.
`div3` can be planned by *CLAM*3 and has the proof plan

```
normal(imply_intro) then
     eval_def([1,1],div3) then
        elementary(...)
```

This is an example for which the not commonly loaded method NORMAL can be transferred. `div3term` can not be planned by *CLAM*3. It can, however, be planned by analogy to `div3`. The reason is that eval_def's preconditions are not satisfied. The basic mapping maps: $y \mapsto times(y,z)$ in the antecedents. Everything else is mapped identically by m_b. Note: for noninductive proofs the match constraints do not apply.

In order to establish the p-justification `true(`$\neg * (y,z) = 0 \vdash \neg y = 0$`)` base-level planning has to be invoked.

10.6 `cnc_theorems`

A class of examples for which the not commonly loaded method NORMAL can be transferred is the class of `cnc..` theorems. For instance,
source theorem: `cnc_plus`: $x = y \rightarrow plus(x,z) = plus(y,z)$
target theorem: `cnc_half`: $x = y \rightarrow half(x) = half(y)$.
All the plans (also for `cnc_times` etc.) have the structure

```
 normal(imply_intro) then
     equal(v0,left) then
         elementary(...)
```

and the planning fails without the method NORMAL.

`cnc_termplus`: $times(u,x) = times(v,y) \rightarrow plus(times(u,x),z) = plus(times(v,y),z)$ is a more complicated target for which the source can be transferred analogically.

10.7 `zerotimes1-zerotimes3`

This is another example for which the not commonly loaded method NORMAL can be transferred.

source theorem: `zerotimes1`: $x = 0 \rightarrow times(x,y) = 0$
target theorem: `zerotimes3`: $times(x,y) = 0 \rightarrow times(times(x,y),y) = 0$.

The source plan is

```
normal(imply_intro) then
     equal(v0,right) then
        eval_def([1,1],times1) then
            elementary(...).
```

The target plan is

```
normal(imply_intro) then
     equal(v0,right) then
        eval_def([1,1],times1) then
            elementary(...).
```

As opposed to a plan produced by base-level planning, the resulting target plan does not start with a GENERALISE method. m_b and the analogical replay do the job necessary. The analogical replay does not consider the precondition `freevarinterm(G,`$times(x,y)$ because it has no corresponding justification.

10.8 assapp-assplus1

source theorem: $x <> (y <> z) = (x <> y) <> z$
target theorem: $x + (x + x) = (x + x) + x$.

This is an example in which target variables are generalized apart in order to meet the b4 constraint of m_b. The result is the reformulated target `assplus2`, given below, for which the analogical replay works. *app* is not indexed because all *app*s are mapped to $+$.

The extended mapping maps for the step case `app2` to `plus2` and `app1` to `plus1` by m_e $::_{1,2}\mapsto s$ and for the base case `app1` to `plus1` by $m_e : nil \mapsto 0$. The source plan is

```
induction([v0::l],[l:pnat list]) then
  [eval_def([1,1],app1) then
     eval_def([1,2,1],app1) then
       elementary(...),
   wave([1,1],[app2,equ(left)]) then
     wave([1,2,1],[app2,equ(left)]) then
       wave([2,1],[app2,equ(left)]) then
         fertilize(weak(v1)) then
           elementary(...)
  ]
```

Source	m_b	Target
$app(l, app(m,n)) = app(app(l,m),n)$	$app \mapsto +$	After the renaming of variables $+(x, +(y,y)) = +(+(x,y),y)$

source rewrites	m_e	target rewrites
app2:$app(h ::_1 m, n) = h ::_2 app(m,n)$	$app \mapsto +$ $::_{1;2} \mapsto s$	$+(s(x),y) = s(+(x,y))$

Step case replay:

source goals	source justif.	target goals
$app(v_0 ::_1 l, app(m,n)) =$ $app(app(v_0 ::_1 l, m)n)$	app2	$+(s(x), +(y,y)) = +(+(s(x),y),y)$
$v_0 ::_2 app(l, app(m,n)) = \dots$	app2	$s(+(x, +(y,y))) = \dots$
$\dots = app(v_0 ::_2 app(l,m),n)$	app2	$\dots = +(s(+(x,y)),y)$
$\dots = v_0 ::_2 app(app(l,m),n)$	app2 $::_1 = ::_2$	$\dots = s(+(+(x,y),y))$
$v_0 ::_2 app(app(l,m),n) = \dots$	part_ih_match	$s(+(+(x,y),y)) = \dots$
$true$		$true$

Base case replay:

source rewrites	m_e	target rewrites
app1: $app(nil, l) = l$	$nil \mapsto 0$	$+(0,y) = y$

source goals	source justif.	target goals
$app(nil, app(m,n)) =$ $app(app(nil,m),n)$		$+(0, +(m,n)) = +(+(0,m),n)$
$app(m,n) = \dots$	app1	$+(m,n) = \dots$
$\dots = app(m,n)$	app1	$\dots = +(m,n)$
$true$		$true$

The resulting target proof plan is

```
induction([s(x)],[x:pnat]) then
  [eval_def([1,1],plus1) then
     eval_def([1,2,1],plus1) then
       elementary(...),
   wave([1,1],[plus2,equ(left)]) then
     wave([1,2,1],[plus2,equ(left)]) then
       wave([2,1],[plus2,equ(left)]) then
         fertilize(weak(v0)) then
           elementary(...)
  ]
```

10.9 `halfplus2-lenapp`

This is an example for the application of the reformulation *2to1*. `part_ih_match` is short for the justification `partial_ind_hyp_match(...)`.

Source	m_b	Target
$half_1(x +_1 y) = half_2(y +_2 x)$	$half_{1;2} \mapsto len$ $+_{1;2} \mapsto <>$	$len(x <> y) = len(y <> x)$

The step case replay

source rewrites	m_e	target rewrites
half3: $half_3(s_1(s_1(x))) = s_2(half_3(x))$ plus2: $s_3(x) +_3 y = s_4(x +_3 y)$ lemma:$x{+_1}s_3(s_3(y)) = s_5(s_6(x{+_1}y))$	$s_1(s_1()) \mapsto v_0 ::$ $s_2() \mapsto s()$ $half_3 \mapsto len$ $+_3 \mapsto <>$ $s_{3;4} \mapsto v_0 ::$	$len(v_0 :: x) = s(len(x))$ $(v_0 :: x) <> y = v_0 :: (x <> y)$

source goals	source justif.	target goals
$half_1(x +_1 s_3(s_3(y))) =$ $half_2(s_3(s_3(y)) +_2 x)$	plus2	after 2to1 reformulation $len(x <> (v_0 :: y)) =$ $len((v_0 :: y) <> x)$
$\ldots = half_2(s_4(s_3(y) +_3 x)$	plus2 $+_2 = +_3$	after 2to1 collapsed 2 methods $\ldots = len(v_0 :: (y <> x))$
$\ldots = half_2(s_4(s_4(y +_3 x)))$	plus2	collapsed method
$\ldots = s_2(half_3(y +_3 x))$	half3 $half_2 = half_3$ $s_4 = s_1$ (causes 2to1)	$\ldots = s(len(y <> x))$
$half_1(s_5(s_6(x +_1 y))) = \ldots$	lemma	by suggested lemma $len(v_0 :: (x <> y)) = \ldots$
$s_2(half_3(x +_1 y)) = \ldots$	half3 $half_1 = half_3$ $s_5 = s_1$ $s_6 = s_1$	$s(len(x <> y)) = \ldots$
$s_2(half_2(y +_2 x)) = \ldots$	part_ih_match $half_3 = half_1$	$s(len(y <> x)) = \ldots$
$true$	$half_2 = half_3$ $+_2 = +_3$	$true$

The source lemma provided by a critic is $x +_1 s_3(s_3(y)) = s_5(s_6(x +_1 y))$. Suggested target lemma is: $x <> (v_0 :: y) = v_0 :: (x <> y)$.

10.10 `halfplus2-evenplus`

source theorem: $half_1(x +_1 y) = half_2(y +_2 x)$
target theorem: $even(x + y) \leftrightarrow even(y + x)$.
This is an example, where the matching constraint b1 has to be relaxed and in which the application of a lemma discovery critic can be avoided by analogy.

Source	m$_b$	Target
$half_1(x +_1 y) = half_2(y +_2 x)$	$half_{1;2} \mapsto evn$ $+_{1;2} \mapsto +$	$even(x + y) \leftrightarrow even(y + x)$

step case replay:

source rewrites	m$_e$	target rewrites
	$+_3 \mapsto +$	
$s_1(x) +_3 y = s_2(x +_3 y)$	$half_3 \mapsto even$	$s(x) + y = s(x + y)$
$half_3(s_3(s_3(x))) = s_4(half_3(x))$	$s_{1;2;3} \mapsto s$	$even(s(s(x))) \leftrightarrow even(x)$
	$s_4 \mapsto$ -	

source goals	source justif.	target goals
$half_1(s_1(s_1(x)) +_1 y) =$ $half_2(y +_2 s_1(s_1(x)))$		$even(s(s(x) + y) \leftrightarrow even(y + s(s(x))$
$half_1(s_2(s_1(x) +_3 y)) = \dots$	$+_1 = +_3$	$even(s(s(x) + y)) \leftrightarrow \dots$
$half_1(s_2(s_2(x +_3 y))) = \dots$	$+_3 = +_3$	$even(s(s(x + y))) \leftrightarrow \dots$
$s_4(half_3(x +_3 y)) = \dots$	$s_2 = s_3$ $half_1 = half_3$	$even(x + y) \leftrightarrow \dots$
$\dots = s_4(half_3(y +_3 x))$	lemma	by suggested lemma $\dots \leftrightarrow even(y + x)$
$s_4(half_3(x +_3 y)) = s_4(half_3(x +_3 y))$	ih_match $half_3 = half_1$ $half_3 = half_2$ $+_3 = +_{1;2}$	$even(y + x) \leftrightarrow even(y + x)$
$true$		$true$

The source lemma, introduced by a critic, is $half_2(y +_2 s_1(s_1(x))) = s_4(half_3(y +_3 x))$. Correspondingly, the suggested target lemma is $even(y + s(s(x))) \leftrightarrow even(y + x)$.

10.11 `lenapp-halfplus2`

Source theorem: $len_1(a <>_1 b) = len_2(b <>_2 a)$
Target theorem: $half(a + b) = half(b + a)$

This is an example for the application of the reformulation *1to2* that avoids the application of the induction revision critic in the target. The replay can transfer the source base case b=0, a=0 to the target base case b=0, a=0. It replays the step case of b=0 as shown below. It breaks down for the target base case b=0,a=s(0) because of the IF-equation $nil_3 = nil_4$ which is mapped to $s(0) = 0$. As shown below the replay of the source base case b=0 for the target base case b=s(0) yields a gap at the first method eval_def(app1) because of the IF-equation $nil = nil_4$ (nil is introduced by the target induction) which is mapped to $s(0) = 0$. It produces the island conclusion $half(+(a, s(0))) =?$. With this island the replay can be continued yielding a final subgoal $s(?) =??$. The instantiation of the gap variable by $half(s(a))$ allows to complete the replay of this step case of b=s(0). The source base cases b=0, a=0 can be replayed for the target base case b=s(0), a=0 if the island is instantiated. The replay of the base case b=0, a=0 for the target base case b=s(0), a=s(0) fails because of the IF-equation $nil_3 = nil_4$ which is mapped to $s(0) = 0$.

Source	m_b	Target
$len_1(a <>_1 b) = len_2(b <>_2 a)$	$len_{1;2} \mapsto half$ $<>_{1;2} \mapsto +$	$half(a+b) = half(b+a)$

source rewrites	m_e	target rewrites
app2: $(v_0 ::_2 a) <>_3 b = v_0 ::_3 (a <>_3 b)$ len2:$len_3(v_0 ::_1 a) = s_1(len_3(a))$	$len_3 \mapsto half$ $::_1 \mapsto s(s())$ $s_1 \mapsto s$ $<>_3 \mapsto +$ $::_{2;3} \mapsto s$	$s(a) + b = s(a+b)$ $half(s(s(a))) = s(half(a))$

The step case replay:

source goals	source justif.	target goals
$len_1(a <>_1 v_0 ::_2 b) =$ $len_2(v_0 ::_2 b <>_2 a)$	app2	After 1to2 ref. because of (*) $half(a+s(s(b))) = half(s(s(b))+a)$
$\ldots = len_2(v_0 ::_3 (b <>_3 a))$	app2 $<>_2 = <>_3$	double wave method because of 1to2 $\ldots = half(s(s(b+a)))$
$\ldots = s_1(len_3(b <>_3 a))$	len2 $len_2 = len_3$ (*) $::_3 = ::_1$	$s(half(b+a))$
$s_2(len_3(a <>_3 b)) = \ldots$	lemma	by suggested lemma $s(half(a+b)) = \ldots$
$s_1(len_2(b <>_2 a)) = \ldots$	$len_3 = len_1$ $<>_3 = <>_1$	$s(half(b+a)) = \ldots$
$true$	$s_1 = s_2$ $len_2 = len_3$ $<>_2 = <>_3$	$true$

Source lemma: $len_1(a <>_1 v_0 ::_2 b) = s_2(len_3(a <>_3 b))$.
Suggested target lemma: $half(a + s(s(b))) = s(half(a+b))$.

The source base case $b = 0$ (step case) replay for the target case base $b = 0$ (step case):

source rewrites	m$_e$	target rewrites
app1: $app_6(nil_4, x) = x$	$app_6 \mapsto +$	$+(0, x) = x$
length1: $len_5(nil_5) = 0$	$nil \mapsto 0$	$half(0) = 0$
app2	$::_{2;3} \mapsto s$	plus2
length2	$::_1 \mapsto s(s())$	half3
	$len_5 \mapsto half$	

source goals	source justif.	target goals
$len_1(app_1(a, nil)) =$ $len_2(app_2(nil, a))$		$half(+(a, 0)) = half(+(0, a))$
$len_1(app_1(a, nil)) = len(a)$	app1 $app_2 = app_5$ $nil = nil_4$	$half(+(a, 0)) = half(a)$
induction on a $len_1(app_1(v_0 ::_2 a, nil)) =$ $len_2(v_0 ::_2 a)$ **step case:**	app2	$half(+(s(s(a)), 0)) = half(s(s(a)))$
$len_1(v_0 ::_3 app_4(a, nil)) = \ldots$	app2 $app_3 = app_1$	double wave[plus2] because of 1to2 $half(s(s(+(a, 0)))) = \ldots$
$s_1(len_3(app_4(a, nil))) = \ldots$	length2 $len_1 = len_3$ $::_3 = ::_1$ (causes 1to2)	$s(half(+(a, 0))) = \ldots$
$\ldots = s_1(len(a))$	length2 $len_2 = len_3$ $::_2 = ::_1$	$\ldots = s(half(z))$
$s_1(len_3(a)) = \ldots$	$len_1 = len_3$ $app_4 = app_1$	$s(half(a)) = \ldots$
true		*true*

The source base case b=0 replay for the target base case b=s(0):

source rewrites	m$_e$	target rewrites
app1: $app_6(nil_4, x) = x$	$nil \mapsto s(0)$	$+(0,x) = x$
len1: $len_5(nil_5) = 0$	$nil_4 \mapsto 0$	$half(0) = 0$
etc.		$half(s(0) = 0$

source goals	source justif.	target goals
$len_1(app_1(a, nil)) =$ $len_2(app_2(nil, a))$		$half(+(a, s(0))) =$ $half(+(s(0), a))$
$\ldots = len_2(a)$	app1 $app_2 = app_5$ $nil = nil_4$ (causes failure)	$\ldots =?$
induction on a $len_1(app_1(v_0 ::_2 a, nil)) =$ $len_2(v_0 ::_2 a)$ **step case:**	app2	$half(+(s(s(a)), s(0))) =?$
$len_1(v_0 ::_3 app_4(a, nil)) = \ldots$	app2 $app_3 = app_1$	double wave[plus2] because of 1to2 $half(s(s(+(a, s(0))))) =?$
$s_1(len_3(app_4(a, nil))) = \ldots$	length2 $len_1 = len_3$ $::_3 = ::_1$ (causes 1to2)	$s(half(+(a, s(0)))) =?$
$\ldots = s_1(len(a))$	length2 $len_2 = len_3$ $::_2 = ::_1$	$\ldots =??$
$s_1(len_3(a)) = \ldots$	$len_1 = len_3$ $app_4 = app_1$	$s(?) =??$ replay stopped

10.12 plus2right-doubleplus2right

Source theorem: $+_1(x, s_1(y)) = s_2(+_2(x, y))$
Target theorem: $double(+(x, s(y))) = s(s(double(+(x, y))))$

This is an example for the application of the reformulation *blow*.

In the step case m$_e$ maps s_4 to $\lambda x.s(s(x))$ because of the IF-equation $s_2 = s_4$ in the justification of ELEMENTARY. In the base case m$_e$ maps 0_1 to 0 because of the

IF-equation $0_1 = 0$. S_3 remains a meta-variable which is introduced because no target lemma corresponding to plus2 is available.

At the first WAVE(plus2) node the replay suggests the target lemma $double(+(S_3(x), y) = s(s(double(+(x, y))))$ corresponding to the source rewrite plus2. This lemma contains the meta variable S_3 that has to be instantiated (e.g., to s).

The replay of the base case suggests the target rewrite $double(+(0, x)) = x$ based on the source rewrite `plus1` and the IF-equations $+_1 = +_4, 0 = 0_1$. The transfer of the base case $x = 0$ fails because the suggested lemma $double(+(0, x)) = x$ is disproved. The replay stops as soon as $? = ??$ occurs. Besides, the IF-equation $s_1 = s_2$ in the elementary node of the case base contradicts the basic mapping but this reason for the replay failure is not even considered.

Source	m_b	Target
$+_1(x, s_1(y)) = s_2(+_2(x, y))$	$+_{1,2} \mapsto$ $double(+)$ $s_1 \mapsto s$ $s_2 \mapsto s(s())$	$double(+(x, s(y))) =$ $s(s(double(+(x, y))))$

source rewrites	m_e	target rewrites
plus2:$+_3(s_3(x), y) = s_4(+_3(x, y))$	$s_3 \mapsto S_3$ $s_4 \mapsto s(s())$	nothing for $double(+())$

The step case replay:

source goals	source justif.	target goals
$+_1(s_3(x), s_1(y)) = s_2(+_1(s_3(x), y))$	plus2	$double(+(S_3(x), s(y))) =$ $s(s(double(+(S_3(x), y))))$
$s_4(+_3(x, s_1(y))) = \dots$	plus2 $+_1 = +_3$	by suggested lemma $s(s(double(+(x, s(y))))) = \dots$
$\dots = s_2(s_4(+_3(x, y)))$	plus2 $+_1 = +_3$	$\dots = s(s(s(s(double(+(x, y))))))$
$s_4(s_2(+_2(x, y))) = \dots$	part_ih_match $+_3 = +_1$	$s(s(s(s(double(+(x, y)))))) = \dots$
true	$s_2 = s_4$ $+_2 = +_3$	*true*

Base case replay:

source rewrites	m_e	target rewrites
plus1: $+_4(0, x) = x$	$0_1 \mapsto 0$	suggested rewrite $double(+(0, x)) = x$ disproved

source goals	source justif.	target goals
$+_1(0, s_1(y)) = s_2(+_2(0, y))$		$double(+(0, s(y))) =$ $s(s(double(+(0, y))))$
$s_1(y) = \dots$	plus1 $+_1 = +_4$ $0 = 0_1$	no transfer $? = s(s(double(+(0, y))))$
$\dots = s_2(y)$	plus1 $+_2 = +_4$ $0 = 0_1$	no transfer ?=??
true		$s_1 = s_2$

References

[Bundy 88] A. Bundy. The use of explicit plans to guide inductive proofs. In E. Lusk and R. Overbeek, editors, *Proc. 9th International Conference on Automated Deduction (CADE)*, volume 310 of *Lecture Notes in Computer Science*, pages 111–120, Argonne, 1988. Springer.

[Bundy *et al* 91a] A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324, 1991. Earlier version available from Edinburgh as DAI Research Paper No 413.

[Bundy *et al* 91b] A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324, 1991.

[Bundy *et al* 93] A. Bundy, Stevens A, F. Van Harmelen, A. Ireland, and A. Smaill. A heuristic for guiding inductive proofs. *Artificial Intelligence*, 63:185–253, 1993.

[Carbonell 83] J.G. Carbonell. Learning by analogy: Formulating and generalizing plans from past experience. In R.S. Michalsky, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 371–392. TiogaPubl., Palo Alto, 1983.

[Carbonell 86] J.G. Carbonell. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R.S. Michalsky, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 371–392. Morgan Kaufmann Publ., Los Altos, 1986.

[Gordon *et al* 79] M. Gordon, R. Milner, and C.P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation.* Lecture Notes in Computer Science 78. Springer, Berlin, 1979.

[Hesketh 91] J.T. Hesketh. *Using Middle-Out Reasoning to Guide Inductive Theorem Proving.* Unpublished PhD thesis, University of Edinburgh, 1991.

[Huang *et al* 94a] X. Huang, M. Kerber, M. Kohlhase, E. Melis, D. Nesmith, J. Richts, and J. Siekmann. Omega-MKRP: A Proof Development Environment. In *Proc. 12th International Conference on Automated Deduction (CADE)*, Nancy, 1994.

[Huang *et al* 94b] X. Huang, M. Kerber, M. Kohlhase, and J. Richts. Methods - the basic units for planning and verifying proofs. In *Proceedings of Jahrestagung für Künstliche Intelligenz*, Saarbrücken, 1994. Springer.

[Hutter 90] D. Hutter. Guiding inductive proofs. In M.E. Stickel, editor, *Proc. of 10th International Conference on Automated Deduction (CADE)*, volume Lecture Notes in Artificial Intelligence 449. Springer, 1990.

[Hutter 94] D. Hutter. Synthesis of induction orderings for existence proofs. In A. Bundy, editor, *Proc. of 12th International Conference on Automated Deduction (CADE)*, Lecture Notes in Artificial Intelligence 814, pages 29–41. Springer, 1994.

[Ireland & Bundy 94] A. Ireland and A. Bundy. Productive use of failure in inductive proof. Technical report, Department of AI Edinburgh, 1994. Available from Edinburgh as DAI Research Paper 716.

[Kolbe & Walther 94] Th. Kolbe and Ch. Walther. Reusing proofs. In *Proceedings of ECAI-94*, Amsterdam, 1994.

[Kolbe & Walther 95] Th. Kolbe and Ch. Walther. Patching proofs for reuse. In N. Lavrac and S. Wrobel, editors, *Proceedings of the 8th European Conference on Machine Learning 1995*, Kreta, 1995.

[Kraan 94] I. Kraan. *Proof Planning for Logic Program Synthesis.* Unpublished PhD thesis, University of Edinburgh, 1994.

[Madden 91] P. Madden. *Automated Program Transformation Through Proof Transformation.* Unpublished PhD thesis, University of Edinburgh, 1991.

[Melis 95] E. Melis. A model of analogy-driven proof-plan construction. In *Proceedings of the 14th Interna-*

tional Joint Conference on Artificial Intelligence, Montreal, 1995.

[vanHarmelen *et al* 93] F. van Harmelen, A. Ireland, S.Negrete, A. Stevens, and A. Smail. The CLAM proof planner, user manual and programmers manual. Technical Report version 2.0, University of Edinburgh, Edinburgh, 1993.

[Veloso 94] M.M. Veloso. *Planning and Learning by Analogical Reasoning*. Springer, Berlin, New York, 1994.

[Villafiorita & Sebastiani 94] A. Villafiorita and R. Sebastiani. Proof planning by abstraction. In *Proceedings ECAI-94 workshop on interactive systems*, Amsterdam, 1994.