Analogy in CLAM*

Erica Melis †

Abstract

CLAM 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 CLAM can be applied to CLAM and it presents analogy-driven proof plan construction as a control strategy of CLAM. 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 CLAM 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 CLAM 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 $\Omega$-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
Analog in CIA

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. CIA’s search for methods, however, is quite restricted already because inductive theorem proving is a rather goal-directed type of mathematical reasoning and CIA uses efficient search heuristics such as rippling. Therefore analogy might be helpful or efficiently used only if CIA’s proof planning breaks down. Hence, the focus of applying analogy in CIA 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 CIA and the model of analogy-driven proof plan construction. In section 4 the purposes of analogy in CIA 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 CIA 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 CIA. For a comprehensive introduction see [Bundy et al 91a].

2 Proof Planning in CIA

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 CIA [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 \textit{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 \textit{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
\[ (s(x) + y) + z = s(x) + (y + z). \]
If we remove the structure of the not underlined parts of the boxes, then we obtain the \textit{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 C I³M have the form (see [vanHarmelen et al 93])

\begin{verbatim}
method(name(...Args...), % name slot: Prolog term
    H==>G, % input slot: sequent,
    [...]Preconditions..., % precondition-slot: list of conjuncts
    [...]Postconditions..., % postcondition-slot: list of conjuncts
    [...]Outputs..., % output slot: list of sequents
    tactic(...Args...) % tactic slot: Prolog term
\end{verbatim}

Version 3 of C I³M, to which I refer throughout this paper, works with the following methods:

- \textbf{ELEMENTARY} applies if current method is easily provable using some propositional and equational reasoning, and very elementary facts

- \textbf{EQUAL} checks whether there is any equality $\text{Term}=\text{Var}$ among the hypotheses. If so, it uses the equality to rewrite all hypotheses and the goal in a definite order @j.

- \textbf{EVAL_DEF} symbolically evaluates a term in the goal by applying one of its defining equations.

- \textbf{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 halves 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).

**CIAM3** 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. **CIAM** 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 \( \Omega \)-MKRP method calling an automated theorem prover can provide different formal proofs in the source in the target when executed. Similarly, the \textsc{elementary} method can yield different formal proofs for source and target when executed; and \textsc{wave or eval} 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 \textit{plans} if a transfer of hierarchically superior methods is possible only and the transfer of details fails.

\textit{Reformulations} do more than just instantiate variables. They are mappings of a proof plan to a proof plan\textsuperscript{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\textsuperscript{4} in order to construct a target plan. The source plan is \textit{linearised}, i.e., ordered by the sequence in which the nodes have been added to the source plan. As in [Veloso 94], \textit{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.

\textsuperscript{3}Reformulations usually, but not necessarily, preserve the verifiability of methods in a plan. In the \( \Omega \)-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 \( CI^4M \).

\textsuperscript{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 CLAM

What can analogy buy for inductive theorem proving with CLAM? 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 CLAM 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 \texttt{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 \texttt{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 \texttt{halfplus2-lenapp},

4. The analogy can guide non-inductive subproofs, e.g., in \texttt{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: \texttt{assp-assm} in case the distributivity axiom is not given for \texttt{assm},

6. Analogy may find generalisations via analogy, e.g., for \texttt{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. \texttt{NORMAL}, if this method was loaded for the source planning and is transferred to the target, for instance, for \texttt{div3-div3term},

8. Analogy can guide the application of a method even if a heuristic precondition does not hold as, e.g., for \texttt{zerotimes1-zerotimes3}. Analogy is a control strategy for planning that can overwrite heuristic preconditions.

5 \textbf{Analogy in CLAM}

In order to use the analogy-driven proof plan construction for \textit{CLAM} 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)\textsuperscript{5}. For this purpose, source function symbols at different positions are differentiated by additional indices. Because of

\textsuperscript{5}Even more elaborate mappings are conceivable that map arbitrary functions (lambda terms), to functions
the rippling heuristic in \( C I^A M \), 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 \( C I^A M \) 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 \textsc{elementary}. The IF-equations can be obtained by slightly extending the matching in \( C I^A M \). The IF-equations belong to the justifications of the respective method and restrict the analogical mappings.

\( C I^A M3 \) 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 \( C I^A M \) 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 \( C I^A M \).

### 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.
ripping 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 \texttt{lenapp} and \texttt{halfplus2} with the ripple paths as bold lines. In the figure the source IE-variable is $x$ and the target IE-variable is $x$.

A basic match establishes the basic mapping $m_b$ in order to match the parameterized 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.

\section*{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 \textit{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 $m_b(t_{s_r}) = t_{t_r}$ 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 $t_{s_r}$.

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

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.  

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: $app_2 = app_3$, $len_2 = len_3$, $::: = :::1$, $len_4 = len_1$,

$s_1 = s_2$, $len_2 = len_4$, $app_2 = app_4$, $app_4 = app_1$,

$m_b : len_{1:2} \mapsto half$, and $m_b : app_{1:2} \mapsto +$,

with the source rewrites

**app2**: $app_3((v_0 :: a, b) = v_0 :: (app_4(a, b))$ and  

**length2**: $len_3(v_0 :: a) = s_1(len_4(a))$, 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.

$\text{shrink}$ 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.\(^{10}\)

$\text{blow}$ If $m_b$ maps a source function symbol $f$ to a function $\lambda x.g_1(g_2(x))$ that has the
same (combined) labelled fragment as $f$, then $m_e$ suggests a corresponding

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 $\text{blow}$ is the introduction of a function that has the same
fragment as the identity function.

$\text{reduce}$ A normalizing reformulation\(^{11}\) adds a method $M$ (or subplan) on top of the
proof plan for which $\text{output}(M) \rightarrow \text{input}(M)$ holds. The first option of the
$\text{reduce}$ 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

\(^{10}\) This is a special case of $\lambda x.f_1(f_2(x)) \mapsto \lambda w.g(w)$ 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 \texttt{reduce} could eventually have other parameters (or submethods) that guide its actual work.

\texttt{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 $m_b : t_1 \mapsto t'_1, t_2 \mapsto t'_2$. In order to trigger \texttt{reduce}, the matching procedure has to be extended slightly.

\texttt{reduce} can ease the analogy that otherwise might suggest difficult to prove lemmas. It can also shorten target proof plans substantially\textsuperscript{12}. \texttt{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: \texttt{reduce} or \texttt{blow}, e.g., prefer \texttt{reduce} to \texttt{blow} if the mapping $m_b : t_1 \mapsto t'_1, t_2 \mapsto t'_2$ maps function symbols to function symbols only.

\texttt{symm} If the second-order match of a source theorem with a target theorem permutes arguments of $f$ for $m_b(f)$, then the labelled fragment of $f$ and $m_b(f)$ have to be permuted in the same way.

For instance, in \texttt{apprev-plussumsum} $m_b : \text{app}_1(w_1, w_2) \mapsto \lambda w_1 \lambda w_2.\text{plus}(w_2, w_1)$ requires a labelled fragment of \texttt{plus} that is symmetric to the labelled fragment of \texttt{app}_1. Those symmetric labelled fragment of \texttt{app} and \texttt{plus} are provided by the rewrites \texttt{assapp} and \texttt{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 \texttt{apprev - plussumsum} in section 10.

\texttt{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 \texttt{condt} reformulation of the proof plan replaces (weak) \texttt{fertilize} by Kraan's \texttt{very} (\texttt{weak}) \texttt{fertilize} [Kraan 94]. In the step case\textsuperscript{13} \texttt{condt} introduces into the target planning the instantiated antecedents' $C_{t_{bij}}$ of the target base cases as an additional subgoal and introduces subplans for their proof or disproof into the plan. \texttt{condt} transfers the source base case if $C_{t_{bij}}$ can be proven, and it terminates a base case if $C_{t_{bij}}$ is disproved.\textsuperscript{14} For a detailed example see \texttt{halfdouble-doublehalf} in section 10.

\textsuperscript{12}E.g., this reformulation yields a plan of \texttt{halfplus} with 13 methods, as opposed to base-level planning that comes up with a plan with 57 methods and lemma calculation.

\textsuperscript{13}\texttt{very} (\texttt{weak}) \texttt{fertilize} produces the additional lemma $C'_i \rightarrow C_t$ to be proved, where $C'_i$ is the antecedents of the induction conclusion and $C_t$ is the antecedents of the induction hypothesis.

\textsuperscript{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 \texttt{weak fertilize} by \texttt{very weak fertilize} for those specific cases because the precondition \texttt{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 \( i \)th 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 \( i \)th argument of \( f \) in the source may contain IE-variables.

### 5.2.2 IF-Reformulation

As the basic reformulation was guided by the basic match, the IF-reformulation is guided by IF-equations. This procedure can apply the reformulations Ito2 or 2to1:

**Ito2** 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 \( m_c(f_j)(m_c(f_j)) = m_c(f_i) \), 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 \( m_c \). Certain features of \( m_c \) require additional changes of the source proof plan:

**abs** If \( m_c \) maps a function symbol \( f \) to a lambda term with an unspecified non-constant argument \( o \) that can have different instantiations, then a replayed
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. 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 won't 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: \texttt{plus}(s(x), y) = s(\texttt{plus}(x, y)) \text{ to}

union4: \neg \texttt{member}(x, y) \rightarrow \texttt{union}(h :: x, y) = h :: \texttt{union}(x, y)

by m_e : \texttt{plus} \mapsto \texttt{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

---

15Latest when the instantiation is relevant to satisfy a method justification

16The name **assp** 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 \(?\). 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, \texttt{assp-assm} or \texttt{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 \texttt{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 \texttt{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 \textit{CLAM} justifications exist in a bigger variety and play a more important role than they do in \textit{OMKRP}, so far. The reason is that \textit{CLAM}'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 \textit{CLAM} method exist. I analyse how justifications can be constructed for nodes of \textit{CLAM}'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 \( C I^A M \) methods have hybrid preconditions, e.g., \texttt{trivial(H=>C)} that is classified \( h,p \). The actual classification of the preconditions of \( C I^A M^3 \) 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 \( C I^A M^3 \), e.g., an l-precondition of the method \texttt{EQUAL} is that an equation occurs in an hypothesis.

- **e-preconditions yield e-justifications.** In \( C I^A M^3 \) 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 \( C I^A M^3 \), e.g., a heuristic precondition of the method \texttt{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 \( C I^A M^3 \), e.g., a p-precondition of the method \texttt{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 chosen 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 justified 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 asp-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 IF-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 I-justification cannot be satisfied and does not contain a gap-meta-variable $?_j$, then the source method is not transferred. If an I-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 $?_1$ 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, asp-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 \((H \Rightarrow C)\) 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: \(\text{div3-div3term}\).

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 pa-
   rametrized source that have not yet been mapped by \(m_0\) and \(m_0\). (For an
   example see \(\text{plus2right-doubleplus2right}\).) The suggestion of rewrites
   should be accompanied by a disprove procedure\(^\text{17}\). (See, e.g., base case of
   example \(\text{plus2right-doubleplus2right}\) 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
  \(\text{zerotimes1}: x = 0 \rightarrow \text{times}(x, y) = 0\) is replayed for the target
  \(\text{zerotimes3}: \text{times}(x, y) = 0 \rightarrow \text{times}(\text{times}(x, y), y) = 0\), then the h-
  precondition of the \(\text{EQUAL}\) method \(\text{freevarinterm}(G, \text{Var})\) that holds in
  the source with \(\text{Var} = x\) does not hold for \(m_0(x) = \text{times}(x, y)\). Since this
  precondition is ignored for the justifications, the replay transfers the \(\text{EQUAL}\)
  method nevertheless.

- The same holds for the class of \(\text{cnc}\-examples\) such as
  source: \(\text{cnc\_times}: x = y \rightarrow \text{app}(x, z) = \text{app}(y, z)\)
  target: \(\text{cnc\_termplus}: \#(u, x) = \#(v, y) \rightarrow +(#(u, x), z) = +(#(v, y), z)\)

- If we took the precondition \(\text{trivial}(H \Rightarrow C)\) as a justification, the analogical
  replay of the following example would not work:
  source: \(\text{div3}: \neg y = 0 \rightarrow \text{div}(+(y, x), y) = s(\text{div}(x, y))\)
  target: \(\text{div3term}: \neg \text{*(}(y, z) = 0 \rightarrow \text{div}(+(y, x), y) = s(\text{div}(x, y))\).

The h,p-precondition \(\text{trivial}(H \Rightarrow C)\) of the method \(\text{EVAL\_DEF}\) yields the
source p-justification \((\neg y = 0 \rightarrow \neg y = 0)\) which is mapped to the target
justification \((\neg \text{*(}(y, z) = 0 \rightarrow \neg y = 0)\). This target justification can be
established by base-level planning for \((\neg \text{*(}(y, z) = 0 \rightarrow \neg y = 0)\). Since the
planning succeeds, the replay transfers the method \(\text{EVAL\_DEF}\).

---

\(^\text{17}\) available in \(\text{CLAM3.2}\) that can prove many wrong suggestions to be false
6 Related Work

Villaflorita [Villaflorita & 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 structures 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 $CIA$. 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 $CIA$ will be continued and will examine more challenging examples that $CIA$ 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 $CIA$ and discussions on analogy in $CIA$, in particular Alan Bundy, Andrew Ireland, Alan Smaill, and Jon Whittle.
9 Appendix I

The classification of preconditions, as it follows for CI^4M3 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,R)],
    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 \( m : + \mapsto \lambda a \lambda b. \text{half}(+(a,b)) \) 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 \( \text{REDUCE(EQUALITY)} \) 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(...)
    ]
]
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

\[
\text{induction}([s_1(x)], [x: \text{pnat}]) \text{ then }
\]
\[
\text{eval_def}([1,1], \text{plus1}) \text{ then }
\]
\[
\text{eval_def}([1,2,1], \text{plus1}) \text{ then }
\]
\[
\text{elementary}(\ldots),
\]
\[
\text{wave}([1,1], [\text{plus2, equ(left)}]) \text{ then }
\]
\[
\text{wave}([1,2,1], [\text{plus2, equ(left)}]) \text{ then }
\]
\[
\text{wave}([2,1], [\text{plus2, equ(left)}]) \text{ then }
\]
\[
\text{fertilize(weak(v0)) then }
\]
\[
\text{elementary}(\ldots)
\]

- 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:

\[
+_3(s_1(x), y) \Rightarrow s_2(+_5(x, y))
\]

and the only given target rewrite is

\[
*(s(x), y) = +(*(x, y), y).
\]

The IF-equations of the source are:

\[
+_1 = +_3
\]

\[
+_3 = +_5
\]

\[
+_4 = +_5
\]

\[
s_1 = s_2.
\]

\( m_e \) maps \( s_1 \mapsto s(w_1) \) and \( s_2 \mapsto +_3(w, o) \) for an abstract symbol \( o \) that may have different instantiations.

- The first method is \( \text{INDUCTION}(s_1(x)) \), justified by plus2.

- The step case goal in the source is: \( +_1(s_1(x), +_2(y, z)) = +_4(+_3(s_1(x), y), z) \)

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:
\[
s_2(\frac{+_3(x, +_2(y, z))}{} = \ldots \text{ and the corresponding target goal is}
\]
\[
+((x, *(y, z)), o_1) = \ldots \text{ The instantiation } *(y, z) \text{ of the meta-variable } o_1 \text{ is obtained by running the wave method and it is stored.}
\]

• The next method, \textsc{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:
\[
s_2(\frac{+_5(x, +_2(y, z))}{} = +_4\left(s_2(\frac{+_5(x, y)}{}), z\right). \text{ The corresponding target goal is}
\]
\[
+((x, *(y, z)), o_1) = *(+((x, y), o_2), z). \text{ The instantiation } y \text{ of the meta-variable } o_2 \text{ is obtained by running the wave method and it is stored.}
\]

• The next method, \textsc{wave(plus2)(\textbf{*})}, 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 \textbf{gap} and produces an \textbf{island} node with no method and with the conclusion
\[
\text{I: } +((x, *(y, z)), o_1) = ?.
\]

• The next source method, \textsc{fertilize(weak)}, has \(+_5 = +_1\) as a justification which holds in the target. An \(l\)-justification of the node is

\[\text{partial ind hyp match}(\omega) \equiv [+1(x, +_2(y, z))] = [+4(s_2(\frac{+_5(x, y)}{}), z)], [s_2(\frac{+_5(x, y)}{}), z]) = s_2(\frac{+_5(x, y)}{}, z). \text{ The corresponding target justification}
\]
\[\text{partial ind hyp match}(\omega) \equiv *[*(x, y), z)] = *[*(x, y), z)] = ?), \text{ holds. The method yields the subgoal } +((x, y), z), o_1) = ?.
\]

• The next source method, \textsc{elementary}, has the justifications \(+_4, +_3 = +_5\), which holds in the target, and \textbf{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 \(\ldots = *(+(x, y), z)\). The \textsc{wave} method justified by the distributivity rewrite (\textsc{distr}) can close this gap. In fact, the justification \(s_1 = s_2\) of \textsc{wave(plus2)(\textbf{*})} is mapped to \(+(\omega, o) = (\omega, o)\) in case \(m_e\) maps \textbf{plus2} to \textbf{distr}: \(+(x, z), y) = +(x, y), *(z, y))\).

If \textbf{distr} was given as a target rewrite, the source rewrite \textbf{plus2} might have two different images under \(m_e\), namely \textbf{times2} and \textbf{distr}. In such a situation the extended matching provides \(m_e : +_51 \mapsto *, s_{11} \mapsto s, s_{21} \mapsto +((\omega, o)\) (for \textbf{plus2} \mapsto \textbf{times2}) and
\[
m_e : +_52 \mapsto *, s_{12} \mapsto +(\omega, o), s_{22} \mapsto +(\omega, o) \text{ (for } \textbf{plus2} \mapsto \textbf{distr}). \text{ In order to satisfy}
\]
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 $*(0, *(y, z)) = *(0, y)$, 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:

$+_1 = +_7$,
$0 = 0_1$,
$+_3 = +_7$,
$+_2 = +_4$.

$m_e : +_7 \mapsto *, 0_1 \mapsto 0$. 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

\[
\text{induction}([s(x)], [x:\text{pnat}]) \text{ then}\]
\[
\begin{array}{l}
\text{eval_def}([1,1], \text{times1}) \text{ then} \\
\quad \text{eval_def}([1,2,1], \text{times1}) \text{ then} \\
\quad \quad \text{eval_def}([2,1], \text{times1}),\text{then} \\
\quad \quad \quad \text{elementary}(\ldots), \text{%%}
\end{array}
\]

\[
\begin{array}{l}
\text{wave}([1,1], [\text{times2, equ(left)}]) \text{ then} \\
\quad \text{wave}([1,2,1], [\text{times2, equ(left)}]) \text{ then} \\
\quad \text{wave}([2,1], [\text{distr, equ(left)}]) \text{ then} \\
\quad \quad \text{fertilize(weak(v0)) then} \\
\quad \quad \quad \text{elementary}(\ldots)
\end{array}
\]
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 $m$ and sets of IF-equations.

Source theorem: $\text{app}_1(\text{rev}(y), \text{rev}(x)) = \text{rev}(\text{app}_2(x, y))$

Target theorem: $+(\text{sum}(x), \text{sum}(y)) = \text{sum}(\text{app}(x, y))$

Source rewrites:
\begin{align*}
\text{rev}_2 &: \text{rev}_2(h :: l) = \text{app}_3(\text{rev}_2(l), h :: \text{nil}) \\
\text{app}_2 &: \text{app}_3(h :: l_1, l_2) = h :: \text{app}_3(l_1, l_2) \\
\text{assapp} &: \text{app}_6(l, \text{app}_7(m, n)) = \text{app}_8(\text{app}_6(l, m), n).
\end{align*}

The **IF-equations** of the source planning are:
\begin{align*}
\text{app}_1 &= \text{app}_6 \\
\text{app}_2 &= \text{app}_3 \\
\text{app}_6 &= \text{app}_7 \\
\text{rev} &= \text{rev}_2 \\
::1 ::2 ::3.
\end{align*}

The source proof plan is
\begin{verbatim}
induction([v0::m],[m:int list]) then
  [eval_def([1,2,1],app1) then
    eval_def([2,1,1],rev1) then
      generalise(\text{rev}(1),v0:int list) then
        induction([v1::v0],[v0:int list]) then
          [eval_def([1,1],app1) then
            elementary(...),
            wave([1,1],[\text{app}_2,\text{equ(left)}]) then
              fertilize(\text{weak}(\text{v2})) then
                elementary(...)
            ]
        ],
        wave([1,2,1],[\text{app}_2,\text{equ(left)}]) then
        wave([2,1],[\text{rev}_2,\text{equ(left)}]) then
        wave([2,1,1],[\text{rev}_2,\text{equ(left)}]) then
        wave([1,1],[\text{assapp},\text{equ(left)}]) then
            fertilize(\text{weak}(\text{v1})) then
                elementary(...)
  ]
\end{verbatim}

Given target rewrites are:
\begin{align*}
\text{app2} &: \text{app}(h :: l_1, l_2) = h :: \text{app}(l_1, l_2) \\
\text{sum2} &: \text{sum}(h :: l) = +(h, \text{sum}(l)) \\
\text{plus2} &: +(s(x), y) = s(+s(x), y).
\end{align*}
The basic mapping is

\[
m_b: \text{app}_2(w_1, w_2) \mapsto \text{app}(w_1, w_2),
\]

\[
\text{app}(w_1, w_2) \mapsto +({w_2}, w_1)
\]

\[
\text{rev}(w_1) \mapsto \text{sum}(w_1).
\]

\[
m_c: \text{app}_3(w_1, w_2) \mapsto \lambda w_1 \lambda w_2. \text{plus}(w_2, w_1)
\]

requires a labelled fragment of \text{plus} that is symmetric to the labelled fragment of \text{app}_1 that originates in \text{assapp}. \text{plus}_2 provides such a symmetric labelled fragment.

The extended match yields \( m_c : \cdots \mapsto \cdots (w_1, w_2) \mapsto \lambda w_1 \lambda w_2. \text{app}_3(w_1, w_2) \mapsto +({w_2}, w_1) \) matches \text{rev}_2 with \text{sum}_2 and \text{app}_2 with \text{app}_2. \text{assapp} can not be matched consistently with \text{plus}_2 but \( m_b \) and \( m_c \) suggest a target lemma \text{assplus}:

\[
+({+(n, m), l}) = +({n, +(m, l)}).
\]

\( m_c \) has been constructed following the sequence \text{app}_2, \text{rev}_2, \text{assapp}, \) in which the rewrites were used in the source planning. If the \( m_c \) 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 \text{app}_2, \text{assapp}, \text{rev}_2, \) we obtain \( m_c : \text{app}_3 \mapsto \lambda w_1 \lambda w_2. s(w_1) \) mapping \text{assapp} to \text{plus}_2. Then a target lemma \( s(h : l) = s(s(l)) \) corresponding to \text{rev}_2 would be suggested which is wrong since it yields \( s(s(l)) = +({h, s(l)}) \).

The source base case conjecture is

\[
\text{app}_1(\text{rev}(l), \text{rev}(\text{nil})) = \text{rev}(\text{app}_2(\text{nil}, l))
\]

and the target base case conjecture is

\[
+({\text{sum}(\text{nil}), \text{sum}(l)}) = \text{sum}(\text{app}(\text{nil}, l)).
\]

The source rewrites in the base case are

\[
\text{app}_1: \text{app}_1(\text{nil}_3, l) = l
\]

\[
\text{rev}_1: \text{rev}_1(\text{nil}) = \text{nil}_2
\]

\[
\text{sum}_1: \text{sum}(\text{nil}) = 0_2
\]

The IF-equations in the source base case are

\[
\text{app}_1 = \text{app}_3
\]

\[
\text{app}_1 = \text{app}_4
\]

\[
:\cdots = :\cdots.
\]

The base case replay of the \text{apprev- plussumsum} has the following steps:

- The first method, \text{eval}_d(\text{app}_1) \) can be transferred to method \text{eval}_d(\text{app}_1) \) because the IF-equation and preconditions are satisfied by \( m_c : \text{nil} \mapsto \text{nil}. \)

- The next method, \text{eval}_d(\text{rev}_1), \) can be transferred to method \text{eval}_d(\text{sum}_1) \) because the justifications are satisfied.

- The next method, \text{generalise}, can be transferred and the term to be generalized is \( \text{sum}(l) \) instead of the source term \( \text{rev}(l) \).
The resulting subgoal is \((0, v_0) = v_0\) which is an instance of the target rewrite \texttt{plus1}. Thus the rest of the branch is skipped because of redundancy.

(Otherwise the next method, induction justified by \texttt{app2}, could be transferred only if its justification reformulated by \(m_0 : \text{app1} \mapsto \lambda w_1 \lambda w_2.+(w_2, w_1), \text{app2} \mapsto \lambda w_1 \lambda w_2.+(w_2, w_1)\) to \(+(l_2, F_1(h, l_1)) = F_1(h, +(l_2, l_1))\)\footnote{Because of the IF-equation \(\vdash_2\vdash_1\), \(F_1\) is the meta variable for \(\vdash_1\) and \(\vdash_2\) in the parametrized \texttt{app2} which is not yet instantiated by \(m_0\) or \(m_e\).} 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

\[
\text{induction([v0::x],[x::nat list]) then}
\]

\[
[\text{eval_def([1,2,1],app1) then}
\]

\[
\text{eval_def([1,1,1],sum1) then}
\]

\[
\text{generalise(sum(y),v0:int) then}
\]

\[
\text{eval_def([1,1],plus1) then}
\]

\[
\text{elementary(...),}
\]

\[
\text{%POSITION CHANGED by symm}
\]

\[
\text{eval_def([1,2,1],app2,equ(left)]) then}
\]

\[
\text{wave([2,1],[sum2,equ(left)]) then}
\]

\[
\text{wave([1,1],[sum2,equ(left)]) then}
\]

\[
\text{wave([1,1],[assplus,equ(left)]) then}
\]

\[
\text{fertilize(weak(v1)) then}
\]

\[
\text{elementary(...)}
\]

\[
\text{]}
\]


10.4 halfdouble-doublehalf

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

source theorem halfdouble: \( \text{half}(\text{double}(n)) = n \)
target theorem doublehalf: \( \text{even}(n) \rightarrow \text{double}(\text{half}(n)) = n \).

The source plan is

\[
\text{induction}([s(n)], [n:\text{pnat}]) \quad \text{then}
\quad \begin{array}{c}
\text{eval}([1,1,1], \text{double1}) \quad \text{then} \\
\quad \begin{array}{c}
\text{eval}([1,1], \text{half1}) \quad \text{then} \\
\quad \text{elementary}(...),
\end{array} \\
\quad \begin{array}{c}
\text{wave}([1,1,1], [\text{double2}, \text{equ(left)}]) \quad \text{then} \\
\quad \begin{array}{c}
\text{wave}([1,1], [\text{half3}, \text{equ(left)}]) \quad \text{then} \\
\quad \text{fertilize}(<\text{weak}(v0)>) \quad \text{then} \\
\quad \text{elementary}(...) \\
\end{array},
\end{array}
\end{array}
\]

The basic mapping \( m_b : \text{double} \mapsto \text{half}, \text{half} \mapsto \text{double} \).
The extended mapping \( m_e : \text{half}_{1,2} \mapsto \text{double} \)
double_{1,2} \mapsto \text{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
\( \text{even}(s(s(n))) \rightarrow s(s(\text{double}(\text{half}(n)))) = s(s(n)). \)
VERY WEAK FERTILIZATION results in the (transferred) subgoal \( s(s(n)) = s(s(n)) \) and in the additional subgoal lemma:\( \text{even}(s(s(n))) \rightarrow \text{even}(n). \)

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

The resulting target proof plan is

\[
\text{induction}([s(s((n))], [n:\text{pnat}]) \quad \text{then}
\quad \begin{array}{c}
\text{eval}([1], \text{even1}) \quad \text{then} \quad \%\text{ADDITIONAL EVALUATION} \\
\quad \begin{array}{c}
\text{eval}([1,1,1,2], \text{half1}) \quad \text{then} \\
\quad \begin{array}{c}
\text{eval}([1,1,2], \text{double1}) \quad \text{then} \\
\quad \text{elementary}(...),
\end{array} \\
\end{array} \\
\quad \begin{array}{c}
\text{eval}([1,1], \text{half2}) \quad \text{then} \\
\quad \begin{array}{c}
\text{eval}([1,2], \text{double2}) \quad \text{then} \\
\text{wave}([1,1,2,2], [\text{half2}, \text{equ(left)}]) \quad \text{then} \\
\text{wave}([1,1,2], [\text{double3}, \text{equ(left)}]) \quad \text{then} \\
\text{fertilize}(<\text{very\_weak}(v0), \text{lemma1}) \quad \text{then} \\
\text{elementary}(...),
\end{array}
\end{array} \\
\end{array}
\]

\[ ]
10.5 div3–div3term

source theorem: \( \text{div3} \): \(-y = 0 \rightarrow \text{div}(\text{plus}(y, x), y) = s(\text{div}(x, y))\)

target theorem: \( \text{div3term} \): \(-\text{times}(y, z) = 0 \rightarrow \text{div}(\text{plus}(y, x), y) = s(\text{div}(x, y))\).

div3 can be planned by CL\(^3\)M and has the proof plan

\[
\text{normal(\text{imply_intro}) then}
\]

\[
\text{eval_def}([1,1], \text{div3}) \text{ then}
\]

\[
\text{elementary(\ldots)}
\]

This is an example for which the not commonly loaded method \text{NORMAL} can be transferred. div3term can not be planned by CL\(^3\)M. It can, however, be planned by analogy to div3. The reason is that eval_def's preconditions are not satisfied. The basic mapping maps: \( y \mapsto \text{times}(y, z) \) in the antecedents. Everything else is mapped identically by \( m_0 \). Note: for noninductive proofs the match constraints do not apply.

In order to establish the p-justification \( \text{true}(\neg \ast (y, z) = 0 \vdash \neg y = 0) \) base-level planning has to be invoked.

10.6 cnc_theorems

A class of examples for which the not commonly loaded method \text{NORMAL} can be transferred is the class of cnc.. theorems. For instance,

source theorem: \( \text{cnc\_plus} \): \( x = y \rightarrow \text{plus}(x, z) = \text{plus}(y, z) \)

target theorem: \( \text{cnc\_half} \): \( x = y \rightarrow \text{half}(x) = \text{half}(y) \).

All the plans (also for \( \text{cnc\_times} \) etc.) have the structure

\[
\text{normal(\text{imply_intro}) then}
\]

\[
\text{equal(v0,left) then}
\]

\[
\text{elementary(\ldots)}
\]

and the planning fails without the method \text{NORMAL}.

\( \text{cnc\_termpplus} \): \( \text{times}(u, x) = \text{times}(v, y) \rightarrow \text{plus}(\text{times}(u, x), z) = \text{plus}(\text{times}(v, y), z) \)

is a more complicated target for which the source can be transferred analogically.

10.7 zero\text{times1}–zero\text{times3}

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

source theorem: \( \text{zero\_times1} \): \( x = 0 \rightarrow \text{times}(x, y) = 0 \)

target theorem: \( \text{zero\_times3} \): \( \text{times}(x, y) = 0 \rightarrow \text{times}(\text{times}(x, y), y) = 0 \).

The source plan is
normal(\text{implies intro}) \text{ then}
\quad \text{equal}(v_0, \text{right}) \text{ then}
\quad \text{eval-def}([1,1], \text{times1}) \text{ then}
\quad \text{elementary}(\ldots).

The target plan is

\begin{align*}
\text{normal}(\text{implies intro}) \text{ then} \\
\text{equal}(v_0, \text{right}) \text{ then} \\
\text{eval-def}([1,1], \text{times1}) \text{ then} \\
\text{elementary}(\ldots).
\end{align*}

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

### 10.8 assapp-\text{assplus1}

source theorem: $x \leftrightarrow (y \leftrightarrow z) = (x \leftrightarrow y) \leftrightarrow z$

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 \text{assplus2}, given below, for which the analogical replay works. \text{app} is not indexed because all \text{apps} are mapped to $\plus$.

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

\begin{align*}
\text{induction}([v_0;1],[1:\text{pnat list}]) \text{ then} \\
\quad [[\text{eval-def}([1,1], \text{app1}) \text{ then} \\
\quad \quad \text{eval-def}([1,2,1], \text{app1}) \text{ then} \\
\quad \quad \text{elementary}(\ldots), \\
\quad \quad \text{wave}([1,1], [\text{app2}, \text{equ(left)})]) \text{ then} \\
\quad \quad \text{wave}([1,2,1], [\text{app2}, \text{equ(left)})]) \text{ then} \\
\quad \quad \text{wave}([2,1], [\text{app2}, \text{equ(left)})]) \text{ then} \\
\quad \quad \text{fertilize}(\text{weak}(v_1)) \text{ then} \\
\quad \quad \text{elementary}(\ldots) \\
\quad ]
\end{align*}
Analogy in \(C^AM\)

<table>
<thead>
<tr>
<th>Source</th>
<th>(m_b)</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>(app(l, app(m, n)) = app(app(l, m), n))</td>
<td>(app \mapsto +)</td>
<td>After the renaming of variables</td>
</tr>
<tr>
<td>(s)ource (r)e(w)(rites)</td>
<td>(m_c)</td>
<td>target (r)e(w)(rites)</td>
</tr>
<tr>
<td>(app_2 app(h ::_1 m, n) = h ::_2 app(m, n))</td>
<td>(app \mapsto +)</td>
<td>(+s(x), y) = +s(+x, y))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step case replay:</th>
<th>source goals</th>
<th>source justif.</th>
<th>target goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>(app(v_0 ::_1 l, app(m, n)) =)</td>
<td>(app(app(v_0 ::_1 l, m)n))</td>
<td>(app_2)</td>
<td>(+s(x), y) = +s(+x, y))</td>
</tr>
<tr>
<td>(v_0 ::_2 l, app(m, n)) = \ldots)</td>
<td>(app_2)</td>
<td>(+s(x), y) = \ldots)</td>
<td></td>
</tr>
<tr>
<td>(\ldots = app(v_0 ::_2 l, m, n))</td>
<td>(app_2)</td>
<td>(\ldots = s(+x, y), y))</td>
<td></td>
</tr>
<tr>
<td>(\ldots = v_0 ::_2 app(l, m, n))</td>
<td>(app_2)</td>
<td>(\ldots = s(+x, y), y))</td>
<td></td>
</tr>
<tr>
<td>(v_0 ::_2 app(app(l, m), n) = \ldots)</td>
<td>(part_match)</td>
<td>(+s(+x, y), y) = \ldots)</td>
<td></td>
</tr>
<tr>
<td>(true)</td>
<td></td>
<td>(true)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Base case replay:</th>
<th>source (r)e(w)(rites)</th>
<th>(m_c)</th>
<th>target (r)e(w)(rites)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(app1: app(nil, l) = l)</td>
<td>(nil \mapsto 0)</td>
<td>(+0, y) = y)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>source goals</th>
<th>source justif.</th>
<th>target goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>(app(nil, app(m, n)) =)</td>
<td>(app(app(nil, m), n))</td>
<td>(+0, +m, n) = +(+0, m, n))</td>
</tr>
<tr>
<td>(app(m, n) = \ldots)</td>
<td>(app1)</td>
<td>(+m, n) = \ldots)</td>
</tr>
<tr>
<td>(\ldots = app(m, n))</td>
<td>(app1)</td>
<td>(\ldots = +m, n))</td>
</tr>
<tr>
<td>(true)</td>
<td></td>
<td>(true)</td>
</tr>
</tbody>
</table>
The resulting target proof plan is

\[
\text{induction}([s(x)], [x: \text{nat}]) \text{ then}
\]
\[
\begin{align*}
\text{eval_def}([1,1], \text{plus1}) \text{ then} \\
\text{eval_def}([1,2,1], \text{plus1}) \text{ then} \\
\text{elementary}(\ldots), \\
\text{wave}([1,1], [\text{plus2}, \text{equ(left)}]) \text{ then} \\
\text{wave}([1,2,1], [\text{plus2}, \text{equ(left)}]) \text{ then} \\
\text{wave}([2,1], [\text{plus2}, \text{equ(left)}]) \text{ then} \\
fertilize(\text{weak(v0)}) \text{ then} \\
\text{elementary}(\ldots)
\end{align*}
\]
### 10.9 halfplus2-lenapp

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

<table>
<thead>
<tr>
<th>Source</th>
<th>$m_b$</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>$half_1(x_1 + y) = half_2(y + 2x)$</td>
<td>$half_{1,2} \mapsto \text{len}$</td>
<td>$\text{len}(x &lt;&gt; y) = \text{len}(y &lt;&gt; x)$</td>
</tr>
</tbody>
</table>

The step case replay

- **source rewrites**
  - $s_1(s_1()) \mapsto v_0 ::$
  - $s_2() \mapsto s()$
  - $half_3 \mapsto \text{len}$
  - $+3 \mapsto <>$
  - $s_{3,4} \mapsto v_0 ::$

- **target rewrites**
  - $\text{len}(v_0 :: x) = s(\text{len}(x))$
  - $(v_0 :: x) <> y = v_0 :: (x <> y)$

<table>
<thead>
<tr>
<th>source goals</th>
<th>source justif.</th>
<th>target goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>$half_1(x_1 + s_3(s_3(y))) =$</td>
<td><code>plus2</code></td>
<td>after <code>2tol</code> reformulation</td>
</tr>
<tr>
<td>$half_2(s_3(s_3(y)) + 2x)$</td>
<td><code>plus2</code></td>
<td>$\text{len}(x &lt;&gt; (v_0 :: y)) =$</td>
</tr>
<tr>
<td>...</td>
<td><code>plus2</code></td>
<td>$\text{len}((v_0 :: y) &lt;&gt; x)$</td>
</tr>
<tr>
<td>... = $half_2(s_4(s_3(y) + 3x))$</td>
<td><code>plus2</code></td>
<td>after <code>2tol</code> collapsed 2 methods</td>
</tr>
<tr>
<td>... = $half_2(s_4(s_4(y + 3x)))$</td>
<td><code>half3</code></td>
<td>collapsed method</td>
</tr>
<tr>
<td>... = $s_2(half_3(y + 3x))$</td>
<td><code>half3</code></td>
<td>$\text{len}(y &lt;&gt; x)$</td>
</tr>
<tr>
<td>$half_1(s_5(s_6(x + y))) =$</td>
<td><code>lemma</code></td>
<td>by suggested lemma</td>
</tr>
<tr>
<td>...</td>
<td><code>half3</code></td>
<td>$\text{len}(v_0 :: (x &lt;&gt; y)) =$</td>
</tr>
<tr>
<td>$s_2(half_3(x + 1y)) =$</td>
<td><code>half3</code></td>
<td>$\text{len}(x &lt;&gt; y) =$</td>
</tr>
<tr>
<td>$s_5 = s_1$</td>
<td><code>half1</code></td>
<td>$s(\text{len}(x &lt;&gt; y)) =$</td>
</tr>
<tr>
<td>$s_6 = s_1$</td>
<td><code>half3</code></td>
<td>$s(\text{len}(y &lt;&gt; x)) =$</td>
</tr>
<tr>
<td>$s_2(half_2(y + 2x)) =$</td>
<td><code>part_ih_match</code></td>
<td></td>
</tr>
<tr>
<td>$half_3 = half_1$</td>
<td><code>half3</code></td>
<td>$s(\text{len}(y &lt;&gt; x)) =$</td>
</tr>
<tr>
<td>true</td>
<td>$half_2 = half_3$</td>
<td>true</td>
</tr>
<tr>
<td>$+2 = +3$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The source lemma provided by a critic is $x + 1 s_3(s_3(y)) = s_5(s_6(x + y))$. Suggested target lemma is: $x <> (v_0 :: y) = v_0 :: (x <> y)$. 
10.10 halfplus2-evenplus

source theorem: \(\text{half}_1(x + 1 \cdot y) = \text{half}_2(y + 2 \cdot x)\)

target theorem: \(\text{even}(x + y) \leftrightarrow \text{even}(y + x)\).

This is an example, where the matching constraint \(b_1\) has to be relaxed and in which the
application of a lemma discovery critic can be avoided by analogy.

<table>
<thead>
<tr>
<th>Source</th>
<th>(m_b)</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{half}_1(x + 1 \cdot y) = \text{half}_2(y + 2 \cdot x))</td>
<td>(\text{half}_1: 2 \leftrightarrow \text{even}) (+1: 2 \leftrightarrow +)</td>
<td>(\text{even}(x + y) \leftrightarrow \text{even}(y + x))</td>
</tr>
</tbody>
</table>

step case replay:

<table>
<thead>
<tr>
<th>source rewrites</th>
<th>(m_r)</th>
<th>target rewrites</th>
</tr>
</thead>
</table>
| \(s_1(x) + 3 \cdot y = s_2(x + 3 \cdot y)\) | \(+3 \leftrightarrow +\) | \(s_1(x) + 3 \leftrightarrow \text{even}\) \(s_1: 2: 3 \leftrightarrow s\)
| \(\text{half}_3(s_3(s_3(x))) = s_4(\text{half}_3(x))\) | \(\text{half}_3 \leftrightarrow \text{even}\) \(s_2 = s_3\) | \(\text{even}(s(s(x))) \leftrightarrow \text{even}(x)\) \(\text{half}_1 = \text{half}_3\) |

<table>
<thead>
<tr>
<th>source goals</th>
<th>source justif.</th>
<th>target goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{half}_1(s_1(s_1(x)) + 1 \cdot y) = \text{half}_2(y + 2 \cdot s_1(s_1(x))))</td>
<td>(+1 = +3)</td>
<td>(\text{even}(s(s(x)) + y) \leftrightarrow \text{even}(y + s(s(x))))</td>
</tr>
<tr>
<td>(\text{half}_1(s_2(s_1(x) + 3 \cdot y)) = \ldots)</td>
<td>(+3 = +3)</td>
<td>(\text{even}(s(s(x) + y)) \leftrightarrow \ldots)</td>
</tr>
<tr>
<td>(\text{half}_1(s_2(s_2(x + 3 \cdot y)) = \ldots)</td>
<td>(+3 = +3)</td>
<td>(\text{even}(s(s(x + y)) + y) \leftrightarrow \ldots)</td>
</tr>
<tr>
<td>(s_4(\text{half}_3(x + 3 \cdot y)) = \ldots)</td>
<td>(s_2 = s_3)</td>
<td>(\text{even}(x + y) \leftrightarrow \ldots)</td>
</tr>
<tr>
<td>(\ldots = s_4(\text{half}_3(y + 3 \cdot x)))</td>
<td>(\text{half}_1 = \text{half}_3)</td>
<td>(\text{by suggested lemma})</td>
</tr>
<tr>
<td>(\ldots \leftrightarrow \text{even}(y + x))</td>
<td>(\text{ih} \cdot \text{match})</td>
<td></td>
</tr>
</tbody>
</table>

The source lemma, introduced by a critic, is \(\text{half}_2(y + 2 \cdot s_1(s_1(x))) = s_4(\text{half}_3(y + 3 \cdot x))\).
Correspondingly, the suggested target lemma is \(\text{even}(y + s(s(x))) \leftrightarrow \text{even}(y + x)\).
10.11 lenapp-halfplus2

Source theorem: \( \text{len}_1(a <>, b) = \text{len}_2(b <>, a) \)
Target theorem: \( \text{half}(a + b) = \text{half}(b + a) \)

This is an example for the application of the reformulation \( \text{Ito2} \) 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 \( \text{nil}_3 = \text{nil}_4 \) 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 \( \text{eval-def(app1)} \) because of the IF-equation \( \text{nil} = \text{nil}_4 \) (\( \text{nil} \) is introduced by the target induction) which is mapped to \( s(0) = 0 \). It produces the island conclusion \( \text{half}(+(a, s(0))) = ? \). With this island the replay can be continued yielding a final subgoal \( s(?) = ?? \). The instantiation of the gap variable by \( \text{half}(s(a)) \) 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 \( \text{nil}_3 = \text{nil}_4 \) which is mapped to \( s(0) = 0 \).
Analogy in CLAM

Source
\[ len_1(a \triangleleft \triangleright b) = len_2(b \triangleleft \triangleright a) \]

Target
\[ m_b \quad \text{len}_1;_2 \mapsto \text{half} \]
\[ \triangleleft \triangleright _{1;_2} \mapsto + \]
\[ \text{half} (a + b) = \text{half} (b + a) \]

Source rewrites
\[ m_c \quad \text{len}_3 \mapsto \text{half} \]
\[ :_1 \mapsto s(s()) \]
\[ s_1 \mapsto s \]
\[ \triangleleft \triangleright _3 \mapsto + \]
\[ \triangleleft \triangleright ;_3 \mapsto s \]

Target rewrites
\[ s(a) + b = s(a + b) \]
\[ \text{half}(s(s(a))) = s(\text{half}(a)) \]

The step case replay:

Source goals

Target goals

---

\[ len_1(a \triangleleft \triangleright 1 v_0 :\triangleright 2 b) = \]
\[ len_2(v_0 :\triangleright 2 b \triangleleft \triangleright 2 a) \]

app2
After 1to2 ref. because of (*)
\[ \text{half}(a + s(s(b))) = \text{half}(s(s(b)) + a) \]

\[ \ldots = len_2(v_0 :\triangleright 3 b \triangleleft \triangleright 3 a) \]

app2
double wave method because of 1to2
\[ \triangleleft \triangleright 2 = \triangleleft \triangleright 3 \]
\[ \ldots = \text{half}(s(s(b + a))) \]

\[ \ldots = s_1(len_3(b \triangleleft \triangleright 3 a)) \]

len2
\[ len_2 = len_3 \]
\[ (*) :_3 = :_1 \]
\[ s(\text{half}(b + a)) \]

s2(len3(a \triangleleft \triangleright 3 b)) = \ldots

lemma
by suggested lemma
\[ s(\text{half}(a + b)) = \ldots \]

s1(len2(b \triangleleft \triangleright 2 a)) = \ldots

\[ len_3 = len_1 \]
\[ \triangleleft \triangleright 3 = \triangleleft \triangleright 1 \]
\[ s(\text{half}(b + a)) = \ldots \]

true
\[ len_2 = len_3 \]
\[ \triangleleft \triangleright 2 = \triangleleft \triangleright 3 \]

Source lemma: \[ len_1(a \triangleleft \triangleright 1 v_0 :\triangleright 2 b) = s_2(len_3(a \triangleleft \triangleright 3 b)) \]

Suggested target lemma: \[ \text{half}(a + s(s(b))) = s(\text{half}(a + b)) \].
The source base case \( b = 0 \) (step case) replay for the target case base \( b = 0 \) (step case):

\[
\begin{align*}
\text{source rewrites} & \quad m_e & \quad \text{target rewrites} \\
\text{app1: } app_0(nil_4, x) = x & \quad app_1 \mapsto + \quad +(0, x) = x \\
\text{length1: } len_5(nil_5) = 0 & \quad nil \mapsto 0 \quad half(0) = 0 \\
\text{app2} & \quad ::_2 ::_3 \mapsto s \quad \text{plus2} \\
\text{length2} & \quad ::_1 \mapsto s(s()) \quad \text{half3} \\
\text{len}_5 \mapsto {\text{half}} \\
\end{align*}
\]

\[
\begin{align*}
\text{source goals} & \quad \text{source justif.} & \quad \text{target goals} \\
len_1(app_1(a, nil)) = & \quad \text{app1} & \quad half(+ (a, 0)) = half(+ (0, a)) \\
len_2(app_2(nil, a)) = & \quad app_2 = app_5 & \quad \text{half} (a) \\
len_1(app_1(a, nil)) = len(a) & \quad nil = nil_4 \\
\text{induction on } a & \quad app_2 \\
len_1(app_1(v_0 ::_2 a, nil)) = & \quad app_2 \\
len_2(v_0 ::_2 a) = & \quad half (+ (s(s(a)), 0)) = half (s(s(a))) \\
\text{step case:} \\
len_1(v_0 ::_3 app_4(a, nil)) = & \quad app_2 \\
app_3 = app_1 & \quad \text{half} (s (+ (a, 0))) = \ldots \\
lens_1 = len_3 & \quad \text{length2} \\
\vdots ::_3 ::_1 & \quad len_1 = len_3 & \quad s(half(+ (a, 0))) = \ldots \\
\text{causes 1to2} & \quad \text{length2} & \\
\ldots = s_1(len(a)) & \quad len_2 = len_3 & \quad \ldots = s(half(z)) \\
::_2 ::_1 & \quad \text{length2} \\
s_1(len_3(a)) = \ldots & \quad len_1 = len_3 \\
app_4 = app_1 & \quad s(half(a)) = \ldots \\
true & \quad true \\
\end{align*}
\]
The source base case \( b = 0 \) replay for the target base case \( b = s(0) \):

<table>
<thead>
<tr>
<th>Source rewrites</th>
<th>( m_e )</th>
<th>Target rewrites</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{app1: } \text{app}_5(\text{nil}_4, x) = x )</td>
<td>( \text{nil} \mapsto s(0) )</td>
<td>( + (0, x) = x )</td>
</tr>
<tr>
<td>( \text{len1: } \text{len}_5(\text{nil}_5) = 0 )</td>
<td>( \text{nil} \mapsto 0 )</td>
<td>( \text{half}(0) = 0 )</td>
</tr>
<tr>
<td>etc.</td>
<td>etc.</td>
<td>etc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source goals</th>
<th>Source justif.</th>
<th>Target goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{len}_1(\text{app}_1(a, \text{nil})) = )</td>
<td>( \text{app1} )</td>
<td>( \text{half}(+(a, s(0))) = )</td>
</tr>
<tr>
<td>( \text{len}_2(\text{app}_2(\text{nil}, a)) )</td>
<td>( \text{app2} = \text{app}_5 )</td>
<td>( \text{half}(+(s(s(a)), s(0))) =? )</td>
</tr>
</tbody>
</table>

... = \( \text{len}_2(a) \)  
\( \text{nil} = \text{nil}_4 \)  
( causes failure )

**induction on \( a \)**  
\( \text{len}_1(\text{app}_5(v_0 :: 2 \ a, \text{nil})) = \)  
\( \text{len}_2(v_0 :: 2 \ a) \)  
**step case:**  
\( \text{len}_1(v_0 :: 3 \ \text{app}_4(a, \text{nil})) = \ldots \)  
\( \text{app2} \)  
\( \text{app}_3 = \text{app}_1 \)  
\( \text{len} = \text{len}_3 \)  
\( :: 3 :: 1 \)  
\( \text{length}2 \)  
\( s(\text{half}(+(a, s(0)))) =? \)  
\( \text{(causes 1to2)} \)  
\( \ldots = s(\text{len}(a)) \)  
\( \text{len}_2 = \text{len}_3 \)  
\( :: 2 :: 1 \)  
\( \text{length}2 \)  
\( \ldots = ?? \)  

\( s(\text{len}_3(a)) = \ldots \)  
\( \text{len}_1 = \text{len}_3 \)  
\( s(?) = ?? \)  
\( \text{app}_4 = \text{app}_1 \)  
replay stopped

### 10.12 plus2right-doubleplus2right

Source theorem: \( +_1(x, s_1(y)) = s_2(+_2(x, y)) \)  
Target theorem: \( \text{double}(+(x, s(y))) = s(s(\text{double}(+(x, y)))) \)  
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
\[ \text{double}(+(S_3(x), y)) = s(s(\text{double}(+(x, y)))) \]
 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 $\text{double}(+(0, x)) = x$ 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 $\text{double}(+(0, x)) = x$ 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.
### Analogy in CIAM

<table>
<thead>
<tr>
<th>Source</th>
<th>$m_b$</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+1(x, s_1(y)) = s_2(+2(x, y))$</td>
<td>$+1,2 \mapsto double(+)$</td>
<td>$\text{double}(+(x, s(y))) = s(s(\text{double}(+(x, y))))$</td>
</tr>
<tr>
<td>$s_1 \mapsto s$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$s_2 \mapsto s(s())$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

source rewrites

plus2: $+3(s_3(x), y) = s_4(+3(x, y))$

| $m_c$ | $s_3 \mapsto s_3$ | $s_4 \mapsto s(s())$ |

nothing for $\text{double}(+())$

#### The step case replay:

<table>
<thead>
<tr>
<th>source goals</th>
<th>source justif.</th>
<th>target goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+1(s_3(x), s_1(y)) = s_2(+1(s_3(x), y))$</td>
<td>plus2</td>
<td>$\text{double}(+(S_3(x), s(y))) = s(s(\text{double}(+(S_3(x), y))))$</td>
</tr>
<tr>
<td>$s_4(+3(x, s_1(y))) = ...$</td>
<td>plus2</td>
<td>by suggested lemma</td>
</tr>
<tr>
<td>$+1 = +3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$s(s(\text{double}(+(x, s(y)))))) = ...$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$... = s_2(s_4(+3(x, y))))$</td>
<td>plus2</td>
<td>$... = s(s(s(\text{double}(+(x, y))))))$</td>
</tr>
<tr>
<td>$+1 = +3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$s(s(s(\text{double}(+(x, y)))))) = ...$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$s_4(s_2(+2(x, y)))) = ...$</td>
<td>partial match</td>
<td>$s(s(s(\text{double}(+(x, y)))))) = ...$</td>
</tr>
<tr>
<td>$+3 = +1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$s_2 = s_4$</td>
<td>$s_2 = s_4$</td>
<td>$true$</td>
</tr>
<tr>
<td>$+2 = +3$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Base case replay:

source rewrites

plus1: $+4(0, x) = x$

<table>
<thead>
<tr>
<th>$m_e$</th>
<th>$0_1 \mapsto 0$</th>
<th>target rewrites</th>
</tr>
</thead>
<tbody>
<tr>
<td>suggested rewrite</td>
<td>$\text{double}(+(0, x)) = x$ disproved</td>
<td></td>
</tr>
</tbody>
</table>

source goals

<table>
<thead>
<tr>
<th>source justif.</th>
<th>target goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+1(0, s_1(y)) = s_2(+2(0, y))$</td>
<td>plus1</td>
</tr>
<tr>
<td>$+1 = +4$</td>
<td></td>
</tr>
<tr>
<td>$0 = 0_1$</td>
<td></td>
</tr>
<tr>
<td>$\text{no transfer}$</td>
<td>$?=s(s(\text{double}(+(0, y))))$</td>
</tr>
<tr>
<td>$s_1(y) = ...$</td>
<td>plus1</td>
</tr>
<tr>
<td>$+2 = +4$</td>
<td></td>
</tr>
<tr>
<td>$0 = 0_1$</td>
<td></td>
</tr>
<tr>
<td>$?=??$</td>
<td></td>
</tr>
<tr>
<td>$... = s_2(y)$</td>
<td></td>
</tr>
</tbody>
</table>

$true$ | $s_1 = s_2$ |
References


tional Joint Conference on Artificial Intelligence, Montreal, 1995.

