MAX-PLANCK-INSTITUT FÜR INFORMATIK

A Recursion Planning Analysis of Inductive Completion

Richard Barnett
David Basin
Jane Hesketh

MPI–I–94–230 June 1994

INFORMATIK

Im Stadtwald
D 66123 Saarbrücken
Germany

Authors' Addresses

Richard Barnett ICL Cavendish Road, Stevenage, Herts SG1 3DS, UK
R.D.Barnett@ste0404.wins.icl.co.uk
David Basin, Max-Planck-Institut für Informatik Im Stadtwald, D-6600 Saarbrücken, Germany
basin@mpi-sb.mpg.de
Jane Hesketh University of Edinburgh 80 South Bridge Edinburgh EH1 1HN Scotland, U.K.
jane@ai.ed.ac.uk

Publication Notes

This paper has been submitted to the Annals of Mathematics and Artificial Intelligence.

Acknowledgements

The authors thank Alan Bundy for suggesting this direction of research and his subsequent interest and advice. We also thank Leo Bachmair for very helpful feedback and enlightening discussions.

Abstract

We use the AI proof planning techniques of *recursion analysis* and *rippling* as tools to analyze so called *inductionless induction* proof techniques. Recursion analysis chooses induction schemas and variables and rippling controls rewriting in explicit induction proofs. They provide a basis for explaining the success and failure of inductionless induction both in deduction of critical pairs and in their simplification. Furthermore, these explicit induction techniques motivate and provide insight into advancements in inductive completion algorithms and suggest directions for further improvements. Our study includes an experimental comparison of Clam, an explicit induction theorem prover, with an implementation of Huet and Hullot's inductionless induction.

1 Introduction

An important research area in mechanized reasoning is automating proofs involving some form of mathematical induction. One of the dominant branches of work in this area originated with the research of Boyer and Moore and their students [6, 1] and is based on analyzing recursive definitions and using their structure to suggest induction schemata and variables. Other research in this branch of so called *classical* or *explicit* induction has been carried forth by Bundy *et al* [20, 7, 8] and the INKA group [5]. A largely separate branch of research has taken place in the term rewriting community where researchers have examined the problem of determining when an equation E is true with respect to a standard model, the initial algebra (denoted $\mathcal{I}(\mathcal{E})$), of a set of equations $\mathcal{E}$. When E is true, it is said to be an *inductive consequence* (or an *inductive theorem*) of $\mathcal{E}$. Proof techniques for determining inductive consequence have often been based, more or less, upon Knuth-Bendix completion; they attempt to generate a rewrite system corresponding to $\mathcal{E} \cup \{E\}$ that is (in)consistent precisely when E is (not) an inductive consequence of $\mathcal{E}$. The test for consistency depends on the properties of $\mathcal{E}$ and the proof procedure used.

Rewrite based work has been referred to in different ways, for example *Inductive Completion* to emphasize the relationship with Knuth-Bendix completion or *Proof by Consistency* to emphasize the relationship between truth and consistency. These techniques have also been referred to as *Inductionless Induction* as they do not appear to construct an induction proof in the classical sense. However the relationship between these techniques and proof by explicit induction is very tight. Theoretically, many of these techniques can be seen as performing induction over a nœtherian ordering given by the equations $\mathcal{E}$ considered as a uniformly terminating term rewriting system [19]. On the practical side, researchers have commented on the close relationship between proof obligations that arise in their techniques and similar obligations in explicit induction proof.

Our aim in this report is to give a concrete analysis of the differences between the two paradigms and suggest how ideas from explicit induction might be used by the inductive completion community. To this end our analysis begins with a comparison of explicit induction (as embodied in the Clam theorem prover [14]) and inductionless induction (as embodied in the Huet-Hullot Completion Procedure, subsequently called HHC [15]). This comparison is both theoretical and experimental; the experimental part consists of a comparison on a suite of twenty theorems. Afterwards, we step back and suggest how ideas arising from this comparison can be used to increase the success rate of less restrictive kinds of completion.

We have selected Clam since it is well suited for such a comparison as much effort has been made to simplify and make explicit its proof search heuristics. Clam is based on an extension of Boyer and Moore's techniques called *recursion analysis* that selects induction schemata and variables; it also employs a controlled form of rewriting called *rippling* to control post-induction rewriting.

The choice of a completion algorithm was more difficult. There are many proposals, some of which are so far removed from Musser's original proposal [18], eg [19, 21] that they are better understood as classical induction than completion-based. For crispness of comparison we have chosen the relatively unsophisticated completion procedure of Huet and Hullot. Their technique is simple and execution does lead to choice points that require heuristics similar to those used to control explicit induction proofs (eg selection of complete positions, cover sets, application of lemmas, and the like). Furthermore, this choice enables us to see which of the problems with induction completion are best viewed as "self-made" and how, when improvements are made, the same kind of control problems and need for heuristics arise as in explicit induction.

Overall, our comparison demonstrates that proof planning based upon recursion analysis and the use of rippling to control proof search in Clam provides a foundation from which completion based techniques may be understood and assessed. Against the backdrop of Clam, we can explain the success and failure of HHC over a wide spectrum of problems. In particular, recursion analysis provides insight into when critical-pair generation is likely to fail and rippling enables us to better understand simplification and use of lemmas. Based on this, we suggest how these heuristics might be incorporated in more recent techniques.

Our report is organized as follows. Sections 2 and 3 provide background material on explicit induction proofs and HHC completion. Sections 4 and 5 provide a theoretical comparisons along the lines of induction schema and variable selection and post-induction rewriting and control. Section 6 provides details on the experimental comparison. The final section draws conclusions by suggesting how lessons learned from the comparison can suggest heuristics useful in inductive completion.

2 Recursion Analysis, Rippling and Clam

[20, 7, 8] present theories of rippling and recursion analysis, a rational reconstruction and extension of the method used to guide induction proofs in the Boyer-Moore Theorem Prover. What follows will be a brief recapitulation ideas presented there, sufficient to provide a basis for our comparison. The references may be consulted for a deeper account.

Recursion Analysis

Given a goal, recursion analysis attempts to find a suitable schema and a universally quantified variable for induction. The difficulty is that there may be many induction schemata to choose from, and multiple variables over which to induct. Choice of the wrong schema or variable is likely to lead to failure.

The specifics of recursion analysis are best explained through an example. Consider the conjecture

$$even(X) \wedge even(Y) \Rightarrow even(X+Y) \tag{1}$$

to which we will refer as EVENP. In this (as in all our equations) free variables are implicitly universally quantified. How is this proved by induction given the following definitions for + and even (defined over the Peano natural numbers 0, $s0$, $ss0$, ...)?

$$\begin{aligned} 0+X &= X & (2)\\ \underline{s}X+Y &= s(X+Y) & (3)\\ even(0) &= true & (4)\\ even(s0) &= false & (5)\\ even(\underline{ss}X) &= even(X) & (6) \end{aligned}$$

Here, we have underlined the *recursion construction* that occurs in the recursive argument position of the above definitions. Note that + steps down in single steps, whereas even steps down in double steps.

To each recursion schema there corresponds a dual induction schema. The dual to 1-step recursion is the schema

$$P(0) \wedge \forall X.\{P(X) \Rightarrow P(sX)\} \Rightarrow \forall X.\ P(X) \tag{7}$$

while

$$P(0) \wedge P(s0) \wedge \forall X.\{P(X) \Rightarrow P(ssX)\} \Rightarrow \forall X.\ P(X) \tag{8}$$

is the dual to 2-step recursion. $P(X)$ ranges schematically over formulæ containing a free occurrence of X.

Recursion analysis locates the recursive functions in a conjecture. Each occurrence of a recursive function F with a variable X in its recursive argument position gives rise to a raw induction schema and suggests the induction variable X. The induction schema suggested is the one dual to the form of recursion used to define F. In the above example, recursion analysis produces the raw induction suggestions given in Table 1.

Each occurrence of X gives rise to a raw induction suggestion. In this example there are two suggestions, but fortunately the 2-step schema *subsumes* the 1-step schema since the recursion term in the former schema consists of repeated nestings of the recursion term in the latter. In general, there may not be a schema for a variable that subsumes all others, but the schemata may suggest one that does [20]. For example, a 2-step and a 3-step induction schema may be *merged* into a 6-step schema that subsumes them both.

The end result of merging is a set of suggestions covering every distinct (ie non-subsumed) induction schema that can be derived from the raw induction schemata. All that remains is to chose the final induction suggestion. Note in our example, that the 2-step X schema will produce an induction conclusion where the term ssX' replaces each occurrence of X, and that, by design, a recursive definition will match the term

Variable	Function	Schema	Recursion Term	Status
X	even	2-step	ssX	unflawed
Y	even	2-step	ssY	flawed
X	$+$	1-step	sX	subsumed

Table 1: Induction Suggestions for the EVENP Example

immediately dominating each of these occurrences in the induction conclusion, namely $even(ssX')$ and $ssX' + Y$. The same is not true of the 2-step Y schema. The second occurrence of Y did not play a role in formation of this induction suggestion, and no recursive definition will match the term $X + ssY'$ which it gives rise to in the induction conclusion. The replacement of the second occurrence of Y with ssY' is regarded as *unsuitable*. Boyer and Moore classify such suggestions as *flawed*.[1] Flawed suggestions are rejected if any unflawed ones remain; hence, in the example above, the 2-step X suggestion is the one that is finally chosen. In the event of a tie the induction subsuming the largest number of raw suggestions is chosen.

The induction schema and variable chosen by recursion analysis are dependent on the forms of recursion used to define the operators used in the conjecture. There are examples where this leads to an inadequate induction schema (eg the classic version of the prime factorization theorem, or the arithmetic/geometric mean theorem); but, in general, recursion analysis is extraordinarily successful and the induction it suggests leads to a proof.

Rippling

Once an induction schema and variable have been chosen the proof remains to be completed by rewriting and appeal to the induction hypothesis. This is the purpose of rippling. The idea behind rippling is that the induction conclusion will be an image of the induction hypothesis except that *constructor terms*, such as the successor function s, will be wrapped around the induction variables. Rippling attempts to move the constructor functions out through the term leaving behind an exact image of the induction hypothesis within the induction conclusion. For example, corresponding to the induction hypothesis given by Equation 1 is the induction conclusion

$$even(\boxed{ss(\underline{X})}) \wedge even(Y) \Rightarrow even(\boxed{ss(\underline{X})} + Y). \tag{9}$$

We have used "box and underline" notation to mark *wave fronts*. In general, a wave front is a term (marked with the box) with a proper subterm deleted (marked by underlining). The wave-front represents the part of the term which rippling must eliminate or move to the outside of the term if a goal is to match the induction hypotheses (here Equation 1).

Recursion analysis suggests initial wave fronts: those constructor symbols given by the induction schema that immediately dominate the induction variable. The role of rippling is to move these wave fronts outwards, leaving the original structure of the induction hypothesis behind. Rippling works from the induction conclusion to the induction hypothesis using *wave rules* which are rewrite rules of the form:[2]

$$F(\boxed{S(\underline{U})}) \rightarrow \boxed{T(\underline{F(U)})}$$

where F, S and T are terms with one distinguished argument. T may be empty, but S and F may not be. Note the effect of applying such a wave rule is to move out (and possibly change or delete) the constructor symbols through the term.

[1] See [6] or [20] for a full definition of unsuitable replacements and flawed suggestions.

[2] We are greatly oversimplifying; Clam allows more general wave rules and rippling strategies, as well as both constructor and destructor induction schemata [8].

It should be emphasized that the wave fronts themselves are part of the rewrite rule and the term being rewritten, and are used to control rewriting. For example, rippling should not loop even if we have both versions of a rewrite such as that for associativity:

$$\boxed{(X + \underline{Y})} + Z \rightarrow \boxed{X + \underline{(Y + Z)}} \tag{10}$$

$$X + \boxed{(\underline{Y} + Z)} \rightarrow \boxed{\underline{(X + Y)} + Z} \tag{11}$$

Each application must move a wave front one step up through the rewritten term, hence rippling will terminate. Furthermore, the annotation restricts the applicability of wave-rules, so they can generally be applied without backtracking. These properties make rippling a very desirable kind of rewriting.

The wave rules in our examples correspond to the recursive case of + (Equation 3) and even (Equation 6). Recast as wave rules, they are:

$$\boxed{s(\underline{X})} + Y \quad \rightarrow \quad \boxed{s(\underline{X + Y})} \tag{12}$$

$$even(\boxed{ss(\underline{X})}) \quad \rightarrow \quad even(X) \tag{13}$$

The application of wave rules terminates successfully when the induction conclusion contains an exact image of the induction hypothesis. At this point, the induction hypothesis may be used as a rewrite rule (this is called *fertilization* in the literature), and the proof is often completed in a trivial manner; alternatively, if there are still universally quantified variables in the conclusion, further induction may be required. However, the rippling process may fail when wave fronts are neither eliminated, nor moved outside the image of the induction hypothesis, and no wave rule applies. We then say that rippling is *stuck*; the proof is unlikely to be completed using the induction chosen.

Clam

The Edinburgh Clam system is a proof planning system that implements a family of proof planners. These planners use various search control strategies (eg depth-first, iterative deepening search, etc) to construct *meta-level proofs*. These proofs are constructed via the application of operators called *methods*; the primary methods being recursion analysis and rippling. Other methods include goal generalization, propositional tautology checking, and expression simplification. When Clam succeeds in constructing a meta-level proof, it produces a program called a *tactic* whose execution should produce a proof in the object-level logic (a constructive type theory, although this is unimportant for our comparison). Clam has been tested extensively on a large number of inductive theorems, most of which are drawn from the Boyer-Moore corpus. Of the 68 examples it was tested on, it proved 65 of them, the remaining three failing due to difficulties with the object level logic, and not because of limitations in the proof methods [8].

In our example, after picking the 2-step induction schema and induction variable X, Clam must prove two base cases and a step case. The base cases are

$$even(0) \wedge even(Y) \Rightarrow even(0 + Y) \tag{14}$$

$$even(s0) \wedge even(Y) \Rightarrow even(s0 + Y) \tag{15}$$

and the step case is

$$even(\boxed{ss(\underline{X})}) \wedge even(Y) \Rightarrow even(\boxed{ss(\underline{X})} + Y). \tag{16}$$

These base cases are trivially proved using symbolic simplification and propositional reasoning. In general, the difficulty in inductive proofs is proving the step cases, and this is the primary focus of this paper.

In the step case, application of wave rule 13 yields

$$even(X) \wedge even(Y) \Rightarrow even(\boxed{ss(\underline{X})} + Y).$$

Two applications of wave rule 12 yields

$$even(X) \wedge even(Y) \Rightarrow even(\boxed{ss(\underline{X+Y})}).$$

The proof is completed by another application of wave rule 13 and appeal to the induction hypothesis.

3 Huet-Hullot Completion

Inductive completion techniques attempt to determine if an equation E (which we call the *goal equation*) is an inductive consequence of a set of equations $\mathcal{E}$. There are no complete proof techniques for determining this, but there are sound techniques, which include explicit induction. Another such technique is Huet-Hullot Completion [15]. HHC requires that the function symbols $\mathcal{F}$ of the theory are divided into a set of *constructors* $\mathcal{C}$ and a set of *defined operators* $\mathcal{D}$; the algorithm's input is a set of equations $\mathcal{E} \cup \{E\}$ where $\mathcal{E}$ must satisfy the condition (which Huet and Hullot call the *principle of definition* or POD) that every ground term g is equationally equal to a unique ground term g' built solely from constructors. HHC is based on the fact that POD for $\mathcal{E}$ is preserved by extension by E if and only if E is valid in $\mathcal{I}(\mathcal{E})$.

HHC determines if POD is preserved through a variant of Knuth-Bendix Completion (KBC). Recall that a *critical pair* of two rules $\lambda_1 \rightarrow \rho_1$ and $\lambda_2 \rightarrow \rho_2$ is the pair $\langle \lambda_1[\rho_2]\sigma, \rho_1\sigma \rangle$ where σ is the most general unifier of a non-variable subexpression t of $\lambda_1[t]$ with λ_2. KBC attempts to orient a set of equations $\mathcal{E}$ as a uniformly terminating set of rewrite rules $\mathcal{R}$. If each critical pair of $\mathcal{R}$ conflates (both terms in the pair rewrite to an identical term) then $\mathcal{R}$ is a canonical set of rewrite rules. Otherwise the non-conflating critical pair is oriented as a rewrite rule and the process continues. HHC is essentially KBC with the additional constraint on the addition of a rule $\lambda \rightarrow \rho$:

- If either λ or ρ is headed by a constructor, and the other is a variable or is headed by a different constructor, then halt with disproof.
- If $\lambda = C(t_1, \ldots, t_n)$ and $\rho = C(s_1, \ldots, s_n)$ for some $C \in \mathcal{C}$, then for each i add $t_i = s_i$ to the list of equations yet to be considered by KBC.
- If λ is headed by a constructor and ρ is headed by a defined operator, halt with failure.

If this modified KBC when applied to $\mathcal{E} \cup \{E\}$ (where $\mathcal{E}$ has POD) halts without failure or disproof, then E is an inductive theorem of $\mathcal{E}$; if it halts with disproof, this equation is not an inductive theorem of $\mathcal{E}$; and if it halts with failure, or fails to terminate, we can say nothing about whether E is an inductive theorem of $\mathcal{E}$.

Later we begin our comparison by applying HHC to the EVENP example. For now we illustrate its application on the simpler example of proving that

$$h(X+X) = X \tag{17}$$

is an inductive consequence of the following equations.

$$\begin{aligned} 0+Y &= Y & (18)\\ sX+Y &= s(X+Y) & (19)\\ h(0) &= 0 & (20)\\ h(s0) &= 0 & (21)\\ h(ssX) &= sh(X) & (22) \end{aligned}$$

Here, h is intended to be a "halving" function. Taking the set of constructors $\mathcal{C}$ to be $\{0, s\}$, the equations for $+$ and h satisfy the requisite properties for HHC. Orienting Equations 17-22 from left to right as rewrite

Rules	Critical Expression	Critical Pair
17,18	$h(0+0)$	$\langle 0, h(0)\rangle$
17,19	$h(sX+sX)$	$\langle sX, h(s(X+sX))\rangle$

Table 2: "Halving Example" Critical Pairs

rules, the only critical pairs (and their *critical expression* which shows how they overlap) of this system are given in Table 2.

The first critical pair conflates using Equation 20; however, the second is irreducible and the rule $R_0 : h(s(X+sX)) \rightarrow sX$ is generated. This introduced rule initiates the generation of an infinite sequence $\{R_i\}$ of further rules, where R_{i+1} is derived from the critical pair between rules 19 and R_i. Hence, in this example, HHC will loop (without the addition of lemmas) and the theorem is not proven.

Inspection reveals that conflation of the critical pairs generated reflect the proof obligations that would arise in an explicit induction proof and that the rewrite rules required to conflate these pairs correspond to lemmata needed to prove the equalities resulting in the corresponding induction proof. These are the topics of the following sections.

4 Induction Schemata

Recursion analysis presents a principled way of choosing induction schemata and variables that are likely to succeed. Are there any similar principles underlying HHC? In this section we answer this question and demonstrate how given a goal equation oriented as a rewrite rule, HHC generates and attempts to prove *all* base and step cases resulting from *all* applicable raw induction schemata.

In what follows, assume we have a set of equations $\mathcal{E}$ completely defining a primitive recursive set of operators. Let $\mathcal{R}$ be their orientation into a set of rewrite rules compatible with some term ordering and such that definitional cases of operators in $\mathcal{D}$ appear on the left-hand side of rules; $\mathcal{R}$ possesses POD [15]. Similarly, assume we can orient the goal equation E as $e_1 \rightarrow e_2$.

The execution of HHC can be partioned into two phases (we will say more about this decomposition in Section 7):

1. Generation of *inductive critical pairs* between $e_1 \rightarrow e_2$ and R.
2. Attempted conflation of inductive critical pairs and further critical pair generation.

We refer to the critical pairs generated in the first phase as inductive critical pairs as our claim is that they correspond to subgoals resulting from induction over all raw induction schemata suggested by recursion analysis of e_1. To see this, consider a definition by 1-step recursion (where $\vec{Y}$ represents possibly multiple variable occurrences).

$$f(0,\vec{Y}) \rightarrow g(\vec{Y}) \tag{23}$$
$$f(sX,\vec{Y}) \rightarrow h(X,\vec{Y},f(X,\vec{Y})) \tag{24}$$

Now, suppose we try to prove an inductive theorem

$$P[f(X,\vec{Y})] = Q(X,\vec{Y}) \tag{25}$$

oriented left to right. Table 3 contains the inductive critical pairs and the corresponding subgoals from explicit induction. They illustrate the exact relationship between inductive critical pairs resulting from a 1-step recursion schema and the subgoals that result from explicit 1-step induction. Specifically, if we view each critical pair as expressing a desired equality, then each pair is one step removed from a base or step case

Rules	Critical Expression	Critical Pair	Explicit Induction
25,23	$P[f(0,\vec{Y})]$	$\langle P[g(\vec{Y})], Q(0,\vec{Y})\rangle$	$P[f(0,\vec{Y})] = Q(0,\vec{Y})$
25,24	$P[f(sX,\vec{Y})]$	$\langle P[h(X,\vec{Y},f(X,\vec{Y}))], Q(sX,\vec{Y})\rangle$	$P[f(sX,\vec{Y})] = Q(sX,\vec{Y})$

Table 3: 1-Step Critical Pairs

induction subgoal. The induction uses the schema dual to the definition (23 and 24), in this case 1-step using X as the induction variable; the rewrite step is the application of rule 23 or 24 to the base or step subgoal respectively.[3] Pairs between theorem and 'base' definitions correspond to base subgoals and similarly 'step' definitions correspond to step subgoals. Note that the goal equation serves as an induction hypothesis that may be applicable as a rewrite rule to simplify the critical pair in the step case.

This relationship generalizes both to functions defined on multiple variables and to more complex recursion schemata and suggest a notion of flawed and unflawed variable occurrence that is analogous to flawed/unflawed explicit inductions. Let t be a term and $u = u_1 \cdots u_m$ a term address such that t/u is a variable, and $t/u_1 \cdots u_{m-1}$ has head operator f. We say u is an *IC-unflawed* occurrence (wrt $\mathcal{R}$) if:

1. $\mathcal{R}$ contains a set $\mathcal{R}_{f,u_m}$ of rules defining f recursively on its u_m-th argument, and
2. $t/u_1 \cdots u_{m-1}$ unifies with the left-hand side of each rule in $\mathcal{R}_{f,u_m}$.

For example, given our previous definitions for $+$ (Equations 18 and 19 oriented as rules), in the expression $X + (Y + Z)$ the occurrences of X and Y are IC-unflawed while that of Z is IC-flawed.

Unfortunately HHC performs (and demands proof for) all inductions whether flawed or unflawed. More exactly, HHC effectively requires a proof corresponding to every raw induction schema that explicit recursion analysis of the left-hand side of the goal equation would suggest; these correspond to IC-unflawed occurrences of variables.[4] If such variables have IC-flawed occurrences in the goal, or if the schema is subsumed by another suggestion, then critical pairs generated are unlikely to conflate. In the halving example of the previous section there was only one IC-unflawed variable in the left-hand side of the goal equation (Equation 17). But in general this will not be the case and HHC will generate critical pairs corresponding to flawed induction schemata.

This is best illustrated by example, and we return to the EVENP theorem. The equations used by HHC are the same as those used for Clam (Equations 1–6), except in the HHC setting, they must be recast as rewrite rules.

$$
\begin{aligned}
0 + X &\rightarrow X && (26)\\
sX + Y &\rightarrow s(X+Y) && (27)\\
even(0) &\rightarrow true && (28)\\
even(s0) &\rightarrow false && (29)\\
even(ssX) &\rightarrow even(X) && (30)\\
(even(X) \wedge even(Y) \Rightarrow even(X+Y)) &\rightarrow true && (31)
\end{aligned}
$$

Here, Equation 31 is the conjecture. Also in the database will be appropriate definitions for $\wedge$ (conjunction) and $\Rightarrow$ (implication) as rewrite rules.[5] These equations possess POD.

[3] This step, implicit in critical pair generation, guarantees that the rule derived from 25 (the "inductive hypothesis") is only applied to smaller terms in the well-order associated with the uniformly terminating rewrite system $\mathcal{R}$. See [19].

[4] If $u_1 \ldots u_{m+1}$ is an IC-unflawed variable then $u_1 \ldots u_m$ is a complete position for superposition in the sense of [11]. However, HHC may produce additional proof obligations due to superpositions at non-complete positions although not all will correspond to legitimate induction schemata.

[5] These need not concern us here. They are defined on variables and the constants true and false, and will not take part in critical pair generation.

Inductive critical pairs of the above rules are given in Table 4, together with a classification of the induction subgoal corresponding to conflation of the pair by type (inductive, base: 0, and so on), operator, induction variable (underlined), and the induction schema used for the induction.

Rules	Critical Expression Critical Pair	Induction Scheme	Induction Subgoal
31,26	$even(0) \wedge even(Y) \Rightarrow even(0+Y)$ $\langle true, even(0) \wedge even(Y) \Rightarrow even(Y) \rangle$	sX	base: 0 $\underline{X}+Y$
31,27	$even(sX) \wedge even(Y) \Rightarrow even(sX+Y)$ $\langle true, even(sX) \wedge even(Y) \Rightarrow even(s(X+Y)) \rangle$	sX	inductive $\underline{X}+Y$
31,28	$even(0) \wedge even(Y) \Rightarrow even(0+Y)$ $\langle true, true \wedge even(Y) \Rightarrow even(0+Y) \rangle$	ssX	base: 0 $even(\underline{X})$
31,29	$even(s0) \wedge even(Y) \Rightarrow even(s0+Y)$ $\langle true, false \wedge even(Y) \Rightarrow even(s0+Y) \rangle$	ssX	base: $s0$ $even(\underline{X})$
31,30	$even(ssX) \wedge even(Y) \Rightarrow even(ssX+Y)$ $\langle true, even(X) \wedge even(Y) \Rightarrow even(ssX+Y) \rangle$	ssX	inductive $even(\underline{X})$
31,28	$even(X) \wedge even(0) \Rightarrow even(X+0)$ $\langle true, even(X) \wedge true \Rightarrow even(X+0) \rangle$	ssY	base: 0 $even(\underline{X})$
31,29	$even(X) \wedge even(s0) \Rightarrow even(X+s0)$ $\langle true, even(X) \wedge false \Rightarrow even(X+s0) \rangle$	ssY	base: $s0$ $even(\underline{Y})$
31,30	$even(X) \wedge even(ssY) \Rightarrow even(X+ssY)$ $\langle true, even(X) \wedge even(Y) \Rightarrow even(X+ssY) \rangle$	ssY	inductive $even(\underline{Y})$

Table 4: Inductive Completion of Even+

Inspection reveals that effectively *three* different inductions are carried out. This is unfortunate as a single induction, namely the 2-step induction corresponding to the recursive definition of $even$ on the variable X (lines 3 – 5) would be sufficient to complete the proof. Indeed, these critical pairs conflate, whereas the step cases corresponding to the other two inductions fail to conflate. (This is largely because the second occurrence of Y in 31 is IC-flawed and the 1-step schema corresponding to the IC- unflawed second occurrence of X is subsumed by that corresponding to the first.) This leads to failure; in such cases HHC will fail to terminate.

5 Rippling and Simplification

The previous section presented HHC as a two phase process. The second phase consists of simplifying (and hopefully conflating) inductive critical pairs and subsequently generated critical pairs. If this is to succeed, then HHC must derive a canonical rewrite system (with POD) for a possibly enlarged equational theory. When the equations $\mathcal{E}$ can be oriented into a canonical rewrite system, the remaining "proof" involves conflating the critical pairs corresponding to the HHC "inductions". If these cannot be conflated, they are added as rules and may in turn generate further critical pairs.

Given the possibility of additional critical pair generation, this phase is different from simplification in Clam. However the simplification of critical pairs with rewriting in explicit induction can be directly compared. In particular, if we focus on the "step cases" of the induction we find that the recursive cases of the rewrite rules all correspond to wave-rules. This is no coincidence. The name rippling comes from *rippling-out* a term coined by Aubin [1], a student of Boyer and Moore's, during his study of generalization in inductive theorem proving. It is based on an observation that one can iteratively unfold (as in [9]) recursive functions in the induction conclusion, preserving the structure of the induction hypothesis while unfolding.

Since structure is preserved as constructor symbols are moved outwards, such rewrite rules may always be annotated with wave-fronts and used in rippling.

The significance of the above point is that rewrite rules in $\mathcal{R}$ appropriate for the step case may be used as wave rules but not conversely. That is, consider the two orientations of associativity of plus given in Equations 10 and 11. Neither come from recursive definitions and both cannot be together oriented as rewrite rules. Now consider trying to prove

$$(y + z) * x = y * x + z * x \tag{32}$$

from the rewrite rules

$$R = \{0 + X \to X,\ s(X) + Y \to s(X + Y),\ 0 * X \to X,\ s(X) * Y \to Y + X * Y\}$$

Superimposing the step case of addition against the goal yields the critical pair

$$\langle s(y + z) * x, s(y) * x + z * x\rangle. \tag{33}$$

On the left this simplifies to $x + (y + z) * x$ which can be further reduced using Equation 32 to $x + (y * x + z * x)$. On the right we can simplify to $(x + y * x) + z * x$ which is irreducible. Since the two sides are not identical, the proof has failed. However, if this proof were directed by rippling, and the two terms were annotated as

$$\langle \boxed{s(\underline{y + z})} * x, \boxed{s(\underline{y})} * x + z * x\rangle$$

then the same reduction steps would be carried out. Moreover, after the above reductions, the right hand term would be annotated as $\boxed{(x + \underline{y * x})} + z * x$. At this point, we can ripple once more with the associativity wave-rule given by Equation 10; this successfully completes the proof.

The ability to add lemmas like associativity is strongly restricted in the framework of HHC. Recall, for example, the halving example (Equation 17) that HHC failed to prove. If we add a lemma (true in $\mathcal{I}(\mathcal{E})$) such as $X + sX = s(X + X)$ as a rewrite rule, the critical pair corresponding to the inductive step conflates. However, incorporating this lemma into the set of rewrite rules gives rise to further critical pairs; as in this example, the result may be the generation of an infinite sequence of critical pairs arising from the new equation. In more recent procedures this problem has been solved; however the solutions create new problems of their own, namely the control of their application. This will be a topic of Section 7.

6 Experimental Comparison

This section presents an experimental comparison of Clam with an implementation of HHC [3]. Twenty theorems are presented along with the inductions performed in the Clam proof and the inductions arising from the inductive completion process.

Table 5 shows the various theorems considered in the orientation used by the inductive completion algorithm (an equation $\lambda = \rho$ became the rewrite rule $\lambda \to \rho$). Some notational changes have been made; for example, we use Prolog-like notation for lists although the operators cons and nil were used in the input. All the operators were defined in the obvious way by primitive recursion on the first suitable argument (their definitions can be found in [3]); p_eq is an equality predicate on Peano naturals, and mem is a member predicate.

The results are given in Table 6. For Clam inductions the *Initial* entry $\mathcal{V}_n$ indicates that the variable $\mathcal{V}$ and an n-step induction schema (for data of the sort of $\mathcal{V}$) were chosen by Clam for the first induction. *Subsequent* contains the lemmata (if any) which were automatically proved by induction during the proof. For the HHC inductions, the *Initial* entry $\mathcal{V}\ (F)$ indicates that the occurrence of the variable $\mathcal{V}$ as an argument of the function $\mathcal{F}$ in the left-hand side of the theorem was IC-unflawed, and hence critical pairs were generated corresponding to induction on this variable using the schema with which $\mathcal{F}$ was defined on $\mathcal{V}$. *Subsequent*

Name	Theorem
APPREV	$rev(app(X,Y)) = app(rev(Y), rev(X))$
ASSAPP	$app(app(X,Y),Z) = app(X, app(Y,Z))$
ASSP	$(X+Y)+Z = X+(Y+Z)$
DIST	$X\cdot(Y+Z) = (X\cdot Y)+(X\cdot Z)$
DISTTWO	$(X+Y)\cdot Z = (X\cdot Z)+(Y\cdot Z)$
DOUBLEHALF	$p\_eq(double(half(X)), X) = even(X)$
EVENP	$even(X) \wedge even(Y) \Rightarrow even(X+Y) = true$
HALFDOUBLE	$half(double(X)) = X$
IDENTRM	$X\cdot s0 = X$
LENREV	$len(rev(X)) = len(X)$
LENSUM	$len(app(X,Y)) = len(X)+len(Y)$
MEMAPP	$mem(X,Y) \Rightarrow mem(X, app(Y,Z)) = true$
MEMAPP2	$mem(X,Y) \Rightarrow mem(X, app(Z,Y)) = true$
MEMAPP3	$mem(X, app(Y,Z)) = mem(X,Y) \vee mem(X,Z)$
MEMREVA	$mem(X, rev(Y)) = mem(X,Y)$
REVREV	$rev(rev(X)) = X$
TAILREV	$rev(app(X,[Y])) = [Y \mid rev(X)]$
TAILREV3	$rev(app(rev(X),[Y])) = [Y \mid X]$
ZEROPLUS	$p\_eq(X+Y,0) = p\_eq(X,0) \wedge p\_eq(Y,0)$
ZEROTIMES	$p\_eq(X\cdot Y,0) = p\_eq(X,0) \vee p\_eq(Y,0)$

Table 5: Comparison Theorems

contains the lemmata (if any) which were introduced by the user with the hope that the algorithm would then terminate with success — no further comment indicates that this was indeed the case. The failure of MEMAPP2 was due to interaction between the first two lemmata which generated a non-terminating sequence of rules; replacements that would have given successful completion could not be found.

Out of 20 theorems, Clam successfully proved 19, and HHC proved 10 automatically and an additional 7 with a human carefully adding lemmata to the defining equations. HHC picked 16 of 20 induction schemata correctly by the standards of recursion analysis and was able to prove all of these theorems, although aid was still required with 6. In the remaining 4 theorems, HHC performed unnecessary inductions. Of these, 1 could still be completed with aid, but the remaining 3 could not be proved due to problems caused by the introduction of lemmata.

Much care should be taken in interpreting these comparison figures. The most significant caveat is that Clam's repertoire of wave rules is not a fixed set and can be extended by the user to reflect rewrites that have proven useful in practice. This is similar to the approach taken in the Boyer-Moore prover whereby the user effectively extends the power of the prover by adding new facts to a theorem database. Another caveat is that Clam has a built-in module for propositional reasoning. This was one source of difficulty and required lemmata suggestions in 5 HHC proofs.

Despite these caveats, the results are in accord with our expectations. HHC's failures[6] arose as additional lemmata were required to prevent a non-terminating sequence of rules arising from the inductive critical pairs corresponding to the flawed induction schema. For example, in the DIST example, a non-orientable equation arose from the two recursive definitions of multiplication.[7] And even when only the proper induction is

[6] Due either to the generation of a non-orientable equation (as in the DIST example), or the interaction of two rules of the definitions and lemmata producing an infinite sequence of rules (as in the MEMAPP2 example, where the lemmata $X \Rightarrow X \wedge Y \rightarrow true$ and $(X \wedge Y) \wedge Z \rightarrow X \wedge (Y \wedge Z)$ caused this behaviour).

[7] The critical expression $sX \cdot sY$ gives the pair $\langle sX + sX\cdot Y, sY + X\cdot sY\rangle$ which further reduces to the pair $\langle s(X+(Y+X\cdot Y)), s(Y+(X+X\cdot Y))\rangle$.

	Clam Inductions		**HHC Inductions**	
Theorem	Initial	Subsequent	Initial	Subsequent
APPREV	X_1	$app(X, []) = X$ ASSAPP	X (app)	$app(X, []) = X^\star$ ASSAPP
ASSAPP	X_1	none	X (app)	none
ASSP	X_1	none	X $(+)$	none
DIST	X_1	$X + sY = s(X + Y)$ $X + (Y + Z) = Y + (X + Z)$ $(X + Y) + (U + V)$ $= (X + U) + (Y + V)$	X $(\cdot)$ Y $(+)$	$X\cdot\,!0 = 0^\star$ $X\cdot sY = X + X\cdot Y^\star$ FAILS (non-orientable equation)
DISTTWO	X_1	$X + 0 = X$ $X + sY = s(X + Y)$ $(X + Y) + Z = (X + Z) + Y$ $X + Y = Y + X$	X $(+)$	ASSP
DOUBLEHALF	X_2	$ssX = ssY \Leftrightarrow X = Y$	X $(half)$	none
EVENP	X_2	none	X $(even)$ Y $(even)$ X $(+)$	$X + sY = s(X + Y)^\star$
HALFDOUBLE	X_1	none	X $(double)$	none
IDENTRM	X_1	$X + s0 = sX$	X $(\cdot)$	none
LENREV	X_1	$len(app(X, [Y])) = s(len(X))$	X (rev)	$len(app(X, [Y])) = s(len(X))$
LENSUM	X_1	none	X (app)	none
MEMAPP	Y_1	none	Y (mem) Y (app)	$X \Rightarrow X \vee Y = true$ $(X \vee Y) \Rightarrow (X \vee Z)$ $= X \vee (Y \Rightarrow Z)$
MEMAPP2	Z_1	none	Y (mem) Z (app)	$X \Rightarrow (Y \vee X) = true$ $(X \vee Y) \vee Z = X \vee (Y \vee Z)$ $X \Rightarrow X = true$, MEMAPP3 FAILS (see text)
MEMAPP3	Y_1	none	Y (app)	$(X \vee Y) \vee Z = X \vee (Y \vee Z)$
MEMREVA	Y_1	none	Y (rev)	$(X \vee Y) \vee Z = X \vee (Y \vee Z)$ $mem(X, app(Y, Z))$ $= mem(X, Y) \vee mem(X, Z)$ FAILS (non-orientable equation)
REVREV	X_1	TAILREV	X (rev)	none
TAILREV	X_1	none	X (app)	none
TAILREV3	FAILS	FAILS	X (rev)	none
ZEROPLUS	X_1	none	X $(+)$	none
ZEROTIMES	X_1	$X + sY = s(X + Y)$ $X\cdot sY = X + X\cdot Y$ $(X\cdot Y) + Y = 0$ $\Leftrightarrow (sX = 0) \vee (Y = 0)$	X $(\cdot)$	$X \wedge (Y \vee X) = X$ ZEROPLUS

Table 6: CLAM/HHC Comparison

performed, the strong requirement of confluence often requires additional lemmata.

The Clam failure on TAILREV3 is also rather interesting and demonstrates how critical pairs can be used as lemmata in explicit induction proofs. An examination of the HHC proof reveals that the TAILREV conjecture is generated from a (non-inductive) critical pair and added as a new rule. This in turn subsumes TAILREV3 (as it rewrites the left hand side) which simplifies to REVREV. The HHC proof terminates as these two rules can themselves be proved by HHC without lemmata and they conflate the inductive critical pairs. Currently, Clam is unlikely to find these lemmata as they do not occur as subgoals during explicit induction or arise from Clam's generalization procedure. So here the deduction of additional critical pairs was an advantage for HHC. In general, this phenomenon seems to occur very infrequently, and it does not seem possible to predict in advance. Hence, it is hard to imagine heuristics that might be used to add such lemma generation to explicit induction (or even to linear inductive completion algorithms).

7 Heuristics in Inductive Completion

While it has not been our aim to give a survey of inductive completion, in this section we step back from HHC and consider induction completion more abstractly and how the heuristics of recursion analysis and rippling might be used. In what follows $\mathcal{R}$ will be a confluent or ground confluent set of rewrite rules, $\mathcal{L}$ a set of equations to be used as lemmas, and $\mathcal{C}$ a set of theorems to be proved.

At an abstract level, most inductive completion techniques have two main components:

1. Deduction: the generation of critical pairs.
2. Simplification: rewriting with $\mathcal{R}$, $\mathcal{L}$ and $\mathcal{C}$.

For example, in HHC, $\mathcal{C}$ is a singleton set $\{E\}$, and $\mathcal{L} = \{\}$. Furthermore, $\mathcal{R} \cup \mathcal{C}$ must be turned into a canonical set of rewrite rules. As we have seen, this is very restrictive since deduction is too strong (too many critical pairs) and simplification too weak (no lemmas). The former means that recursion analysis cannot be applied; the latter means that we cannot ripple with additional lemmas.

More sophisticated techniques have weakened these restrictions. For example, [11] (building on [16]) and [2] have introduced linear proof procedures in which $\mathcal{R}$ need only be ground confluent, and $\mathcal{L}$ may be non-empty. Recursion analysis may be used in these two techniques. As in Table 4, superposition of the goal conjectures against the rewrite rules suggests induction schemata and induction variables. Recursion analysis can then be used to choose among the flawed and unflawed schemata (eg Table 1). There are limitations to this however. In particular, when superposition is used to select induction schemata, then these are limited to those recursions present in the given equations. One could not find a 6-step induction by merging a 2-step and a 3-step schema found by superposition, for example. This limited selection is sometimes claimed as an advantage of inductive completion: the induction schemata selected are "appropriate" for the conjecture and rewrite rules at hand. Basically this appropriateness simply means at least one rewrite step from $\mathcal{R}$ will apply to at least one term position. Note that the one-step look ahead of rippling is stronger since this is a look-ahead in all positions, and it uses all wave-rules (eg $\mathcal{L}$ as well as $\mathcal{R}$). It is possible to use more powerful notions of cover-sets for schemata generation, eg [19], but automatically generating them or selecting among them leads to the same need for heuristics as in explicit induction.

In stronger frameworks, rippling should be directly applicable to control simplification of critical pairs. This requires, of course, first annotating such critical pairs with wave-fronts, as well as the terms in $\mathcal{R}$ and $\mathcal{L}$. Both can easily be done automatically, eg using ideas like *difference matching* presented in [4]. Afterwards, rippling can be used to direct simplification, perhaps (depending on the IC algorithm used) with the constraint that equations in $\mathcal{C}$ must be applied to terms that have been reduced in the appropriate term ordering.

As an example, recall the proof that multiplication distributes over addition given in Section 5. This proof could be directly carried out, say, using the proof by consistency procedure suggested by [2]. In this case, recursion analysis could choose, say among the complete sets of superpositions,

$$\{\langle z * x, 0 * x + z * x\rangle,\ \langle s(y+z) * x, s(y) * x + z * x\rangle\}$$

as a cover set of critical pairs. The first critical pair, corresponding to the base case is simplified (and hence can be deleted) automatically using the rewrite rules $\mathcal{R}$ corresponding to the base cases of plus and times. In the step case, the proof goes through (including the associativity step) using rippling as described in Section 5; note that $\mathcal{C}$ is not applied to any term made larger by a $\mathcal{L}$-rewriting step so the ordering requirements of [2] are satisfied.

One can loosen restrictions in inductive completion even more, and the resulting procedures becomes more and more like explicit induction. For example, if the disproof of non-theorems is not desired, ground confluence and fairness in simplification are not required (eg [19]). Indeed, it is even possible to incorporate techniques like generalization [13], which are needed in inductive theorem proving as cut-free proofs do not always exist (as they do in equational logic). In the end, as inductive completion become more flexible, the "self-made" problems fall away and only the control problems of explicit induction remain and the potential for incorporating ideas from explicit induction grows.

References

[1] R Aubin. *Some Generalization Heuristics in Proofs by Induction*. Actes du Colloque Construction: Amelioration et Verification des Programmes, INRIA 1975.

[2] L Bachmair. *Proof by Consistency in Equational Theories*. Proc LICS 1988.

[3] R Barnett. *An Implementation and Evaluation of Inductive Completion*. University of Edinburgh MSc Thesis 1990.

[4] David Basin and Toby Walsh. Difference matching. In *Proc. of 11th International Conference On Automated Deduction (CADE-11)*, pages 295–309, Saratoga Springs, New York, June 1992. Springer-Verlag.

[5] S. Biundo, B. Hummel, D. Hutter, and C. Walther. The karlsruhe induction theorem proving system. In *8th International Conference On Automated Deduction*, Oxford, UK, 1986.

[6] R Boyer, J Moore. *A Computational Logic*. Academic Press, 1979 (ACM Monograph Series).

[7] A Bundy, F van Harmelen, J Hesketh, A Smaill, A Stevens. *A Rational Reconstruction and Extension of Recursion Analysis*. University of Edinburgh DAI Research Paper 419, 1989.

[8] A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. Research Paper 567, Dept. of Artificial Intelligence, Edinburgh, 1991. To appear in Artificial Intelligence.

[9] R.M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the Association for Computing Machinery*, 24(1):44–67, 1977.

[10] N Dershowitz. *Applications of the Knuth-Bendix Completion Procedure*. Proc of the Seminaire d'Informatique Theorique, 1982.

[11] L Fribourg. *A Strong Restriction of the Inductive Completion Procedure*. Proc ICALP 13, 1986 (Springer-Verlag LNCS 226).

[12] R Göbel. *A Specialized Knuth-Bendix Algorithm for Inductive Proofs*. Proc. Combinatorial Algorithms in Algebraic Structures 1985.

[13] B Gramlich. *Inductive Theorem Proving using Refined Unfailing Completion Techniques*. SEKI Report SR-89-14.

[14] F van Harmelen 1989. *The Clam Proof Planner*. University of Edinburgh DAI Technical Paper 4.

[15] G Huet, J-M Hullot. *Proofs by Induction in Equational Theories with Constructors*. INRIA 1982.

[16] J-P Jouannaud, E Kounalis. *Automatic Proofs by Induction in Equational Theories without Constructors*. Proc LICS 1986.

[17] W Küchlin. *Inductive Completion by Ground Proof Transformation*. University of Delaware Dept of Computer and Information Sciences Technical Report 87-08.

[18] David R Musser. On proving inductive properties of abstract data types. In *ACM Symposium on Principles of Programming Languages*, pages 1154–162, Las Vegas, 1980.

[19] U Reddy. *Term Rewriting Induction*. Proc. CADE 10, 1990 (Springer-Verlag LNAI 449).

[20] A Stevens. *A Rational Reconstruction of Boyer and Moore's Technique for Constructing Induction Formulas*. Proc ECAI-88; University of Edinburgh DAI Research Paper 360.

[21] H Zhang, D Kapur, M Krishnamoorthy. *A Mechanizable Induction Principle for Equational Specifications.* Proc CADE 9, 1988 (Springer-Verlag LNCS 310).