The Power of Combining Resonance with Heat*

Larry Wos

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439-4801

Abstract

In this article, I present experimental evidence of the value of combining two strategies each of which has proved powerful in various contexts. The *resonance strategy* gives preference (for directing a program's reasoning) to equations or formulas that have the same shape (ignoring variables) as one of the patterns supplied by the researcher to be used as a resonator. The *hot list strategy* rearranges the order in which conclusions are drawn, the rearranging caused by immediately visiting and, depending on the value of the heat parameter, even immediately revisiting a set of input statements chosen by the researcher; the chosen statements are used to *complete* applications of inference rules rather than to *initiate* them. Combining these two strategies often enables an automated reasoning program to attack deep questions and hard problems with far more effectiveness than using either alone. The use of this combination in the context of cursory proof checking produced most unexpected and satisfying results, as I show here. I present the material (including commentary) in the spirit of excerpts from an experimenter's notebook, thus meeting the frequent request to illustrate how a researcher can make wise choices from among the numerous options offered by McCune's automated reasoning program OTTER. I include challenges and topics for research and, to aid the researcher, in the Appendix a sample input file and a number of intriguing proofs.

1. The Value of Strategy and of a Chronicle of Experiments

When the object of one's research is the completion of a sought-after proof, one encounters the obstacle of deciding where to focus one's attention. An automated reasoning program faces the same obstacle. In contrast, the obstacle of immediately using one or more axioms at any point in the reasoning is faced by the program but not by the researcher. The severity of each of these obstacles, for an automated reasoning program, can be lessened markedly by relying, respectively, on the *resonance strategy* and on the *hot list strategy*. When these two strategies are combined in an appropriate manner (discussed in this article), as the experimental evidence I present suggests, the result often is most satisfying. More generally, for sharply reducing the severity of virtually any obstacle, strategy frequently is the key.

Indeed, from the mid-1960s until the late 1980s, I voiced the opinion repeatedly that far too little research focused on the formulation of additional strategies intended either to *restrict* or to *direct* the actions of an automated reasoning program. Perhaps the explanation for my effrontery or chutzpah rests with the successes of my research, successes that I still trace directly to the employment of strategy. I can report with great satisfaction that the period between the late 1980s and the mid-1990s has, in sharp contrast to the earlier years, witnessed the introduction of a number of intriguing strategies.

Included among the new strategies are the kernel strategy [Wos93] (for studying combinatory logic); McCune's ratio strategy [McCune92]; the subtautology strategy [Wos95a]; McCune's tail strategy [Wos90,Wos95a]; the recursive tail strategy [Wos95a]; and the two strategies that are of prime concern here, the resonance strategy [Wos91,Wos95b] and the hot list strategy [Wos95c]. (All of the cited strategies, as well as others not cited, are offered by McCune's automated reasoning program OTTER [McCune93], whose use plays a vital role in the research reported here.) It seems hardly

*This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

coincidental that the cited period has also witnessed a sharp increase in the rate of deep questions answered and hard problems solved with substantial assistance of an automated reasoning program (see, for example, [Wos93,Padmanabhan95a,Padmanabhan95b]).

Although a given strategy may have aesthetic appeal and even beauty, its power is the ultimate test of its value. Here I chronicle various experiments, in part to show how one event leads to another and thus provide insight into research, and in part to show (in response to frequent requests) how one can effectively choose from among the numerous options offered by McCune's automated reasoning program OTTER; see Chapter 8 of [Wos95a] for additional guidelines for option choosing. Therefore, the style of this article will be somewhat reminiscent of an experimenter's notebook.

As noted, featured in the experiments are two strategies, the resonance strategy and the hot list strategy, briefly reviewed in Sections 1.1 and 1.2, respectively. By itself, each of the two strategies has proved powerful in various contexts; in Section 5 I present experimental evidence of the value of combining (in two different ways) these two strategies. For example, the use of each combination in the context of cursory proof checking produced most unexpected and satisfying results, as I show in Section 5. Also playing a significant role in the program's search for new conclusions, stored as clauses, is Overbeek's *weighting strategy* [McCharen76,Wos87,Wos92] (which directs the search by complexity preference selection of clauses to initiate applications of the inference rules), as well as other strategies that include McCune's *ratio strategy*; see [Wos92] for a detailed discussion of the cited strategies and for a discussion of the clause language for representing information. (Throughout this article, I often use the term ''clause'' [Wos87,Wos92] interchangeably with the term conclusion, for conclusions that are drawn and input statements that characterize the problem under study are stored by OTTER as clauses.)

More generally, while the focus in this article is on the resonance strategy combined with the hot list strategy, various strategies taken in combination have played a vital role in my research. Indeed, employment of strategies in combination often is the key to obtaining a sought-after result.

1.1. The Resonance Strategy

To use the resonance strategy, the researcher chooses equations and formulas because their symbol pattern (whose variables are treated as indistinguishable) appears to make each appealing for directing the program's attack. Such chosen equations or formulas are called *resonators*. The appeal rests with the conjecture that any equation or formula that matches the shape of an included resonator (symbol pattern of the type under discussion) merits immediate or almost immediate attention for driving the program's reasoning. Two equations or two formulas, one of which is a resonator, match (in the sense used here) if and only if they are identical when all the variables are treated as indistinguishable. For each resonator that the researcher chooses to include in the input, a value is assigned (by the researcher) to reflect its conjectured significance. The smaller the value, the greater the significance. Resonators direct a program's search for new conclusions by giving to any matching equation or formula present in the database the value of the resonator, which in turn (typically) gives the equation or formula preference for being chosen to drive the program's reasoning.

Sources for potentially powerful resonators include the elements of the *special hypothesis* of the theorem under study, the conclusions of the theorem, deduced steps of proofs of related theorems, and deduced steps of proofs completed while attempting to prove the (main) theorem in question. (The special hypothesis of a theorem of the form **if** *P* **then** *Q* refers to that part of *P*, if such exists, that excludes the underlying axioms and lemmas. For example, in the study of the theory that asserts that rings in which the cube of $x = x$ are commutative, the special hypothesis consists of the equation $xxx = x$.) In general, when one begins the study of a new area, Overbeek suggests accumulating a set of resonators just as one accumulates a set of lemmas, where one simply accumulates the proof steps of the theorems as they are proved, perhaps starting with simple theorems.

At least in the Argonne paradigm, one important effect of including resonators concerns the potential of rapidly adding to the pool of conjectured-to-be-useful lemmas. Indeed, when a conclusion is retained that matches a resonator, because of the (usually) small weight (value) that is assigned to the resonator and therefore assigned to the retained conclusion, the conclusion is quickly chosen as the focus of attention to direct the program's reasoning. Then, once it has served its purpose for initiating

applications of the chosen inference rules, the conclusion is then made available to be used in the same manner as lemmas included in the input list(usable) are used. Input clauses included in list(usable) are used to *complete* applications of an inference rule, and are never used to *initiate* applications.

1.2. The Hot List Strategy

Use of the hot list strategy typically requires the researcher to choose from among the input statements that present the question or problem those that are conjectured to merit immediate visiting (as hypotheses) and, if the value assigned to the heat parameter is greater than 1, even immediate revisiting. The chosen statements are placed in the (input) hot list, and they are used one at a time or in combination (depending on the inference rule being applied) to *complete* inference rule applications, rather than to *initiate* applications. Where resonators are restricted to be correspondents of unit clauses, clauses free of the logical **or** symbol ‘‘|’’, members of the hot list are *not* subject to this restriction. Potentially powerful choices for members of the hot list include the elements of the special hypothesis and elements of the initial set of support. Also, when an inference rule in use relies on the use of some clause as a nucleus (as in hyperresolution), then placing a copy of such a clause in the hot list is often profitable.

When the program decides to retain a new conclusion, *before* another conclusion is chosen from list(sos) as the focus of attention to drive the program’s reasoning, the hot list strategy causes the new conclusion to initiate applications of the inference rules in use, with the remaining hypotheses (or parents) that are needed all chosen from the hot list. Members of the hot list, whether input or adjoined during a run, are not subject to back demodulation or back subsumption; in other words, clauses in the hot list are never rewritten with the discovery of new demodulators (rewrite rules), nor are they purged because of being captured by a newly retained clause.

If one wishes the program *during* the run to adjoin new members to the hot list, one uses the *dynamic hot list strategy* [Wos95c] (which McCune formulated as an extension of the hot list strategy). If that is the choice, then one assigns an integer, positive or negative, to the dynamic_heat_weight parameter with the intention that conclusions that are retained and that have a *pick_given weight* (complexity) less than or equal to the assigned value will be adjoined to the hot list. (A clause technically has two weights, its pick_given weight which is used in the context of choosing clauses as the focus of attention to drive the program’s reasoning, and its purge_gen weight which is used in the context of clause discarding; often the two weights are the same.) The dynamic_heat_weight assignment places an upper bound on the pick_given weight of clauses that can be adjoined to the hot list during the run.

When the heat parameter is assigned the value 1, conclusions that result from consulting the hot list (whether one is using the hot list strategy or the dynamic hot list strategy) have *heat level* 1. (By definition, input clauses have level 0, and a deduced clause has a level one greater than that of its parents.) If the program decides to retain a conclusion of heat level 1 and if the (input) heat parameter is assigned the value 2, then *before* another conclusion is chosen as the focus of attention, the heat-level-1 conclusion is used to initiate the search for conclusions of heat level 2 (with the hypotheses needed to complete the corresponding application of the inference rule chosen from the hot list). The heat parameter can be assigned to whatever positive integer the researcher chooses, with the objective of placing corresponding emphasis on the immediate use of the members of the hot list. If the parameter is assigned the value 0, then the hot list strategy will not be used.

1.3. The Resonance Strategy versus the Hot List Strategy

Whereas resonators have no true or false value, members of the hot list are treated as true, as lemmas or their equivalent. A second significant difference between resonators and members of the hot list concerns what I call *temporary keying* versus *constant keying*. Regarding temporary keying, a deduced clause that matches a resonator is assigned the value (typically a small positive integer) of the corresponding resonator and is, therefore, usually given great preference for keying the program’s search for information; but that preference is brief in the following sense. After a clause *A* is chosen from list(sos) to be the focus of attention and considered with all of the input clauses that were placed in list(usable) (and hence not given set of support) and all of the clauses that were moved to list(usable) from list(sos) having served as the focus of attention, *A* is never again used to initiate inference rule

applications. Once such a clause *A* ceases to be the focus of attention to direct or drive the program's reasoning, it is considered only as often as are the other clauses in list(usable) and not keyed on again, which is why I use the term temporary keying. On the other hand, each clause that is a member of the hot list is constantly keyed on (of course, assuming that the heat parameter is assigned a value greater than or equal to 1), in the sense that it is immediately considered with every newly retained clause, even if the retained clause is *never* chosen as the focus of attention.

2. The Field of Interest and Rules of Play

Throughout this article, I focus on two-valued sentential (or propositional) calculus and feature the objective of finding shorter proofs. (Nevertheless, as suggested by research not reported here, the approaches presented in this article are promising for other areas and for other objectives.) For the experiments I present, I choose as the axiom system for the calculus that of Lukasiewicz [Lukasiewicz63] consisting of *L*1, *L*2, and *L*3—the following expressed as clauses in OTTER notation, where the function *i* can be interpreted as *implication*, the function *n* as *negation*, and the predicate *P* as *provable*. (One can view *L*1 as transitivity of implication.)

(*L*1) P(i(i(x,y),i(i(y,z),i(x,z)))).
(*L*2) P(i(i(n(x),x),x)).
(*L*3) P(i(x,i(n(x),y))).

The inference rule I use is condensed detachment [Kalman78,Kalman83] captured by the use of hyper-resolution [Wos87,Wos92] and the following clause, in which ''-'' denotes logical **not** and '' | '' denotes logical **or**.

-P(i(x,y)) | -P(x) | P(y).

My study focuses on various other axiom systems of two-valued sentential calculus: those of Church, Frege, Hilbert, (an alternate of) Lukasiewicz, and Wos [Lukasiewicz63,Wos91]. In each experiment, I have the goal of deducing from the Lukasiewicz axiom system one or more of the other axiom systems under consideration. However, rather than seeking *any* proof, my goal is a set of proofs that are shorter than those of Lukasiewicz and, even better, proofs that offer research challenges in regard to their length. I shall show how I was able to obtain the desired shorter proofs and, far more satisfying, obtain them often in reasonable CPU time and with ease on my part.

I shall, where possible, comment on why I made various choices and pursued various paths. By doing so, I intend to provide a means for someone to extend the reported research and, even better, to apply some of the given methodology to attacking other areas, perhaps culminating in the answers to questions that are currently open. I also have the objective of fulfilling a frequent request to illustrate how a researcher can wisely choose from among the numerous options offered by OTTER.

As part of the game, I shall offer challenges and topics for study (see, for example, Section 7) and, in the Appendix, a sample input file and various pleasing proofs to aid the researcher. I issue the following important warning: Because this article (as others I have recently completed) represents current research, even before it appears, I may have obtained better results. In no way (in my view) does that compromise or lessen the intrigue or long-term value of attempting to meet one of the offered challenges. Indeed, quite the contrary, for an attempt to meet a challenge—even if it has already been met by another—can produce sharply different results, sometimes culminating in a significant advance for a field.

3. Combining Strategies

When one combines two different strategies, occasionally one produces a strategy with not-so-obvious properties. A powerful example of this occurrence is provided by combining the resonance strategy with the dynamic hot list strategy. A second example offering power focuses on combining the resonance strategy with the (static) hot list strategy. First in order is an illustration of using the main strategies of interest in this article, followed by examples of combining strategies.

3.1. Illustrating the Use of the Featured Strategies

With the resonance strategy, the researcher supplies resonators in the input, equations or formulas that are unit clauses, free of logical **or**. For OTTER, the resonators are usually placed in one of three weight lists: weight_list(pick_given), weight_list(pick_and_purge), and (far less often) weight_list(purge_gen). The first of the three cited weight_lists is used by the program to aid the program in deciding which clause on which to focus next to drive its reasoning. The second weight_list is used for the same purpose but, in addition, is used to aid in deciding which clauses to discard upon generation. The third weight_list is used only to aid in deciding which clauses to discard upon generation. If no member of a weight_list matches a clause (where variables are treated as indistinguishable), the clause is given a (pick_given) weight equal to its symbol count; the same is true for its purge_gen weight.

Because OTTER treats variables in weight templates as indistinguishable, one can conveniently include resonators for the program's use. To each resonator, as noted, the researcher assigns its value. One must always keep in mind that an included resonator is *not* a lemma or its equivalent; rather, a resonator is a symbol pattern used to give priority to clauses. When a clause matches a resonator (with, of course, the variables ignored), the priority assigned to it equals the value assigned to the matching resonator. The lower the value, the higher the priority.

For the program OTTER, the following example illustrates the use of a resonator, a resonator that corresponds to what Lukasiewicz calls thesis 35; see Section 4 for a list of theses that merit experimentation.

```
weight_list(pick_and_purge).
weight(P(i(i(x,i(y,z)),i(i(x,y),i(x,z)))),2).
end_of_list.
```

The presence of this resonator causes OTTER to assign a weight of 2—indeed a small weight—to any deduced clause that matches it, a clause such as the following.

```
P(i(i(x,i(y,z)),i(i(y,x),i(y,z)))).
```

The clause contained in the resonator differs from the preceding clause only in the particular variables that occur; specifically, some occurrences of *y* are replaced by *x* and some of *x* by *y*. From the viewpoint of the resonance strategy, the two clauses match, for variables are considered indistinguishable. To mechanically test a formula or equation for possibly matching a resonator, one first replaces in the resonator all occurrences of a variable with variables such that no two variables are identical, and then one determines whether a renaming of the variables yields the formula or equation; if such a renaming exists, then the formula or equation matches a resonator. Because of being assigned a weight of 2, the preceding clause, if deduced and retained, will be given great preference for driving the program's reasoning. Further, if the clause is deduced, it will be retained unless the max_weight is assigned a value (integer) strictly less than 2 or unless some procedure such as subsumption interferes.

If one wishes to experience the impressive impact of the inclusion of the cited resonator, one simply seeks a deduction of thesis 35 *with* and *without* its presence, using as axioms the three Lukasiewicz axioms given in Section 4. Indeed, I have an example where the inclusion of the pattern for thesis 35 sharply reduced the CPU time to prove the entire Church axiom system (consisting of theses 18, 35, and 49), in part by enabling OTTER to find a proof in which the last step is thesis 35 and that preceding it has the same shape.

In contrast to resonators having no true or false value, members of the hot list are treated as true, as lemmas or their equivalent. Such members play an active role in the program's reasoning, even though they must wait to *complete* applications of inference rules rather than to *initiate* applications. Members of the hot list also differ from resonators in their treatment of variables; indeed, variables in the former are treated as any variable in an equation or formula is treated, in contrast to variables in the latter being treated as indistinguishable and (in effect) place holders for some variable.

For example, the following two clauses (just used to illustrate the resonance strategy) are treated by the hot list strategy as distinctly different clauses.

```
P(i(i(x,i(y,z)),i(i(x,y),i(x,z)))).
```

P(i(i(x,i(y,z)),i(i(y,x),i(y,z)))).

Indeed, the presence of the correspondent of either clause as a resonator suffices to cause the program to key on either if retained, whereas to cause the program to key on the two clauses when the hot list strategy is used, one must include both clauses in the hot list (in the following manner).

```
list(hot).
P(i(i(x,i(y,z)),i(i(x,y),i(x,z)))).
P(i(i(x,i(y,z)),i(i(y,x),i(y,z)))).
end_of_list.
```

3.2. Examples of Combining Strategies

In the context of combining strategies, sometimes the researcher knows of one or more lemmas or facts (expressible as a unit clause) that, if provable, would make excellent members of the input hot list. Of course, such statements should not be cavalierly placed in the hot list, for they may not be provable. Also, sometimes the notion is that effectiveness would be sharply increased with the adjunction of one or more of the conjectured-to-be-valuable lemmas to the hot list *during* the run, if the program succeeds in proving any. In other words, sometimes the researcher would like the program, **if** a proof is produced of a given fact (expressed in the form of a unit clause), to **then** adjoin to the hot list the fact to be immediately visited with each newly retained clause. Intuitively, the wish is to give the program the ability to draw a conclusion and, **if** the conclusion matches a chosen target, **then** almost immediately use it to complete deductions. From the perspective of mathematics and logic, the idea is to give the program the ability to add *during* the run lemmas to immediately consider with each newly retained conclusion **if** the lemmas are proved. (In case one is tempted to simply say that a bit of programming would suffice for this type of **if-then** action, I note that the scope of combining both the resonance strategy and the dynamic hot list strategy is far greater than **if-then**.)

Just what is needed is provided by a judicious use of the resonance strategy combined with the dynamic hot list strategy. One places in the chosen weight_list, either the pick_and_purge weight_list or the pick_given weight_list, the resonators that match (ignoring variables) the desired lemmas, assigning to each resonator a value of, say, 2, and assigning to the dynamic_heat_weight a value of 2. With such an assignment, no clause whose pick_given weight is strictly greater than 2 can be adjoined (during the run) to the hot list. When and if the program deduces (proves) one of the desired lemmas, the corresponding clause will be assigned a pick_given weight of 2 because the matching resonator has been assigned a pick_given weight of 2. That new clause will be retained unless, of course, the max_weight is strictly less than 2 or subsumption interferes. Then, because the dynamic_heat_weight has been assigned the value 2, the newly retained clause (having a pick_given weight of 2) will immediately be adjoined to the hot list (*during* the run).

One does run the risk of having clauses adjoined to the hot list that one does not expect to be adjoined, namely, those that are not among the desired lemmas but, instead, match a resonator (with the variables ignored). (For example, the program would adjoin to the hot list during the run either or both of the clauses, if retained, that were used to illustrate the use of the resonance strategy if either were present as a resonator.) The fortune that results from this (so-to-speak) generality can be large or small, good or bad. For example, on the (possible) down side, if one combines the strategies in the cited manner and includes resonators that are easily matched, then the effectiveness of the program can be dramatically reduced, for too many clauses may be adjoined to the hot list during the run. The structure of the conclusions that are drawn by the program during a particular study sometimes provides a warning sign that effectiveness may be sharply reduced if the cited combination of strategies is used. Indeed, the inclusion of formulas (to be used as resonators) from equivalential calculus [Kalman78,Kalman83] in a study of that area of logic can be dangerous, for many formulas can match a single resonator.

Rather related to the combination of strategies just discussed is that in which one uses the resonance strategy and the hot list strategy, the latter *not* in a dynamic manner. Such a combination acts like a mixture, in contrast to the preceding which acts like a compound. Indeed, when the resonance strategy is combined with the dynamic hot list strategy in the manner described, an interaction takes place, for, among other factors, the nature of the hot list can be materially affected by the nature of the

list of resonators. Such is not the case when the hot list strategy is used in a static (rather than dynamic) manner. As the experiments discussed in Section 5 show, each of the two combinations is quite promising.

For a combination of strategies that may merit study and that possesses sharply different properties when compared with either of the preceding combinations, I suggest combining the hot list strategy with level saturation. By definition, the level of input clauses is 0; the level of a deduced clause is one greater than the maximum of the levels of its parents. With the choice of level saturation to direct a program's search, the program is instructed first to deduce all clauses of level 1 within the constraints of max_weight and the like, next to deduce all clauses of level 2, and to continue until stopped by some condition; in other words, a breadth-first search is conducted. As noted, regarding the hot list strategy, assigning the value 1 (for example) to the heat parameter instructs a reasoning program to deduce clauses of heat level 1, and assigning the value 2 to deduce clauses of heat level 2 (both relative to the hot list). If one assigns the heat parameter the value, say, 3 and uses level saturation, then, when the program is deducing clauses of level *j*, it is also deducing *some* clauses of level *j*+3. In other words, the combination of the hot list strategy and level saturation enables the program to ''look ahead'' into levels greater than that being explored. The aspect of looking ahead is also possessed by the use of the hot list by itself, for the immediate consideration of a newly deduced clause, rather than waiting until that clause is chosen (if ever) from the set of support to be used as the focus of attention, enables the program to look ahead by deducing clauses far earlier than they otherwise might be deduced.

Perhaps the question that comes to mind concerns the properties of combining all three strategies: resonance, the dynamic hot list, and level saturation. That question is left for future research, perhaps by others. What is next in order is a brief account of how the research that is the basis of this article commenced.

4. A Wellspring for Research

Next in order is the background and some results that together provide the basis for the research that is central to this article and that is reported in Section 5. The highlights of this section are displayed at its close.

In earlier research (to be documented in a forthcoming paper), my goal was to demonstrate the value of using the hot list strategy to bring within range the proofs of theorems that, before the formulation of the resonance strategy, had continually resisted the attack of an automated reasoning program. The area of study was two-valued sentential (or propositional) calculus, axiomatized by the three Lukasiewicz axioms (*L*1, *L*2, and *L*3) given near the beginning of Section 2. The inference rule in use was condensed detachment, captured by the use of hyperresolution and the three-literal clause given in Section 2. I had placed in the input hot list the clauses corresponding to the three Lukasiewicz axioms and the clause corresponding to condensed detachment. The target theorems were the following 68, called theses by Lukasiewicz [Lukasiewicz63]; theses 1, 2, and 3 are, respectively, *L*1, *L*2, and *L*3.

(thesis_04) i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))
(thesis_05) i(i(x,i(y,z)),i(i(u,y),i(x,i(u,z))))
(thesis_06) i(i(x,y),i(i(i(x,z),u),i(i(y,z),u)))
(thesis_07) i(i(x,i(i(y,z),u)),i(i(y,v),i(x,i(i(v,z),u))))
(thesis_08) i(i(x,y),i(i(z,x),i(i(y,u),i(z,u))))
(thesis_09) i(i(i(n(x),y),z),i(x,z))
(thesis_10) i(x,i(i(i(n(x),x),x),i(i(y,x),x)))
(thesis_11) i(i(x,i(i(n(y),y),y)),i(i(n(y),y),y))
(thesis_12) i(x,i(i(n(y),y),y))
(thesis_13) i(i(n(x),y),i(z,i(i(y,x),x)))
(thesis_14) i(i(i(x,i(i(y,z),z)),u),i(i(n(z),y),u))
(thesis_15) i(i(n(x),y),i(i(y,x),x))
(thesis_16) i(x,x)
(thesis_17) i(x,i(i(y,x),x))
(thesis_18) i(x,i(y,x))
(thesis_19) i(i(i(x,y),z),i(y,z))

(thesis_20) i(x,i(i(x,y),y))
(thesis_21) i(i(x,i(y,z)),i(y,i(x,z)))
(thesis_22) i(i(x,y),i(i(z,x),i(z,y)))
(thesis_23) i(i(i(x,i(y,z)),u),i(i(y,i(x,z)),u))
(thesis_24) i(i(i(x,y),x),x)
(thesis_25) i(i(i(x,y),z),i(i(x,u),i(i(u,y),z)))
(thesis_26) i(i(i(x,y),z),i(i(z,x),x))
(thesis_27) i(i(i(x,y),y),i(i(y,x),x))
(thesis_28) i(i(i(i(x,y),y),z),i(i(i(y,u),x),z))
(thesis_29) i(i(i(x,y),z),i(i(x,z),z))
(thesis_30) i(i(x,i(x,y)),i(x,y))
(thesis_31) i(i(x,y),i(i(i(x,z),u),i(i(y,u),u)))
(thesis_32) i(i(i(x,y),z),i(i(x,u),i(i(u,z),z)))
(thesis_33) i(i(x,y),i(i(y,i(z,i(x,u))),i(z,i(x,u))))
(thesis_34) i(i(x,i(y,i(z,u))),i(i(z,x),i(y,i(z,u))))
(thesis_35) i(i(x,i(y,z)),i(i(x,y),i(x,z)))
(thesis_36) i(n(x),i(x,y))
(thesis_37) i(i(i(x,y),z),i(n(x),z))
(thesis_38) i(i(x,n(x)),n(x))
(thesis_39) i(n(n(x)),x)
(thesis_40) i(x,n(n(x)))
(thesis_41) i(i(x,y),i(n(n(x)),y))
(thesis_42) i(i(i(n(n(x)),y),z),i(i(x,y),z))
(thesis_43) i(i(x,y),i(i(y,n(x)),n(x)))
(thesis_44) i(i(x,i(y,n(z))),i(i(z,y),i(x,n(z))))
(thesis_45) i(i(x,i(y,z)),i(i(n(z),y),i(x,z)))
(thesis_46) i(i(x,y),i(n(y),n(x)))
(thesis_47) i(i(x,n(y)),i(y,n(x)))
(thesis_48) i(i(n(x),y),i(n(y),x))
(thesis_49) i(i(n(x),n(y)),i(y,x))
(thesis_50) i(i(i(n(x),y),z),i(i(n(y),x),z))
(thesis_51) i(i(x,i(y,z)),i(x,i(n(z),n(y))))
(thesis_52) i(i(x,i(y,n(z))),i(x,i(z,n(y))))
(thesis_53) i(i(n(x),y),i(i(x,y),y))
(thesis_54) i(i(x,y),i(i(n(x),y),y))
(thesis_55) i(i(x,y),i(i(x,n(y)),n(x)))
(thesis_56) i(i(i(i(x,y),y),z),i(i(n(x),y),z))
(thesis_57) i(i(n(x),y),i(i(x,z),i(i(z,y),y)))
(thesis_58) i(i(i(i(x,y),i(i(y,z),z)),u),i(i(n(x),z),u))
(thesis_59) i(i(n(x),y),i(i(z,y),i(i(x,z),y)))
(thesis_60) i(i(x,i(n(y),z)),i(x,i(i(u,z),i(i(y,u),z))))
(thesis_61) i(i(x,y),i(i(z,y),i(i(n(x),z),y)))
(thesis_62) i(i(n(n(x)),y),i(x,y))
(thesis_63) i(x,i(y,y))
(thesis_64) i(n(i(x,x)),y)
(thesis_65) i(i(n(x),n(i(y,y))),x)
(thesis_66) i(n(i(x,y)),x)
(thesis_67) i(n(i(x,y)),n(y))
(thesis_68) i(n(i(x,n(y))),y)
(thesis_69) i(x,i(n(y),n(i(x,y))))
(thesis_70) i(x,i(y,n(i(x,n(y)))))
(thesis_71) n(i(i(x,x),n(i(y,y))))

When (in my earlier research) the hot list strategy essentially unaided (by the use of other strategies or by guidance from the researcher) proved 44 of the 68 theses, I was not totally pleased, despite the slightly encouraging results. (For those who enjoy history, the study of the given 68 theses led

directly to the formulation of the resonance strategy; its first use [Wos91] proved all 68 in less than 16 CPU-minutes on a SPARCstation-1+.) In addition, in the study reported here, I was after bigger game than proving a large fraction of the 68 theses. Specifically, I was intent upon proving a significant theorem, namely, that the three Lukasiewicz axioms provide an axiomatization for two-valued sentential calculus, where the targets were one or more of the other known axiom systems. The other axiom systems are (an alternate) from Lukasiewicz, consisting of theses 19, 37, and 59; from Church, consisting of theses 18, 35, and 49; from Frege, consisting of theses 18, 21, 35, 39, 40, and 46, of which thesis 21 is dependent on the other five; from Hilbert, consisting of theses 3, 18, 21, 22, 30, and 54, of which thesis 30 is dependent on the other five; and from Wos, consisting of theses 19, 37, and 60. Of course, since the members of the target axiom systems are all among the 68 theses, a proof of all 68 theses would (in effect) win the bigger game (by deducing each of the other five known axiom systems). I note that (in that earlier study) I was *not* focusing on *any* particular properties of the sought-after proofs, for example, proof length. By proof length, I mean the number of deduced steps, and I require that every deduced step be justified by the application of some inference rule.

I had an inspiration. If (for the study reported here) I could find a means to have potentially powerful clauses adjoined to the hot list *during* the run, then the effectiveness of the hot list strategy could be further increased. The added power might enable OTTER to reach both objectives, that of proving more than 44 of the 68 theses, and that of deducing one or more of the other five known axiom systems, of course, using as hypotheses the three axioms of Lukasiewicz. Because my experiments have repeatedly shown that resonators of various types offer substantial power, my notion was to combine the resonance strategy with the dynamic hot list strategy.

The idea was to include in the input highly attractive resonators. For this research, a highly attractive resonator is a resonator such that, among the clauses that match it (ignoring variables), there exists at least one that, if deduced, would make a good addition to the hot list. So the burden shifted to deciding which clauses I would like adjoined to the hot list, if deduced. I reasoned (I believe) that, if including the three Lukasiewicz axioms in the hot list had in earlier research proved a good move—which it in fact had—then the inclusion of members of other known axiom systems would be a good move. Of course, as is probably obvious, I could not simply include (in the initial hot list) such members in the input, for I was not allowed to assume their deducibility.

My solution was, other than *L*3, to place as a resonator in weight_list(pick_and_purge) the template corresponding to each of the (additional fourteen) members of the other five axiom systems, assigning to each a weight of 2. I also assigned the value 2 to the dynamic_heat_weight. The consequences of these two actions is that, if a clause matching one of the fourteen resonators is deduced, it will be assigned a pick_given weight of 2. If none of the other mechanisms (such as subsumption or max_weight) interfere, the clause will be retained. Because its weight is 2 and the dynamic_heat_weight is 2, the clause will immediately be adjoined to the hot list. I was more than willing to run the risk that clauses (other than the desired fourteen) not identical to, but similar to (if variables are ignored), a resonator would be adjoined to the hot list.

For my experiment, I chose as options to assign the value 20 to max_weight, 5 to max_distinct_vars, 5 to the pick_given_ratio, 1 to the heat parameter, and 2 to the dynamic_heat_weight. The first two values were chosen because all 68 theses expressed as clauses have weight (in symbol count) less than or equal to 20—thesis 7 in fact has a weight of 20—and because none of the cited clauses relies on strictly more than five distinct variables. *During* the run, OTTER adjoined 41 new clauses to the hot list. Regarding the 68 theses, 52 were deduced—not extremely impressive. However, among those proved are theses 46 and 54, important because of being members, respectively, of the Frege axiom system and the Hilbert axiom system. More significant, I caught my bigger game: The Hilbert axiom system was proved, in approximately 1039 CPU-seconds on a SPARCstation-10 with a length of 50 and a level of 26. (The cited proof length is 54, the greater number explained by the presence of duplicate steps resulting from the use of the dynamic hot list strategy. In particular, when a clause is added to the hot list during the run, that clause is a copy, but with a different number, of the clause that is then added to list(sos). When duplicate steps occur in a proof, the cause rests with the use of a clause and also of its copy from the hot list. The designation heat=n asserts that the hot list was consulted at the *n*-th level to yield the corresponding clause. Note that the notation heat=n concerns the history of the clause, and says nothing about how the clause is used.)

These (encouraging to me) results obtained by combining the resonance strategy and the dynamic hot list strategy (in the cited manner) provided the wellspring for the following research that is reported in Section 5.

Highlights of Section 4

- With the hot list strategy alone, OTTER proved 44 of the target 68 theses cited in this section, but proved none of the entire target axiom systems.
- Using as resonators members of known axiom systems, the combination of the resonance strategy with the dynamic hot list strategy proved 52 of the 68 theses and also proved the Hilbert axiom system.

5. The Actual Research

To aid the researcher, appended to the end of this section are the displayed highlights.

The just-cited success regarding the deduction of the Hilbert axiom system naturally called for further testing of the power of combining resonance with heat. Both combinations merited experimentation, the resonance strategy with the hot list strategy and the resonance strategy with the dynamic hot list strategy. Of the myriad of possibilities for experimentation, rather than seeking proofs *without* regard to their properties, my choice was to seek shorter proofs than were familiar to most researchers. The area of study was two-valued sentential (or propositional) calculus, axiomatized by the three Lukasiewicz axioms ($L1$, $L2$, and $L3$) given near the beginning of Section 2. The inference rule in use was condensed detachment, captured by the use of hyperresolution and the three-literal clause given in Section 2. The targets were the Lukasiewicz proofs of the other five axiom systems.

Taking into account the facts that Lukasiewicz combines proof steps and occasionally renames variables, his proof of the Church axiom system has length 33, of the Frege axiom system has length 41, of the Hilbert axiom system has length 34, of the alternate Lukasiewicz axiom system has length 37, and of the Wos system has length 39. Of course, technically, he had not proved the Wos axiom system, for it was not reported until 1991. Proof length is measured in number of applications of condensed detachment. My immediate goal was, if possible, to use a combination of resonance and heat to find proofs strictly shorter than those of Lukasiewicz.

The first decision to make concerned the choice of resonators for use by the resonance strategy. The decision was indeed influenced by my deeming it best to attack each of the five axiom systems as a target one at a time. I started with the Church axiom system consisting of theses 18, 35, and 49. A natural first cut at a set of useful resonators focuses on the Lukasiewicz original proof, essentially consisting of 33 applications of condensed detachment. The 33 resonators corresponding to the deduced steps of that proof were obtained by using OTTER in a rigorous proof-checking mode, in contrast to its use in a cursory proof-checking mode; both types of proof checking are discussed shortly and discussed in more detail in Section 5.4. (To reflect their source and their use, I call such resonators Lukasiewicz-Church resonators, and, later in this section, I follow a similar naming convention.) As for the assignment to each resonator, I followed my custom of the past few years, assigning to each the value 2. Since I was intent upon testing my conjecture asserting that effectiveness can be sharply increased if one instructs the program to adjoin *during* the run new members to the hot list, where the new members are chosen because of matching an (input) resonator, I naturally assigned the dynamic_heat_weight the value 2. I was aware of the obvious danger, namely, that, in addition to adjoining to the hot list members of the Lukasiewicz proof, perhaps far too many other clauses would be adjoined during the run because of matching one of the thirty-three resonators.

My choice for the initial members of the (input) hot list was dictated by following my own recommendation: Include in the hot list the members of the initial set of support. (I almost always rely on the use of the set of support strategy [Wos87,Wos92].) In this case, I therefore included the three Lukasiewicz axioms. Of course, I also included in the initial hot list the clause correspondent of condensed detachment; otherwise the hot list strategy would be disabled, for the use of hyperresolution requires (in this case) the nucleus to be a member of the hot list. Regarding the assignment to the (input) heat parameter, I reasoned that, if all went as planned, OTTER would deduce most or all of the

clauses corresponding to the Lukasiewicz proof of the Church axiom system, and then adjoin each to the hot list during the run. Also, since weighting treats variables as indistinguishable, other clauses would most likely be adjoined to the hot list because of matching in shape one of the resonators. Therefore, the size of the hot list might become rather large. I was thus led to assign the value 1 to the heat parameter.

As for the other options, my choices were in part based on more than thirty years of experimentation, in part based on the success reported in Section 4, and in part based on motivation lost to unrecorded history. I assigned max_weight the value 20, max_proofs the value -1 (to permit the program to complete as many proofs as allowed by the constraints placed on memory and CPU time, respectively, 60 megabytes and none), and max_distinct_vars the value 5 (to partially curb the rate of conclusion retention). I chose 20 for the max_weight because, in terms of symbol count and in the presence of the predicate *P*, the largest weight of theses 4 through 71 is 20; thesis 7 has weight 20. I also assigned the pick_given_ratio the value 5. I set hyperresolution as the inference rule (for condensed detachment), order_history (to instruct the program to list the ancestors in the order nucleus, major premiss, minor premiss), and set input_sos_first (to force the program to choose as the focus of attention the members of the initial set of support before choosing any deduced clause for that purpose). Because I chose to use the dynamic hot list strategy, for my weight_list in which to place the chosen resonators, I used weight_list(pick_and_purge). Had I instead used weight_list(purge_gen), then clauses matching a resonator would have been given their pick_given weight based on symbol count, for resonators in that weight_list affect only the computation of weight in the context of discarding clauses. In that event, almost certainly all such (matching) clauses would have failed to be adjoined to the hot list (during the run), contrary to my intention.

As for the input clauses, I placed only those corresponding to the three Lukasiewicz axioms in list(sos), I placed the clause for condensed detachment and those corresponding to the denials or negations of the five target axiom systems in list(usable), and I placed in list(passive) the negations or denials of the individual members of the five target axiom systems. My motivation for my choice regarding the initial set of support was my preference for *initiating* applications of hyperresolution with each of the Lukasiewicz axioms (in the beginning). My choice regarding the members of the (input) list(usable) was based on which clauses I wished to *complete* applications of hyperresolution, including providing a means for detecting when an *entire* axiom system had been deduced. In other words, I thought it possible that, although the Church axiom system was under study and the resonators were taken from a deduction of that system, other axiom systems might be deduced. As for the choice of the members of list(passive), I wished to monitor the progress of the program; clauses in that list are used for forward subsumption and, important in this context, for unit conflict.

I thus began my assault on the problem of finding (if possible) shorter proofs for one, or for all, of the five target axiom systems, at this point emphasizing the study of the Church axiom system. I won the first round, but only barely. Indeed, although all members of the Church axiom system as well as all members of the alternate Lukasiewicz axiom system were deduced (in approximately 79 CPU-seconds on a SPARCstation-10), only the proof of the Church system (which has length 32 in contrast to the Lukasiewicz proof having length 33) completed, while the proof of the alternate Lukasiewicz system did not complete. The program reported a proof length of 38 for the Church axiom system, again explained by the presence of duplicate steps resulting from the use of the dynamic hot list strategy. The proof was completed in approximately 114 CPU-seconds, the added time explained when the discussion turns (almost immediately) to the mechanism for choosing the clauses on which to focus attention. As evidence that the hot list strategy played a key role, 15 of the 32 steps of the proof show heat=1 in their history, meaning that a hot list member participated. (For the researcher interested in finer detail, we note that having the proofs of the individual members of an axiom system is far from satisfying when the goal is a proof of the *entire* system, for, among other drawbacks, the ensemble of individual proofs will almost certainly contain many duplicate steps.) But, one might immediately wonder, if all members of an axiom system were deduced (namely, the alternate of Lukasiewicz), then why was no proof of that axiom system completed?

The answer rests with the mechanism for choosing clauses on which to focus to drive the program's reasoning. In particular, the choice among the retained clauses is based on pick_given weight, which is first determined by included weight templates present either in

weight_list(pick_and_purge) or in weight_list(pick_given), and then (if none apply) by symbol count. The clause correspondent of thesis 59 (the last member of the alternate Lukasiewicz system that was deduced) was not chosen as the focus of attention; its weight is 15. Nor was the clause equivalent of thesis 37 chosen as the focus of attention, although its weight is only 11. Evidently, too many clauses were available with smaller weight. Had I included the resonators corresponding to theses 37 and 59, then the deduction of the corresponding clauses would almost immediately have caused them to be chosen as the focus of attention, resulting in the completion of a proof of the alternate Lukasiewicz axiom system because the remaining member (thesis 19) had in fact been chosen as the focus of attention. The absence of the cited resonators is explained by my intention to study the effects of using as resonators only the steps of a single proof of an entire axiom system in combination with the dynamic hot list strategy (in effect) directed by those resonators. Therefore, any additional resonator inclusion would have compromised my experiment.

Regarding the cited danger in the context of clause adjunction to the hot list during the run, 265 clauses were adjoined. Perhaps those added clauses or perhaps the latitude permitted with the max_weight of 20 caused the program to find the not-altogether-satisfying 32-step proof of the Church axiom system. On the other hand, it was a plus to have the use of the Lukasiewicz-Church resonators lead to a deduction of all members of another axiom system, namely, the alternate of Lukasiewicz. That result supports the claim that proof steps from one theorem are useful in increasing the effectiveness of a reasoning program when attacking related theorems.

As a result of the first round yielding less progress than I was after, I immediately reduced the max_weight to 2, vaguely in the spirit of proof checking. The idea was to confine the program's retention of information to those clauses matching one of the thirty-three resonators. The corresponding experiment also produced a 32-step proof of the Church axiom system, in approximately 133 CPU-seconds; eight of the steps are not present in the 32-step proof produced earlier with max_weight = 20. The partially redeeming property of deducing all members of another axiom system was even absent. I turned to Round 2.

In Round 2, I continued to use the thirty-three resonators corresponding to the Lukasiewicz proof of the Church axiom system, conjecturing that their presence might again lead to other interesting proofs. However, I replaced the dynamic hot list strategy with the hot list strategy, thus avoiding the adjunction of clauses during the run, clauses that might have been the cause of the small amount of progress. In other words, I decided to explore the power of the other combination of resonance and heat. All other parameters and options were left unchanged from Round 1.

Round 2 I rate a success. In approximately 31 CPU-seconds, OTTER deduced the Church axiom system, the proof being of length 29 and level 24; 9 of the deduced steps result from the use of the hot list strategy. In approximately 212 CPU-seconds, OTTER deduced the Frege axiom system, the proof being of length 40 and level 24; 14 of the deduced steps have heat=1 in their history. In approximately 362 CPU-seconds, OTTER deduced the Hilbert axiom system, the proof being of length 31 and level 22; 10 of the deduced steps have heat=1 in their history. Also, although not completing the corresponding proof, the program did deduce all members of the alternate Lukasiewicz axiom system in approximately 563 CPU-seconds. The program failed to complete the proof because the clause correspondent of thesis 59 (the last member of the alternate Lukasiewicz system that was deduced) was not chosen as the focus of attention; its weight is 15. This experiment clearly (to me) yielded additional evidence of the value of using the hot list strategy and of the value of using as resonators steps of the proof of one theorem (the Lukasiewicz proof of the Church axiom system) to sharply increase the likelihood of proving related theorems.

Highlights of Section 5

- With the goal of finding shorter proofs using the dynamic hot list strategy combined with the resonance strategy, where the resonators correspond to the Lukasiewicz 33-step proof of the Church axiom system, a 32-step proof of the Church axiom system was completed.
- When the dynamic hot list strategy was replaced with the hot list strategy, the Church axiom system was completed in 29 steps, the Frege axiom system was completed in 40 steps (in contrast to

the Lukasiewicz 41-step proof), and the Hilbert axiom system was completed in 31 steps (in contrast to the Lukasiewicz 34-step proof).

5.1. A Promising Refinement

The approach formulated in this section is summarized at its close, and highlights are displayed.

The inspiration for Round 3 was based in part on the results of Rounds 1 and 2 and in part on the results reported at the end of Section 4. The plan was to return to the combination of resonance and the dynamic hot list strategy, using as resonators (for my emphasis on the Church axiom system as main target) the thirty-three steps of the Lukasiewicz proof of the Church axiom system, also the members of the five axiom systems under attack, and also the members of the main target axiom system. To permit the hot list to have clauses adjoined to it during the run, but not so rapidly, I assigned the dynamic_heat_weight the value 2, a weight of 1 to the resonators corresponding to the members of the Church axiom system, a weight of 2 to the resonators that corresponded to the remaining members of the five axiom systems (of course, omitting *L*3), and a weight of 3 to the resonators that corresponded to the Lukasiewicz proof. (I include in the Appendix an input file of the type under discussion.)

With the actions just described, because OTTER assigns a weight to a clause based on the earliest template in the input that matches it, the deduction, say, of the clause correspondent to thesis 18 (a member of the Church axiom system) would result in a weight of 1 being assigned to it. Such a clause, if deduced and retained, would be adjoined to the hot list during the run because of the assignment of the value 2 to the dynamic_heat_weight. Also as a consequence of the cited actions, the deduction of a member of the Lukasiewicz proof of the Church axiom system would, in most cases, cause that clause to be assigned a weight of 3. Because the dynamic_heat_weight is assigned the value 2, such a clause would *not* be adjoined to the hot list during the run. Except for max_weight, it seemed reasonable to leave the other options unchanged. Regarding max_weight, two values were appealing: that of 20, and that of 3. Therefore, as with Round 1, Round 3 consisted of the corresponding two experiments.

When the max_weight was assigned the value 20, where proof length is the criterion, I think I can say without hesitation that the experiment failed badly. The proofs ranged in length from 44 for the Church axiom system to 51 for the Wos axiom system. Perhaps the failure resulted from the wrong clauses being adjoined to the hot list during the run, in this case, those corresponding to members of other axiom systems that are (at the same time) not among the members of the Church axiom system. Nevertheless, all five axiom systems were deduced, and approximately 63 CPU-seconds was all that was required.

On the other hand, Round 3 was a success when the max_weight was assigned the value 3. In approximately 6 CPU-seconds, OTTER completed a 27-step proof of level 21 of the Church axiom system (but, because of duplicate steps, announced the proof to be of length 28); in approximately 63 CPU-seconds, the program completed a length 29 and level 23 proof of the Hilbert axiom system. Also of interest, the proof of the Church axiom system is *compact* [Wos95a] in that it is precisely a proof of one of the members of the system, namely, thesis 35; I return to the subject of compact proofs throughout this section and also in Sections 6 and 7. Informally, as an example, if the theorem has the form *P* implies the **and** of *Q*, *R*, and *S*, then a proof of the theorem is compact if and only if it is a proof of precisely one of *Q*, *R*, or *S* (of course, with no extra steps).

Progress was (to me) more than satisfactory. Indeed, perhaps a fairly powerful method for finding shorter proofs had been formulated. I did test and evaluate this refinement, as reported in Section 5.2.

That the hot list strategy played a key role is evidenced by the fact that 9 of the 27 steps in the deduction of the Church axiom system show heat=1 in their history, and 10 of the 29 steps in the deduction of the Hilbert axiom system show heat=1. For the curious, 23 of the 27 proof steps of the Church axiom system are present in the proof of the Hilbert axiom system. Regarding the resonance strategy, in particular, the resonators corresponding to the deduced steps of the Lukasiewicz proof of the Church axiom system, 4 steps of the just-cited 27-step proof and 3 of the just-cited 29-step proof are not among them; 1 of the 27 steps and 2 of the 29 steps do not match, when variables are ignored, any of the 33 resonators corresponding to the Lukasiewicz proof.

Rather than turning immediately to using the refined approach for an attack on finding shorter proofs (than those of Lukasiewicz) for the other axiom systems, I (in effect) paused to proof check my result regarding the 27-step proof of the Church axiom system. Such a move is typical in my research. The proof checking I instructed OTTER to do is of a cursory nature, in contrast to rigorous proof checking; see Section 5.4. In particular, I placed in weight_list(pick_and_purge) resonators corresponding to the deduced steps of the 27-step proof of the Church axiom system, each with an assigned weight of 2. By then assigning a max_weight of 2, which I did, the program is virtually guaranteed to retain a clause only if it matches a resonator. Regarding the other options used in Round 3, I made but one change: I removed the use of the hot list. Therefore—so I have always expected, with almost certainty—the program will simply reproduce (with far less CPU time than originally required) the proof whose steps are captured with weight templates. (This cited strong conviction will play an important role later in this section.) OTTER did essentially return the expected 27-step proof of the Church axiom system. Not astounding to me, the order, and hence the history, was slightly different, but no new deduced steps were present.

Closer inspection provided an explanation for the different step order and history (and hence, technically, different proof). Indeed, the reduction in the assigned max_weight from 3 to 2 had prevented the program from delaying as choice for the focus of attention the clause corresponding to one of the proof steps. In addition, the use of the hot list strategy in Round 3 had enabled OTTER to deduce certain proof steps earlier than occurred in the cursory proof-checking run. An obvious experiment to make then was to assign the max_weight the value 3 and add the use of the hot list strategy, in the manner used in Round 3.

That experiment did in fact yield a proof, but not the one being proof checked. Instead, a 30-step proof was found, 9 steps of which are present because of using the hot list strategy. The proof is not compact. Five of the deduced steps are not among the 33 of which the Lukasiewicz proof consists. Tantalizing (to me), the 30-step proof contains all 27 steps of the proof found in Round 3, but the 27-step proof is *not* a subproof of the 30-step proof. (Of a different nature entirely, tantalizing also is the fact that cursory proof checking can, occasionally, yield no proof at all, a subject discussed in Section 5.4 with an example.)

Before leaving the so-called proof-checking attack on the Church axiom system, one more experiment was in order, that in which the dynamic hot list strategy is added to the preceding options. Instead of the 27-step proof being proof checked, perhaps unforeseen, OTTER found a 29-step proof, announcing a proof of length 33 containing duplicate steps resulting from using the dynamic hot list strategy. Of the 29 deduced steps, 14 show heat=1 in their history. Of the steps of the earlier-cited 27-step proof, all are members of the 29-step proof, but the former is not a subproof of the latter. The explanation for the different results obtained with cursory proof checking in combination with the hot list strategy, the dynamic hot list strategy, or neither rests with the perturbing of the search space that can occur. For example, use of the hot list strategy enables a program to draw conclusions far earlier than it might otherwise draw them.

To close this section and set the stage for presenting the results of testing the refined approach, a summary of it at the general level is in order. The general notion is to combine the resonance strategy with the dynamic hot list strategy in a manner that encourages additions to the hot list during the run, but only if the addition matches a resonator that captures some significant result. Three sets of resonators are placed in weight_list(pick_and_purge), with all members of the first set assigned a weight of i, all members of the second set a weight of j, and all members of the third set a weight of k, where $i < j < k$. The dynamic_heat_weight is assigned the value j, and (usually) the max_weight is assigned the value k. In the typical experiment reported here, $i = 1$, $j = 2$, and $k = 3$.

At the syntactic level, the resonators of the first set are symbolically identical to the clause equivalents of the targets of the experiment. For example, the targets (in the sense used here) of an experiment designed to deduce the Church axiom system are its individual members. Since, by definition, variables in a resonator are ignored, it is the functional shape that the targets provide. The resonators so chosen capture the individual targets. In a similar manner, resonators of the second set are obtained by choosing powerful formulas or equations of the field from which the problem is taken, such as the members of other axiom systems. Finally, the resonators of the third set correspond (in the sense just described) to the steps of some proof of interest in the field.

Because of the assigned weights—if the max_weight is small enough, say 3, to cause the program to exclude from retention any clause unless it matches a resonator—preference for directing a program's reasoning is first given to a clause that matches a resonator in the first set, second to a member that matches a resonator in the second set, and third to a member that matches a resonator in the third set. On the other hand, if one wishes other clauses to be retained and used to direct the reasoning, clauses whose weight will be based solely on symbol count, then one chooses the max_weight accordingly, but greater than *k*. If one wishes the program to (possibly) expand the hot list during the run by adjoining clauses that do not match a resonator, one assigns a value larger than *j* to give the program the needed latitude. In the vast majority of cases, the weight of a clause is strictly greater than 3 when measured by symbol count. If a clause matches two resonators, that appearing earliest in the input dictates the weight of the clause, for weighting (in OTTER) focuses on the first template that matches. Similarly, in the context of adjoining a clause to the hot list during the run, if the clause matches, say, a resonator in the first set and one in the third set, the clause will be adjoined. If the clause matches a resonator in the third set but not in either the first or second, because of the weight it will be assigned (that of the matching resonator) and because of the assigned value to the dynamic_heat_weight, the clause will *not* be adjoined to the hot list.

Intuitively, behind this layered-resonator refinement is the notion that any target that is deduced and retained merits use (more than any other deduction) for driving the program's reasoning and also merits inclusion in the hot list. The inclusion in the hot list of such a clause makes it available for the remainder of the run for immediately *completing* rather than *initiating* applications of an inference rule, where, because of using the hot list strategy, (in this context) the initiation of the the application occurs by immediately focusing on each new clause that is retained *before* any other clause is chosen as the focus of attention. Similar observations apply to any clause that is deduced and retained and that is a (designated) significant formula or equation in the field under study. Such a clause is given slightly less preference for driving the program's reasoning, but, in the context of the hot list, it is treated no differently from a deduced and retained target clause. Finally, deduced and retained conclusions that are steps of the chosen proof are given high preference for directing the reasoning, but less preference than either of the other two types of conclusion. Such a clause is not, however, given the role for immediate consideration with each newly retained clause for completing an application of an inference rule. Because resonators treat variables as indistinguishable, deduced and retained conclusions other than targets, significant formulas and equations, and steps of the chosen proof may receive the type of classification just discussed. But perhaps that is how it should be, for such a conclusion does have the same functional shape as a member of one of the three classes.

Highlights of Section 5.1

- The layered refinement is a strategy that combines the dynamic hot list strategy with the resonance strategy, where three sets of resonators are used, the last assigned a weight greater than the assigned dynamic_heat_weight.
- With the layered refinement—using for the first set of resonators the three members of the Church axiom system, for the second set fourteen members of five target axiom systems, and for the third set the thirty-three deduced steps of the Lukasiewicz proof of the Church system—OTTER completed a 27-step proof of the Church axiom system and a 29-step proof of the Hilbert axiom system.

5.2. Testing and Evaluating the Refinement

Next in order was to test the conjecture that the refinement of combining resonance and heat was indeed fairly powerful. The approach consists of combining the resonance strategy and the dynamic hot list strategy in the following manner. Three sets of resonators are chosen, the first corresponding to the target(s)—for example, the members of the Church axiom system—the second corresponding to formulas or equations that have proved powerful (such as the members of the five axiom systems), and the third corresponding to the proof steps of some published result. All members of the first set are assigned the same small weight, a weight less than the assignment to the dynamic_heat_weight. All members of the second set are assigned the same weight, a weight slightly larger than that assigned to

the members of the first set but still less than or equal to the dynamic_heat_weight. All members of the third set are assigned the same weight, a weight one larger than that assigned to the dynamic_heat_weight. Finally, a small value, say 2, is assigned to the dynamic_heat_weight, and the max_weight is assigned to be one greater, say 3. The idea is to (1) rely on resonators that correspond to useful steps of some proof; (2) prevent an addition to the hot list during the run, as a result of the deduction of any of those resonators (with a few exceptions); and (3) at the same time use as resonators the exceptions and (possibly) other interesting formulas or equations in a manner that permits the size of the hot list to grow not too rapidly during the run.

5.2.1. The Lukasiewicz-Hilbert Resonators

The highlights of this subsection are displayed at its close.

To test the power of the approach by applying it to other axiom systems as the focus of attention, in the context of finding shorter proofs, I first turned to an attack on the Hilbert axiom system (consisting of theses 3, 18, 21, 22, 30, and 54, of which thesis 30 is dependent on the other five). The most appealing approach (to me) was to parallel the most successful experiment focusing on the Church axiom system (discussed in Section 5.1). Therefore, I (in effect) turned to the successful part of Round 3, replacing the 33 resonators corresponding to the Lukasiewicz proof of the Church axiom system with resonators that correspond to the 34 deduced steps of the Lukasiewicz deduction of the Hilbert axiom system. (For the curious, 6 of the deduced steps of the Lukasiewicz proof of the Church axiom system are not present in his proof of the Hilbert axiom system; hence, of the 34 resonators being used, 7 are not used in Round 3. Regarding the resonators corresponding to the 68 theses 4 through 71, 4 of the 33 resonators for the Church study are not among them, and 4 of the 34 resonators for the Hilbert study are not among them.) Each resonator was assigned a weight (priority) of 3. The dynamic hot list strategy was used, with the dynamic_heat_weight assigned the value 2. The max_weight was assigned the value 3. In weight_list(pick_and_purge) I placed the following: first, the 5 resonators corresponding to the members of the Hilbert axiom system (excluding *L*3), each assigned the value 1; second, the 14 correspondents of the members of the five axiom systems under study, each with a weight of 2; and third, the 34 resonators being used. As in the Church study, just in case marvelous events occurred regarding short proofs, I included as possible targets for proof all five axiom systems.

Success was the result. In approximately 3 CPU-seconds on a SPARCstation-10, a deduction of the Hilbert axiom system was completed. The length of the deduction is 26 (but announced as 27 because of the duplicate step resulting from the use of the dynamic hot list strategy), and the level is 18. Regarding the role of the dynamic hot list strategy, 8 of the 26 steps show heat=1. In view of the fact that the Lukasiewicz proof has length 34, the result suggests that the approach being tested is indeed powerful. Regarding the actual deduced formulas, 25 of the 26 are among the 34 deduced steps of the Lukasiewicz proof of the Hilbert axiom system. I find such aspects piquant, that a rearrangement of formulas (and, of course, their respective history) permits omitting (in this case) 9 and requiring but 1 additional.

Even more evidence of the usefulness of the approach under study was provided by a glance at the output. In approximately 2 CPU-seconds, even before the Hilbert axiom system was proved, a proof was completed of the alternate Lukasiewicz axiom system, that consisting of theses 19, 37, and 59. The proof has length 26 (which is also the announced length) and level 19. The Lukasiewicz proof has length 37. Of the 26 deduced steps, 8 show heat=1, and 6 are not present among the 34 steps of the Lukasiewicz proof of the Hilbert axiom system. Taking into account the facts that the max_weight was assigned the value 3, that no formula can have (in symbol count) such a small weight, and that resonators are treated (in effect) as templates with their variables ignored, one has rather substantial evidence of the value of the resonance strategy in general and, in particular, of the resonators that were used. The choice of resonators is dictated by the approach being evaluated.

Still more evidence of the power of the approach is found in the output. In particular, in approximately 2 CPU-seconds, before completing a proof of the Hilbert axiom system, a proof is completed of the Wos axiom system (consisting of theses 19, 37, and 60). The proof has length 27 (which is also the announced length) and level 20. Of the 27 deduced steps, 9 show heat=1, and 7 are not present among the 34 steps of the Lukasiewicz proof of the Hilbert axiom system. The proof, whose last step is thesis

60, contains as a subproof that of the alternate Lukasiewicz axiom system. Then, after deducing the Hilbert axiom system, OTTER completed a proof of the Church axiom system, one of length 28 and level 21. Of the deduced steps, 8 have heat=1 in their history, and 7 are not present among the 34 steps of the Lukasiewicz proof of the Hilbert system.

Highlights of Section 5.2.1

- With the layered refinement—using for the first set of resonators five members (excluding *L*3) of the Hilbert axiom system, for the second set fourteen members of five target axiom systems, and for the third set the thirty-four deduced steps of the Lukasiewicz proof of the Hilbert system—OTTER completed a 26-step proof of the Hilbert axiom system, a 26-step proof of the alternate Lukasiewicz axiom system, a 27-step proof of the Wos axiom system, and a 28-step proof of the Church axiom system.

5.2.2. The Lukasiewicz-Lukasiewicz Resonators

The highlights of this subsection are displayed at its close.

To continue testing and evaluating the refined approach, I replaced the 33 Lukasiewicz-Church resonators (used in Round 3) with those corresponding to the Lukasiewicz proof of his alternate system, the system consisting of theses 19, 37, and 59. The Lukasiewicz proof has length 37. (For the curious, 5 of the deduced steps of the Lukasiewicz proof of the Church axiom system are not present in his proof of the alternate Lukasiewicz axiom system; hence, of the 37 resonators being used, 9 are not used in Round 3. Regarding the resonators corresponding to the 68 theses 4 through 71, 4 of the 37 resonators are not among them.) As dictated by the refined approach, weight_list(pick_and_purge) consisted of three lists: the resonators corresponding to the members of the alternate Lukasiewicz axiom system, each assigned a weight of 1; followed by the resonators corresponding to the members of the five target axiom systems, each assigned a weight of 2; followed by the 37 Lukasiewicz-Lukasiewicz resonators, each assigned a weight of 3. Max_weight was assigned the value 3 and the dynamic_heat_weight the value 2. Again, in case marvelous events occurred regarding short proofs, I included as possible targets for proof all five axiom systems.

In approximately 2 CPU-seconds, OTTER completed a 27-step proof of level 19 of the alternate Lukasiewicz axiom system. The proof is compact, being in fact a proof of thesis 59. Regarding the role of the hot list strategy, 9 of the 27 deduced steps show heat=1. Especially in the context of seeking shorter proofs, the result compares favorably with the Lukasiewicz proof, for his has length 37. A comparison of this 27-step proof and the 26-step proof (of the same axiom system) found when using the Lukasiewicz-Hilbert resonators leads to an example of the delight and piquant behavior sometimes encountered in automated reasoning, mathematics, and logic. In particular, the 27-step proof comes very close to containing the 26-step proof as a subproof, of course, with a renumbering of the clauses. The single step of the longer proof is used once in a later application of condensed detachment, and the result of that application is present in the shorter proof, but with a different pair of ancestors. Further, the two ancestors are present in the longer proof. Therefore, indeed the 26-step proof is almost a subproof of the 27-step proof. Of substantial interest and (to me) satisfaction is the fact that the use of the Lukasiewicz-Hilbert resonators produced a shorter proof than did the use of the Lukasiewicz-Lukasiewicz resonators, for it suggests how useful and powerful steps from one proof can be when used as resonators for attacking a related theorem.

Next, one finds in the output produced by the experiment a 28-step proof of the Wos axiom system, that consisting of theses 19, 37, and 60. The proof was completed in approximately 2 CPU-seconds, having a level of 20. The proof is compact, being in fact a proof of thesis 60. As an indication of the potential offered by the hot list strategy for sharply reducing the CPU time required to complete an assignment, one finds, among the 10 steps with heat=1, the last step of the proof, the deduction of the clause equivalent of thesis 60. Were it not for the hot list strategy causing the program to immediately choose for the focus of attention an ancestor of the last step—at least in the Argonne paradigm—the program would have been forced to wait until that ancestor had been chosen because of its weight (priority). Under different conditions, such an ancestor might not be chosen until much CPU

time had elapsed, or perhaps never chosen, thus substantially delaying or even preventing the program from completing the desired proof.

Then, in approximately 4 CPU-seconds, OTTER completed a proof of the Hilbert axiom system, a proof of length 28 (but announced length 29) and level 19. Of the deduced steps, 10 show heat=1. Finally, the program deduced the Church axiom system, with a proof of length 30 and level 21, 10 of whose steps show heat=1.

Highlights of Section 5.2.2

- With the layered refinement—using for the first set of resonators the three members of the alternate Lukasiewicz axiom system, for the second set fourteen members of five target axiom systems, and for the third set the thirty-seven deduced steps of the Lukasiewicz proof of the alternate Lukasiewicz system—OTTER completed a 27-step proof of the alternate Lukasiewicz axiom system, a 28-step proof of the Wos axiom system, a 28-step proof of the Hilbert axiom system, and a 30-step proof of the Church axiom system.

5.2.3. The Lukasiewicz-Frege Resonators

The highlights of this subsection are displayed at its close.

The next test for the refined approach was to apply it to an attack on finding a shorter proof for the Frege axiom system, which consists of theses 18, 21, 35, 39, 40, and 46, with thesis 21 dependent on the other five theses. Therefore, in addition to the (new) resonators with weight 1 (corresponding to the members of the Frege axiom system) and with weight 2 (corresponding to the 14 members of the five axiom systems), respectively, a new third set to replace that used in the preceding experiment was inserted, namely, a set of 41 resonators (each with a weight of 3) corresponding to the Lukasiewicz proof. In approximately 44 CPU-seconds, a deduction of the Frege axiom system was completed; its length is 37 (but announced as 39), and its level is 21. Of the deduced steps, 11 show heat=1. Last, in approximately 49 CPU-seconds, a 31-step proof of level 22 was completed of the Hilbert axiom system; 10 of the steps show heat=1.

Highlights of Section 5.2.3

- With the layered refinement—using for the first set of resonators the six members of the Frege axiom system, for the second set fourteen members of five target axiom systems, and for the third set the forty-one deduced steps of the Lukasiewicz proof of the Frege system—OTTER completed a 37-step proof of the Frege axiom system and a 31-step proof of the Hilbert axiom system.

5.2.4. The Lukasiewicz-Wos Resonators

There remained but one test, that focusing on the use of 39 resonators corresponding to the Lukasiewicz proof of the Wos axiom system. Of course, he did not know of that axiom system, as far as I know; indeed, as remarked earlier, it was not found until 1991. Because, typically, a proof of the Wos system is very closely related to a proof of the alternate Lukasiewicz axiom system, one might expect to get results similar to those reported earlier when the Lukasiewicz-Lukasiewicz resonators were used. In fact, the results were almost identical in that, except for a renumbering of the clauses, the same proofs were found by OTTER. Even the CPU time was essentially unchanged. Regarding the order in which the proofs were completed, the only change was that the Wos axiom system was proved before the alternate Lukasiewicz system was.

5.3. Testing and Evaluating a Variant of the Refinement

The approach formulated in this section is summarized at its close, and, with the exception of Section 5.3.5, highlights are displayed at the close of each subsection.

Studies not reported in this article suggested that a variant of the layered-resonator refinement merited experimentation. Resonance and heat still provided the basis, but, rather than the dynamic hot

list strategy, the hot list strategy was used. To allow clauses that do *not* match a resonator to play a (possibly) big role, the max_weight was assigned the value 20 rather than 3; if measured in symbol count, 20 is greater than or equal to the weight of each of theses 4 through 71. As in the refinement tested and evaluated in the preceding section, three sets of resonators were used. The first set consisted of the correspondents to members of the axiom system mainly in focus, for example, that of Church, each assigned a weight of 1. The second set consisted of the correspondents of the 14 members (other than *L*3) of the five axiom systems that have been studied throughout this article, each assigned a weight of 2. The third set consisted of the correspondents of the proof steps taken from a success reported earlier here, for example, the deduced steps of the 27-step proof of the Church axiom system (cited in Section 5.1), each with a weight of 3.

Not included as part of the refinement studied in the preceding section, McCune's ratio strategy [Wos92] was used. The following command suffices.

```
assign(pick_given_ratio,3).
```

With the command just cited, OTTER is instructed to choose (for the focus of attention) three clauses by weight, one by first come first serve (or breadth first), then three, then one, and the like. The ratio strategy blends (subject to the assignment of the pick_given_ratio parameter) a bit of level saturation with (usually more) choosing of clauses for the focus of attention by weight.

Finally, as an addition, McCune's ancestor subsumption [Wos92] is used, in turn necessitating (for practical reasons) the use of back subsumption. By comparing derivation proof length whenever a retained conclusion is deduced a second time, the use of ancestor subsumption enables a program to directly and explicitly attack the problem of seeking shorter proofs. With the use of ancestor subsumption, when a conclusion is drawn more than once, the program retains the copy reached by the shorter path. The following two commands are what is needed.

```
set(ancestor_subsume).
set(back_sub).
```

The general plan was to test and evaluate the variant (in which the dynamic hot list strategy is replaced by the hot list strategy and ancestor subsumption is used) by conducting five experiments, each experiment designed to extend the results of its (so-to-speak) earlier counterpart. Each of the five experiments would be what might be called self-referential in the following sense. For example, for the second experiment, the elements of the first of the three sets of resonators to be used correspond to the members of the Hilbert axiom system (excluding *L*3), and the elements of the third set correspond to the 26 proof steps of the proof of the Hilbert axiom system, the proof (cited in Section 5.2.1) obtained by using the Lukasiewicz-Hilbert resonators. I suggest ''self-referential'' as the classification precisely because the source for the third resonator set is a proof of the Hilbert axiom system found by featuring that system and, at the same time, the emphasis in the experiment is on that axiom system. I say ''emphasis'', noting that, although all five axiom systems are targets for finding shorter proofs, the most likely progress will occur with the axiom system being featured in the first and third resonator sets.

Regarding the ensemble of five experiments, one might wonder about the merit of replacing the set of resonators corresponding to a proof by another such set when the two sets have a large intersection. Indeed, the 27 resonators used in the first experiment in which the Church axiom set is featured share in common 21 resonators with the 26 resonators used in the second experiment in which the Hilbert axiom set is featured. So one might hazard that the second experiment might not merit conducting. A partial answer is contained in the observation that one simple but key lemma, if omitted from consideration, can block the discovery of a pleasing proof or the completion of a proof far shorter than one has in hand.

Highlights of Section 5.3

- Quite effective is a variant of the layered refinement; rather than using the dynamic hot list strategy, the variant combines the hot list strategy with the resonance strategy, where three sets of resonators are used, the last corresponding to deduced steps of a proof obtained with the layered refinement.

5.3.1. Experiment 1, Focusing on the Church Axiom System

The highlights are displayed at the close of this subsection.

In Experiment 1 for testing the variant, I used for the first set of resonators the correspondents to the members of the Church axiom system, which consists of theses 18, 35, and 49. For the third set, I used 27 resonators corresponding to the proof steps of the Church axiom system obtained by using the Lukasiewicz-Church resonators. As in all five experiments, I used for the second set of resonators correspondents of the members of the five target axiom systems, excluding the correspondent of $L3$ because of being in the hot list. As noted, (in all five experiments and consistent with the refinement featured earlier) members of the first set are each assigned a weight of 1, of the second set each a weight of 2, and of the third set each a weight of 3. The main goal of Experiment 1 was to find a proof of the Church axiom system in which strictly fewer than 27 applications of condensed detachment are required. Any other breakthroughs regarding ''short'' proofs would be a bonus and would also be evidence of the value of relying on resonators corresponding to the steps of one proof when attacking a related proof.

OTTER completed four proofs of the Church axiom system, of respective lengths 30, 29, 27, and—of some satisfaction—26. The latter has level 19 and was completed in approximately 2004 CPU-seconds on a SPARCstation-10 with retention of clause (49484). The proof is compact, being a proof of thesis 35 and containing as subproofs proofs of theses 18 and 49. Of the 26 deduced steps, 6 show heat=1 and 5 are *not* present in the 27-step proof used to influence the search.

Also in the run, the program completed six proofs of the Hilbert axiom system, of respective lengths 42, 39, 32, 31, 33, and 29, the latter having level 23. That proof was found in approximately 4847 CPU-seconds, completing with retention of clause (96572). Immediately one might wonder at the sequence of proofs getting shorter and shorter, then longer, then even shorter. After all, the use of ancestor subsumption is intended to promote the discovery of ever-shorter proofs. To begin with, a glance at the output file shows that six proofs are found for thesis 54 (a member of the Hilbert axiom system), the last three of respective lengths 36, 29, and 25. Any proof of the Hilbert axiom system must contain a proof of thesis 54 as well as a proof of the other members of the system. Ancestor subsumption has the object of finding shorter proofs of individual conclusions, not of the **and** of a set of conclusions, and the target axiom systems under attack have the form of an **and** of its members. The shorter of two proofs of, say, thesis 54 can have fewer deduced steps by avoiding the use of formulas that, unfortunately, are needed for proving one or more of the other members of the axiom system. In the limiting case, one can imagine finding proofs of the individual members such that, though each is pleasingly short, no deduced step is present in two or more proofs. The resulting proof of the **and** of the members might indeed be lengthened markedly when compared with a proof containing, as subproofs, not so short proofs of the individual members.

Similar remarks hold for seeking shorter proofs for (deductions of) a single thesis. However, in the presence of forward subsumption, one does not find the proof (deduction) of a unit clause followed later by a longer proof of that clause. The clause that would be needed for the second proof is forward subsumed by the earlier copy. The reason one does see longer proofs of the **and** of a set of unit clauses rests with the fact that forward subsumption is not applied to copies of the empty clause and the empty clause is the result of the corresponding application of hyperresolution, where the nucleus is the clause equivalent of the negation of the **and** of the members of the axiom system under study. (One would clearly wish to avoid subsumption being applied to copies of the empty clause, especially when one is after more than one proof in a single run.)

In regard to proof length, for theorems whose conclusion has the form the **and** of elements representable by unit clauses as well as theorems whose conclusion is representable with a single unit clause, the use of ancestor subsumption together with back subsumption can occasionally cause duplicate steps to be present in a proof. Therefore, the proof length given by the program can be greater than the actual proof length. A duplicate step occurs in a proof when (1) a retained clause A is used to deduce a clause B that is in the proof, and hence A as its ancestor is in the proof; (2) later a clause C is deduced that is a copy of A but obtained with a strictly shorter derivation path, and hence C is retained because of ancestor subsumption, and A is purged because of back subsumption; and (3) C is used as an ancestor of a clause D that is also in the proof. (When a clause is removed by back subsumption from

the database of clauses usable for drawing conclusions, it is still available for proof recovery.) In such an event the actual proof length is strictly less than the figure cited by the program. Indeed, no need exists for the clause *C*, for *A* serves nicely in place of it. Further, one may be able to produce a yet shorter proof by purging, if such exists, the clauses that are on the path to *C* if they are not needed elsewhere in the proof. One thus has a taste of how complicated the seeking of shorter proofs is; no practical algorithm exists, as far as I know. Far more on the subject of seeking shorter proofs and on the obstacles that can be encountered is found in Chapter 3 of [Wos95a].

Of more interest, one finds in the run a proof of the alternate Lukasiewicz axiom system, a proof of length 26 and level 19, completed in approximately 1793 CPU-seconds with retention of clause (47689). The interest comes in part from the fact that use of the Lukasiewicz-Lukasiewicz resonators produced a 27-step proof, and it comes in part from the fact that all deduced formulas in the 26-step proof are among those of the 27-step proof. But for a small amount of reshuffling and a slight difference in ancestry, the shorter proof would be a subproof of the longer. Essentially the same relation holds for the 27-step proof of the Wos axiom system found in the same run when compared with the 28-step proof found using the Lukasiewicz-Wos resonators. The shorter proof has level 20, required approximately 1827 CPU-seconds to find, and was completed with retention of clause (47959).

Finally, in the run, one finds eight proofs of the Frege axiom system, ranging in length from 40 to 31 (the last). The 31-step proof, which has level 19, was found in approximately 9797 CPU-seconds, completing with retention of clause (149314). Seven of its deduced steps show heat=1 and, more interesting to me, 7 of its deduced steps are not among the 37-step proof found by using the Lukasiewicz-Frege resonators. The latter figure illustrates how resonators (in this case, 27 from a proof of the Church axiom system) can enable a program to (in effect) outline a major part of a sought-after proof.

Taken as a whole, the preceding results suggest that the variant is promising. Therefore, I turned to the second experiment.

Highlights of Section 5.3.1

- With the variant of the layered refinement—using for the first set of resonators the three members of the Church axiom system, for the second set fourteen members of five target axiom systems, and for the third set the twenty-seven deduced steps of the layered-refinement proof of the Church system—OTTER completed a 26-step proof of the Church axiom system, a 29-step proof of the Hilbert axiom system, a 26-step proof of the alternate Lukasiewicz axiom system, a 27-step proof of the Wos axiom system, and a 31-step proof of the Frege axiom system.

5.3.2. Experiment 2, Focusing on the Hilbert Axiom System

The highlights are displayed at the close of this subsection.

As already noted, Experiment 2 featured the Hilbert axiom system, with the first set of resonators the correspondents of its members (excluding *L*3) and the third set the correspondents of the 26 deduced steps of the proof obtained with the Lukasiewicz-Hilbert resonators. OTTER completed two proofs of the Hilbert axiom system, of respective lengths 26 and 27, both of level 18. The first was completed in approximately 16 CPU-seconds, with retention of clause (3922). The proof is identical to that found with the Lukasiewicz-Hilbert resonators.

In approximately 68 CPU-seconds, with retention of clause (10329), the program found a 29-step proof of the Church axiom system, one of level 20. Of the deduced steps, 7 show heat=1 and 9 are not among those of the 27-step proof of the Church axiom system obtained with the Lukasiewicz-Church resonators.

In approximately 68 CPU-seconds OTTER found a 35-step proof of the Frege axiom system, but, a bit more satisfying, in approximately 14,464 CPU-seconds, the program then found a 34-step proof. That proof has level 20 and was completed with retention of clause (177222). Only in the past few years, as a result of McCune's advances in the design of automated reasoning programs, has it been more than practical to pursue proofs relying on the retention of clauses numbering in the hundreds of

thousands. The cited 34-step proof offers a piquant property: It contains as a subproof the 29-step proof of the Church axiom system. Also of some satisfaction is the shortening in length to 34 when compared with the length of 37 for the proof obtained with the Lukasiewicz-Frege resonators reported in Section 5.2.3. Of the 34 deduced steps, 11 are not among those of the 37-step proof.

Regarding the alternate Lukasiewicz axiom system, two proofs were found of respective lengths 26 and 25. The second was found in approximately 10 CPU-seconds with retention of clause (2903); it has level 18. In view of the fact that the original Lukasiewicz proof has length 37 and the fact that the proof obtained with the Lukasiewicz-Lukasiewicz resonators (reported in Section 5.2.2) has length 27, the shortening of proof length provides more evidence of the value of the variant under testing. The 25-step proof contains but one formula, the following, not present in the 27-step proof obtained with the Lukasiewicz-Lukasiewicz resonators.

P(i(i(i(n(x),y),z),i(i(n(y),x),z))).

As for the Wos axiom system, approximately 7 CPU-seconds sufficed to find a 27-step proof of level 20, completing with retention of clause (2124). Two items are of some interest. First, all formulas of the 27-step proof are among those of the 28-step proof found using the Lukasiewicz-Wos resonators (of Section 5.2.4) and in the same order. In fact, the shorter proof comes very close to being a subproof of the longer. Second, for reasons unknown to me—although, ordinarily, a proof of the Wos axiom system is obtained by adjoining one more condensed detachment step to the corresponding proof of the alternate Lukasiewicz axiom system, a step to deduce thesis 60—such did not occur in the run under discussion.

Highlights of Section 5.3.2

- With the variant of the layered refinement—using for the first set of resonators five members (excluding *L*3) of the Hilbert axiom system, for the second set fourteen members of five target axiom systems, and for the third set the twenty-six deduced steps of the layered-refinement proof of the Hilbert system—OTTER completed a 26-step proof of the Hilbert axiom system, a 29-step proof of the Church axiom system, a 25-step proof of the alternate Lukasiewicz axiom system, a 27-step proof of the Wos axiom system, and a 34-step proof of the Frege axiom system.

5.3.3. Experiment 3, Focusing on the Frege Axiom System

The highlights are displayed at the close of this subsection.

Next in order is Experiment 3, an experiment that proved somewhat rewarding. Featured in the experiment is the Frege axiom system. Indeed, the first set of resonators (each assigned a weight of 1) consisted of the correspondents of its members, namely, theses 18, 21, 35, 39, 40, and 46, of which thesis 21 is dependent on the other five. The third set consisted of the 37 correspondents of the deduced steps of the proof of the Frege axiom system obtained with the Lukasiewicz-Frege resonators. Perhaps of interest, of the 27 resonators in the third set in Experiment 1, all but 2 are among the 37 resonators of the third set of this (Frege) experiment.

OTTER found eight proofs of the Frege axiom system, of respective reported lengths of 36, 39, 35, 36, 33, 37, 37, and 35. The first was completed in approximately 13 CPU-seconds with retention of clause (3365), and the last was completed in approximately 21,497 CPU-seconds with retention of clause (189431). I used the term ‘‘reported proof length’’, for, as commented earlier, the use of ancestor subsumption coupled with back subsumption can lead to the presence of duplicate steps in a proof. Such is the case in the last of the eight proofs; in particular, two copies of the clause equivalent of thesis 39 are present. As demonstrated in Section 5.3.4 when cursory proof checking takes center stage, occasionally proofs of the duplicate-step type serve well in the context of seeking shorter proofs.

In the same run, the program completed three proofs of the Church axiom system, of respective lengths 29, 27, and 25 and of respective levels 21, 20, and 18. The time required for the proofs ranged from approximately 11 CPU-seconds to 21,497 CPU-seconds. The last proof completed with retention of clause (189430). As yet another delight of mathematics, the 25-step proof is a subproof of the cited 34-step proof of the Frege axiom system. (For the curious, the 25-step proof contains three formulas

not present in the 33-step proof of the Frege system, and the 33-step proof contains five formula not in the 34-step proof of the Frege system.) The deduction of a 25-step proof was satisfying, for it was the shortest obtained in this study so far.

As for the Hilbert axiom system, seven proofs were found, of respective lengths (in order) 37, 35, 32, 31, 33, 32, and 29. For whatever it signifies, the last has level 18, and all the others have level 22 or 23. The required time in CPU-seconds ranges from approximately 65 to 8311. Where the first proof completed with retention of clause (8942), the last completed with retention of clause (119991).

Regarding the alternate Lukasiewicz axiom system, in approximately 3331 CPU-seconds, OTTER found a length 26 and level 19 proof, completed with retention of clause (79082); no other proof of that system was found. All 26 deduced formulas are among the 27 of the proof of the system using the Lukasiewicz-Lukasiewicz resonators; in fact, the former is close to being a subproof of the latter.

Contrary to expectation, the single proof of the Wos axiom system that was found does not follow the pattern of being the proof of the alternate Lukasiewicz system (found earlier in the run) with one additional formula adjoined at the end, namely, the clause form of thesis 60. Indeed, the proof has announced length 29 and actual length 28, containing two copies of the clause correspondent of thesis 22, which, incidentally, is not a member of the axiom system. That level 20 proof was found in approximately 3385 CPU-seconds, completing with retention of clause (79344). It contains one formula not present in the 28-step proof obtained with the Lukasiewicz-Wos resonators.

Highlights of Section 5.3.3

- With the variant of the layered refinement—using for the first set of resonators the six members of the Frege axiom system, for the second set fourteen members of five target axiom systems, and for the third set the thirty-seven deduced steps of the layered-refinement proof of the Frege system—OTTER completed a 33-step proof of the Frege axiom system, a 25-step proof of the Church axiom system, a 29-step proof of the Hilbert axiom system, a 26-step proof of the alternate Lukasiewicz axiom system, and a 28-step proof of the Wos axiom system.

5.3.4. Experiment 4, Focusing on the Alternate Lukasiewicz Axiom System

The highlights are displayed at the close of this subsection.

Experiment 4, which featured the alternate Lukasiewicz axiom system, produced an unexpected and most satisfying result, as will be seen. Perhaps the success can be traced in part to the fact that, of the 27 resonators of which the third set consisted in the experiment, 8 are not present in the 27 used in Experiment 1. The members of this third set correspond to the 27 deduced steps of the proof of the alternate Lukasiewicz axiom system obtained by using the Lukasiewicz-Lukasiewicz resonators. In Experiment 4, the first set of resonators correspond to the members of the axiom system, namely, theses 19, 37, and 59.

Four proofs of respective lengths 26, 34, 35, and 34 of the axiom system being featured were found, the first of level 19. Approximately 6 CPU-seconds were required as well as the retention of clause (1921). Except for the presence of one formula of the 27-step proof of the alternate Lukasiewicz axiom system (obtained by using the Lukasiewicz-Lukasiewicz resonators), the order of formulas is the same in both the 26-step and the 27-step proofs of that axiom system. Then, as is so typical, in approximately 6 CPU-seconds the program found a 27-step proof (of level 20) of the Wos axiom system, a proof whose first 26 deduced steps are those of the just-cited 26-step proof of the alternate Lukasiewicz axiom system. As for the Hilbert axiom system, the shortest that was found in Experiment 4 has announced length 37 and level 18, completing in approximately 4904 CPU-seconds with retention of clause (102102). Closer inspection shows that 5 of the 37 steps are duplicates, suggesting (as discussed in Section 6) that a perhaps far shorter proof can be easily obtained. The shortest proof of the Frege axiom system found in the run has length 35 and level 20, requiring approximately 70 CPU-seconds, and completing with retention of clause (10678).

The best I have saved for last. Five proofs of the Church axiom system were found, of respective announced lengths 29, 36, 36, 31, and 30. The first proof completed in approximately 70 CPU seconds

with retention of clause (10687), and the last (of level 19) completed in approximately 4201 CPU-seconds with retention of clause (98477). Although on the surface the cited results are far from striking, hidden within them is a treasure, in particular, within the last two proofs. Each of the two proofs contains three duplicate steps. Therefore, their respective actual lengths are 28 and 27. The promised ''most satisfying result'', rather than resting with the cited actual lengths, rested with a possible implication. Specifically, perhaps one or more of the duplicate steps would be found to be the end of a chain of formulas (used for its derivation) such that no member of the chain was needed elsewhere in the proof. In other words, perhaps hidden in either or both of the last two cited proofs (found in the run featuring as target the alternate Lukasiewicz axiom system) one might find a ''short'' proof of the Church axiom system. I was further encouraged in that regard when I found in the run, among the proofs of thesis 35, two of respective lengths 24 and 23. Indeed, many of the proofs found by OTTER of the Church axiom system are precisely a proof of thesis 35, in other words, are compact.

Therefore, before turning to Experiment 5—featuring the Wos axiom system and expected to produce little more of interest because of the (frequently) close relation of its proof to a proof of the alternate Lukasiewicz axiom system—two runs, each in the spirit of cursory proof checking, were in order. The first of the two calls for the use of 23 resonators, the correspondents of the proof steps of the 23-step proof of thesis 35. The second of the two focuses on the 24-step proof. In both, I assigned the value 3 to each resonator and the value 3 to max_weight. The idea was, in a cursory manner, to proof check each of the two proofs by preventing the program from retaining any clause that failed to match one of the resonators in use. Of course, one could merely read through the proofs to see whether theses 18 and 49 were present to decide whether the proofs were compact and were, in fact, proofs of the entire Church axiom system. However, such is ordinarily not my preference; indeed, I find it of greater interest to see what results from an appropriate OTTER run. Sometimes a prize is won; sometimes insight is gained; sometimes an idea for a new strategy is born.

Two additional experiments, also somewhat in the spirit of proof checking, merited conducting, experiments respectively focusing on the 27-step and 28-step proof of the Church axiom system, each found in the run for Experiment 4. The object would be to settle the question concerning what would occur when and if the proofs were purged of duplicate steps and, possibly, of additional steps used only in the derivation of a duplicate step. Perhaps either or both experiments would unearth a hidden proof, hidden within the longer one in which duplicate steps are present.

Because of their position in the output file—the 24-step proof of thesis 35, the announced 31-step proof of the Church axiom system, the 23-step proof of 35, and the announced 30-step proof of the Church axiom system—I thought it quite likely that the four contemplated experiments would pair off in the following (probably obvious) manner. It seemed likely that use of either the 30 resonators (including 3 duplicates) or the 23 resonators in a cursory proof-checking run would produce the same proof of the Church axiom system. The implication is that, in either case, OTTER would produce a 23-step proof. With similar reasoning, use of either the 31 resonators (including 3 duplicates) or the 24 resonators would lead to the same proof of the Church axiom system, a proof of length 24. Especially when compared with the Lukasiewicz proof of the Church axiom system, which has length 33, the completion of a 23-step proof would mark an important advance and would offer strong evidence of the value of the variant under study.

In less than 1 CPU-second, use of the 23 resonators produced the hoped-for 23-step proof of the Church axiom system, a proof of level 19, completing with retention of clause (57). As for the use of the 30 resonators (including 3 duplicates), the only change concerns the completion on retention of clause (61). Regarding a comparison of the clauses that correspond to the 23 resonators with the 23 deduced steps of the proof, only the order was changed, implying that the proof that was found differed at least somewhat from the 23-step proof of thesis 35. The result also showed that the 27-step proof extractable from the announced 30-step proof does indeed contain unneeded formulas, 4 of them, used in the derivation of the 3 duplicate steps.

Even more exciting were the results of the experiments respectively relying on the 24 resonators and on the 31 resonators (including 3 duplicates). With the 24 resonators, in less than 1 CPU-second with retention of clause (56), OTTER found a level 18 and length 22 proof of the Church axiom system. With the 31 resonators (including 3 duplicates), the result was the same except for relying on the retention of clause (61). In addition to avoiding the use of 2 of the 24 steps in the 24-step proof of

thesis 35 (the following), the order was indeed changed.

P(i(x,i(i(x,y),y))).
P(i(i(i(n(i(x,y)),x),z),z)).

The result also showed that the 28-step proof extractable from the announced 31-step proof does indeed contain unneeded formulas, 6 of them, used in the derivation of the 3 duplicate steps. Probably more important, the success illustrates the usefulness of the type of cursory proof checking under discussion.

Highlights of Section 5.3.4

- With the variant of the layered refinement—using for the first set of resonators the three members of the alternate Lukasiewicz axiom system, for the second set fourteen members of five target axiom systems, and for the third set the twenty-seven deduced steps of the layered-refinement proof of the alternate Lukasiewicz system—OTTER completed a 26-step proof of the alternate Lukasiewicz axiom system, a 27-step proof of the Wos axiom system, a 32-step proof of the Hilbert axiom system, a 35-step proof of the Frege axiom system, and a 27-step proof of the Church axiom system,
- Also completed were a 24-step and a 23-step proof of thesis 35, significant because, often, a proof of that thesis is a proof of the entire Church axiom system.
- In a cursory proof-checking mode, use of the 24-step proof of thesis 35 quickly led to a 22-step proof of the Church axiom system.

5.3.5. Experiment 5, Focusing on the Wos Axiom System

Although little was expected in Experiment 5 (featuring the 28 resonators corresponding to the deduced steps of the proof of the Wos axiom system obtained with the Lukasiewicz-Wos resonators) because of the close relation to Experiment 4 (relying on 27 resonators), it required conducting. The experiment produced the expected results, the same proofs as those produced with Experiment 4 and with little change in CPU time or in clause numbering. The presence of 9 rather than 8 resonators not among those of Experiment 1 was of no consequence.

5.4. Proof Checking with OTTER

This section provides a taste of two types of proof checking that are possible with OTTER, cursory proof checking and rigorous proof checking. (The subject is treated in far greater detail in Chapter 10 of [Wos95a].) Briefly, the difference between the two types of proof checking rests with the precision that is required. When rigorous proof checking is the assignment, success requires that a proof be produced that is identical, including ancestor information, to that being proof checked. Since the proof being checked may be incomplete, may indeed be missing steps that are implicit, the program is required to complete the proof by supplying all steps explicitly. In such an event, obviously one cannot claim with certainty that the program has produced the precise proof that is, for example, supplied by the author in a paper being refereed. I have thought for years that, eventually, an automated reasoning program will be quite useful in materially aiding one in refereeing papers, especially in checking proofs that are often difficult to validate if left to one's own device.

When cursory (rather than rigorous) proof checking is the assignment, success requires that a proof be produced that resembles that being proof checked; in particular, far less precision is required. Indeed, success may be marked by the completion of a proof that is more elegant than that serving as the target, more elegant, for example, by being shorter or by avoiding the use of some class of terms such as those of the form $n(n(t))$ for any term t, where n denotes negation.

Regarding the use of OTTER, in rigorous proof checking (as illustrated shortly) the program is given (where possible) a detailed proof including the ancestors and inference rule used to obtain each deduced step. The program is *prevented* from straying from the detailed proof. When asked to cursory proof check, (as illustrated in the preceding sections and reviewed immediately) the program is given guidelines (in the form of weight templates) to direct its search for a proof, with the implicit permission to stray a bit from the guidelines.

With no guarantee that OTTER will return the proof being checked, to cursory proof check a proof, one places weight templates that correspond to its deduced steps in weight_list(pick_and_purge), each assigned a small weight, say, 3. The idea is to give preference (over other clauses should any be retained) to any clause that resembles (with its variables ignored) one of the deduced steps of the proof. OTTER treats the variables in a weight template as indistinguishable, which turns out to be an advantage in research, as discussed and illustrated in this article. Indeed, the cited treatment enables the researcher to easily use the resonance strategy. To virtually guarantee that the program retain a clause only if it resembles a proof step, one assigns to the max_weight the value assigned to the included weight templates. Despite the use of a tight max_weight, the treatment of variables in weight templates does permit the program to retain clauses other than those in the proof being checked, which is the main reason that cursory proof checking does not guarantee that a proof completed by the program will be identical to that in hand. In addition, other factors may cause the program to draw conclusions in an order different from that of the proof under study.

Section 5.3.4 offers a fine example of how the drawing of conclusions in an order different from that of the proof under study can produce a proof other than that in hand and, even better, produce a more elegant proof. Indeed, when cursory proof checking was applied to a 24-step proof of thesis 35, OTTER produced a 22-step proof all of whose steps are among the 24, but in a different order. As a bonus (and as noted), the proof is of greater interest, for it is a 22-step proof of the entire Church axiom system. The result was most satisfying, especially when compared with the Lukasiewicz proof of length 33.

In contrast to finding a more elegant proof, because the program may consider conclusions in an order different from that of the proof under study, occasionally a less satisfying proof is produced, one with greater length, for example. Even worse, the program may fail to complete *any* proof, a phenomenon that did puzzle me for a while until the following occurred to me.

Let *A* be a step of the proof that is being cursorily proof checked, and assume that *A* is deduced and retained. Next assume that *A* is chosen as the focus of attention, and assume that its use leads to the retention of two clauses, *B* and *E*, such that *B* is a later step in the proof and *E* is not in the proof. Almost certainly, the clause *E* matches a resonator, but, of course, it is not identical to any. Then assume that the choice of *B* as the focus of attention leads to the deduction of *C*, an even later step in the proof, but assume that *E* subsumes *C*, preventing its retention. Where *C* would be used to draw conclusions, the program can use *E*. But if the nature of *E* is such that its use leads to a clause that subsumes the needed clause that the use of *C* would have yielded, and if the subsuming clause does not match any of the resonators, then the small max_weight can cause the program to immediately discard the subsuming clause. The result can be that *no proof* is completable within the conditions under which the program is running.

Now, if the situation just described occurred where the hot list strategy was not used, then its use might perturb the order in which clauses are deduced enough to avoid what I have just hypothesized. If the described situation occurred when the hot list strategy was used, but the dynamic hot list strategy was not, then adding the use of the dynamic hot list strategy might be just what is needed to obtain proofs. On the other hand, use of the dynamic hot list strategy might be the problem, and, therefore, removing its use might be enough to enable the program to find the sought-after proof. I have not verified my hypothetical example by instructing OTTER to set(very_verbose), which places in the output file copious detail, permitting one to carefully analyze what actually occurred.

In sharp contrast to the just-cited hypothetical example that illustrates how cursory proof checking can fail to produce *any* proof, much less that being checked, rigorous proof checking does not encounter this situation. Indeed, when OTTER is instructed to rigorously proof check by using demodulation coupled with weighting in a manner to be illustrated, no conclusions are retained other than those in the proof under consideration. Because of the basic search algorithm used by OTTER and because of the effect weighting has on the choice of clause on which to focus attention, their order will usually differ from that of the proof in hand. The differences are not important in the sense that a reshuffling and renumbering will produce the desired order of deduced steps. In other words, despite the different order of retained conclusions, the program is still (in effect) proof checking the given proof. To retain the conclusions of interest and, at the same time, reject all others, the idea conceptually is to test each conclusion to see if its ancestry is acceptable. If the ancestry passes the test, then,

through demodulation, one of the desired conclusion numbers (corresponding to a step of the proof being checked) is planted in the clause. If the test fails, then demodulation does not plant a desired conclusion number, and through weighting the clause is purged. The following commands provide the promised illustration, giving one a taste of how it works.

To provide the basis for the rigorous proof checking, in the following manner, the clauses that capture the hypotheses of the proof are assigned numbers that correspond to the order of the hypotheses.

```
list(sos).
P(c1,i(i(x,y),i(i(y,z),i(x,z)))).
P(c2,i(i(n(x),x),x)).
P(c3,i(x,i(n(x),y))).
end_of_list.
```

Next, to enable the program to accept conclusion identification numbers and correctly plant the corresponding ancestry history, the clause for condensed detachment (or that which is needed for the type of reasoning in use) is modified in the following manner.

```
% The following modified clause is used with hyperresolution for condensed detachment.
-P(u,i(x,y)) | -P(v,x) | P(rew(u,v),y).
```

Because all retained conclusions will have as an argument an identification number, the denial clauses (for example, that for the Church axiom system) must be modified.

```
-P(x,i(q,i(p,q))) | -P(y,i(i(p,i(q,r)),i(i(p,q),i(p,r)))) |
-P(z,i(i(n(p),n(q)),i(q,p))) | $ANSWER(step_allBEH_Church_FL_18_35_49).
```

As expected, such clauses are placed in list(usable).

With the preceding as the basis for the rigorous proof checking, next in order is an illustration of the type of demodulator that takes the ancestor numbers and plants, if appropriate, the correct conclusion number.

```
list(demodulators).
% Following essentially mirrors the Lukasiewicz proof with history.
EQ(rew(c1,c1),c4).
EQ(rew(c4,c4),c5).
EQ(rew(c4,c1),c6).
EQ(rew(c5,c6),c7).
EQ(rew(c7,c1),c8).
EQ(rew(c1,c3),c9).
end_of_list.
```

As is probably obvious, the preceding is just the initial segment of the Lukasiewicz derivation of theses 4 through 71. Not obvious was the necessity to add condensed detachment steps to cope with his practice of combining steps, of renaming variables, and the like. To (in effect) complement the actions of demodulation in the context of rigorous proof checking, one includes the following for weighting.

```
weight_list(pick_and_purge).
weight(P(rew($(0),$(0)),$(1)),99).
end_of_list.
```

One then includes an assignment of max_weight, say the value 80, that is strictly less than that appearing in the template. (One immediately sees a sharp difference between rigorous and cursory proof checking. In the former, a weight template and a max_weight are included to purge unwanted clauses; in the latter, weight templates and max_weight are included to enable the program to retain desired clauses.) The following close look at the given demodulators and their interaction with the given weight template is merited.

OTTER begins by applying condensed detachment (through the use of the cited, modified three-literal clause) to two copies of the first of the three Lukasiewicz axioms, that denoted by *c*1. Before an attempt to apply demodulators, the following clause (including the ancestor history) is deduced.

P(rew(c1,c1),i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))).

The cited clause is immediately demodulated by using the first of the cited demodulators to produce the following.

P(c4,i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))).

As the notation correctly suggests, the program has deduced the clause equivalent of thesis 4 as its second argument; its first argument is the corresponding identifier. Had the clause remained in the cited undemodulated form—which would occur if the needed demodulator were omitted from the input—then it would be assigned a weight greater than 99. In that event, it would be immediately discarded because of having a weight strictly greater than the assigned max_weight.

For a second example of clause discarding, when the program applies condensed detachment to the third and second Lukasiewicz axioms (denoted, respectively, by *c*3 and *c*2), the application succeeds. However, the corresponding formula is not present in the proof that Lukasiewicz gives of theses 4 through 71. Therefore, no demodulator is included to rewrite the subexpression rew(c3,c2). The clause is then given a weight greater than 99, and it is immediately discarded because of having a weight that exceeds the assigned max_weight. Of course, rather than the few demodulators that are listed, one includes a full set, one for each deduced step of the proof being rigorously checked. Also, as observed, one might be forced to include additional demodulators to cope with combined steps.

If one wishes a research topic relevant to the material of this section, one might attempt to devise a means to cause the program to produce a proof in which the order of its steps precisely matches that of the proof in hand. Of a totally different nature, one might focus on research designed to extend the cited rigorous proof checking to the case in which paramodulation and demodulation occur in the proof to be checked.

6. Best Results

Whether an advance is actually significant is conjectural often until years have passed. Still, when one has sought a result with repeated experiments, much CPU time, and perhaps the most powerful automated reasoning program—of course, I refer to OTTER—and when that result has eluded one, then obtaining it (at least subjectively) is significant. This section features such results, almost all of which rely on the resonance strategy combined with the hot list strategy; see the Appendix for the corresponding proofs. Although many of the cited results were obtained with earlier research, they are included to provide additional evidence of the value of combining resonance and heat and also to set the stage for the challenges issued in Section 7. When compared with the layered refinement or with its variant, that earlier research has the disadvantages of requiring far more from the researcher, being substantially less algorithmic, and being dramatically more complex and lengthy both in real time and in CPU time. Despite the disadvantages, in some cases, intriguing proofs were completed.

To obtain the first result of interest, I began my attack with the use of resonators corresponding to the 22-step proof of the Church axiom system cited in Section 5.3.4. Reminiscent of both the layered refinement and its variant, I assigned a weight of 3 to each of the 22 resonators. My goal was that of finding a shorter proof of that system. A direct attack was made by relying on the use of ancestor subsumption and, of course for practical considerations, therefore also on back subsumption. To give OTTER more latitude, max_weight was assigned the value 32. However, rather than maintaining this assignment—for fear of drowning the program—the max_weight was reduced to 20 after 70 clauses had been chosen as the focus of attention to drive the program's reasoning. For this purpose, the following two commands were included.

assign(change_limit_after,70).
assign(new_max_weight,20).

I chose 20 as the new max_weight, for that is the weight in symbol count of the longest thesis among theses 4 through 71. I assigned the pick_given_ratio the value 2 with the notion that the use of long clauses found early in the output might result in a breakthrough. I also assigned max_distinct_vars the value 5. Finally, borrowing from the promising refinement introduced in Section 5.1 and from the variant introduced in Section 5.3, I included before the set of twenty-two resonators two sets of resonators: the first correspondents of the three members of the Church axiom system, each assigned a weight of 1,

and the second correspondents to fourteen members of known axiom systems, each assigned a weight of 2. For reasons I cannot supply, I chose not to rely on the hot list strategy.

Success occurred, but not mainly in the context of the cited goal regarding the Church axiom system. Indeed, although a 22-step proof of the Church axiom system was found that I had never seen before—it differs by one formula—OTTER completed a 29-step proof of the Frege axiom system, a proof I had never seen before and one that is shorter than any reported earlier in this article for that system. The difference in the two 22-step Church proofs rests with the replacement of the second of the following two formulas by the first.

P(i(n(i(i(n(x),x),x)),y)).
P(i(x,i(i(i(n(x),y),z),i(i(u,y),z)))).

The new proof of the Frege axiom system caused me to turn to a follow-up experiment.

The follow-up experiment was in the spirit of cursory proof checking, as discussed in Section 5.4. The max_weight was assigned the value 3, and 29 resonators corresponding to the steps of the just-cited proof of the Frege axiom system were used, each assigned a weight of 3. For historical reasons, I shall call that proof Proof 2. Consistent with cursory proof checking (in the strictest sense), little else was used from among OTTER's numerous options. On the one hand, based on my rather intense study of this axiom system approximately eighteen months ago, I thought it most likely that the program would merely return the same 29-step proof. That study had also produced a 29-step proof (which I shall call Proof 1), but a proof that differs in ten of its deduced steps. Substantially more effort and CPU time (those months ago) had strongly indicated that a shorter proof would be difficult to discover, if indeed such a proof existed. On the other hand, perhaps the newer 29-step proof (Proof 2) would open the way to finding a shorter proof; after all, in general, the current studies (reported here) were readily yielding new proofs.

To my delight and astonishment, OTTER and cursory proof checking prevailed. In approximately 2 CPU seconds (on a SPARCstation-2) with retention of clause (82), the program completed a level 17 and length 28 proof of the Frege axiom system, a proof given in the Appendix especially for the researcher who wishes to attempt to improve on the result. Regarding compactness of the proof, the longest subproof that is a proof of one of the members of the axiom system has length 23. All 28 deduced steps are among the 29 steps of Proof 2, but in a different order. Contained within the 28-step proof is a 23-step proof of the Church axiom system, a proof that is compact in that it is also a proof of thesis 35; the 23-step proof is in fact a subproof of the 28-step proof. Because adding the use of the hot list strategy sometimes (as seen shortly) yields additional results of interest, I added the strategy with the heat parameter assigned the value 1. Essentially the same 28-step proof was found, showing that the combination of resonance and heat would have sufficed; the order of the deduced steps was perturbed quite a bit.

The following four successes provide an excellent example regarding the role of the hot list strategy (superimposed on the use of the resonance strategy) in yielding additional results of interest. By a sharply different research path, far more lengthy and arduous than that which is the basis of this article, OTTER completed a 22-step proof of the Church axiom system. (The proof is called Proof 4 in [Wos95a].) Proof 4 relies on eight formulas not present in the 22-step proof cited in Section 5.3.4. Cursory proof checking applied to Proof 4 (whose steps were used as resonators) merely returns Proof 4 which, by the way, is compact, being a proof of thesis 35. However, when the hot list strategy is used (with the clause for condensed detachment and those for the three Lukasiewicz axioms in the hot list), the so-called modified cursory proof checking does not return Proof 4. Instead, the program completes a 21-step proof, given in the Appendix as an example of one of the best proofs of which I know. The proof is compact, being a proof of thesis 35. All deduced steps are present in Proof 4, but their order is quite different. Summarizing, the hot list strategy proved to be most useful, its use yielding a proof of length 21; I currently (on March 31, 1995) know of no strictly shorter proof. As the following shows, the cited occurrence is by no means unique, suggesting that use of the resonance strategy combined with the use of the hot list strategy, even in the context of cursory proof checking, can be most fruitful.

When the just-discussed approach was applied to a 24-step proof (whose steps were used as resonators) of the Hilbert axiom system (found through the pursuit of a lengthy and different research path than reported here), with the hot list strategy used in a cursory proof-checking mode a 23-step proof

was completed, and without it the 24-step proof was returned. The 23-step proof is given in the Appendix. All 23 deduced formulas are among the 24 of the 24-step proof, but their order is sharply changed. Within the 23-step proof, the longest subproof of a member of the Hilbert axiom system has length 21.

When the approach was applied to a 25-step proof of the alternate Lukasiewicz axiom system (found with the aforementioned lengthy research path), with the hot list strategy a 24-step proof was found, and without it the same 25-step proof was returned. The 24-step proof is given in the Appendix. All 24 deduced formulas are among the 25 of the 25-step proof, but their order is sharply changed. Within the 24-step proof, the longest subproof of a member of the alternate Lukasiewicz axiom system has length 23.

When the approach was applied to a 26-step proof of the Wos axiom system (found in the lengthy study), with the hot list strategy a 25-step proof was completed, and without it the 26-step proof was returned. The 25-step proof is given in the Appendix. All 25 deduced formulas are among the 26 of the 26-step proof, but their order is sharply changed. Within the 25-step proof, the longest subproof of a member of the Wos axiom system has length 24.

A glance at the various proofs in the Appendix shows how and where the hot list strategy played a role. As noted, the designation heat=n asserts that the hot list was consulted at the *n*-th level to yield the corresponding clause. For those interested in history, I was indeed surprised when cursory proof checking combined with the hot list strategy was applied to the 22-step proof of the Church axiom system and the result was the discovery of a 21-step proof. For many, many months I had sought a proof of that length without success. That success led me immediately to the experiments that culminated in the reported discoveries of the cited proofs for the other four axiom systems. In contrast and most piquant, applying this modified cursory proof checking often leads to nothing more than a rearrangement of the proof steps whose correspondents are used as resonators. Also worth noting is the fact that inclusion of resonators that correspond to the members of the axiom system being featured, each assigned for its weight the value 1, can produce a compact proof where one was lacking, but often a proof that is not shorter and, sometimes, is even longer. The use of additional resonators, the use of the hot list strategy, and other option choices can each perturb the search space materially, which in turn can produce unforeseen results.

If one experiences suspicion at the realization that the approach discussed in this section, when successful, always led to a reduction of one in the length of the completed proof, I can only say that I sympathize. Currently, I have no explanation.

7. Challenges and Possible Research

Regarding challenges, for openers, I suggest seeking shorter proofs than those I found for the five axiom systems. The proofs are presented in the Appendix. For the Church axiom system, the shortest I have found to this date (April 1 1995) has length 21 (applications of condensed detachment, the required inference rule in this context), length 28 for Frege, length 23 for Hilbert, length 24 for (the alternate of) Lukasiewicz, and length 25 for Wos. In some cases, uniqueness is not present in the sense that other proofs exist of the same length for various axiom systems. For example, I know of four proofs of length 21 for the Church axiom system. Were the suggestion a starred exercise, I would give a hint: Perhaps allowing six or more distinct variables in retained conclusions might be fruitful.

For a related challenge, one might seek ''short'' *compact* proofs for each of the five axiom systems. As a reminder, for example, a proof of the Hilbert axiom system (consisting of theses 3, 18, 21, 22, 30, and 54, of which thesis 30 is dependent on the other five) is compact if and only if it is a proof of one of the six members, of course, with no extra steps. Among the few compact proofs of which I know, I have seen four of length 22 for the Church axiom system, and four of length 21 also for that system [Wos95a]. Intuitively, combining proof brevity and proof compactness increases the challenge, for the less room within the proof of a single member of an axiom system, the more difficult it is to squeeze in the proofs of the other members.

For a third challenge, one could (so to speak) turn the tables on Lukasiewicz, using one of the five axiom systems for the hypothesis. For example, one could use as the (starting) axioms the Church axiom system consisting of theses 18, 35, and 49. Then, as a replacement for the Church system as a target, one would have the Lukasiewicz axiom system (consisting of $L1$, $L2$, and $L3$), of course, with

the other four target systems, from Frege, Hilbert, (the alternate of) Lukasiewicz, and Wos. The goal would be that of finding short proofs, compact proofs, and short and compact proofs. One could rely on the approach taken in this article, or one might formulate an entirely different approach.

Of a general nature, for research, I recommend as tough topics the seeking of metarules for deciding when to use which strategy and when to use which combination of strategies. Some light might be shed on the topic by analyzing why one choice of options promotes finding shorter proofs, where another results in finding no proofs at all. Of yet a different character, one might seek other areas for which the resonance strategy, the hot list strategy, the dynamic hot list strategy, and different combinations of these strategies are most effective.

8. Conclusions and Observations

In this article, I have studied the use of strategies in various combinations. I have shown that, when strategies are used in combination—often, but not always—the result is far more effective than used singly. The evidence was gathered from experiments with McCune's automated reasoning program OTTER. Featured are the *resonance strategy*, the *hot list strategy*, and the *dynamic hot list strategy*. Also used are the *ratio strategy* and *weighting*. McCune formulated the ratio strategy and the dynamic hot list strategy (as an extension of the hot list strategy); Overbeek formulated weighting.

Although the context of my experiments focuses on finding shorter proofs and on combinations of strategies, a closer inspection of the material presented here strongly suggests that, for totally unrelated studies, individually or in combination the strategies used throughout offer impressive power for overcoming obstacles that naturally arise. For one example, without using an appropriate strategy, a person or a program can easily get trapped in a valley of conclusions of similar or identical complexity. One way to overcome this obstacle is to use the resonance strategy, which asks the researcher to conjecture which equations or formulas are such that their respective patterns (ignoring variables) merit marked preference for initiating applications of the inference rules in use. For a second example, sometimes the desired proof requires crossing a high plateau, consisting of three or more consecutive and complex conclusions. The resonance strategy can be of assistance here also, and so too can the hot list strategy, the latter asking the researcher to choose from among the statements that characterize the problem those that merit repeated immediate visiting and possibly immediate revisiting to *complete* applications of inference rules rather than to *initiate* applications. Then one sometimes is certain that a level saturation approach is best, but is concerned about forcing the program to search rapidly growing levels of conclusions. If level saturation is combined with the hot list strategy with, say, the heat parameter assigned the value 3, then the program will deduce *some* conclusions at level j+3 when it is actually exploring level j. In other words, this combination enables the program to (so to speak) ''look ahead''. As a final example, should the researcher prefer an automated reasoning program to adjoin *during* the run clauses to the hot list **if** they are proved and **if** they also match a symbol pattern (ignoring variables) that appears appealing, a combination of the resonance strategy with the dynamic hot list strategy may provide precisely what is needed.

My experiments did elicit satisfaction and even surprise. Among the unexpected occurrences, when I applied a version of cursory proof checking to a given proof, rather than successfully proof checking the proof, sometimes OTTER found a more elegant proof. On the other hand, sometimes the proof checking simply failed totally, yielding no proofs, not even of any of the members of the axiom system whose proof was being checked. Of such mysteries automated reasoning is made. But, of such mysteries also the mind of a good mathematician or a good logician is made. Whether mysteries of the cited types will be solved is left to the future. However, access to McCune's powerful automated reasoning program OTTER provides invaluable assistance in bringing that future closer while, in the present, serving as a powerful assistant for research.

Appendix

So often, my research has been fruitful directly because of access to specific proofs. Independent of that aspect, I occasionally am presented with a proof that I had thought *could not exist*. Prompted by

such factors and the intention that such charming ‘‘short’’ proofs (as those cited in this article) not be lost, I include here the most satisfying (to me) proofs of the five axiom systems for two-valued sentential (or propositional) calculus: those of Church, Frege, Hilbert, (an alternate of) Lukasiewicz, and Wos. The proofs may serve well to stimulate research, for, among other reasons, they provide needed targets.

As is no doubt obvious, for each proof, the inference rule that is used is condensed detachment (from the viewpoint of logic) and hyperresolution (from the viewpoint of automated reasoning), and the hypotheses are the three Lukasiewicz axioms, *L*1, *L*2, and *L*3. In each proof (expressed in clause notation acceptable to OTTER), ‘‘-’’ denotes logical **not**, ‘‘ | ’’ denotes logical **or**, the function *i* can be interpreted as *implication*, the function *n* as *negation*, and the predicate *P* as *provable*. When the input contains the command set(order_history), [hyper,28,2712,30] says that clause (28) is the nucleus, clause (2712) is unified with the first literal of (28), and clause (30) is unified with the second literal. Note that, when the dynamic hot list strategy is used, the listed proof length (by OTTER) may be strictly greater than the actual proof length because the use of this strategy can lead to the presence of duplicate steps. Also note that, in the following proofs, the presence of two copies in the input of any clause says that the hot list strategy played a role; when two copies appear, one is in the hot list, and the other is in list(usable) or in list(sos). The clause number gives its relative position in the database of retained clauses. I also include here a sample input file to facilitate the research of others. When a line contains a ‘‘%’’, the characters from the first ‘‘%’’ to the end of the line are treated by the program as a comment.

Sample Input File

```
set(hyper_res).
assign(max_weight,3).  %  Disallow clauses with weight strictly greater than 3.
assign(max_proofs,-1).  %  Place no limit on the number of proofs to be completed.
clear(print_kept).  %  Do not enter into the output file clauses as they are retained.
clear(back_sub).  %  Prevent purging of existing clauses when a new clause captures such.
%  assign(max_seconds,1200).  %  Limit the CPU time.
assign(max_mem,60000).  %  Limit the memory usage in kilobytes.
assign(report,900).  %  Place in output file a statistical summary every 15 CPU-minutes.
assign(max_distinct_vars,5).  %  Do not keep clauses with more than 5 distinct variables.
assign(pick_given_ratio,5).  %  Five clauses are chosen by weight and one by breadth first, repeatedly.
assign(heat,1).  %  Limits the recursion through the hot list.
assign(dynamic_heat_weight,2).  %  Limits weight on clauses adjoined to hot list during run.
set(order_history).  %  List ancestors in the order nucleus,
%  clause unifying with first literal, clause unifying with second.
set(input_sos_first).  %  Choose for driving the reasoning clauses from the input list(sos) before others.
%  set(sos_queue).  %  Direct the search by first come, first serve.

weight_list(pick_and_purge).
%  Following are the members of the Church system.
weight(P(i(x,i(y,x))),1).
weight(P(i(i(x,i(y,z)),i(i(x,y),i(x,z)))),1).
weight(P(i(i(n(x),n(y)),i(y,x))),1).
%  Following are 14 members of known axiom systems, excluding L1 L2 L3.
weight(P(i(x,i(y,x))),2).
weight(P(i(i(i(x,y),z),i(y,z))),2).
weight(P(i(i(x,i(y,z)),i(y,i(x,z)))),2).
weight(P(i(i(x,y),i(i(z,x),i(z,y)))),2).
weight(P(i(i(x,i(x,y)),i(x,y))),2).
weight(P(i(i(x,i(y,z)),i(i(x,y),i(x,z)))),2).
weight(P(i(i(i(x,y),z),i(n(x),z))),2).
weight(P(i(n(n(x)),x)),2).
```

```
weight(P(i(x,n(n(x)))),2).
weight(P(i(i(x,y),i(n(y),n(x)))),2).
weight(P(i(i(n(x),n(y)),i(y,x))),2).
weight(P(i(i(x,y),i(i(n(x),y),y))),2).
weight(P(i(i(n(x),y),i(i(z,y),i(i(x,z),y)))),2).
weight(P(i(i(x,i(n(y),z)),i(x,i(i(u,z),i(i(y,u),z))))),2).
%  Following are correspondents to the Lukasiewicz 33-step proof of the Church system.
weight(P(i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))),3).
weight(P(i(i(i(n(x),y),z),i(x,z))),3).
weight(P(i(i(x,i(y,z)),i(i(u,y),i(x,i(u,z))))),3).
weight(P(i(i(x,y),i(i(i(x,z),u),i(i(y,z),u)))),3).
weight(P(i(i(x,i(i(y,z),u)),i(i(y,v),i(x,i(i(v,z),u))))),3).
weight(P(i(x,i(i(i(n(x),y),z),i(i(u,y),z)))),3).
weight(P(i(i(i(n(i(i(n(x),x),x)),y),z),i(i(u,y),z))),3).
weight(P(i(i(x,i(i(n(y),y),y)),i(i(n(y),y),y))),3).
weight(P(i(x,i(i(n(y),y),y))),3).
weight(P(i(i(n(x),y),i(z,i(i(y,x),x)))),3).
weight(P(i(i(i(x,i(i(y,z),z)),u),i(i(n(z),y),u))),3).
weight(P(i(i(n(x),y),i(i(y,x),x))),3).
weight(P(i(i(x,i(y,z)),i(i(n(z),y),i(x,z)))),3).
weight(P(i(x,i(i(y,x),x))),3).
weight(P(i(i(x,i(y,z)),i(z,i(x,z)))),3).
weight(P(i(x,i(y,x))),3).
weight(P(i(i(i(x,y),z),i(y,z))),3).
weight(P(i(x,i(i(x,y),y))),3).
weight(P(i(i(x,i(y,z)),i(y,i(x,z)))),3).
weight(P(i(i(i(x,i(y,z)),u),i(i(y,i(x,z)),u))),3).
weight(P(i(i(i(x,y),z),i(i(x,u),i(i(u,y),z)))),3).
weight(P(i(i(x,y),i(i(z,x),i(z,y)))),3).
weight(P(i(i(n(x),n(y)),i(y,x))),3).
weight(P(i(i(x,i(n(y),z)),i(i(i(x,z),y),y))),3).
weight(P(i(i(i(x,y),x),x)),3).
weight(P(i(i(i(x,y),z),i(i(z,x),x))),3).
weight(P(i(i(i(i(x,y),y),z),i(i(i(y,u),x),z))),3).
weight(P(i(i(i(x,y),z),i(i(x,z),z))),3).
weight(P(i(i(x,y),i(i(i(x,z),u),i(i(y,u),u)))),3).
weight(P(i(i(i(x,y),z),i(i(x,u),i(i(u,z),z)))),3).
weight(P(i(i(x,y),i(i(y,i(z,i(x,u))),i(z,i(x,u))))),3).
weight(P(i(i(x,i(y,i(z,u))),i(i(z,x),i(y,i(z,u))))),3).
weight(P(i(i(x,i(y,z)),i(i(x,y),i(x,z)))),3).
%  Following are templates corresponding to the 68 theses 4-71 to be proved.
%  weight(P(i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))),2).
%  weight(P(i(i(x,i(y,z)),i(i(u,y),i(x,i(u,z))))),2).
%  weight(P(i(i(x,y),i(i(i(x,z),u),i(i(y,z),u)))),2).
%  weight(P(i(i(x,i(i(y,z),u)),i(i(y,v),i(x,i(i(v,z),u))))),2).
%  weight(P(i(i(x,y),i(i(z,x),i(i(y,u),i(z,u))))),2).
%  weight(P(i(i(i(n(x),y),z),i(x,z))),2).
%  weight(P(i(x,i(i(i(n(x),x),x),i(i(y,x),x)))),2).
%  weight(P(i(i(x,i(i(n(y),y),y)),i(i(n(y),y),y))),2).
%  weight(P(i(x,i(i(n(y),y),y))),2).
%  weight(P(i(i(n(x),y),i(z,i(i(y,x),x)))),2).
%  weight(P(i(i(i(x,i(i(y,z),z)),u),i(i(n(z),y),u))),2).
%  weight(P(i(i(n(x),y),i(i(y,x),x))),2).
%  weight(P(i(x,x)),2).
%  weight(P(i(x,i(i(y,x),x))),2).
```

```
%  weight(P(i(x,i(y,x))),2).
%  weight(P(i(i(i(x,y),z),i(y,z))),2).
%  weight(P(i(x,i(i(x,y),y))),2).
%  weight(P(i(i(x,i(y,z)),i(y,i(x,z)))),2).
%  weight(P(i(i(x,y),i(i(z,x),i(z,y)))),2).
%  weight(P(i(i(i(i(x,i(y,z)),u),i(i(y,i(x,z)),u))),2).
%  weight(P(i(i(i(x,y),x),x)),2).
%  weight(P(i(i(i(x,y),z),i(i(x,u),i(i(u,y),z)))),2).
%  weight(P(i(i(i(x,y),z),i(i(z,x),x))),2).
%  weight(P(i(i(i(x,y),y),i(i(y,x),x))),2).
%  weight(P(i(i(i(i(x,y),y),z),i(i(i(y,u),x),z))),2).
%  weight(P(i(i(i(x,y),z),i(i(x,z),z))),2).
%  weight(P(i(i(x,i(x,y)),i(x,y))),2).
%  weight(P(i(i(x,y),i(i(i(x,z),u),i(i(y,u),u)))),2).
%  weight(P(i(i(i(x,y),z),i(i(x,u),i(i(u,z),z)))),2).
%  weight(P(i(i(x,y),i(i(y,i(z,i(x,u))),i(z,i(x,u))))),2).
%  weight(P(i(i(x,i(y,i(z,u))),i(i(z,x),i(y,i(z,u))))),2).
%  weight(P(i(i(x,i(y,z)),i(i(x,y),i(x,z)))),2).
%  weight(P(i(n(x),i(x,y))),2).
%  weight(P(i(i(i(x,y),z),i(n(x),z))),2).
%  weight(P(i(i(x,n(x)),n(x))),2).
%  weight(P(i(n(n(x)),x)),2).
%  weight(P(i(x,n(n(x)))),2).
%  weight(P(i(i(x,y),i(n(n(x)),y))),2).
%  weight(P(i(i(i(n(n(x)),y),z),i(i(x,y),z))),2).
%  weight(P(i(i(x,y),i(i(y,n(x)),n(x)))),2).
%  weight(P(i(i(x,i(y,n(z))),i(i(z,y),i(x,n(z))))),2).
%  weight(P(i(i(x,i(y,z)),i(i(n(z),y),i(x,z)))),2).
%  weight(P(i(i(x,y),i(n(y),n(x)))),2).
%  weight(P(i(i(x,n(y)),i(y,n(x)))),2).
%  weight(P(i(i(n(x),y),i(n(y),x))),2).
%  weight(P(i(i(n(x),n(y)),i(y,x))),2).
%  weight(P(i(i(i(n(x),y),z),i(i(n(y),x),z))),2).
%  weight(P(i(i(x,i(y,z)),i(x,i(n(z),n(y))))),2).
%  weight(P(i(i(x,i(y,n(z))),i(x,i(z,n(y))))),2).
%  weight(P(i(i(n(x),y),i(i(x,y),y))),2).
%  weight(P(i(i(x,y),i(i(n(x),y),y))),2).
%  weight(P(i(i(x,y),i(i(x,n(y)),n(x)))),2).
%  weight(P(i(i(i(i(x,y),y),z),i(i(n(x),y),z))),2).
%  weight(P(i(i(n(x),y),i(i(x,z),i(i(z,y),y)))),2).
%  weight(P(i(i(i(i(x,y),i(i(y,z),z)),u),i(i(n(x),z),u))),2).
%  weight(P(i(i(n(x),y),i(i(z,y),i(i(x,z),y)))),2).
%  weight(P(i(i(x,i(n(y),z)),i(x,i(i(u,z),i(i(y,u),z))))),2).
%  weight(P(i(i(x,y),i(i(z,y),i(i(n(x),z),y)))),2).
%  weight(P(i(i(n(n(x)),y),i(x,y))),2).
%  weight(P(i(x,i(y,y))),2).
%  weight(P(i(n(i(x,x)),y)),2).
%  weight(P(i(i(n(x),n(i(y,y))),x)),2).
%  weight(P(i(n(i(x,y)),x)),2).
%  weight(P(i(n(i(x,y)),n(y))),2).
%  weight(P(i(n(i(x,n(y))),y)),2).
%  weight(P(i(x,i(n(y),n(i(x,y))))),2).
%  weight(P(i(x,i(y,n(i(x,n(y)))))),2).
%  weight(P(n(i(i(x,x),n(i(y,y))))),2).
end_of_list.
```

```
%  Used to complete applications of inference rules.
list(usable).
%  The following clause is used with hyperresolution for condensed detachment.
-P(i(x,y)) | -P(x) | P(y).
%  The following disjunctions, except those mentioning Scott,
%  are the negation of known axiom systems.

-P(i(p,i(q,p))) | -P(i(i(p,i(q,r)),i(i(p,q),i(p,r)))) | -P(i(n(n(p)),p)) |
-P(i(p,n(n(p)))) | -P(i(i(p,q),i(n(q),n(p)))) | -P(i(i(p,i(q,r)),i(q,i(p,r)))) |
$ANSWER(step_allFrege_18_35_39_40_46_21).  % 21 is dependent.

-P(i(p,i(q,p))) | -P(i(i(p,i(q,r)),i(q,i(p,r)))) |
-P(i(i(q,r),i(i(p,q),i(p,r)))) | -P(i(p,i(n(p),q))) |
-P(i(i(p,q),i(i(n(p),q),q))) | -P(i(i(p,i(p,q)),i(p,q))) |
$ANSWER(step_allHilbert_18_21_22_3_54_30).  % 30 is dependent.

-P(i(p,i(q,p))) | -P(i(i(p,i(q,r)),i(i(p,q),i(p,r)))) |
-P(i(i(n(p),n(q)),i(q,p))) | $ANSWER(step_allBEH_Church_FL_18_35_49).

-P(i(i(i(p,q),r),i(q,r))) | -P(i(i(i(p,q),r),i(n(p),r))) |
-P(i(i(n(p),r),i(i(q,r),i(i(p,q),r)))) | $ANSWER(step_allLuka_x_19_37_59).

-P(i(i(i(p,q),r),i(q,r))) | -P(i(i(i(p,q),r),i(n(p),r))) |
-P(i(i(s,i(n(p),r)),i(s,i(i(q,r),i(i(p,q),r))))) | $ANSWER(step_allWos_x_19_37_60).

-P(i(i(p,q),i(i(q,r),i(p,r)))) | -P(i(i(n(p),p),p)) |
-P(i(p,i(n(p),q))) | $ANSWER(step_allLuka_1_2_3).

%  -P(i(p,p)) | -P(i(p,i(q,p))) | -P(i(i(p,i(q,r)),i(q,i(p,r)))) |
%  -P(i(i(i(p,q),p),p)) | -P(i(i(p,i(q,r)),i(i(p,q),i(p,r)))) |
%  -P(i(n(n(p)),p)) | -P(i(p,n(n(p)))) | -P(i(i(p,q),i(n(q),n(p)))) |
%  $ANSWER(step_allScott_orig_16_18_21_24_35_39_40_46).

%  -P(i(p,p)) | -P(i(p,i(q,p))) | -P(i(i(p,i(q,r)),i(q,i(p,r)))) |
%  -P(i(i(i(p,q),p),p)) | -P(i(i(p,i(q,r)),i(i(p,q),i(p,r)))) |
%  -P(i(n(n(p)),p)) | -P(i(p,n(n(p)))) | -P(i(i(i(n(p),n(q)),i(q,p))) |
%  $ANSWER(step_allScott_orig0_16_18_21_24_35_39_40_49).
end_of_list.

%  Used to initiate applications of inference rules.
list(sos).
%  The following three are Luka, 1 2 3.
P(i(i(x,y),i(i(y,z),i(x,z)))).
P(i(i(n(x),x),x)).
P(i(x,i(n(x),y))).
end_of_list.

%  Used mainly to detect proof completion and to monitor progress.
list(passive).
%  -P(i(i(i(i(q,r),i(p,r)),s),i(i(p,q),s))) | $ANS(neg_th_04).
%  -P(i(i(p,i(q,r)),i(i(s,q),i(p,i(s,r))))) | $ANS(neg_th_05).
%  -P(i(i(p,q),i(i(i(p,r),s),i(i(q,r),s)))) | $ANS(neg_th_06).
%  -P(i(i(t,i(i(p,r),s)),i(i(p,q),i(t,i(i(q,r),s))))) | $ANS(neg_th_07).
%  -P(i(i(q,r),i(i(p,q),i(i(r,s),i(p,s))))) | $ANS(neg_th_08).
```

```
%  -P(i(i(i(n(p),q),r),i(p,r))) | $ANS(neg_th_09).
%  -P(i(p,i(i(i(n(p),p),p),i(i(q,p),p)))) | $ANS(neg_th_10).
%  -P(i(i(q,i(i(n(p),p),p)),i(i(n(p),p),p))) | $ANS(neg_th_11).
%  -P(i(t,i(i(n(p),p),p))) | $ANS(neg_th_12).
%  -P(i(i(i(n(p),q),i(t,i(i(q,p),p)))) | $ANS(neg_th_13).
%  -P(i(i(i(t,i(i(q,p),p)),r),i(i(n(p),q),r))) | $ANS(neg_th_14).
%  -P(i(i(n(p),q),i(i(q,p),p))) | $ANS(neg_th_15).
%  -P(i(p,p)) | $ANS(neg_th_16).
%  -P(i(p,i(i(q,p),p))) | $ANS(neg_th_17).
-P(i(q,i(p,q))) | $ANS(neg_th_18).
-P(i(i(i(p,q),r),i(q,r))) | $ANS(neg_th_19).
%  -P(i(p,i(i(p,q),q))) | $ANS(neg_th_20).
-P(i(i(p,i(q,r)),i(q,i(p,r)))) | $ANS(neg_th_21).
-P(i(i(q,r),i(i(p,q),i(p,r)))) | $ANS(neg_th_22).
%  -P(i(i(i(q,i(p,r)),s),i(i(p,i(q,r)),s))) | $ANS(neg_th_23).
%  -P(i(i(i(p,q),p),p)) | $ANS(neg_th_24).
%  -P(i(i(i(p,r),s),i(i(p,q),i(i(q,r),s)))) | $ANS(neg_th_25).
%  -P(i(i(i(p,q),r),i(i(r,p),p))) | $ANS(neg_th_26).
%  -P(i(i(i(p,q),q),i(i(q,p),p))) | $ANS(neg_th_27).
%  -P(i(i(i(i(r,p),p),s),i(i(i(p,q),r),s))) | $ANS(neg_th_28).
%  -P(i(i(i(p,q),r),i(i(p,r),r))) | $ANS(neg_th_29).
-P(i(i(p,i(p,q)),i(p,q))) | $ANS(neg_th_30).
%  -P(i(i(p,s),i(i(i(p,q),r),i(i(s,r),r)))) | $ANS(neg_th_31).
%  -P(i(i(i(p,q),r),i(i(p,s),i(i(s,r),r)))) | $ANS(neg_th_32).
%  -P(i(i(p,s),i(i(s,i(q,i(p,r))),i(q,i(p,r))))) | $ANS(neg_th_33).
%  -P(i(i(s,i(q,i(p,r))),i(i(p,s),i(q,i(p,r))))) | $ANS(neg_th_34).
-P(i(i(p,i(q,r)),i(i(p,q),i(p,r)))) | $ANS(neg_th_35).
%  -P(i(n(p),i(p,q))) | $ANS(neg_th_36).
-P(i(i(i(p,q),r),i(n(p),r))) | $ANS(neg_th_37).
%  -P(i(i(p,n(p)),n(p))) | $ANS(neg_th_38).
 -P(i(n(n(p)),p)) | $ANS(neg_th_39).
 -P(i(p,n(n(p)))) | $ANS(neg_th_40).
%  -P(i(i(p,q),i(n(n(p)),q))) | $ANS(neg_th_41).
%  -P(i(i(i(n(n(p)),q),r),i(i(p,q),r))) | $ANS(neg_th_42).
%  -P(i(i(p,q),i(i(q,n(p)),n(p)))) | $ANS(neg_th_43).
%  -P(i(i(s,i(q,n(p))),i(i(p,q),i(s,n(p))))) | $ANS(neg_th_44).
%  -P(i(i(s,i(q,p)),i(i(n(p),q),i(s,p)))) | $ANS(neg_th_45).
-P(i(i(p,q),i(n(q),n(p)))) | $ANS(neg_th_46).
%  -P(i(i(p,n(q)),i(q,n(p)))) | $ANS(neg_th_47).
%  -P(i(i(n(p),q),i(n(q),p))) | $ANS(neg_th_48).
-P(i(i(n(p),n(q)),i(q,p))) | $ANS(neg_th_49).
%  -P(i(i(i(n(q),p),r),i(i(n(p),q),r))) | $ANS(neg_th_50).
%  -P(i(i(p,i(q,r)),i(p,i(n(r),n(q))))) | $ANS(neg_th_51).
%  -P(i(i(p,i(q,n(r))),i(p,i(r,n(q))))) | $ANS(neg_th_52).
%  -P(i(i(n(p),q),i(i(p,q),q))) | $ANS(neg_th_53).
-P(i(i(p,q),i(i(n(p),q),q))) | $ANS(neg_th_54).
%  -P(i(i(p,q),i(i(p,n(q)),n(p)))) | $ANS(neg_th_55).
%  -P(i(i(i(i(p,q),q),r),i(i(n(p),q),r))) | $ANS(neg_th_56).
%  -P(i(i(n(p),r),i(i(p,q),i(i(q,r),r)))) | $ANS(neg_th_57).
%  -P(i(i(i(i(p,q),i(i(q,r),r)),s),i(i(n(p),r),s))) | $ANS(neg_th_58).
-P(i(i(n(p),r),i(i(q,r),i(i(p,q),r)))) | $ANS(neg_th_59).
-P(i(i(s,i(n(p),r)),i(s,i(i(q,r),i(i(p,q),r))))) | $ANS(neg_th_60).
%  -P(i(i(p,r),i(i(q,r),i(i(n(p),q),r)))) | $ANS(neg_th_61).
%  -P(i(i(n(n(p)),q),i(p,q))) | $ANS(neg_th_62).
%  -P(i(q,i(p,p))) | $ANS(neg_th_63).
```

```
%  -P(i(n(i(p,p)),q)) | $ANS(neg_th_64).
%  -P(i(i(n(q),n(i(p,p))),q)) | $ANS(neg_th_65).
%  -P(i(n(i(p,q)),p)) | $ANS(neg_th_66).
%  -P(i(n(i(p,q)),n(q))) | $ANS(neg_th_67).
%  -P(i(n(i(p,n(q))),q)) | $ANS(neg_th_68).
%  -P(i(p,i(n(q),n(i(p,q))))) | $ANS(neg_th_69).
%  -P(i(p,i(q,n(i(p,n(q)))))) | $ANS(neg_th_70).
%  -P(n(i(i(p,p),n(i(q,q))))) | $ANS(neg_th_71).
end_of_list.

%  Following list can be used to purge unwanted formulas of various types.
%  list(demodulators).
%  (n(n(x)) = junk).
%  (n(n(n(x))) = junk).
%  (i(i(x,x),y) = junk).
%  (i(y,i(x,x)) = junk).
%  (i(n(i(x,x)),y) = junk).
%  (i(y,n(i(x,x))) = junk).
%  (i(junk,x) = junk).
%  (i(x,junk) = junk).
%  (n(junk) = junk).
%  (P(junk) = $T).
%  end_of_list.

list(hot).
%  The following clause is used with hyperresolution for condensed detachment.
-P(i(x,y)) | -P(x) | P(y).
%  The following three are Luka, 1 2 3.
P(i(i(x,y),i(i(y,z),i(x,z)))).
P(i(i(n(x),x),x)).
P(i(x,i(n(x),y))).
end_of_list.
```

A 21-Step Proof of the Church Axiom System

```
----- Otter 3.0.2b+, Aug 1994 -----
The job was started by wos on altair.mcs.anl.gov, Fri Mar 31 15:04:46 1995
The command was "otter302c".

-----> EMPTY CLAUSE at   1.65 sec ----> 77 [hyper,4,49,71,46]
$ANSWER(step_allBEH_Church_FL_18_35_49).

Length of proof is 21.  Level of proof is 15.

---------- PROOF ----------

1 [] -P(i(x,y)) | -P(x) | P(y).
4 [] -P(i(p,i(q,p))) | -P(i(i(p,i(q,r)),i(i(p,q),i(p,r)))) | -P(i(i(n(p),n(q)),i(q,p)))
| $ANSWER(step_allBEH_Church_FL_18_35_49).
8 [] P(i(i(x,y),i(i(y,z),i(x,z)))).
9 [] P(i(i(n(x),x),x)).
10 [] P(i(x,i(n(x),y))).
25 [] -P(i(x,y)) | -P(x) | P(y).
26 [] P(i(i(x,y),i(i(y,z),i(x,z)))).
```

27 [] P(i(i(n(x),x),x)).
28 [] P(i(x,i(n(x),y))).
---
29 [hyper,1,8,8] P(i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))).
33 [hyper,1,8,10] P(i(i(i(n(x),y),z),i(x,z))).
34 [hyper,1,10,9] P(i(n(i(i(n(x),x),x)),y)).
35 (heat=1) [hyper,25,26,34] P(i(i(x,y),i(n(i(i(n(z),z),z)),y))).
36 [hyper,1,29,29] P(i(i(x,i(y,z)),i(i(u,y),i(x,i(u,z))))).
37 (heat=1) [hyper,25,36,27] P(i(i(x,y),i(i(n(i(y,z)),i(y,z)),i(x,z)))).
38 [hyper,1,33,35] P(i(x,i(n(i(i(n(y),y),y)),z))).
39 [hyper,1,36,37] P(i(i(x,i(n(i(y,z)),i(y,z))),i(i(u,y),i(x,i(u,z))))).
40 [hyper,1,39,38] P(i(i(x,i(n(y),y)),i(z,i(x,y)))).
42 [hyper,1,29,40] P(i(i(n(x),y),i(z,i(i(y,x),x)))).
45 [hyper,1,39,42] P(i(i(x,i(y,z)),i(i(n(z),y),i(x,z)))).
46 (heat=1) [hyper,25,45,28] P(i(i(n(x),n(y)),i(y,x))).
48 [hyper,1,36,45] P(i(i(x,i(n(y),z)),i(i(u,i(z,y)),i(x,i(u,y))))).
49 [hyper,1,33,46] P(i(x,i(y,x))).
51 (heat=1) [hyper,25,26,49] P(i(i(i(x,y),z),i(y,z))).
53 [hyper,1,29,48] P(i(i(n(x),y),i(i(z,i(u,x)),i(i(y,u),i(z,x))))).
55 [hyper,1,51,46] P(i(n(x),i(x,y))).
60 [hyper,1,45,55] P(i(i(n(x),y),i(n(y),x))).
65 [hyper,1,60,55] P(i(n(i(x,y)),x)).
69 [hyper,1,53,65] P(i(i(x,i(y,i(z,u))),i(i(z,y),i(x,i(z,u))))).
71 (heat=1) [hyper,25,69,26] P(i(i(x,i(y,z)),i(i(x,y),i(x,z)))).
77 [hyper,4,49,71,46] $ANSWER(step_allBEH_Church_FL_18_35_49).

A 28-Step Proof of the Frege Axiom System

----- Otter 3.0.2b+, Aug 1994 -----
The job was started by wos on altair.mcs.anl.gov, Wed Mar 22 11:12:34 1995
The command was "otter302c".

-----> EMPTY CLAUSE at 1.96 sec ----> 82 [hyper,2,44,72,60,67,77,46]
$ANSWER(step_allFrege_18_35_39_40_46_21).

Length of proof is 28. Level of proof is 17.

---------- PROOF ----------

1 [] -P(i(x,y)) | -P(x) | P(y).
2 [] -P(i(p,i(q,p))) | -P(i(i(p,i(q,r)),i(i(p,q),i(p,r)))) | -P(i(n(n(p)),p)) |
 -P(i(p,n(n(p)))) | -P(i(i(p,q),i(n(q),n(p)))) | -P(i(i(p,i(q,r)),i(q,i(p,r)))) |
$ANSWER(step_allFrege_18_35_39_40_46_21).
8 [] P(i(i(x,y),i(i(y,z),i(x,z)))).
9 [] P(i(i(n(x),x),x)).
10 [] P(i(x,i(n(x),y))).
---
25 [hyper,1,8,8] P(i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))).
27 [hyper,1,8,10] P(i(i(i(n(x),y),z),i(x,z))).
28 [hyper,1,10,9] P(i(n(i(i(n(x),x),x)),y)).
29 [hyper,1,25,25] P(i(i(x,i(y,z)),i(i(u,y),i(x,i(u,z))))).
30 [hyper,1,25,8] P(i(i(x,y),i(i(i(x,z),u),i(i(y,z),u)))).
31 [hyper,1,29,30] P(i(i(x,i(i(y,z),u)),i(i(y,v),i(x,i(i(v,z),u))))).
32 [hyper,1,30,28] P(i(i(i(n(i(i(n(x),x),x)),y),z),i(i(u,y),z))).
33 [hyper,1,32,9] P(i(i(x,i(i(n(y),y),y)),i(i(n(y),y),y))).

34 [hyper,1,27,33] P(i(x,i(i(n(y),y),y))).
35 [hyper,1,31,34] P(i(i(n(x),y),i(z,i(i(y,x),x)))).
36 [hyper,1,8,35] P(i(i(i(x,i(i(y,z),z)),u),i(i(n(z),y),u))).
37 [hyper,1,36,9] P(i(i(n(x),y),i(i(y,x),x))).
38 [hyper,1,29,37] P(i(i(x,i(y,z)),i(i(n(z),y),i(x,z)))).
39 [hyper,1,31,38] P(i(i(n(x),y),i(i(z,i(u,x)),i(i(y,u),i(z,x))))).
40 [hyper,1,29,38] P(i(i(x,i(n(y),z)),i(i(u,i(z,y)),i(x,i(u,y))))).
41 [hyper,1,38,10] P(i(i(n(x),n(y)),i(y,x))).
44 [hyper,1,27,41] P(i(x,i(y,x))).
46 [hyper,1,40,44] P(i(i(x,i(y,z)),i(y,i(x,z)))).
51 [hyper,1,46,10] P(i(n(x),i(x,y))).
53 [hyper,1,38,51] P(i(i(n(x),y),i(n(y),x))).
54 [hyper,1,8,51] P(i(i(i(x,y),z),i(n(x),z))).
58 [hyper,1,53,51] P(i(n(i(x,y)),x)).
60 [hyper,1,54,9] P(i(n(n(x)),x)).
65 [hyper,1,39,58] P(i(i(x,i(y,i(z,u))),i(i(z,y),i(x,i(z,u))))).
67 [hyper,1,41,60] P(i(x,n(n(x)))).
69 [hyper,1,39,60] P(i(i(x,i(y,n(z))),i(i(z,y),i(x,n(z))))).
72 [hyper,1,65,8] P(i(i(x,i(y,z)),i(i(x,y),i(x,z)))).
77 [hyper,1,69,51] P(i(i(x,y),i(n(y),n(x)))).
82 [hyper,2,44,72,60,67,77,46] $ANSWER(step_allFrege_18_35_39_40_46_21).

A 23-Step Proof of the Hilbert Axiom System

----- Otter 3.0.2b+, Aug 1994 -----
The job was started by wos on altair.mcs.anl.gov, Fri Mar 31 17:12:56 1995
The command was "otter302c".

-----> EMPTY CLAUSE at 2.43 sec ----> 87 [hyper,3,50,53,56,10,81,83]
$ANSWER(step_allHilbert_18_21_22_3_54_30).

Length of proof is 23. Level of proof is 15.

---------- PROOF ----------

1 [] -P(i(x,y)) | -P(x) | P(y).
3 [] -P(i(p,i(q,p))) | -P(i(i(p,i(q,r)),i(q,i(p,r)))) | -P(i(i(q,r),i(i(p,q),i(p,r)))) |
-P(i(p,i(n(p),q))) | -P(i(i(p,q),i(i(n(p),q),q))) | -P(i(i(p,i(p,q)),i(p,q))) |
$ANSWER(step_allHilbert_18_21_22_3_54_30).
8 [] P(i(i(x,y),i(i(y,z),i(x,z)))).
9 [] P(i(i(n(x),x),x)).
10 [] P(i(x,i(n(x),y))).
25 [] -P(i(x,y)) | -P(x) | P(y).
26 [] P(i(i(x,y),i(i(y,z),i(x,z)))).
27 [] P(i(i(n(x),x),x)).
28 [] P(i(x,i(n(x),y))).
----
29 [hyper,1,8,8] P(i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))).
33 [hyper,1,8,10] P(i(i(i(n(x),y),z),i(x,z))).
34 [hyper,1,10,9] P(i(n(i(i(n(x),x),x)),y)).
35 (heat=1) [hyper,25,33,27] P(i(x,x)).
36 (heat=1) [hyper,25,26,34] P(i(i(x,y),i(n(i(i(n(z),z),z)),y))).
37 [hyper,1,29,29] P(i(i(x,i(y,z)),i(i(u,y),i(x,i(u,z))))).
38 (heat=1) [hyper,25,37,27] P(i(i(x,y),i(i(n(i(y,z)),i(y,z)),i(x,z)))).
39 [hyper,1,33,36] P(i(x,i(n(i(i(n(y),y),y)),z))).

40 [hyper,1,37,38] P(i(i(x,i(n(i(y,z)),i(y,z))),i(i(u,y),i(x,i(u,z))))).
41 [hyper,1,40,39] P(i(i(x,i(n(y),y)),i(z,i(x,y)))).
43 [hyper,1,29,41] P(i(i(n(x),y),i(z,i(i(y,x),x)))).
44 [hyper,1,40,43] P(i(i(x,i(y,z)),i(i(n(z),y),i(x,z)))).
45 (heat=1) [hyper,25,44,28] P(i(i(n(x),n(y)),i(y,x))).
47 [hyper,1,37,44] P(i(i(x,i(n(y),z)),i(i(u,i(z,y)),i(x,i(u,y))))).
50 [hyper,1,33,45] P(i(x,i(y,x))).
53 [hyper,1,47,50] P(i(i(x,i(y,z)),i(y,i(x,z)))).
55 (heat=1) [hyper,25,53,28] P(i(n(x),i(x,y))).
56 (heat=1) [hyper,25,53,26] P(i(i(x,y),i(i(z,x),i(z,y)))).
63 [hyper,1,44,55] P(i(i(n(x),y),i(n(y),x))).
70 [hyper,1,47,63] P(i(i(x,i(y,z)),i(i(n(y),z),i(x,z)))).
78 [hyper,1,70,35] P(i(i(n(x),y),i(i(x,y),y))).
81 [hyper,1,53,78] P(i(i(x,y),i(i(n(x),y),y))).
83 [hyper,1,78,55] P(i(i(x,i(x,y)),i(x,y))).
87 [hyper,3,50,53,56,10,81,83] $ANSWER(step_allHilbert_18_21_22_3_54_30).

A 24-Step Proof of the Alternate Lukasiewicz Axiom System

----- Otter 3.0.2b+, Aug 1994 -----
The job was started by wos on altair.mcs.anl.gov, Fri Mar 31 18:33:12 1995
The command was "otter302c".

-----> EMPTY CLAUSE at 2.64 sec ----> 93 [hyper,5,51,63,85]
$ANSWER(step_allLuka_x_19_37_59).

Length of proof is 24. Level of proof is 15.

---------- PROOF ----------

1 [] -P(i(x,y)) | -P(x) | P(y).
5 [] -P(i(i(i(p,q),r),i(q,r))) | -P(i(i(i(p,q),r),i(n(p),r))) |
-P(i(i(n(p),r),i(i(q,r),i(i(p,q),r)))) | $ANSWER(step_allLuka_x_19_37_59).
8 [] P(i(i(x,y),i(i(y,z),i(x,z)))).
9 [] P(i(i(n(x),x),x)).
10 [] P(i(x,i(n(x),y))).
25 [] -P(i(x,y)) | -P(x) | P(y).
26 [] P(i(i(x,y),i(i(y,z),i(x,z)))).
27 [] P(i(i(n(x),x),x)).
28 [] P(i(x,i(n(x),y))).
----
29 [hyper,1,8,8] P(i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))).
33 [hyper,1,8,10] P(i(i(i(n(x),y),z),i(x,z))).
34 [hyper,1,10,9] P(i(n(i(i(n(x),x),x)),y)).
35 (heat=1) [hyper,25,26,34] P(i(i(x,y),i(n(i(i(n(z),z),z)),y))).
36 [hyper,1,29,29] P(i(i(x,i(y,z)),i(i(u,y),i(x,i(u,z))))).
37 (heat=1) [hyper,25,36,27] P(i(i(x,y),i(i(n(i(y,z)),i(y,z)),i(x,z)))).
38 [hyper,1,33,35] P(i(x,i(n(i(i(n(y),y),y)),z))).
39 [hyper,1,36,37] P(i(i(x,i(n(i(y,z)),i(y,z))),i(i(u,y),i(x,i(u,z))))).
40 [hyper,1,39,38] P(i(i(x,i(n(y),y)),i(z,i(x,y)))).
42 [hyper,1,29,40] P(i(i(n(x),y),i(z,i(i(y,x),x)))).
44 [hyper,1,39,42] P(i(i(x,i(y,z)),i(i(n(z),y),i(x,z)))).
45 (heat=1) [hyper,25,44,28] P(i(i(n(x),n(y)),i(y,x))).
47 [hyper,1,36,44] P(i(i(x,i(n(y),z)),i(i(u,i(z,y)),i(x,i(u,y))))).
49 [hyper,1,33,45] P(i(x,i(y,x))).

51 (heat=1) [hyper,25,26,49] P(i(i(i(x,y),z),i(y,z))).
54 [hyper,1,47,42] P(i(i(x,i(i(i(y,z),z),u)),i(i(n(z),y),i(x,u)))).
55 [hyper,1,47,49] P(i(i(x,i(y,z)),i(y,i(x,z)))).
57 (heat=1) [hyper,25,55,28] P(i(n(x),i(x,y))).
58 (heat=1) [hyper,25,55,26] P(i(i(x,y),i(i(z,x),i(z,y)))).
63 [hyper,1,8,57] P(i(i(i(x,y),z),i(n(x),z))).
67 [hyper,1,8,58] P(i(i(i(i(x,y),i(x,z)),u),i(i(y,z),u))).
72 [hyper,1,47,63] P(i(i(x,i(y,z)),i(i(i(z,u),y),i(x,z)))).
78 [hyper,1,67,72] P(i(i(x,y),i(i(i(y,z),u),i(i(u,x),y)))).
85 [hyper,1,54,78] P(i(i(n(x),y),i(i(z,y),i(i(x,z),y)))).
93 [hyper,5,51,63,85] $ANSWER(step_allLuka_x_19_37_59).

A 25-Step Proof of the Wos Axiom System

----- Otter 3.0.2b+, Aug 1994 -----
The job was started by wos on altair.mcs.anl.gov, Fri Mar 31 20:47:12 1995
The command was "otter302c".

-----> EMPTY CLAUSE at 3.61 sec ----> 110 [hyper,6,51,63,101]
$ANSWER(step_allWos_x_19_37_60).

Length of proof is 25. Level of proof is 16.

---------- PROOF ----------

1 [] -P(i(x,y)) | -P(x) | P(y).
6 [] -P(i(i(i(p,q),r),i(q,r))) | -P(i(i(i(p,q),r),i(n(p),r))) |
-P(i(i(s,i(n(p),r)),i(s,i(i(q,r),i(i(p,q),r))))) | $ANSWER(step_allWos_x_19_37_60).
8 [] P(i(i(x,y),i(i(y,z),i(x,z)))).
9 [] P(i(i(n(x),x),x)).
10 [] P(i(x,i(n(x),y))).
25 [] -P(i(x,y)) | -P(x) | P(y).
26 [] P(i(i(x,y),i(i(y,z),i(x,z)))).
27 [] P(i(i(n(x),x),x)).
28 [] P(i(x,i(n(x),y))).
----
29 [hyper,1,8,8] P(i(i(i(i(x,y),i(z,y)),u),i(i(z,x),u))).
33 [hyper,1,8,10] P(i(i(i(n(x),y),z),i(x,z))).
34 [hyper,1,10,9] P(i(n(i(i(n(x),x),x)),y)).
35 (heat=1) [hyper,25,26,34] P(i(i(x,y),i(n(i(i(n(z),z),z)),y))).
36 [hyper,1,29,29] P(i(i(x,i(y,z)),i(i(u,y),i(x,i(u,z))))).
37 (heat=1) [hyper,25,36,27] P(i(i(x,y),i(i(n(i(y,z)),i(y,z)),i(x,z)))).
38 [hyper,1,33,35] P(i(x,i(n(i(i(n(y),y),y)),z))).
39 [hyper,1,36,37] P(i(i(x,i(n(i(y,z)),i(y,z))),i(i(u,y),i(x,i(u,z))))).
40 [hyper,1,39,38] P(i(i(x,i(n(y),y)),i(z,i(x,y)))).
42 [hyper,1,29,40] P(i(i(n(x),y),i(z,i(i(y,x),x)))).
44 [hyper,1,39,42] P(i(i(x,i(y,z)),i(i(n(z),y),i(x,z)))).
45 (heat=1) [hyper,25,44,28] P(i(i(n(x),n(y)),i(y,x))).
47 [hyper,1,36,44] P(i(i(x,i(n(y),z)),i(i(u,i(z,y)),i(x,i(u,y))))).
49 [hyper,1,33,45] P(i(x,i(y,x))).
51 (heat=1) [hyper,25,26,49] P(i(i(i(x,y),z),i(y,z))).
54 [hyper,1,47,42] P(i(i(x,i(i(i(y,z),z),u)),i(i(n(z),y),i(x,u)))).
55 [hyper,1,47,49] P(i(i(x,i(y,z)),i(y,i(x,z)))).
57 (heat=1) [hyper,25,55,28] P(i(n(x),i(x,y))).
58 (heat=1) [hyper,25,55,26] P(i(i(x,y),i(i(z,x),i(z,y)))).

63 [hyper,1,8,57] P(i(i(i(x,y),z),i(n(x),z))).
67 [hyper,1,8,58] P(i(i(i(i(x,y),i(x,z)),u),i(i(y,z),u))).
73 [hyper,1,47,63] P(i(i(x,i(y,z)),i(i(i(z,u),y),i(x,z)))).
81 [hyper,1,67,73] P(i(i(x,y),i(i(i(y,z),u),i(i(u,x),y)))).
89 [hyper,1,54,81] P(i(i(n(x),y),i(i(z,y),i(i(x,z),y)))).
101 [hyper,1,58,89] P(i(i(x,i(n(y),z)),i(x,i(i(u,z),i(i(y,u),z))))).
110 [hyper,6,51,63,101] $ANSWER(step_allWos_x_19_37_60).

References

[Kalman78] Kalman, J., ''A shortest single axiom for the classical equivalential calculus'', *Notre Dame J. Formal Logic* **19,** 141-144 (1978).

[Kalman83] Kalman, J., ''Condensed detachment as a Rule of Inference'', *Studia Logica* **42,** 443-451 (1983).

[Lukasiewicz63] Lukasiewicz, J., *Elements of Mathematical Logic,* Pergamon Press: Oxford, 1963.

[McCharen76] McCharen, J., Overbeek, R., and Wos, L., ''Complexity and related enhancements for automated theorem-proving programs'', *Computers and Mathematics with Applications* **2,** 1-16 (1976).

[McCune92] McCune, W., and Wos, L., ''Experiments in automated deduction with condensed detachment'', pp. 209-223 in *Proceedings of the Eleventh International Conference on Automated Deduction* (CADE-11), *Lecture Notes in Artificial Intelligence*, Vol. 607, ed. D. Kapur, Springer-Verlag, New York, 1992.

[McCune93] McCune, W., ''Otter 3.0'', Preprint MCS-P399-1193, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, November 1993.

[Padmanabhan95a] Padmanabhan, P., and McCune, W., ''Automated Reasoning about Cubic Curves'', *Computers and Mathematics with Applications* **29,** no. 2, 17-26 (January 1995).

[Padmanabhan95b] Padmanabhan, P., and McCune, W., ''Single identities for ternary Boolean algebras'', *Computers and Mathematics with Applications* **29**, no. 2, 13-16 (January 1995).

[Wos87] Wos, L., *Automated Reasoning: 33 Basic Research Problems,* Prentice-Hall: Englewood Cliffs, N.J., 1987.

[Wos90] Wos, L., ''Meeting the challenge of fifty years of logic'', *J. Automated Reasoning* **6**, no. 2 (June 1990) 213-232.

[Wos91] Wos, L., ''Automated reasoning and Bledsoe's dream for the field'', pp. 297-345 in *Automated Reasoning: Essays in Honor of Woody Bledsoe,* ed. R. S. Boyer, Kluwer Academic Publishers: Dordrecht, 1991.

[Wos92] Wos, L., Overbeek, R., Lusk, E., and Boyle, J., *Automated Reasoning: Introduction and Applications*, 2nd ed., McGraw-Hill: New York, 1992.

[Wos93] Wos, L., ''The kernel strategy and its use for the study of combinatory logic,'', *J. Automated Reasoning* **10**, no. 3 (June 1993) 287-343.

[Wos95a] Wos, L., *The Automation of Reasoning: An Experimenter's Notebook with OTTER Tutorial*, accepted for publication by Academic Press (1995).

[Wos95b] Wos, L., ‘‘The resonance strategy’’, *Computers and Mathematics with Applications* **29,** no. 2, 133-178 (January 1995).

[Wos95c] Wos, L., **Searching for Circles of Pure Proofs***, *J. Automated Reasoning* (accepted for publication).