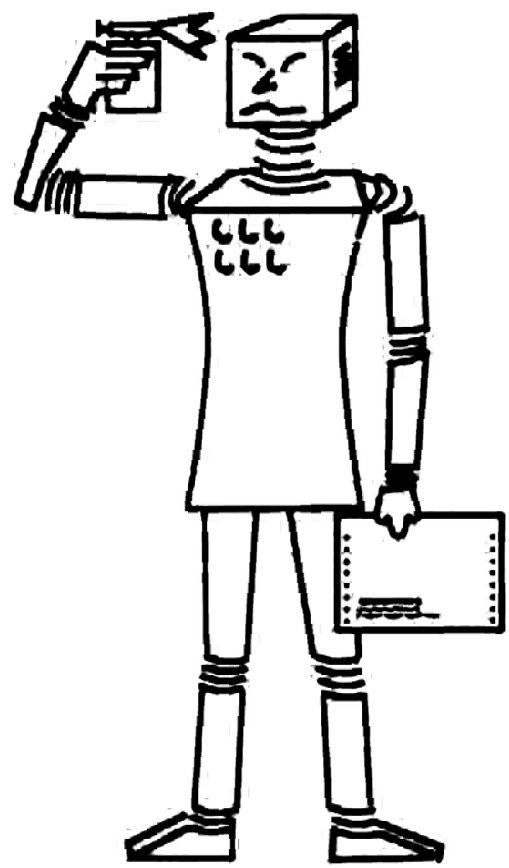
Fachbereich Informatik
Universität Kaiserslautern
D-67663 Kaiserslautern

SEKI - REPORT



WALDMEISTER: Development of a High Performance Completion-Based Theorem Prover

Arnim Buch, Thomas Hillenbrand

SEKI Report SR-96-01

WALDMEISTER: Development of a High Performance Completion–Based Theorem Prover

Arnim Buch Thomas Hillenbrand

SEKI–Report SR-96-01

Universität Kaiserslautern,
Fachbereich Informatik,
D-67653 Kaiserslautern, FRG
email: {buch,hillen}@informatik.uni-kl.de

December 3, 1996

Abstract: In this report we give an overview of the development of our new WALDMEISTER prover for equational theories. We elaborate a systematic stepwise design process, starting with the inference system for unfailing Knuth–Bendix completion and ending up with an implementation which avoids the main diseases today's provers suffer from: overindulgence in time and space.

Our design process is based on a logical three–level system model consisting of basic operations for inference step execution, aggregated inference machine, and overall control strategy. Careful analysis of the inference system for unfailing completion has revealed the crucial points responsible for time and space consumption. For the low level of our model, we introduce specialized data structures and algorithms speeding up the running system and cutting it down in size — both by one order of magnitude compared with standard techniques. Flexible control of the mid–level aggregation inside the resulting prover is made possible by a corresponding set of parameters. Experimental analysis shows that this flexibility is a point of high importance. We go on with some implementation guidelines we have found valuable in the field of deduction.

The resulting new prover shows that our design approach is promising. We compare our system's throughput with that of an established system and finally demonstrate how two very hard problems could be solved by WALDMEISTER.

Contents

1 Introduction

A closer look at automated deduction systems reveals three levels: basic operations, aggregrated inference machine and overall control strategy. A lot of work is put into the development of sophisticated heuristics on the top level. Making provers learn from previous runs is employed in order to enable the programs to use their experience from the past ([Fuc96b]). Another successful approach is to take domain knowledge into account and thus to adjust the prover to new problem classes independently ([Sti84], [DS96]). Furthermore, distributing automated theorem proving on the high level ([AD93], [CMM90]), or parallelizing it on the low level ([BGK96]) is tried in order to reduce the wall–clock run time of provers. Apart from that, the development of data structures and algorithms for specialized high–speed operations like retrieval of generalizations and instances of terms has made good progress ([McC92], [Gra96]). In contrast, not enough attention is yet paid to the question how the single inference steps should be aggregated into an inference machine.

If research focusses on one of the three levels, usually only standard techniques are applied on the other levels in order to build quickly a prototype — which then of course is not uniformly optimized. Establishing sophisticated selection strategies will never yield an extraordinary high performance if the underlying inference machine is not accordingly optimized on the levels both of aggregation and basic operations. Naturally, fast but stupid provers must be considered unsatisfactory as well. This focussing on just one level might be the reason why provers run out of bounds even *before* attacking challenging problems. For building successful provers, all approaches must be combined.

In this report we describe the first part of this task for the field of equational deduction: We start with the inference system for unfailing Knuth–Bendix completion (Sect. 2). We introduce a logical three–level system model of saturation–based provers. Aiming at efficiency in terms of time and space, we propose the paradigm of an *engineering approach* for design and implementation. Afterwards we outline the systematic, stepwise design process (Sect. 3) we followed in the making of WALDMEISTER.

We derive a sketch of algorithm and name its crucial points in terms of time and space consumption. For the level of basic operations, we introduce specialized data structures and algorithms speeding up the running system and cutting it down in size – both by one order of magnitude compared to standard techniques (Sect. 4, 5, 6). For the mid level, we confirm an appropriate aggregration of the inference steps into an efficient inference machine by careful evaluation of our measurements on the open questions (Sect. 7). In Sect. 8 we describe how we realized an efficient implementation, following the paradigm of imperative programming.

The resulting new prover WALDMEISTER[1] shows that our design approach is promising (Sect. 9). Although we have not yet spent much effort on specialized heuristics we gain impressive results on standard problems. Furthermore, WALDMEISTER solves two problems from [LW92] never proved unaided with standard unfailing completion be-

[1] In German this means *woodruff* and is composed of *forrest* and *master*.

fore (Sect. 10). The completely machine generated proof of one of the problems can be found in App. A, whereas App. B contains the problem specifications we used for our experiments.

2 Dealing with Equational Reasoning

What is the typical task of a deduction system? A fixed *logic* has been operationalized towards a *calculus* the application of which is steered by some *control* strategy. Before we will describe the design process that culminated in WALDMEISTER, we will state the starting point of the whole process, i.e. the inference system for unfailing Knuth–Bendix completion.

Unfailing completion

In [KB70], Knuth and Bendix introduced the completion algorithm which tries to derive a set of convergent rules from a given set of equations. With the extension to unfailing completion in [BDP89], it has turned out to be a valuable means of proving theorems in equational theories (positive first–order unit equality clauses).

The following nine inference rules form a variant of unfailing completion which is suitable for equational reasoning. It is based both on [BDP89], which the notation is following, and [AD93]. The inference system works on triples E, R, G of sets of termpairs (equations, rules, and goals). The termpairs in $E \cup R$ are positive unit equality clauses which are implicitely all–quantified. The goals are negated and skolemized all–quantified equations; hence, they contain ground expressions only. For convenience, they are also written with the equality symbol.[2] The symbol $\succ$ denotes an arbitrary reduction ordering which is total on equivalent ground terms and $\rhd$ the specialization ordering ($s \rhd t$ if and only if some subterm of s is an instance of t, but not vice versa).

Orient

$$\frac{E \cup \{s \doteq t\}, R, G}{E, R \cup \{s \to t\}, G} \quad \text{if } s \succ t$$

Generate

$$\frac{E, R, G}{E \cup \{s = t\}, R, G} \quad \text{if } s = t \in CP(R, E)$$

Simplify an equation

$$\frac{E \cup \{s \doteq t\}, R, G}{E \cup \{u \doteq t\}, R, G} \quad \text{if } s \to_R u \text{ or } s \to_{E_\succ, l=r} u \text{ with } s \rhd l$$

Delete an equation

$$\frac{E \cup \{s = s\}, R, G}{E, R, G}$$

[2] WALDMEISTER is also able to deal with existentially quantified goals, which requires additional inference rules for narrowing. See [Den93] for more details.

Subsume an equation

$$\frac{E \cup \{s \doteq t, u[\sigma(s)] \doteq u[\sigma(t)]\}, R, G}{E \cup \{s \doteq t\}, R, G}$$

Simplify the right–hand side of a rule

$$\frac{E, R \cup \{s \to t\}, G}{E, R \cup \{s \to u\}, G} \quad \text{if } t \to_{R \cup E_\succ} u$$

Simplify the left–hand side of a rule

$$\frac{E, R \cup \{s \to t\}, G}{E \cup \{u = t\}, R, G} \quad \text{if } s \to_{R \cup E_\succ, l=r} u \text{ with } s \rhd l$$

Simplify a goal

$$\frac{E, R, G \cup \{s \doteq t\}}{E, R, G \cup \{u \doteq t\}} \quad \text{if } s \to_{R \cup E_\succ} u$$

Success

$$\frac{E, R, G \cup \{s = t\}}{SUCCESS} \quad \text{if } s \equiv t$$

An informal illustration of the inference system is given in Fig. 1. The boxes represent sets in the mathematical sense and contain pairs of terms: namely rules, equations, goals, and critical pairs. Initially, E includes the axioms and G the hypotheses. The set $CP(R, E)$ is a function of the current sets R and E and holds the essence of all the local divergencies. These arise whenever on the same term two different rewrite steps are applicable.

The arrows denote single inference steps, and each shifts a single termpair from one set to another. Along the *simplify* arrows, the terms alter according to the single–step rewrite relation induced by R and E. An equation with identical sides may be *deleted.* The same applies to an equation that is *subsumed* by a more general one. As soon as one side is greater than the other with respect to a fixed reduction ordering, the equation may be *oriented* into a rule. If the *left–hand side* of a rule is simplified, this rule becomes an equation again. This does not happen when reducing the *right–hand side.* A new equation is *generated* by selecting a termpair from $CP(R, E)$. As soon as left–hand side and right–hand side of a goal are identical, the corresponding hypothesis is *successfully* proved.

Instantiating this inference system leads to proof procedures. To achieve semi–completeness, a control constraint — the so–called fairness — has to be considered: From every critical pair the parents of which are persistent an equation must be generated some time.

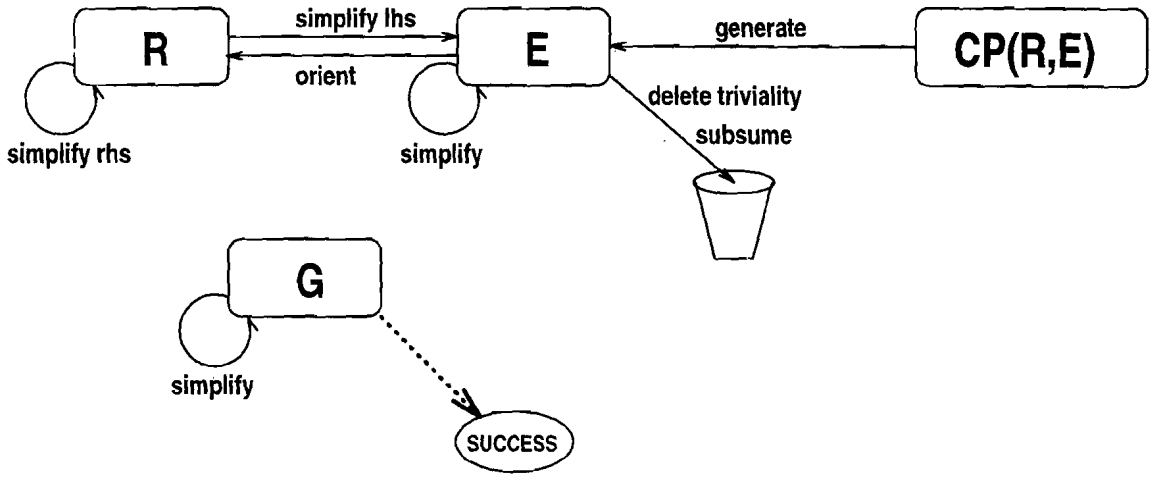


Figure 1: The inference system for unfailing completion

3 Design: Deduction Meets Software Engineering

The main goal during the development of WALDMEISTER was overall efficiency. To achieve this goal, one has to follow the paradigm of an *engineering approach:* Analyzing typical saturation–based provers, we recognize a logical three–level structure (see below). We will analyze each of these levels with respect to the critical points responsible for the prover's overall performance. Wherever necessary, the question of how to tackle these critical points will have to be answered by extensive experimentation and statistical evaluation. In this context, an experiment is a fair, quantitative comparison of different realizations of the same functionality within the same system. Hence, modularity has to be a major principle in structuring the system.

Now, what does that mean in practice? Imagine a certain functionality needed for constructing say an unfailing completion procedure, e.g. normalization of terms. Out of the infinitely many different realizations, one could choose an arbitrary one and forget about the rest. However, there is no justification for restricting a system to a single normalization strategy, since different strategies are in demand for different domains. Hence, the system should include numerous normalization routines (allowing easy addition of other realizations) and leave the choice between them up to the user. As this applies to many choice points, an open system architecture produces a set of system parameters for fixing open choice points in one of the supported ways, and thus allows an easy experimental comparison of the different solutions. It is not before now that an optimal one can be named — if there is one: in some cases different domains require different solutions.

In this section we will depict the afore–mentioned three–level model, and describe the systematic stepwise design process we followed during the development of WALDMEISTER. The seven steps we have followed can be generalized towards applicability to arbitrary (synthetic) deduction systems.

The three–level model

We aim at automated deduction systems including a certain amount of control coping with many standard problems — in contrast to interactive provers. The basis for our development is the afore–mentioned logical *three–level model* of completion–based provers (see Fig. 2). As every calculus that is given as an inference system, unfailing completion has many inherent indeterminisms. When realizing completion procedures in provers, this leads to a large family of deterministic, but parameterized algorithms. Two main parameters are typical for such provers: the reduction ordering as well as the search *heuristic* for guiding the proof. Choosing them for a given proof task forms the top level in our model. The mid–level is that of the *inference machine,* which aggregates the inference rules of the proof calculus into the main completion loop. This loop is deterministic for any fixed choice of reduction ordering and selection heuristic. Since there is a large amount of freedom, many experiments are necessary to assess the differing aggregations before a generally useful one can be found. The lowest

level provides efficient algorithms and sophisticated data structures for the execution of the most frequently used *basic operations,* e.g. matching, unification, storage and retrieval of facts. These operations consume most of the time and space.

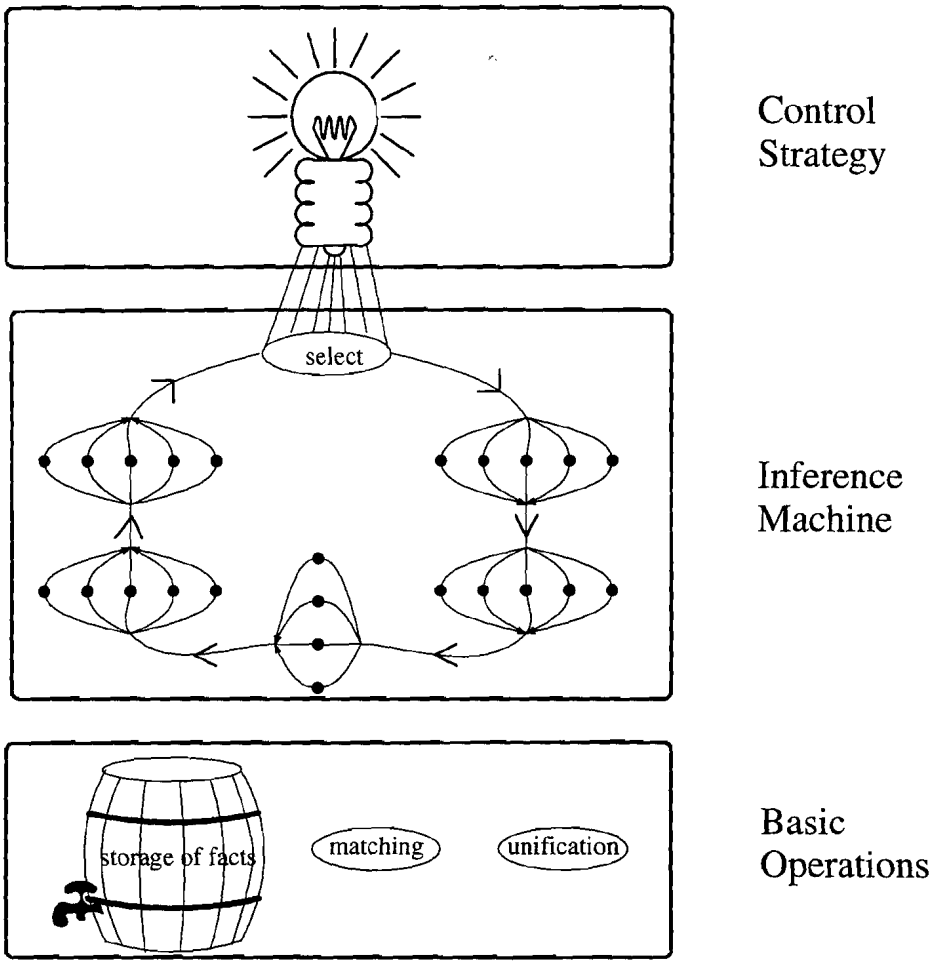


Figure 2: A three–level model for saturation–based deduction systems

Seven steps for designing an efficient prover

Trying to solve a given problem by application of an inference system, one has to proceed as follows:

while the problem is not yet solved do
(a) select an inference rule $\mathcal{R}$
(b) select a tuple of facts $\vec{x}$
(c) apply $\mathcal{R}$ to $\vec{x}$
end

Proceeding that way, one will usually be confronted with two different kinds of decisions to be made. The first one is (a) to select an inference rule, which we will call a "global choice point". If these selections are steered by some kind of control, their sequence already can be seen as an aggregation of the inference rules into an algorithm. Aiming at efficiency, this aggregation must base on deep understanding of the inference system.

The second kind of decisions deals with what we call "local choice points" (b): Each inference rule is instantiated with a tuple of facts which has to be chosen from all the tuples to which it is applicable at the moment.

With these notions, the seven steps in the design process are the following:
(1) Identify the local choice points.
(2) Name your experiences.
(3) Group the inference rules into a sketch of algorithm.
(4) Define a set of system parameters.
(5) Reveal the crucial points and develop appropriate data structures & algorithms.
(6) Implement carefully.
(7) Find a default parameter setting.

The first two steps are important for all the three levels. Steps 3 and 4 deal with the inference machine, step 5 concerns the basic operations level. Step 6 is obviously inevitable for all levels. Finally, step 7 refers to the upper two levels, and accomplishes the engineering approach.

(1) Identify the local choice points

The first step in our design process is to identify the local choice points. Let us have a look at the inference system for unfailing completion:

- *Orienting* an equation into a rule is instantiated with an equation and the given reduction ordering that is fixed during the proof run.

- The four *simplifying* inference rules each have to be instantiated with the object, namely the equation, rule or goal they are applied to, the position in the term to be simplified and the rule or equation used for simplification.

- The decision which equation should be *deleted* from the set E due to triviality or due to *subsumption* obviously is of minor importance. Nevertheless, as subsumption testing takes time, it should be possible to turn it off if the proof process does not benefit from it.

- Finally, there is the *generation* of new equations which depends on the "parent" rules/equations and a term position, restricted by the existence of a unifier.

(2) Name your experiences

Naming the experiences gathered either from prototyping, from the intention behind the inference system, or from manual computation of many examples should be done before further design decisions are met. Additional completeness constraints have to be taken into consideration as well. Statistical results gained from mechanical experiments with a prototypical implementation have to be correlated with the test environment.

(3) Group the inference rules into a sketch of algorithm

With these experiences in mind, a first draft of the system will arise from grouping the inference rules into a sketch of algorithm. Since we are following the imperative programming paradigm, we use iteration, branching and sequencing as basic construction elements. At this design step, object classes and access points for a high–level control strategy should be identified as well. Only those global and local choice points should be fixed that could definitely be cleared by experience.

	Experience says	
	fix it!	don't know
global choice points	(sketch of) algorithm	design challenge or interactive prototype
local choice points	restricted functionality of inference steps	parameter–dependent behaviour

Table 1: Characterization of choice points

We will now derive such a sketch for the case of unfailing completion. Naturally, *generation* of a new equation will be followed by applying *simplification*, *orientation* or *deletion* to that very equation. For that reason, *generation* can be seen as the selection of the critical pair from $CP(R, E)$ to be processed next. Of course only those critical pairs are of concern that have not yet been considered.

Experience shows that — besides the choice of the reduction ordering — this selection is the most important choice point of all: It determines success or failure in the given space and time bounds. Although the fairness constraint restricts the *select* function there is still a large variety of different realizations. Providing *select* with as much profitable information as possible is a major requirement a good inference machine has to meet. This means especially that *select* should have access to all the critical pairs from $CP(R, E)$ in order to assess them with respect to their value in the proof process.

It is generally accepted that a valuable analysis of critical pairs relies on extracting information from their terms. As long as *select* only considers raw critical pairs, it will often be misled. Consequently, a kind of preprocessing should precede the selection. This preprocessing usually includes a certain amount of simplification with respect to

the current rewrite relation. On this occasion, joinable and subsumable critical pairs can be easily detected and hence removed.

Doing preprocessing every time an equation is selected from $CP(R, E)$ would be far too expensive. Hence, not only the critical pairs but also the information computed via preprocessing should be stored. This may include the already generated (and possibly simplified) terms as well as information required for an easier selection, such as the length or depth of the terms. In agreement with the inference system, we must consider this set as a subset of E because preprocessing may require the application of *simplify*. In contrast, all the other inference rules will not be applied to this special subset of E. We will call the resulting set *SUE*, the **S**et of **U**nselected **E**quations. Accordingly, $E_{selected} = E \setminus SUE$ will be called the set of selected equations, and its members simply 'equations'.

Since selection can be seen as the search for the "best" element in *SUE*, it is of the same complexity as a minimum–search. Taking a look at the changes that appear in *SUE* between two successive applications of *select* reveals that exactly one element has been removed whereas a few elements might have been added. As every selection strategy induces an ordering on this set, we can realize *SUE* as a priority queue.

Storing all the unselected but preprocessed equations in such a manner induces serious space problems (cf. Sect. 5). Hence, we follow two approaches to keep this set small. On the one hand, the total number of generated equations should be minimized. As only critical pairs built with rules and equations from $R \cup E_{selected}$ will be added to *SUE*, we must guarantee that $R \cup E_{selected}$ holds rules and equations which are minimal in the following sense: A rule or equation is minimal if each of the following conditions holds:

- both sides cannot be simplified by any other minimal rule or equation,
- it is not trivial and cannot be subsumed,
- it has not been considered superfluous by subconnectedness criteria,
- it has been assessed valuable and thus once been *selected*, and
- it has been oriented into a rule if possible.

On the other hand, unselected equations should be eliminated whenever possible — if they are trivial or subsumable or even joinable with respect to the current rewrite relation. This (combined with preprocessing) causes innumerable invocations of *simplify*, although the application of these inference rules is restricted to rewriting with minimal rules and equations. Even more equations may be deleted from *SUE* as the fairness constraint only requires that all the critical pairs stemming from persistent rules/equations have to be built (and processed) some time. If a rule or equation has been removed from $R \cup E_{selected}$ all the descending critical pairs that are stored as unselected equations in *SUE* can be removed ("orphan murder", see also [Ave95]).

The minimality property guarantees the following:

- simplified right–hand sides shorten simplification chains,
- orientation whenever possible saves calls of the reduction ordering,
- generated equations do not impose superfluous simplifications, and finally
- no superfluous equations (descending from parents that can be

removed via interreduction) will be generated.

The minimality property can be guaranteed by executing interreduction on $R \cup E_{selected}$ with every equation that has been selected from SUE and appeared to be minimal as well. Interreduction means that rules the left–hand sides of which could be simplified, and equations any side of which could be simplified will be shifted from R respectively $E_{selected}$ to SUE.[3]

Another common design decision is to repeat the application of *simplify* unless an irreducible successor of the given term is found. This results in a parameterized normalization procedure instead of the stepwise execution of *simplify*.

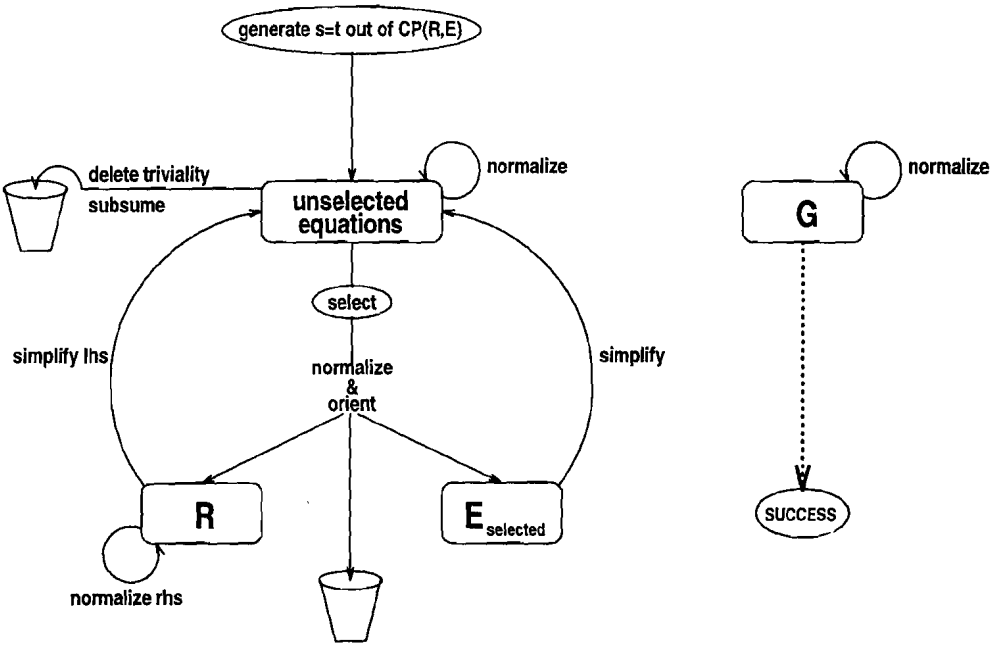


Figure 3: Sketch of algorithm

Having seperated E into minimal (thus selected) and yet unselected equations, hence $E = E_{selected} \dot{\cup} SUE$, and restricting both *generation* of new equations and *simplifying* to minimal rules and equations, we come get a specialization of the inference system that is shown in Fig. 3. It still corresponds to the basic inference system from Fig. 1, since it only restricts *generate* and *simplify* to R and $E_{selected}$. Fairness can be ensured as long as *select* fulfills certain properties.

Now it is easy to derive a sketch of algorithm: A newly selected equation will first be normalized (applying *simplify*), then *deleted* or *subsumed* if possible. Then the new member of $R \cup E_{selected}$ is used to modify the facts already in $R \cup E_{selected}$ (interreduction), or the critical pairs in SUE. Afterwards, it will be added to R if it could

[3]An interesting point is that the distinction between selected and unselected equations is also obtained if one tries to direct the execution of the inference steps *simplify* in such a manner that no superfluous work is done: Rewriting is done with minimal rules/equations only.

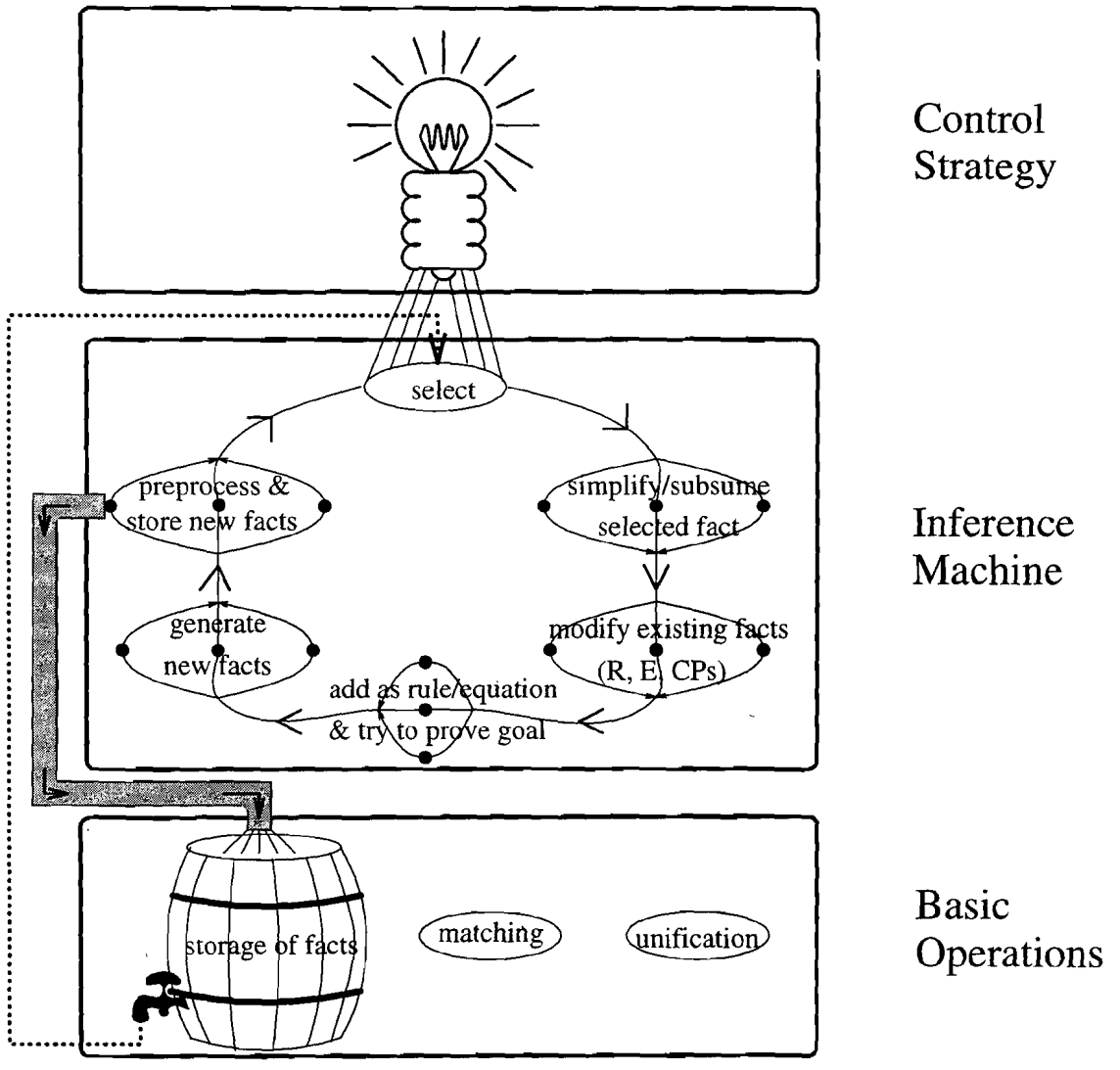


Figure 4: The three–level model instantiated for unfailing Knuth–Bendix completion

be *oriented*, otherwise it becomes a member of $E_{selected}$. Former rules or equations that have been removed during interreduction will be added to *SUE* again. Finally, all the critical pairs that can be built with the new rule or equation are generated, preprocessed, and added to *SUE* which closes the cycle. From time to time, one must take a look at the goals and check whether they can already be joined. Furthermore, reprocessing the elements of *SUE* should be taken into consideration (intermediate reprocessing, *IRP*). In Fig. 4 we have instantiated the general three–level model according to this sketch of algorithm.

(4) Define a set of system parameters

Having designed a sketch of algorithm, we now have to deal with those choice points which could not definitely be fixed by experience and thus — in the framework of an open system architecture as induced by the engineering approach — shall be left open to the user. More precisely, some of the system's functionality will have to be realized in different ways: A certain task will be fulfilled by several components; and the user (or the control strategy) must be able to select one of them at run time. This leads to a corresponding set of system parameters. As soon as the deduction system is completed, statistical evaluations of different parameter settings should result in a kind of default setting suitable for most problem classes.

In the context of unfailing completion, we found three groups of remaining choice points.

- High–level control parameters:
  - the reduction ordering yielding oriented equality, and
  - the *select* function guiding the selection of equations from *SUE*.
- Treatment of unselected equations:
  - the organization of intermediate reprocessing which includes simplification and/or re–classification of the yet unselected equations,
  - the criteria that shall be applied immediately after new equations have been generated (e.g. subconnectedness criteria), and
  - the amount of simplification preceeding the classification of newly generated equations (preprocessing).
- Normalization of terms:
  - the term traversal strategy (e.g. leftmost–outermost, leftmost–innermost),
  - the way of combining R and $E_{selected}$ which may differ for the various situations normalization is applied,
  - the priority of rules when simplifying with respect to $R \cup E_{selected}$,
  - the question which of the possibly more than one applicable rules and/or equations shall be applied, and
  - the backtracking behaviour after the execution of a single rewrite–step.

Although a few of these choice points might appear quite unimportant, all of them have proven valuable during our experimental analysis (see Sect. 7), or at least of significant influence on the prover's performance.

(5) Reveal the crucial points and develop appropriate data structures & algorithms

Now, a closer look at the sketch of algorithm must reveal the crucial points where time or memory are consumed. It is the low level that holds responsible for the system's throughput (inference steps per time unit) and how far the computation can be taken in the space limits given by the environment. If on this level appropriate data structures and algorithms are employed, the execution of the critical inference steps will be improved significantly. Consequently, the whole system will be sped up and cut down in size.

What are the crucial points in the case of unfailing completion? Preprocessing the newly generated equations usually employs normalization. Since this causes most of the system's time consumption, data structures and algorithms for fast normalization are in demand — put in concrete terms, this means rewriting with high efficiency (Sect. 4). To avoid expensive recomputation, the information already generated should be kept, which basically concerns the critical pairs generated during a proof run. As millions of them may arise during such a run, the set of unselected equations must be represented in a space saving way (Sect. 5, 6).

(6) Implement carefully

Implementing transforms the design of efficient algorithms and data structures into a program that runs efficiently on real machines (Sect. 8). This must not be underestimated: For example, efficient memory management can cut down the run times by half and thus should be done before brooding over sophisticated unification routines.

(7) Find a default parameter setting

Detailed statistical information should be available for the user, because precise analysis of the system behaviour is the basis for a better understanding of the proof process. This is eased by the fine-grained influences on the inference machine. Statistical analysis of the open choice points should be done with the finished system in order to find a default parameter setting, or at least to name tendencies on how these parameters affect the system's behaviour. For the case of unfailing completion as realized in WALDMEISTER see Sect. 7.

(*) Let the system run

The user running the system and getting fascinated by its (hopefully) ultimative performance, it will not take too long until new desires arise. Our seven step design process yields an open architecture that not only allows but even encourages further enhancements and improvements. In our case, especially the representation of *SUE* was improved more and more during our work with the system. This was because the high performance retrieval operations led to a very large amount of facts computed

in quite a short time span — and hence we got in trouble with colleagues working on machines on which WALDMEISTER was running.

To end with, let us make a few remarks on software development in the field of deduction. Usually, we have some constant top–level requirements — as in our case 'build a good prover based on Knuth–Bendix completion', where 'good' means economic use of the resources time and space. During the design phase, developers and users coincide. The experience with the program and the knowledge about its behaviour increase, naturally leading to changing requirements on a lower level such as "Let us try this or that, or better both!" (cf. Sect. 7). Now, how can we deal with this kind of changing requirements? Here are a couple of hints we found valuable:

- Flexibility: Leave as many influences to the user as possible. Even try to leave the "This point is absolutely clear, and everybody who says something different is a complete fool."–points open — and be it only to prove everybody a complete fool who says something different.
- Rigidity: Fix only those points (and none else) that are 150% clear. Points that have been confirmed by measurements can be fixed later.
- Interchangeability: Keep in mind that it may be necessary to exchange complete subsystems (open architecture).
- Economicalness: Always keep in mind that time and space are valuable and should not be wasted to superfluous operations or data.

4 Data Structures for Fast Rewriting

The rewriting task is the following: Given a set R of rules and E of equations, furthermore a single term t and a reduction ordering $>$, derive a successor of t. Every derivation step is done by applying a suitable element $l \to r$ of R or $l \doteq r$ of E where a rule is suitable iff its left-hand side is more general than the actual term or one of its subterms; and the match of the left–hand side will be replaced by the right–hand side under the corresponding substitution. Equations may only be applied if additionally the former subterm is greater than its substitute. Employing different term traversal strategies, normalization routines are constructed via subsequent application of single rewrite steps.

As during completion the set of rules grows larger and larger, the bottle–neck in the normalization procedure is the search for generalizations of a single query term in the whole set of rules. How this bottle–neck is tackled sets the starting–point for the design of the fundamental data structures. Our solution consists of specialized indexing techniques (see [Gra96] for an overview) going hand in hand with an appropriate term representation.

4.1 Flatterms

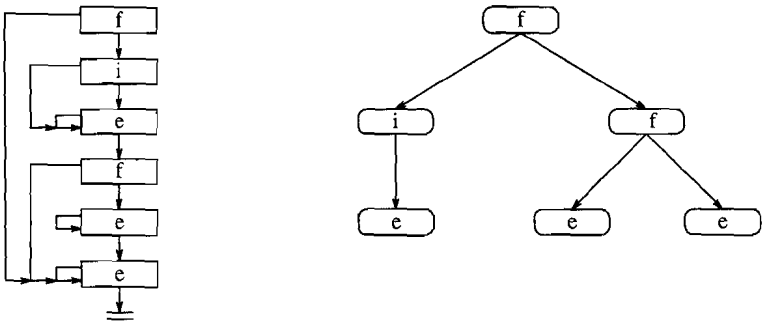


Figure 5: Flatterms vs. tree terms

Flatterms as an alternative to tree–like recursive term representations were introduced by Christian in [Chr89]. The function symbols and variables — represented by non–negative resp. negative integers — are collected in a doubly linked list just in the order of a preorder term traversal. For each list cell, there is an extra entry pointing to the last cell employed by the corresponding subterm. This allows skipping and fast traversal of subterms. As recursions are replaced by simple iterations, the use of flatterms speeds up fundamental operations like copying or equality testing. Furthermore, they correspond perfectly to discrimination trees because the latter base their retrieval operations on preorder term traversal, too. For his HIPER system, Christian states an "overall speedup... in the neighborhood of 30 percent." In our opinion, in rewriting it

is sufficient to use singly linked lists instead of doubly linked ones.[4]

Using flatterms in combination with free–list based memory management has given us yet another advantage: Flatterms can be disposed of in constant time. To this end, the list of free term cells is just appended to the term to discard, now seen as list of term cells. This is done by changing one pointer in the last cell of the term. This technique does not only apply to the case of discarding joinable critical pairs: When applying a rewrite rule such as $f(x, y, z) \rightarrow g(y)$ with substitution σ, the now unnecessary subterms $\sigma(x)$ and $\sigma(z)$ are disposed the same way. This kind of explicit memory management clearly outweighs implicit ones using garbage collection.

4.2 Refined Perfect Discrimination Trees

Indexing techniques have been used in automated theorem proving since the early 70ies. McCune published comparative experiences on this field in [McC92]. He suggested the use of special tries, the so–called standard discrimination trees. Every edge of such a tree is marked with a function symbol or a special wildcard symbol instead of variables. The leafs contain rules with left–hand sides identical but for the variables. Each path from the root to a leaf carries the corresponding term on the edges. Retrieval operations such as searching for generalizations are based on tree traversal using backtracking.

A variant known as perfect discrimination tree distinguishes distinct variables inside the index. Indexed terms must be normalized with respect to their variables: Only $x_1, x_2, \ldots$ are allowed; and the numbering inside of a term must accord to the order of appearance in a left–to–right term traversal. However, when indexing rewrite rules, this is no restriction at all.

The refinement we propose consists in shrinking all branches leading to a single leaf node. So only the information needed to discriminate the prefixes of the indexed terms is stored along the edges, and the remainder belongs to the leaf node. Have a look at Fig. 6 for an example, namely the Waldmeister discrimination tree (WDT) for the completed set of rewrite rules for the standard group axiomatization.

To give an impression of the importance of shrinking, Table 2 shows the number of nodes needed to index different sets of rules with and without truncation. The sets are the final ones when running the proof or completion examples stated in the left column. In the average, shrinking linear branches to leaf nodes saves about 40% of all tree nodes. As a consequence, all retrieval operations are speeded up enormously, for there are much less backtracking points, and the index switches to term–to–term operations as soon as a given query has only one candidate in the current subtree.

What are the functional requirements the indexing structure has to meet? Besides insertion and deletion of arbitrary termpairs, the index must support retrieval of

- generalizations to find matching rules and equations for reduction,
- subsuming equations for forward subsumption,

[4]This holds also true if an innermost normalization strategy is employed.

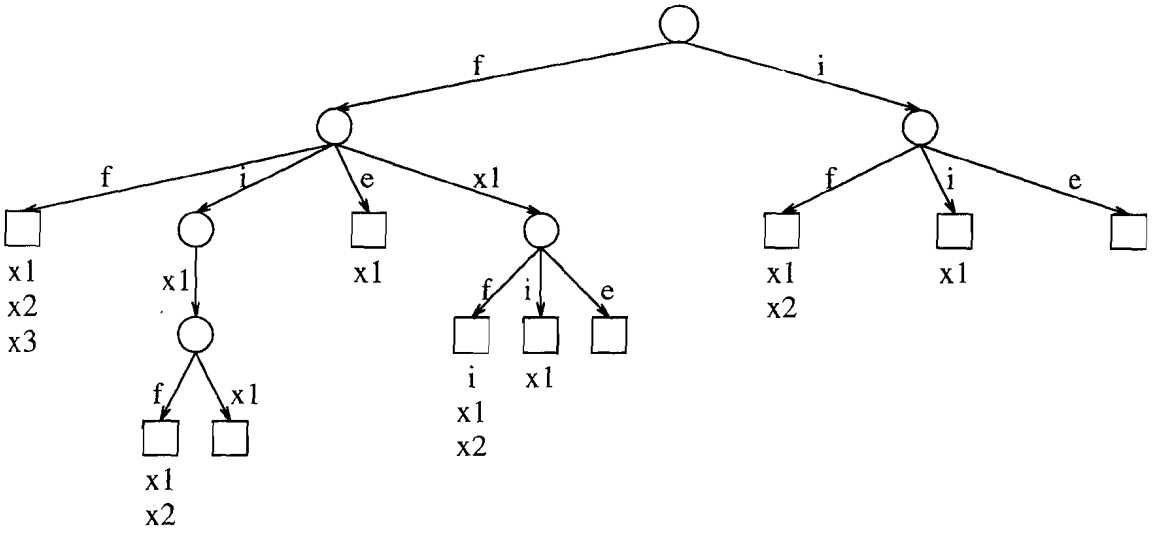


Figure 6: Waldmeister Discrimination Tree for completed group axiomatization

Example	Nodes in R		Savings
	without Truncation	with Truncation	in %
mv2.pr	1859	869	53.3
mv4.pr	1571	747	52.5
p9a.pr	1759	1379	21.6
ra2.pr	213	131	38.5
ra3.pr	284	179	37.0
Lusk5.pr	852	464	45.5
Lusk6.pr	1062	493	53.6
sims2.cp	13	7	46.2
Luka10.pr	970	448	53.8
P16lpo.cp	183	108	41.0
jar10.2.1.pr	25	11	56.0

Table 2: Shrinking linear paths

- instances for interreduction of the sets of rules and equations, and
- unifiable entries for superpositions on top level.

Furthermore, the subterms of the indexed left–hand sides are retrieved for

- instances — just like above for interreduction, and
- unifiable entries for superpositions into rules and equations.

In the context of rewriting, the indexed sets are tremendously more often queried for generalizations than they are changed by insertion or deletion of elements. This is why the speed of insertion and deletion routines is not that important here, whereas the retrieval operations — first of all retrieval of generalizations, second of unifiable entries — have to be supported to a maximum. More precisely, precalculating and storing partial results (e.g. subterms starting at a given node, which is useful for variable bindings into an index when unifying) yields speedups at the expense of only few organization effort both in terms of time and space. (Remember that most of the system's memory consumption arises from storing unselected equations; the memory

requirements of the index are neglectable.)

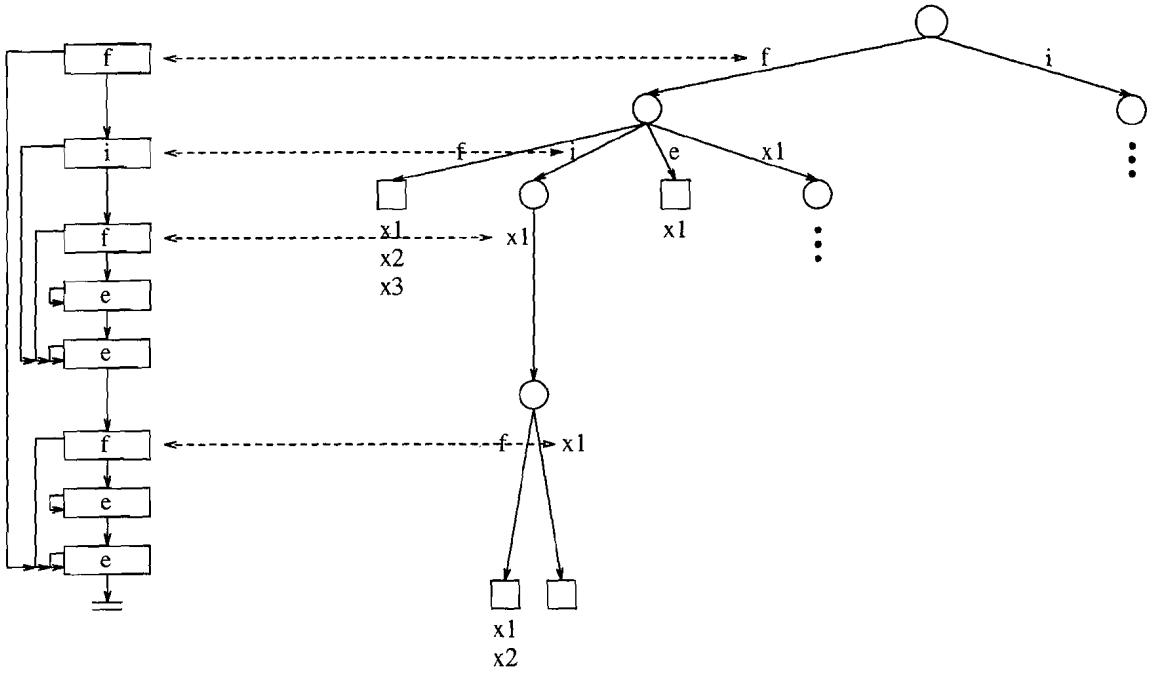


Figure 7: Matching with a WDT

Fig. 7 illustrates how the retrieval of generalizations works: Starting at the root node, one always tries to descend to a child along an edge carrying the same symbol as the top–most one of the current part of the query term, which is traversed in leftmost–outermost order. If such an edge does not exist, the variable edges have to be tried, which always succeeds in case of yet unbound variables, but may fail due to indirect clashes in the other case.[5] Backtracking is invoked when no descent is possible. Having reached a leaf node, the attached remainder and the rest of the query term are processed just as single terms.

Substitutions are represented as arrays mapping variables onto subterms of the query term so that bindings can be established and removed by one assignment each which is necessary for easy backtracking. Therefore, when unifying, the case $x \stackrel{?}{=} t$, where $x \notin Var(t)$, has to be resolved without application of the substitution $\{x \leftarrow t\}$ to the remaining subterms. Instead, whenever a bound variable is encountered, its substitute is processed.

[5]This information is passed to the backtracking routine, as is explained in Sect. 7.4.2.

4.3 Implementational Details

As an efficient data structure for the nodes of standard discrimination trees, Christian proposed an array–based hash table which maps each function symbol onto its corresponding successor node if such a node exists (cf. [Chr93]). There is an additional slot for the wildcard symbol $*$ replacing variables. The hash table especially supports descending from a parent node to a child with respect to a given symbol — keeping the children in a list would require a list traversal.

There are severals ways to apply this approach to perfect discrimination trees, which distinguish distinct variables. [Gra96] states that "since indexed terms are normalized, either a linked list of variable slots must be maintained or the hash tables have to be extended such that there is a slot for all different indicator variables x_i occuring in the set of normalized indexed terms." Fortunately, this is not necessary, for the number of variables that may occur at a given node is fixed and independent of the number of variables in the whole set. There must be one slot for x_1 inside the root node. For any arbitrary inner node, let us assume that its predecessor has variable slots for $x_1, \ldots, x_n$. If the node was reached via an edge carrying x_n, then — as the indexed terms are normalized — no other variables than $x_1, \ldots, x_n, x_{n+1}$ may appear. If the node was reached via any other edge, no more variables than in the predecessor may follow.

As function symbols and variables are represented by integers (of different sign), we uniformly use an array for accessing the successor nodes. Moreover, for every array we keep the smallest and the largest symbol to which a successor exists. This saves lookup attempts to non–existing child nodes whenever each child node has to be considered.

Every inner node also contains a so–called jump list holding the non–variable subterms starting at it together with the nodes that are reached via corresponding descent. This is useful when binding variables into the index on occasion of unifying. The entries of all those lists are collected in different subterm lists with respect to their top symbols. Without additional effort, we gain simple indexing of subterms.

An interesting observation is that the retrieval operations are sped up if the stack that stores the backtracking information is integrated into the discrimination tree. We have measured accelarations of the whole WALDMEISTER system of about 9%, which should be due to improved memory locality of the index.

Table 3 shows the speedups achieved by our indexing technique in comparison with top symbol hashing. We set WALDMEISTER against a program version where rules and equations are kept in a list which is hashed with respect to the top symbols. Special attention was paid to ensure that both versions execute exactly the same inference steps. The columns hold the following numbers: critical pairs generated; rules plus selected equations; calls of the matching routine (for both rules and equations); resulting reductions; timings for runs with resp. without using indices. The speedups range between 1.13 and 10.95. As one expects, they grow as the indexed sets become larger and more complex.

Example	Crit. Pairs	R+E	Match queries	Reductions	T(hashed)	T(indexed)
mv2.pr	404 977	484	8 516 000	756 992	1 797.8	164.2
mv4.pr	943 607	851	23 785 000	1 956 268	6 285.7	685.7
p9a.pr	60 996	1 026	4 770 000	99 556	78.4	17.9
ra2.pr	19 195	119	966 000	37 913	36.4	27.7
ra3.pr	111 989	252	8 047 000	227 063	1 030.9	690.7
Lusk5.pr	190 904	337	9 031 000	345 999	617.9	73.8
Lusk6.pr	93 587	521	5 178 000	291 538	256.8	80.1
sims2.cp	15 999	338	1 597 000	57 964	24.7	8.8
Luka10.pr	53 782	267	836 000	85 477	49.5	5.9
P16lpo.cp	117 106	3 424	11 058 000	236 149	28.7	21.2
jar10.2.1.pr	28 496	193	1 785 000	29 501	29.6	26.3

Table 3: Comparison of top symbol hashing and WDT indexing

Starting from our isolated WDT indexing routines, Fuchs has build a specialized prover DiCoDe (cf. [Fuc96a]) for problems in the domain of condensed detachement (see [Luk70]). Comparing a version with indexing to another version without, he measured substantial speedups as well.

4.4 Comparison of Indexing Techniques

In his PhD thesis [Gra96], Graf gave an overview of existing indexing techniques and also introduced some new ones. Now, what is his favourite technique for the purpose of retrieving generalizations? Graf compared the run–times of implementations taken from his ACID–toolbox (cf. [Gra94]) on average term sets, and the winner was the so–called substitution tree. In its nodes, this tree collects pairs of variables and substituted terms. Via composition, every branch from the root node to a leaf yields a substitution $\{x \leftarrow t\}$ where t can be interpreted e.g. as indexed left–hand side.

For the employment in rewriting we believe that our data structure is more appropriate. First, only perfect discrimination trees allow retrieval of subterms of indexed terms, which is useful both for the generation of critical pairs and for interreduction. Second, tree traversal is many times less costly, as descending from a parent node to a child is a quite simple operation: One has to examine the actual symbol from the query term, and the successor is found in an array indexed by function symbols and variables. This is simplified even more by the use of flatterms, which additionally support preorder term traversal. In the case of substitution trees, however, one has to consider a list of variable substitutes, which themselves are lists made up of symbols.

To justify this estimation, we isolated the Waldmeister discrimination tree (WDT) indexing and compared it with two ACID structures: perfect discrimination tree (PDT) and substitution tree (ST). The results are depicted in Table 4. The names of the indexed sets are given in the leftmost column. The sets are the same Graf used for evaluation purposes; we added four sets taken from typical proof problems.

The middle part of the table holds the number of nodes each tree needs for indexing the respective set. In comparison with PDTs, the WDTs need significantly fewer nodes (the relative savings are given in %), but without any additional expense with respect to retrieval operations. Not always do the STs contain fewer nodes than the WDTs.

Set	Size of index	Queries	Passes	Nodes				Run-times		
				PDT	WDT	Sav.	ST	PDT	WDT	ST
AVG	5001	5001	20	15581	6101	60.8	6214	16.02	10.78	22.12
BO+ +	6000	6000	44	19914	12011	39.7	9919	30.45	10.12	44.52
BO+ -	6000	6000	61	19914	12011	39.7	9919	54.86	10.13	50.37
BO- +	6000	6000	324	28436	14298	49.7	10527	116.59	10.31	52.63
BO- -	6000	6000	64	28436	14298	49.7	10527	59.49	10.12	28.10
CL+ +	1000	1000	54	18883	4960	73.7	1961	45.76	10.11	13.84
CL+ -	1000	1000	127	18883	4960	73.7	1961	159.71	10.07	8.56
CL- +	1000	1000	2673	65766	8634	86.9	1951	1125.52	10.31	37.31
CL- -	1000	1000	71	65766	8634	86.9	1951	125.28	10.16	49.10
DEEP	5001	5001	18	97192	7136	92.7	7371	41.76	10.98	25.42
EC+ +	500	500	356	3262	1228	62.4	944	56.09	10.03	32.49
EC+ -	500	500	357	3262	1228	62.4	944	189.69	10.03	32.59
EC- +	500	500	2500	16720	1515	90.9	993	260.35	10.11	32.05
EC- -	500	500	175	16720	1515	90.9	993	110.92	10.09	28.71
WIDE	5001	5001	26	210615	8250	96.1	8760	101.21	10.46	68.15
Juxta	459	106	11300	595	539	9.6	519	12577.34	9.85	44.77
Luka10	88	50000	25	353	167	52.8	150	53.42	10.50	32.23
Lusk6	50	50000	24	176	97	45.2	87	68.15	10.56	34.43
Sims2	34	25000	75	371	55	85.2	48	267.14	10.28	59.36

Table 4: Using a toolbox

However, the savings are diminishing in case of the four rewrite examples; and when retrieving, much more work has to be done at an ST node.

The rightermost part of Table 4 shows the run–times for the retrieval of generalizations. The ACID–Toolbox was instantiated with our linear term representation. All the indexing routines used free–list based memory management. The run–times are normalized: For every indexed set, we calculated a number of passes with the whole query set so that the WDT time would reach about ten seconds. Afterwards, these passes were executed within a single measurement. Otherwise, some of the run–times would have been too close to the granularity of the internal clock.

Our implementation of WDTs does not only outperform the ACID–PDTs but also the ACID–STs. The misery of the PDT implementation is due to several facts. Every query term is transformed into an internal linear representation before retrieval instead of doing this transformation "on the fly". The node format is suboptimal: For accessing successor nodes, not array–based hash tables (see above) but unsorted lists are used. Whether a successor w.r.t. a given symbol exists can only be recognized by means of list traversal. In our opinion, such an implementation cannot form a solid basis for comparative studies of different indexing schemes.

For ST indexing, conversion effort is necessary as well, but efficiently done on the fly. It is doubtful whether this effort alone is responsible for the weaker performance of STs. The more complex structure of the information kept in the nodes and edges must play an important role as well.

Obviously, using indexing toolboxes is less efficient as soon as conversion effort has to be spent. However, they save implementation effort for low–level retrieval operations. Finally, one should not employ a toolbox without careful examination of its innards, especially when aiming at high efficiency.

5 Data Structures for Providing Large Term Sets

As we have explained earlier, providing the complete set of unprocessed equations (*SUE*) causes most of the system's memory consumption while R and $E_{selected}$ grow. Before discussing data structures for a high density storage of this special set let us recall the statements we made in Sect. 3:

- Deletion of unselected equations with non–persistent parents (orphan–murder) is necessary for keeping the set small. For this reason, information on the parents of each equation in *SUE* must be available in order to enable an easy detection of outdated ones. As we will see later, keeping *SUE* absolutely free of orphans will lead to increased management overhead — either in terms of time or space.

- The selection strategy induces an ordering on the stored equations. Since every selection only retrieves the minimal element, the set should be realized as a priority queue. Only slight changes appear between two successive invocations of select, and thus pre–ordering should be maintained on the set.

- Storing preprocessing information like the generated (and usually normalized) terms, their length or their depth, reduces the effort spent when comparing two elements. Involving the terms in each comparison would still be quite expensive, and hence a mapping should take place that replaces the comparison of equations by comparison of simple integer weights.

Now let us state the resulting functional and non–functional requirements for the system component that will realize the set *SUE*. The **functional requirements** are the following:

- Store termpairs together with an integer weight entry and parent information.
- If equal weights are encountered, employ a parameterized strategy.
- Return termpairs with smallest weight and persistent parents.
- Support "Walk–Throughs" for *IRP* (intermediate reprocessing).

We have to minimize memory consumption, and hence we focus on this aspect as the major **non–functional requirement** and thus reduce

- the number of termpairs with non–persistent parents ("orphans") held in the set,
- the size of the given termpairs, and
- the memory consumed by the set management itself for organization purposes.

Considering all that we find that a newly generated and already preprocessed equation — given as two flatterms and additional information like parent rules/equations, position — will pass the following stages:

1. Classification: Maps the valuableness of the equation (given as a pair of flatterms and its parents) to a weight entry. This entry should suffice for comparing two stored equations according to the search strategy.

2. Transformation: Transforms the flatterms into a termpair in a certain representation. This termpair must suffice for restoring the flatterms from the parent rules/equations and the pair itself, since only the parents must be stored along with the unselected equations any time.

3. Storage: Stores the termpair along with its parents and the weight, and holds responsible for "orphan–murder". In order to minimize the time for retrieval of the lightest entry the storage should maintain a (partial) ordering on the entries.

In the following we introduce data structures for the latter two stages which are responsible for the space consumption of *SUE*.

5.1 Termpair Representation

Reducing the size of the given termpairs by changing their representation from flatterms to one of higher density has to be paid with transformation effort whenever equations are stored respectively retrieved from the storage. This effort obviously increases even more when an intermediate normalization of all the unselected equations is invoked (intermediate re–processing, *IRP*) as it requires restoring the flatterm representation, processing, and transformation into the high density representation again. Additional effort that would arise whenever the comparison of two equations involves the flatterms has been avoided by mapping the valuableness of equations to integer weight entries before inserting them into the set of unselected equations.

We employ termpair representations of higher density mainly for problems that would run out of memory when storing flatterms. Surprisingly we even gain speedups for smaller problems due to less memory allocation effort — and also due to increased locality.

Changing the representation of termpairs from flatterms to **stringterms** where every symbol takes only one byte cuts down the size of termpairs to about 10%. Of course we store left and right hand side of termpairs in a single string in order to reduce the amount of memory wasted due to alignment.

Furthermore, we evolved a minimal representation of constant size storing no terms at all but superposition information to recompute them (**implicit representation**). For this kind of representation only one single integer number for each unselected equation must be stored. (Remember that the parent information is stored anyway). This number corresponds to the position in the rule/equation the superposition was found inside. As no terms will be stored, this representation requires re–doing all simplification steps so far executed on the terms. For that reason, problems requiring *IRP* should be tackled with stringterms as they save that additional amount of normalization. Representing unselected equations implicitly comes close to a single stepping procedure as only the superposition itself will be stored. The difference to single stepping is that the termpairs will be generated as soon as possible, processed (i.e. normalized to a certain amount), and analyzed by the weighting function before they are stored as superpositions only.

Fig. 8 illustrates the three different ways terms may be represented by WALDMEISTER. For the equation $f(x_1, f(e, x_2)) = f(x_1, x_2)$, the flatterm representation requires $8 \cdot 3 = 24$ words $= 96$ bytes, the corresponding stringterm takes 8 bytes, and the implicit representation is restricted to 1 word $= 4$ bytes only. This is independent of the length of the terms!

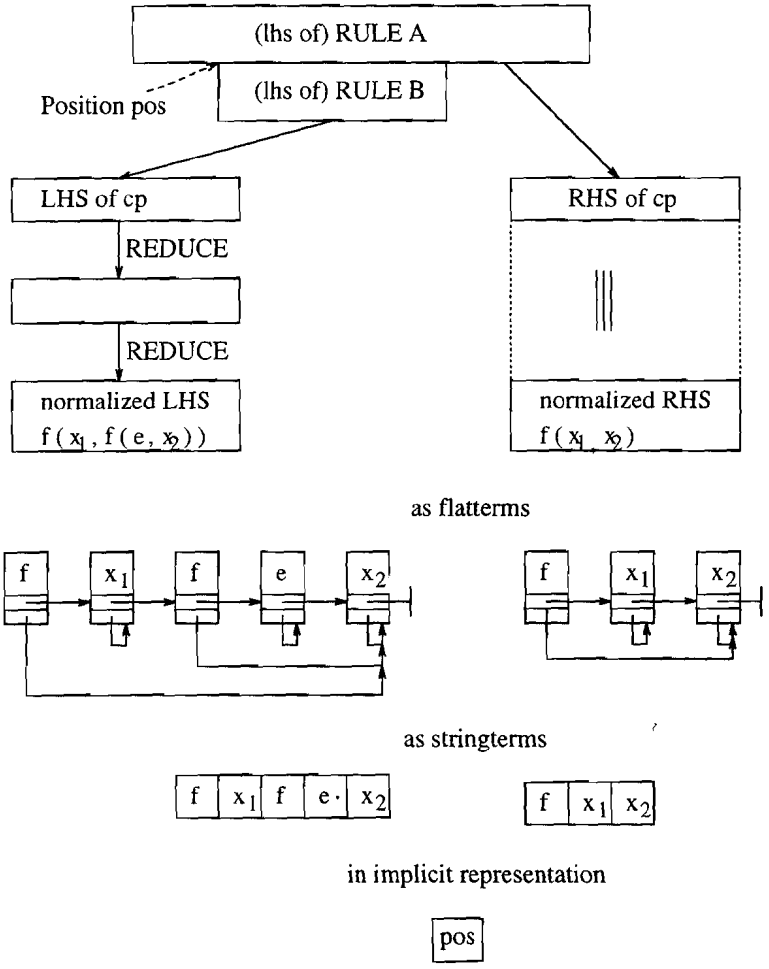


Figure 8: Three ways to store $f(x_1, f(e, x_2)) = f(x_1, x_2)$

5.2 Storage: Priority Queue

An ideal data structure for realizing the priority queue would

- contain no orphans,
- cause no additional management overhead,
- hold — for deleted equations — no left-behind management overhead,
- consume no time for organizing, and
- allow easy reweighting of stored elements, especially during *IRP*.

In the following, we introduce three different data structures and compare them with respect to these five aspects. The figures we present show the status of the data structures after the insertion of seven critical pairs [6] (generated with the rules $1, 2, 3, 4$) and after the deletion of rule 3. The pairs are (1,1), (2,1), (2,2), (3,1), (3,2), (4,1), and (4,3), where (x,y) means that the critical pair has been generated with a new rule x and rule y. Deletion of rule 3 makes the critical pairs (3,1), (3,2) and (4,3) orphans which will be marked with $+$.

5.2.1 Binary Heap

Arbitrary insertion and retrieval of the minimal element is provided by the binary heap structure (cf. Fig. 9), *heap* for short. Representing it as an array (as in *heapsort*) holding the weight, the termpair and its parents obviously minimizes the management overhead but induces major problems: orphan equations cannot be deleted easily unless they have reached the heap's top position. Unfortunately this implies either large quantities of orphans held in the set or time–consuming search for and deletion of orphans, followed by an even more time–consuming reorganization of the whole heap. Another problem when storing unselected equations in a heap arises when the set grows larger and larger: the array–based heap requires a large consecutive area of memory (cf. Sect. 6).

Fig. 9 shows that the critical pair (3,2) is the only orphan that can be deleted immediately as it is on the top position. The orphans (3,1) and (4,3) will remain in the heap.

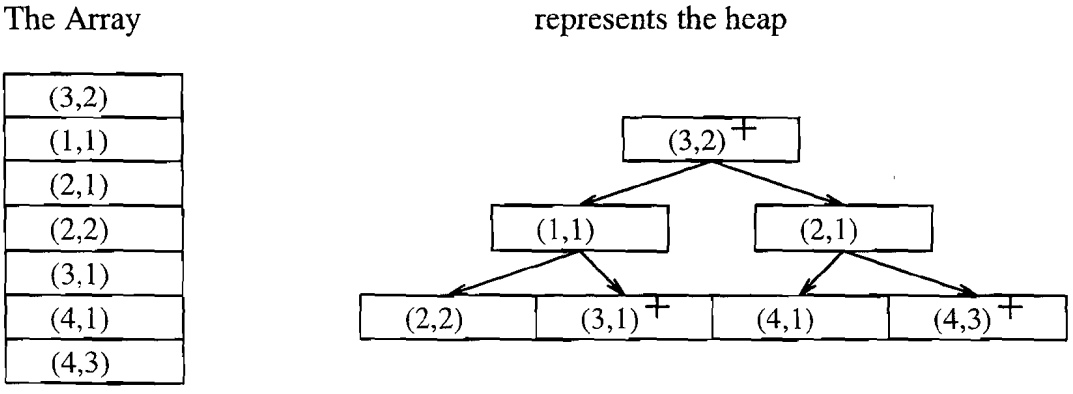


Figure 9: An array–based heap

[6] From now on, the term "critical pair" shall be applied to equations from the stage of generation until the very moment they are turned into a member of $R \cup E_{seleceted}$.

5.2.2 Pairing–Heap

In a first approach, the reduction of reorganization effort caused by orphan murder led us to a related dynamic data structure, the so–called pairing–heap (cf. Fig. 10) which was introduced in [FSST86] (see [SV87] for an evalutation). Double–chaining its entries allows arbitrary deletion and reweighting of elements in constant time. For the purpose of immediate orphan murder, the unprocessed equations are additionally double–chained in two lists each with respect to their parents. Whenever a rule or equation is invalidated, the descending equations are deleted in linear time. Clearly, this approach reduces the number of orphans held in the set to the absolute minimum of zero.

Fig. 10 illustrates how the additional double–chaining allows to delete all the three orphans from the set as soon as rule 3 has been removed. Note the overhead for double–chaining both in the heap itself and twice with respect to the parent rules/equations—which is at least six pointers for each unselected equation. This is not significant if termpairs are stored as flatterms, but quite annoying if they are stored as stringterms (70% management overhead for each termpair in the average). A pairing–heap behaves even worse if implicit termpairs must be stored: The relation between stored information and management overhead in this case is 1:3.

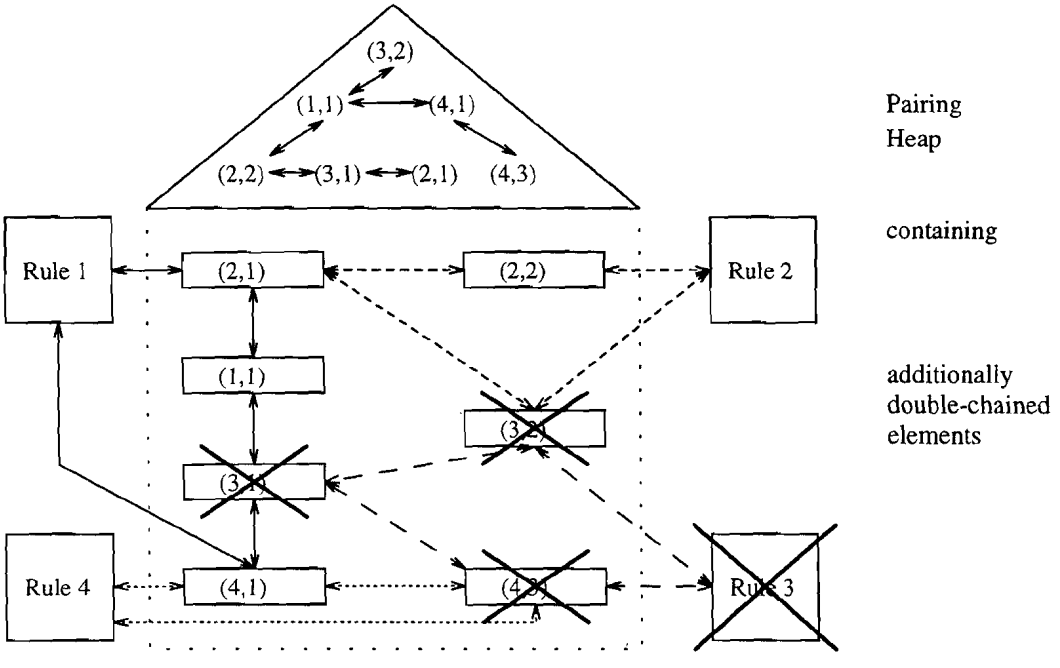


Figure 10: Extended pairing heap

5.2.3 Heap of Heaps

Having developed data structures suitable for lazy (heap) and immediate (pairing–heap) orphan murder, we evolved a two–level data structure, the heap of heaps (cf. Fig. 11) as a compromise. Its lower level consists of array–based heaps. Each of them holds the initial critical pairs of a new rule or equation, that are all the superpositions with itself and with the elder rules and equations that could not be erased due to triviality, joinability, subsumption, or criteria. The upper level is an array–based heap again and contains references to the subheaps, each of them weighted via its top element. The minimal equation is found at the top of the lightest subheap. As soon as a rule or equation has been invalidated its initial critical pairs can be deleted easily. Measurements show that between 70% and 80% of all possible deletions are executed — almost without any additional management effort: Compared to a pairing–heap, we have some orphans left in the set, but in the end save a lot more space due to the minimized management overhead.

Set reduction on this structure will cause reorganization effort, but measurements show that only very few yet unprocessed equations will be deleted or reweighted during walk–throughs, and thus this effort should be neglectable. On the occasion of such an *IRP* walk–through even the remaining orphans can be erased.

The heap of heaps gives yet another advantage compared with the simple array–based heap: it does not require large, consecutive areas of memory. If combined with a specialized memory management (**Memory Beams**, cf. Sect. 6), WALDMEISTER processes do no longer grow monotonously but get smaller whenever subheaps are erased.

Fig. 11 shows that the initial critical pairs of Rule 3, $(3,2)$ and $(3,1)$, could be erased immediately. Only $(4,3)$ will remain in the set until it has reached the top position. The overhead entry of Rule 3 will be removed from the top–level heap as soon as the next selection operation takes place. Note that a critical pair (x,y) can be represented as (y) only for it is located in the subheap belonging to rule x. This saves even more space.

5.2.4 Results

Table 5 shows for the three problems (Lusk6, ra3, and mv4; see App. B) how the different data structures we employ for storing the *SUE* behave on different sizes of *SUE*. All runs employ stringterms rather than flatterms, and special attention was paid to ensure that all three runs execute the same inference steps. The table shows a snapshot of the moment the proof was found.

In the first part of the table one will find information on the run: the number of critical pairs generated, the number of rules and equations encountered, the number of match queries and resulting reductions. Apart from that, we find the average number of critical pairs computed with a new rule resp. equation (during the whole run), and the total number of unselected equations that have become orphans as their parents

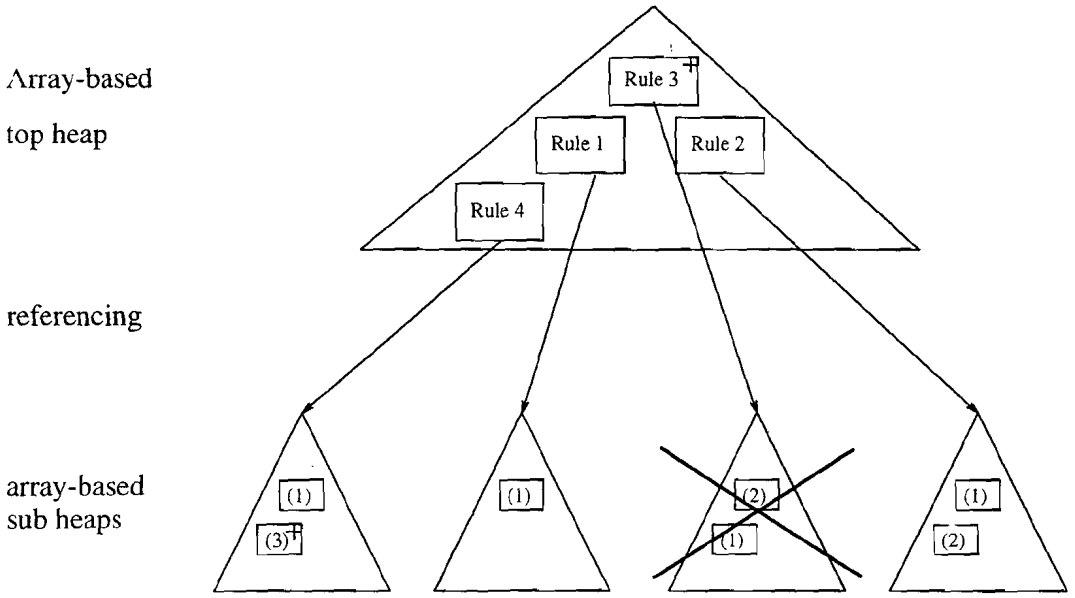


Figure 11: Heap of heaps

were removed from R or $E_{selected}$.

The following parts of the table state for each of the three structures the information characterizing their behaviour: This is the maximum size of *SUE*, the size of *SUE* when the proof was found, the percentage of orphans in the current set, the percentage of orphans removed during the run, and the total amount of memory consumed during the run. Furthermore, for heap and heap of heaps, there is some information on the behaviour of the underlying memory beams (see Sect. 6). This information is not stated for the pairing–heap as this is a dynamic structure and hence does not use memory beams.

The table confirms our considerations from above. We find that the pairing–heap wastes too much memory for management purposes — although it holds no orphans at all. The regular heap keeps the system smaller than the heap of heaps for Lusk6. The reason for this is the wasted memory at the end of each subheap (cf. Sect. 6) and the quite small number of elements in each single subheap. However, the larger the examples grow, the better works the partial orphan murder as applied by the heap of heaps: about 75 % of all orphans have been removed!

Table 6 gives an overview of the advantages and disadvantages of the three data structures. The best technique is marked by ++, the worst one by −−. Good, bad and average performance are denoted by +, −, and ○, respectively. We have added the standard heap with additional orphan murder (executed every time a removed rule or equation would leave orphans in the set). Summing up, we believe that the standard heap will suffice for small runs, whereas the heap of heaps clearly outperforms the other data structures as the runs grow larger.

Example	Lusk6	ra3	mv4
Crit. pairs	79 306	114 313	2 024 100
Rules + equs.	426	268	1 264
Match queries	4 495 438	7 942 487	53 815 644
Reductions	231 028	214 272	4 483 077
Avg. initial cps (R)	107	223	408
Avg. initial cps (E)	288	736	1 194
Total number of orphans	20 763	35 466	310 129
Heap			
Max. set size	50 789	92 418	483 777
Final set size	50 789	92 415	483 424
Orphans in %	39.5	37.9	56.1
Orphan hit rate in %	3.6	1.2	12.5
Total mapped memory (KB)	1 024	1 884	9 708
Max. mapped memory (KB)	1 024	1 884	9 708
Total memory consumption (KB)	5 926	8 202	24 724
Pairing heap			
Max. set size	42 925	76 209	303 341
Final set size	30 765	57 365	212 027
Orphans in %	0.0	0.0	0.0
Orphan hit rate in %	100.0	100.0	100.0
Total memory consumption (KB)	6 950	10 038	27 291
Heap of heaps			
Max. set size	44 461	76 611	303 934
Final set size	35 848	65 012	284 837
Orphans in %	14.2	17.7	25.6
Orphan hit rate in %	75.5	78.4	76.5
Total mapped memory (KB)	1 966	2 171	11 387
Max. mapped memory (KB)	1 356	1 757	6 754
Total memory consumption (KB)	6 094	7 831	19 172

Table 5: Comparison of the three data structures

Data structure	Heap with orphan m.	Heap without orphan m.	Pairing heap	Heap of heaps
Management overhd.	++	++	−−	++
Orphans held	++	−−	++	+
left-behind overhead for deleted parents	++	+	++	+
Time spent for orphan murder	−−		○	++

Table 6: Summary of the different data structures for *SUE*

6 Data Structures for Efficient Memory Management

Deduction systems heavily rely on dynamic data structures since they permanently create new facts and discard unnecessary ones. Consequently, dynamic memory allocation is necessary. In a UNIX environment, using an imperative programming language such as C, this can be left up to the run–time system employing `malloc`, `realloc` and `free`, or supported by data structures that fit better for the given task. In the following we describe the well known free–list based memory management, and introduce memory beams for efficient handling of large, growing arrays.

6.1 Memory Management Based on Free–Lists

Given an arbitrary deduction system, most of its memory resources will be consumed for enormous numbers of tiny objects, most of them having equal size. For every dynamic object such as a single term cell, one could allocate memory by a separate call of the run–time system via `malloc`. A more efficient and well–known technique is based on free–lists (cf. [Gra96]): For every object size, a memory block for e.g. thousand objects is allocated and inscribed a singly–linked list of elements of that size. Creating and discarding objects thereby is reduced to simple list operations.

Example	Time(`malloc`)	Time (free–lists)	Size (`malloc`)	Size (free–lists)
mv2.pr	296.9	59.8 %	38188	53.4 %
Lusk6.pr	113.7	58.6 %	24126	52.6 %
Luka10.pr	20.7	58.5 %	2994	71.9 %
jar10.2.1.pr	23.5	39.6 %	5114	56.7 %
mv4.pr	3001.2	18.9 %	99366	52.4 %
sims2.cp	68.2	59.1 %	15550	54.5 %
ra3.pr	160.2	45.5 %	19418	53.9 %

Table 7: Memory management: `malloc` vs. free–lists

As an open architecture allows to exchange software components, we are able to name the savings earned using this kind of management. The run–times are cut by half. The same applies to the memory consumption, as the run–time system stores a header entry for every piece of memory dynamically allocated. Table 7 states the timings and process sizes of runs with and without the free–list based memory management. The process sizes are given in memory pages (4,096 bytes for SUNOS).

6.2 A Closer Look: Locality Increased

For a precise analysis of the distribution of memory references over the employed address space in the course of time, the following experiment was performed — once with free–list based memory management and once with a `malloc`–based one.

Instead of continuos real time, discrete time slices of 500 milliseconds were considered. For each time slice, the number of memory pages referenced during that slice

(i.e. the size of the working–set) was measured while WALDMEISTER was solving the problems Lusk6.pr respectively mv2.pr (cf. App. B). Fig. 12 contains the graphs of both experiments, but with relative timings for easier comparison. (Remember that the application of `malloc` does not only consume more space but also more time.)

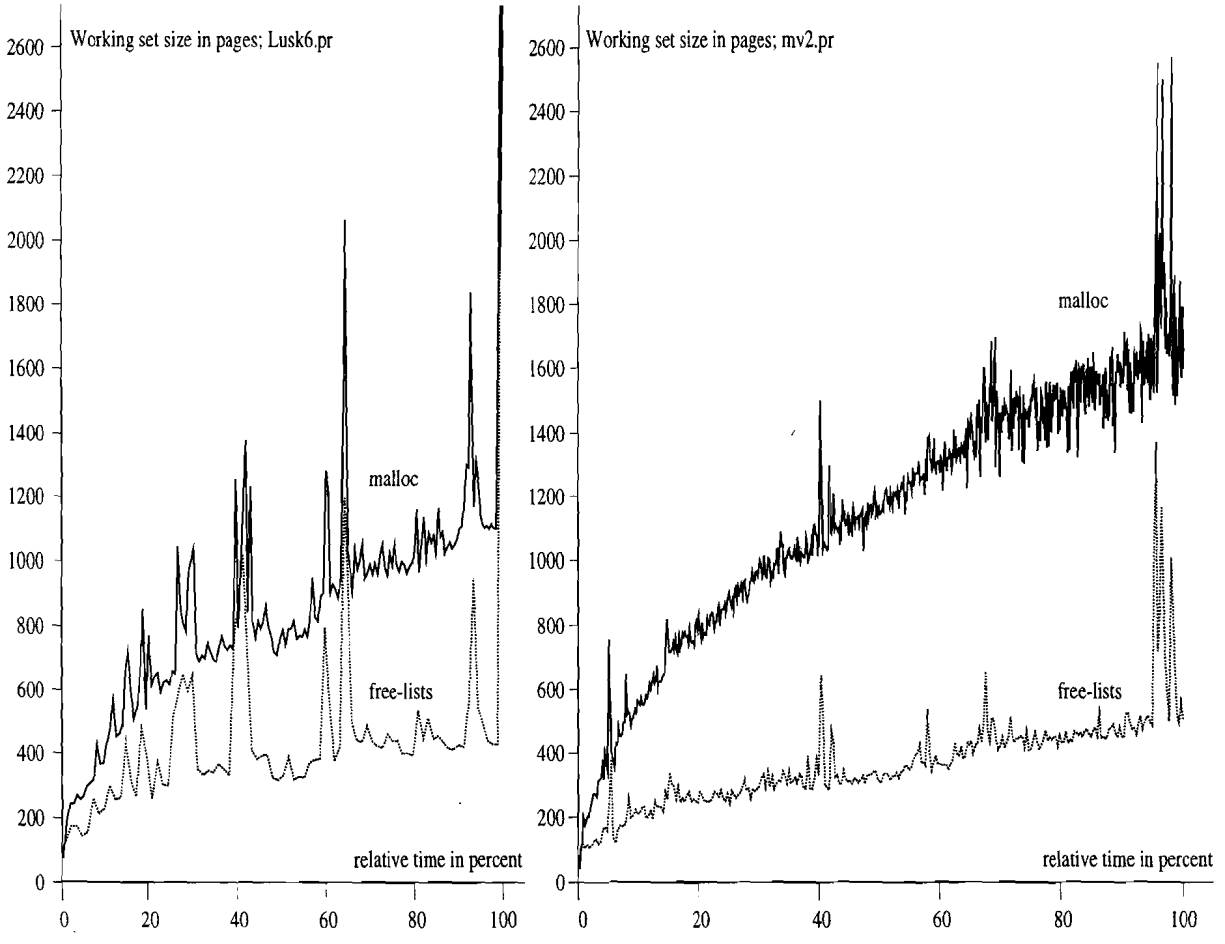


Figure 12: Experiment: comparison of size of working–sets

The peaks occuring in the graphs are due to removing lots of orphans from *SUE* at once after some rules and equations have been removed during interreduction.[7] This "up and down" behaviour is typical for most proof tasks.

Apparently, the use of free–list based memory management reduces the size of the working–set by more than one half, that is, the locality of the system is increased significantly. The difference of the run–times is partially due to this phenomenom. Another interesting observation is that the larger the proof tasks grow, the more the difference in locality increases: The `malloc` version starts paging much earlier.

[7]We used page protection mechanisms for these measurements, and for that reason we had to employ a pairing heap for the *SUE* as the other data structures require the same protection mechanism for the underlying **memory beams**.

6.3 Memory Beams

We have outlined in Sect. 5 that the system component which by far consumes the most amount of memory is the set of unselected equations. For an efficient realization of heaps — which is necessary both for the binary heap and also for the heap of heaps — expandable dynamic arrays are in demand. They may hold hundreds of thousands of entries. As for the heap of heaps, some of the arrays will have to be discarded completely.

In a conventional solution, each array would be allocated via a call to `malloc`, and its actual size kept in some variable n. For a write attempt at position i, it has to be checked whether this fits into the actual array range, which i and n are compared for. If $i > n$, the array is expanded calling `realloc`. This routine will look for a free, consecutive area of memory of the required size and copy the contents of the formerly used memory to the beginning of the new, larger one.[8] The now unnecessary memory is only marked free, but not returned to the operating system; it remains with the process. The longer the system is running, the harder it gets to find appropriate memory areas, whereas the formerly used pieces — scattered all over the address space — cannot be re–employed, since they are too small. Thus, a deplorable memory fragmentation is produced.

Memory beams (cf. Fig. 13) avoid both the range checking and the copying operation. This is achieved by modifying the virtual address mapping. Even more, the need for large consecutive areas of memory is reduced to consecutive areas in the virtual address space. This is why no fragmentation is encountered.

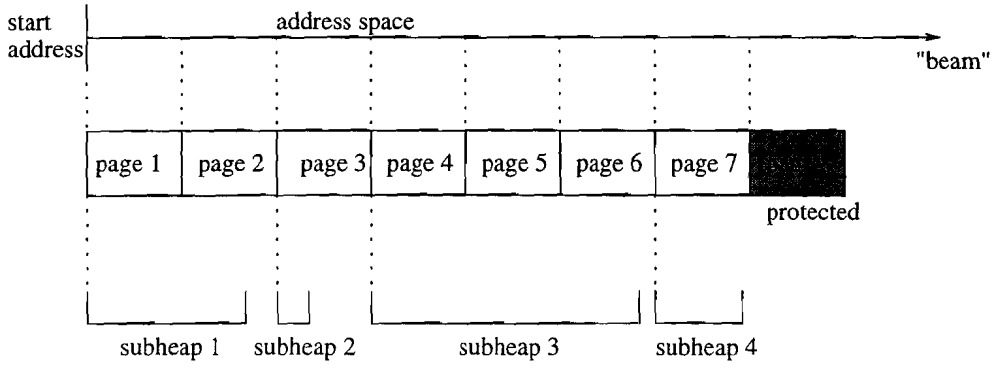


Figure 13: Memory beam before deletion of subheap

Now, how does it work? First of all, one has to choose an area from the virtual address space that will not be used by `malloc`. This start address marks the beginning of a new beam. The following e.g. 32 Kbytes of the address space will be connected to e.g. 8 pages of real machine memory, each holding 4 K (SUNOS system page size), by an `mmap` system call. The last page is read– and write–protected. Consequently, every

[8] In our case, the overall requests for dynamic memory are too frequent and the arrays usually too large to allow expanding them in the same place.

R/W–operation to an address in the protected area will cause the operating system to send a special signal (segmentation violation, SIGSEGV) to the running process. A corresponding signal handler unprotects that page, maps another 8 pages to the following 32 K of addresses, protects the last of the new pages, and returns to the running process.

Another striking advantage of beams becomes obvious when a heap of heaps is used to store the unselected equations. As only the last of all the array–based subheaps is subject to insertion, all of them can be stored in a single beam (cf. Fig. 13). Since every subheap might become a victim of orphan murder, we let the corresponding arrays start at page addresses only. This causes some wasted memory at the end of each array, but allows to delete that very subheap simply by unmapping its memory pages (cf. Fig. 14).[9] These pages are returned to the operating system and can be mapped to the end of the beam again, or used for something completely different: Orphan murder has reduced the process size! Summing up, memory beams consume consecutive address space only, whereas `malloc` consumes consecutive memory.

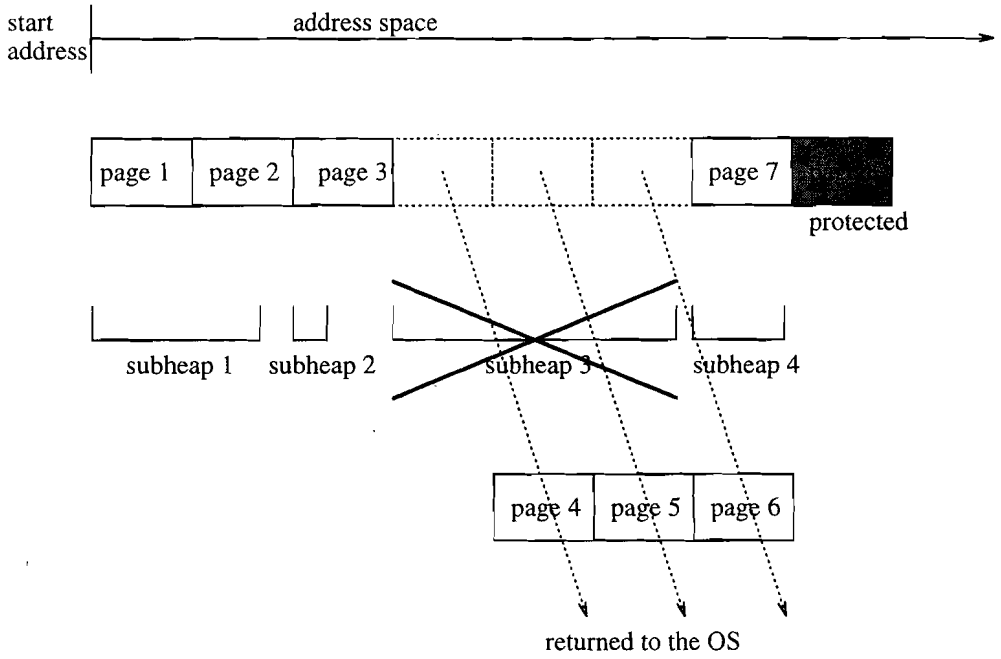


Figure 14: Beam memory after deletion of subheap

[9] As stated in Sect. 5, subheaps usually employ several pages of memory, and thus the wasted part is neglectable. See also Table 5.

7 Looking for an Appropriate Parameter Setting

Our design approach has produced an open system architecture, and thus a set of parameters corresponding to the choice points intentionally left open. What we now have to do is to evaluate the different possible settings for the single choice points in order to identify the best one — or at least to name tendencies if a general statement cannot be made.

We have evaluated these open points via measurements on a set of problems that can be found in App. B. We ran our experiments on Sun SPARCstations 20/712, 196 Mbytes main memory and 670 Mbytes swap space, with Solaris 2.4 as operating system. The timings we state throughout this report are the minimum of at least six identical runs on the same machine. The meaning of the columns in all the tables we present in this section can be found in Table 8.

Crit. Pairs	Total number of critical pairs generated
R + E	Number of rules and (selected) equations produced during the run
Match Queries	Number of calls to the matching routine with both the sets of rules and selected equations
Reductions	Number of resulting reductions
Time	Total process plus system time in seconds. Runs were usually stopped after 1000 seconds. In these cases, the tables state '> 1000'. If the system ran out of memory, this is stated without the time that elapsed up to then.

Table 8: Structure of the tables

As a starting point, we fixed the relevant parameters of the system following what we consider common sense:

- reduction of critical pairs on generation (preprocessing) and selection with rules and equations plus subsumption,
- application of the subconnectedness–criterion with rules and equations,
- weighting of critical pairs by the add–weight heuristic ($weight(l, r) = length(l) + length(r)$) with additional preference for older critical pairs when identical weights are met,
- computation of normal forms with the leftmost–outermost strategy,
- no additional reduction of the SUE during the run (IRP), and
- no early orphan murder.

The data–structure used to hold the critical pair set is an ordinary heap, the terms will be stored as flatterms. This parameter setting is what we expected to be appropriate for most problems, as every effort (apart from early orphan murder) is made to keep the sets R, $E_{selected}$ and SUE small.

Please keep in mind that the results of our experiments are strongly connected with the selection–strategy for unselected equations (add–weight, in our case). Using other

strategies may lead to different results but we believe that the relationship between the various parameters will remain worth an examination if another strategy is employed. Table 9 illustrates the performance of this parameter setting on the set of examples.

Example	Crit. pairs	R + E	Match Queries	Reductions	Time
P16kbo.cp	215 540	3 796	18 446 415	401 632	33.3
P16lpo.cp	82 639	2 306	9 826 527	100 579	16.2
sims2.cp	55 065	576	7 146 127	250 496	46.7
z22.cp	3 005	166	131 766	6 451	0.7
mv2.pr	1 077 750	750	22 201 391	1 993 321	417.0
mv4.pr	1 597 746	1 045	38 053 337	3 141 228	> 1000
p9a.pr	34 771	684	2 262 609	54 883	10.3
ra2.pr	12 094	93	610 731	22 342	18.3
ra3.pr	125 966	260	9 415 150	234 242	758.1
Lusk4.pr	1 659	68	52 589	3 742	0.3
Lusk5.pr	14 912	101	662 764	27 375	5.5
Lusk6.pr	80 274	430	4 604 075	238 648	85.5
Luka10.pr	137 709	394	2 118 209	238 826	16.1
jar10.2.1.pr	143 147	349	8 330 082	140 210	303.8
jar10.3.1a.pr	5 610	104	208 200	4 242	2.3
jar10.3.7b.pr	863	33	21 097	675	0.3

Table 9: Performance of the common sense defaults

7.1 Sensitive Heuristics

A search strategy realized as a weighting function on critical pairs in most cases is not an injective one, and thus will compute identical weights for different critical pairs, and so does the one we chose for our experiments. For that reason it usually is left up to the implementation which one out of the equally weighted equations will be treated first. Strategies like add–weight or max–weight can be heavily disturbed by manipulating the order of treatment of critical pairs with identical weight as Table 10 will show.

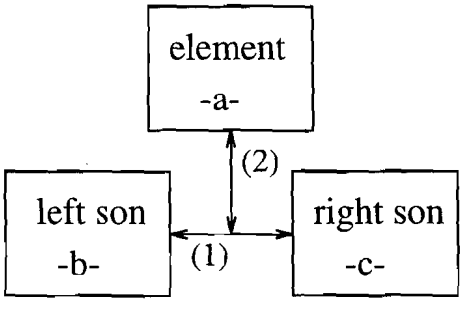


Figure 15: Sinking in elements into a heap

Employing a binary heap as the priority queue for storing the unselected equations requires comparison of elements each time an element sink in. Take a look at Fig. 15 where element a sinks in. On this occasion, the successors of a, namely b and c, will be compared (1), and afterwards the greater one will be compared with a (2). Hence

two comparisons take place at each invocation of the "sinking" procedure, and both can encounter equally weighted entries. If $b = c$ holds true either b or c can be selected (w.l.o.g. b), and if $a = b$ holds true, either a or b can be selected. Combined, we get four different realizations induced by the data structure (e–1, ..., e–4), where each prefers either the left or the right son if comparison (1) reveals equal weights, and each prefers either a or its lightest son if comparison (2) does so. Furthermore, we have employed two extended weightings discriminating older or younger elements. Finally, WALDMEISTER supports random preference for equally weighted entries.

Strategy	Crit. pairs	R+E	Match Queries	Reductions	Time
mv2.pr					
e–1	673 800	631	13 093 451	1 246 328	255.9
e–2	599 420	584	11 158 288	1 109 691	202.0
e–3	771 513	667	15 568 479	1 417 899	297.3
e–4	922 572	708	18 394 158	1 691 035	353.2
random	651 077	608	12 275 159	1 203 072	283.1
random	1 017 794	750	20 831 304	1 881 970	404.6
older	874 723	698	16 536 665	1 619 330	315.9
younger	1 077 750	750	22 201 391	1 993 321	428.9
mv4.pr					
e–1	1 310 181	1 055	31 994 110	2 817 654	> 1000
e–2	1 619 661	1 021	37 200 593	3 096 066	> 1000
e–3	1 297 528	851	28 874 996	2 374 808	566.0
e–4	1 659 366	1 069	39 102 156	3 257 928	> 1000
random	1 178 202	808	25 153 545	2 174 911	505.4
random	1 521 019	1 058	36 132 266	2 991 955	953.2
random	1 658 261	1 078	39 651 348	3 288 453	> 1000
older	1 676 224	1 095	39 521 850	3 311 045	> 1000
younger	1 594 274	1 044	37 955 293	3 128 657	> 1000
p9a.pr					
e–1	74 263	1 127	5 928 210	122 747	26.7
e–2	40 022	813	2 911 509	55 110	12.1
e–3	69 475	1 059	5 018 683	113 587	24.7
e–4	66 658	1 038	5 027 034	107 701	23.1
random	31 148	684	2 212 833	52 045	9.7
random	89 001	1 191	7 039 649	154 353	37.2
older	85 058	1 269	7 343 201	144 024	26.3
younger	34 771	684	2 262 609	54 883	10.7
ra2.pr					
e–1	21 929	110	1 168 321	43 772	61.8
e–2	14 471	104	824 489	32 788	30.5
e–3	16 855	104	899 317	32 962	36.2
e–4	17 618	112	947 835	35 061	30.1
random	12 690	100	644 857	24 198	19.7
random	23 671	126	1 314 962	50 467	68.6
older	11 319	96	536 789	22 188	18.0
younger	12 094	93	610 731	22 342	18.6
ra3.pr					
e–1	105 025	251	7 203 359	201 038	721.6
e–2	124 182	267	8 588 019	240 669	802.0
e–3	99 647	239	6 857 153	185 457	674.6
e–4	94 204	231	6 297 949	174 701	651.8
random	97 290	242	6 624 028	188 598	666.6
random	153 858	295	11 424 882	313 028	948.6
older	127 120	298	10 352 681	297 005	926.3
younger	125 966	260	9 415 150	234 242	763.2
Lusk6.pr					
e–1	73 466	447	4 127 077	223 408	73.8

Strategy	Crit. pairs	R+E	Match-Queries	Reductions	Time
e-2	142 942	574	8 188 478	476 236	162.8
e-3	80 243	465	4 628 095	258 104	81.1
e-4	169 991	594	9 822 669	627 422	183.7
random	82 171	518	4 478 900	252 505	74.3
random	186 316	616	11 445 532	634 950	256.5
older	151 766	674	8 527 915	531 696	121.5
younger	80 274	430	4 604 075	238 648	87.0
Luka10 pr					
e-1	415 246	713	6 689 689	660 586	54.0
e-2	28 254	178	378 013	44 707	2.7
e-3	33 087	210	455 981	51 492	3.1
e-4	319 240	656	5 136 898	514 031	39.8
random	16 436	147	213 302	24 284	1.7
random	386 868	720	6 314 472	625 949	50.4
older	177 115	488	2 987 170	299 892	22.4
younger	137 709	394	2 118 209	238 826	16.1
P16kbo.cp					
e-1	14 247	1 388	2 638 320	26 373	4.0
e-2	14 200	1 381	2 797 239	26 230	4.2
e-3	14 175	1 382	2 505 578	26 906	3.8
e-4	12 357	1 336	2 631 623	22 319	4.1
random	11 444	1 322	2 472 901	20 555	3.8
random	24 688	1 582	3 452 806	47 540	5.5
older	6 731	1 202	2 155 067	11 095	3.1
younger	215 540	3 796	18 446 415	401 632	33.4
P16lpo.cp					
e-1	14 560	1 399	2 620 447	29 629	4.0
e-2	12 360	1 340	2 544 128	22 048	3.8
e-3	8 013	1 236	2 159 648	13 630	3.2
e-4	14 189	1 391	2 753 364	26 369	4.2
random	11 974	1 334	2 480 486	22 751	3.8
random	22 783	1 571	3 319 154	45 690	5.7
older	80 667	2 753	6 823 046	186 789	12.3
younger	82 639	2 306	9 826 527	100 579	16.2
jar10.2.1.pr					
e-1	219 841	322	13 286 077	265 661	> 1000
e-2	208 353	297	12 749 373	237 268	> 1000
e-3	221 331	318	13 517 635	263 526	> 1000
e-4	93 178	309	5 283 153	87 643	151.6
random	117 836	331	7 048 228	118 606	216.3
random	220 858	317	13 611 061	266 785	> 1000
older	106 308	322	6 314 557	103 820	185.8
younger	143 147	349	8 330 082	140 210	304.7
jar10.3.1a.pr					
e-1	10 486	158	372 918	13 016	5.0
e-2	16 189	206	583 104	20 437	8.0
e-3	5 907	105	219 462	5 174	2.6
e-4	5 687	104	207 823	5 034	2.4
random	5 965	115	221 867	5 217	2.4
random	16 191	208	587 238	20 985	8.2
older	16 767	210	613 801	22 184	8.1
younger	5 610	104	208 200	4 242	2.4

Table 10: Variations of the add–weight heuristic

Table 10 shows the unexpected influence of these parameters on the runs. The times differ by factors between 2 (mv2), 10 (P16kbo), 32 (Luka10) and even can decide between success and timeout (mv4). The reason for this is that the weight distribution of add–weight produces a concentration of thousands of critical pairs on only few different values. Fig. 16 shows the distribution of weights produced by the add-weight

heuristic at the end of the run of problem Lusk6. The x–axis holds the weights, the y–axis the number of cps with the given weight. The total number of cps here was 98,128, the maximum number of cps with the same weight was 12,498 (with weight 24).

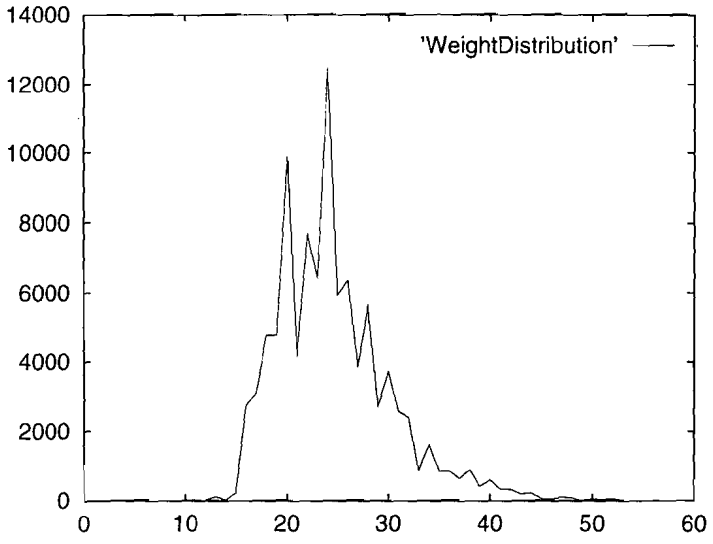


Figure 16: A weight distribution of add–weight

Result: Heuristics like add–weight which produce a large number of critical pairs with the same weight hold indeterminisms that can completely blur the effects gained by changing other system parameters. This is an inherent problem of rough granularity heuristics and should not be left up to the implementation but avoided on the level of the heuristics. For that reason we ran all the other experiments in this report with an extended add–weight heuristic prefering the older ones among the unselected equations.

7.2 Treatment of Critical Pairs

In this part we will try to figure out how the treatment of critical pairs should be realized. Critical pairs will be preprocessed after their generation and normalized again on selection. Furthermore they might be treated additionally at any time the power of the rewrite relation has been increased by introducing new rules or equations. Now the question is not only *when* the critical pairs should be treated but also *how*. This means that the computation of normal forms might not take place at all, use only rules, or both rules and equations. Furthermore subsumption might be invoked or not. The tables in this part hold a strategy column that describes the treatment of equations: the letters 'r' and 'e' indicate the application of rules resp. equations by the normalization procedure. An additional 's' indicates that (forward) subsumption was applied in order to delete more equations. As an example consider 'rs' which means that normalization is done with rules only followed by subsumption testing.

7.2.1 Preprocessing after Generation

Table 11 shows how the system behavior is affected when the preprocessing varies. We assess any combination of applying or not R-normalization, E-normalization, and subsumption. The problems Luka10, P16lpo, and P16kbo do not produce any equations, and hence only preprocessing with resp. without rules is measured.

Preprocessing	Crit. pairs	R+E	Match Queries	Reductions	Time
mv2.pr					
–	2 220 043	1 148	15 113 015	1 674 006	386.6
e	2 214 353	1 147	51 715 849	1 704 092	529.3
r	1 287 680	820	20 016 905	2 362 485	429.5
s	2 220 043	1 148	15 113 015	1 674 006	392.4
es	2 214 353	1 147	51 715 849	1 704 092	532.4
re	1 077 750	750	22 204 135	1 993 321	416.7
rs	1 287 680	820	20 014 086	2 362 481	441.3
res	1 077 750	750	22 201 391	1 993 321	417.0
mv4.pr					
–	3 870 135	1 485	26 765 773	2 964 743	MEMORY
e	3 831 327	1 480	89 716 571	2 979 924	MEMORY
r	1 909 108	1 186	34 089 618	3 838 661	> 1000
s	3 846 668	1 482	26 596 741	2 943 439	MEMORY
es	3 855 730	1 483	90 253 837	3 000 648	MEMORY
re	1 512 758	991	35 395 057	2 916 572	> 1000
rs	1 878 341	1 171	33 275 308	3 751 068	> 1000
res	1 597 746	1 045	38 053 337	3 141 228	> 1000
p9a.pr					
–	18 233	418	683 120	10 033	2.8
e	17 347	412	841 667	13 819	5.3
r	34 168	668	1 923 981	43 151	7.0
s	18 233	418	681 935	9 875	2.9
es	17 347	412	841 323	13 819	5.4
re	34 771	684	2 267 845	54 883	10.2
rs	34 168	668	1 920 901	43 134	7.1
res	34 771	684	2 262 609	54 883	10.3
ra2.pr					
–	507 034	495	2 120 018	17 536	355.3
e	129 350	234	4 550 984	44 750	> 1000
r	16 419	106	511 237	23 763	6.7
s	507 034	495	1 950 509	17 488	330.1
es	145 390	253	5 524 134	63 448	> 1000
re	11 613	91	615 211	22 091	18.4
rs	16 419	106	493 149	23 749	6.3
res	12 094	93	610 731	22 342	18.3
ra3.pr					
–	586 183	403	1 493 421	13 067	> 1000
e	187 628	292	7 917 176	131 297	> 1000
r	119 071	259	4 567 068	164 601	112.2
s	480 735	383	1 231 504	12 973	> 1000
es	124 876	221	4 443 866	56 160	> 1000
re	124 858	258	9 595 978	233 924	783.1
rs	120 559	262	4 374 332	166 408	88.5
res	125 966	260	9 415 150	234 242	758.1
Lusk6.pr					
–	1 720 858	1 319	9 069 793	114 755	MEMORY
e	254 630	493	8 488 458	133 232	> 1000
r	161 775	566	6 319 008	470 993	79.2
s	1 713 674	1 317	8 770 171	114 346	MEMORY
es	260 075	499	8 680 887	135 420	> 1000
re	80 274	430	4 655 965	238 650	85.1
rs	161 775	566	6 227 684	470 816	80.3

Preprocessing	Crit. pairs	R+E	Match Queries	Reductions	Time
res	80 274	430	4 604 075	238 648	85.5
Luka10.pr					
–	272 616	541	2 286 563	203 416	27.1
r	137 709	394	2 118 209	238 826	16.1
P16kbo.cp					
–	839 205	8 261	132 183 029	720 231	295.2
r	215 540	3 796	18 446 415	401 632	33.3
P16lpo.cp					
–	1 988 947	12 821	268 197 582	1 217 227	632.6
r	82 639	2 306	9 826 527	100 579	16.2
jar10.2.1.pr					
–	76 809	233	286 277	3 174	63.7
e	36 671	186	1 332 060	19 791	62.5
r	31 210	219	1 066 266	19 822	15.3
s	88 100	242	238 396	2 009	77.2
es	57 939	219	1 806 971	37 784	142.0
re	18 944	175	1 204 249	19 813	20.5
rs	130 317	341	4 004 926	63 103	93.9
res	143 147	349	8 330 082	140 210	303.8
jar10.3.1a.pr					
–	5 383	104	30 694	548	0.9
e	8 559	149	237 503	4 544	3.6
r	2 710	70	69 908	989	0.6
s	9 954	147	48 418	951	1.8
es	28 920	263	713 992	23 949	17.6
re	5 437	105	217 478	4 157	2.3
rs	6 086	104	126 416	2 653	1.4
res	5 610	104	208 200	4 242	2.3

Table 11: Preprocessing critical pairs

The table shows that in some cases the lazy way[10] is the fastest combination that can be found (p9a, mv2). On the other hand it can fail completely due to lack of memory (mv4, Lusk6) or it takes much longer than other combinations (ra2). Overall we can say that not too much work should be done after generation of critical pairs as only few of them will ever be processed. The reason for this behavior is that selection strategies based on nothing but the terms can completely go bananas when normal forms and original terms differ significantly.

Run–times use to grow when additionally normalizing with equations because of the very small percentage of matches found with the set of equations that finally lead to a reduction. To be more precise, in contrast to the application of rules, rewriting with equations from $E_{selected}$ guarantees Noetherian only if the resulting term is smaller with respect to the reduction ordering $>$. Obviously this requires a $>$–test every time a matching equation could be found. Experiments have shown that one match query with the set of equations usually causes between two and ten refused matches due to failing $>$–test. On average, less than 1% of all matches found in $E_{selected}$ lead to a reduction, compared with at least twice as much successful match queries with rules — that cause no additional $>$–testing.

Result: We suggest to reduce critical pairs after generation only with rules as a compromise between keeping the selection strategy in focus, not wasting too much time in the treatment of critical pairs on generation and keeping *SUE* small.

[10] i.e. not processing the critical pairs on generation at all

7.2.2 Treatment after Selection

According to the minimality property we proposed for $R \cup E_{selected}$ equations selected from SUE will be tackled with full diminishing strength before they join R or $E_{Selected}$. As the open architecture of WALDMEISTER allows us to check even this we did so and found our proposal confirmed as Table 12 shows.

Treatment	Crit. pairs	R+E	Match Queries	Reductions	Time
mv2.pr					
–	2 299 611	1 872	40 753 845	4 242 532	> 1000
e	2 835 961	2 102	50 414 515	5 266 972	> 1000
r	1 083 607	752	22 119 740	1 989 192	417.4
s	2 614 890	1 985	45 968 969	4 792 924	> 1000
es	2 264 465	1 887	40 385 411	4 181 590	> 1000
re	1 077 750	750	22 201 391	1 993 321	418.1
rs	1 083 607	752	22 119 740	1 989 192	414.1
res	1 077 750	750	22 201 391	1 993 321	417.0
mv4.pr					
–	2 887 159	2 088	50 939 817	5 351 533	> 1000
e	2 898 884	2 125	51 727 804	5 392 380	> 1000
r	1 619 071	1 078	38 423 300	3 184 614	> 1000
s	2 918 602	2 100	51 556 928	5 415 102	> 1000
es	2 841 271	2 104	50 526 427	5 276 861	> 1000
re	1 600 604	1 047	38 148 399	3 149 579	> 1000
rs	1 623 368	1 080	38 565 443	3 194 700	> 1000
res	1 597 746	1 045	38 053 337	3 141 228	> 1000
p9a.pr					
–	72 261	1 093	4 197 482	112 141	25.1
e	72 406	1 056	4 182 026	113 392	25.9
r	35 487	694	2 313 553	55 372	10.5
s	72 261	1 093	4 197 482	112 141	25.3
es	72 406	1 056	4 182 026	113 392	25.8
re	34 771	684	2 262 609	54 883	10.8
rs	35 487	694	2 313 553	55 372	12.0
res	34 771	684	2 262 609	54 883	10.3
ra2.pr					
–	55 593	294	2 618 807	121 132	126.3
e	55 507	294	2 622 639	120 755	127.1
r	12 248	94	612 803	22 707	18.3
s	55 593	294	2 618 807	121 132	127.0
es	55 507	294	2 622 639	120 755	127.2
re	12 094	93	610 731	22 342	18.4
rs	12 248	94	612 803	22 707	18.4
res	12 094	93	610 731	22 342	18.3
ra3.pr					
–	141 295	455	12 294 661	330 426	> 1000
e	143 117	458	12 524 959	336 414	> 1000
r	125 973	264	9 350 490	234 109	740.1
s	143 108	458	12 453 012	336 392	> 1000
es	143 207	459	12 534 888	336 734	> 1000
re	125 966	260	9 415 150	234 242	759.3
rs	125 973	264	9 350 490	234 109	740.1
res	125 966	260	9 415 150	234 242	758.1
Lusk6.pr					
–	235 857	1 206	13 399 159	820 864	238.6
e	229 804	1 185	13 070 957	796 747	232.5
r	97 468	458	5 481 926	298 434	104.4
s	235 857	1 206	13 399 159	820 864	239.4
es	229 804	1 185	13 070 957	796 747	231.9
re	80 274	430	4 604 075	238 648	85.0
rs	97 468	458	5 481 926	298 434	106.3

Treatment	Crit. pairs	R+E	Match Queries	Reductions	Time
res	80 274	430	4 604 075	238 648	85.5
Luka10.pr					
–	551 343	1 260	8 460 836	963 399	64.0
r	137 709	394	2 118 209	238 826	16.1
P16kbo.cp					
–	178 316	4 460	21 999 658	333 713	35.7
r	215 540	3 796	18 446 415	401 632	33.3
P16lpo.cp					
–	125 636	2 836	11 998 :49	301 745	21.1
r	82 639	2 306	9 826 527	100 579	16.2
jar10.2.1.pr					
–	381 144	759	20 788 622	561 121	> 1000
e	387 163	787	21 219 161	571 874	> 1000
r	143 468	349	8 321 452	140 162	300.1
s	221 489	574	11 870 061	302 462	> 1000
es	221 032	573	11 961 813	298 159	> 1000
re	143 147	349	8 330 082	140 210	302.0
rs	143 468	349	8 321 452	140 162	299.3
res	143 147	349	8 330 082	140 210	303.8
jar10.3.1a.pr					
–	10 439	205	348 530	9 492	4.1
e	10 292	201	346 189	9 350	4.1
r	5 689	105	210 264	4 268	2.3
s	10 439	205	348 530	9 492	4.1
es	10 292	201	346 189	9 350	4.1
re	5 610	104	208 200	4 242	2.3
rs	5 689	105	210 264	4 268	2.3
res	5 610	104	208 200	4 242	2.3

Table 12: Treatment of critical pairs after selection

Result: According to the minimality property proposed during the design phase critical pairs should be treated with the full diminishing strength after they have been selected from *SUE*.

7.2.3 Periodically Reprocessing the Set of Critical Pairs

Although we have found out that it is sufficient to use only R when preprocessing critical pairs, it is still not clear whether intermediate reprocessing of the preprocessed but yet unselected equations (intermediate re–processing, *IRP*) leads to a better system performance. The most important reasons to believe in this are

1. that the weight of the unselected equations is computed only once after the preprocessing that follows the generation, and thus might change if the stronger rewrite relation is applied to the terms, and

2. that many unselected equations might have become joinable by now and thus the number of equations that have to be held in the set can be reduced significantly inducing less memory consumption.

Have a look at Table 13 where the letters in the leftmost column correspond to Table 11, whereas the numbers state the interval: *res 50* means that *IRP* was executed by the system every 50 insertions of elements into $R \cup E_{selected}$, where normalization

was done w.r.t. rules *and* equations and where subsumption was applied as well. The additional columns state the numbers of critical pairs touched, reduced, removed, and re–weighted, respectively.

Strat.	Crit.Pairs	R+E	Match Queries	Reductions	CPs tcd	red	rem	rew	Time
mv2.pr									
r 1	959 875	692	631 647 000	1 766 556	34 373 000	10 161	9 861	289	901.5
r 10	903 167	670	71 765 000	1 668 269	3 028 000	8 556	8 456	95	> 1000
r 50	1 079 634	747	35 589 000	2 002 864	749 000	7 904	7 771	128	531.4
r 100	1 077 750	750	29 546 000	1 999 042	415 000	7 198	7 084	109	487.6
res 1	959 875	692	631 656 000	1 766 580	34 373 000	10 163	9 909	242	904.4
res 10	620 871	560	70 080 000	1 148 496	1 684 000	6 915	6 902	13	> 1000
res 50	961 781	709	45 322 000	1 768 498	738 000	8 248	8 089	157	698.8
res 100	960 008	712	33 355 000	1 764 724	404 000	7 211	7 146	60	548.0
–	1 077 750	750	22 201 391	1 993 321					417.0
mv4.pr									
r 1	1 006 131	706	681 771 000	1 851 290	37 013 000	10 215	9 915	289	> 1000
r 10	903 167	670	72 415 000	1 668 272	3 064 000	8 590	8 485	100	> 1000
r 50	1 487 146	968	58 770 000	2 894 816	1 328 000	21 682	13 124	4 627	> 1000
r 100	1 537 530	1 009	50 255 000	3 020 152	768 000	18 761	12 951	3 293	> 1000
res 1	1 013 117	709	690 859 000	1 864 211	37 495 000	10 232	9 978	242	> 1000
res 10	641 662	570	73 111 000	1 185 955	1 759 000	7 037	7 024	13	> 1000
res 50	1 220 752	800	63 817 000	2 246 683	1 036 000	9 022	8 863	157	> 1000
res 100	1 349 799	900	55 264 000	2 557 791	679 000	12 083	9 512	1 459	> 1000
–	1 597 746	1 045	38 053 337	3 141 228					> 1000
p9a.pr									
r 1	30 092	635	22 319 000	50 298	1 504 000	1 288	801	345	27.2
r 10	30 139	637	4 027 000	50 158	151 000	627	556	70	18.0
r 50	30 299	640	2 383 000	50 236	28 000	556	485	70	10.9
r 100	30 278	640	2 215 000	50 232	16 000	494	423	70	10.2
res 1	30 092	635	22 321 000	50 309	1 503 000	1 288	872	275	27.1
res 10	30 139	637	6 044 000	50 243	150 000	745	679	20	44.1
res 50	30 299	640	2 763 000	50 295	28 000	638	572	16	15.6
res 100	30 278	640	2 428 000	50 303	16 000	582	531	16	12.8
–	34 771	684	2 262 609	54 883					10.3
ra2.pr									
r 1	8 682	81	3 236 000	17 893	138 000	1 840	672	1 166	15.1
r 10	11 165	90	967 000	22 507	19 000	1 043	398	645	19.0
r 50	11 154	89	629 000	20 371	3 000	60	48	12	17.3
res 1	8 682	81	3 235 000	18 220	137 000	1 924	1 105	817	16.0
res 10	10 877	88	1 357 000	21 694	18 000	1 103	763	440	49.4
res 50	11 154	89	701 000	20 375	3 000	62	54	7	22.7
–	12 094	93	610 731	22 342					18.3
ra3.pr									
r 1	95 118	233	198 533 000	220 855	7 050 000	24 567	2 964	21 573	965.0
r 10	103 088	241	29 610 000	206 783	805 000	6 446	1 248	5 198	729.6
r 50	107 021	242	11 819 000	201 119	143 000	753	205	548	679.1
r 100	107 021	242	10 409 000	201 119	91 000	719	174	545	675.3
res 1	81 794	206	159 810 000	197 818	5 595 000	17 333	3 113	14 202	> 1000
res 10	52 477	128	8 747 000	79 306	110 000	183	2 399	0	> 1000
res 50	61 255	150	7 072 000	96 213	62 000	160	817	0	> 1000
res 100	85 838	200	9 350 000	164 237	48 000	215	795	48	> 1000
–	125 966	260	9 415 150	234 242					758.1
Lusk6.pr									
r 1	78 754	416	117 836 000	246 286	4 831 000	6 566	3 192	2 729	179.0
r 10	79 302	419	15 672 000	243 989	474 000	3 798	2 078	1 678	145.8
r 50	82 054	437	6 870 000	252 147	92 000	2 600	1 151	1 411	99.9
r 100	82 116	437	6 115 000	251 985	58 000	2 302	945	1 327	96.1
res 1	78 754	416	117 424 000	248 040	4 806 000	6 527	4 444	1 481	181.0
res 10	79 302	419	26 594 000	244 404	470 000	3 850	3 075	951	564.3

Strat.	Crit.Pairs	R+E	Match Queries	Reductions	CPs tcd	red	rem	rew	Time
res 50	82 054	437	8 967 000	252 496	92 000	2 642	1 970	828	182.3
res 100	82 116	437	7 469 000	252 279	58 000	2 342	1 592	824	148.5
–	80 274	430	4 604 075	238 648					85.5
Luka10.pr									
r 1	132 602	382	40 174 000	220 000	2 623 000	2 193	2 180	13	49.4
r 10	132 602	382	5 969 000	220 000	274 000	1 859	1 847	12	37.2
r 50	89 337	315	1 720 000	151 000	29 000	733	724	9	11.8
r 100	137 558	393	2 362 000	239 000	17 000	576	568	8	17.3
–	137 709	394	2 118 209	238 826					16.1
P16kbo.cp									
r 1	140 278	3 273	241 778 000	295 000	28 551 000	37 993	12 544	12 465	255.4
r 10	146 831	3 313	43 407 000	290 000	3 179 000	21 465	3 942	14 225	113.5
r 50	150 666	3 317	25 376 000	304 000	720 000	21 559	1 807	17 330	51.6
r 100	156 053	3 375	22 689 000	315 000	361 000	17 397	1 330	14 297	42.2
–	215 540	3 796	18 446 415	401 632					33.3
P16lpo.cp									
r 1	80 539	2 276	122 024 000	143 000	14 469 000	25 183	2 307	11 935	128.7
r 10	81 064	2 276	21 022 000	102 000	1 447 000	4 393	1 182	1 868	51.6
r 50	81 687	2 290	12 008 000	99 000	283 000	1 332	780	461	22.9
r 100	82 639	2 306	10 861 000	102 000	133 000	1 244	799	354	19.1
–	82 639	2 306	9 826 527	100 579					16.2
jar10.2.1.pr									
r 1	107 283	312	87 093 000	227 164	2 665 000	53 377	10 140	42 947	391.4
r 10	190 412	277	31 500 000	234 013	711 000	5 470	1 276	4 194	> 1000
r 50	96 387	303	7 325 000	94 623	56 000	2 019	575	1 444	154.9
r 100	96 484	303	6 237 000	94 808	19 000	1 636	450	1 186	150.6
res 1	166 968	298	168 314 000	248 021	5 337 000	28 282	18 792	9 489	> 1000
res 10	136 085	290	28 439 000	150 039	347 000	8 656	6 025	3 728	> 1000
res 50	153 848	250	14 371 000	168 640	95 000	5 043	4 989	952	> 1000
res 100	142 577	348	12 231 000	150 152	67 000	7 708	1 688	6 131	455.9
–	143 147	349	8 330 082	140 210					303.8
jar10.3.1a.pr									
r 1	2 191	56	400 000	2 382	11 000	1 143	124	1 010	1.4
r 10	2 678	64	157 000	1 485	1 000	108	38	70	1.3
r 50	3 391	79	137 000	1 672	414	68	34	34	1.4
r 100	5 608	104	208 000	3 792	44	22	16	6	2.4
res 1	2 191	56	414 000	2 397	11 000	1 135	308	824	1.6
res 10	2 678	64	202 000	1 530	1 000	136	82	72	1.8
res 50	3 391	79	143 000	1 673	414	68	57	14	1.4
res 100	5 608	104	208 000	3 792	44	22	31	3	2.4
–	5 610	104	208 200	4 242					2.3

Table 13: Reducing the set of critical pairs

The table shows that the additional effort spent on reducing all the unselected equations held by the system does in general not lead to a significant speedup. As the times use to get smaller when the intervals grow we believe that the small number of critical pairs that could be reduced, reweighteded or removed does not justify the effort. However, there are situations when this approach can result in significant speedups: sometimes the system produces a rule or equation that diminishes the set of rules and equations heavily via interreduction. In these cases the rewrite relation has gathered so much additional strength that many critical pairs can be reduced, reweighted, and thus a significant reordering is enforced. For example, ra3 was solved faster in the settings "r 50" resp. "r 100".

Another result that can be drawn from Table 13 is that applying *IRP* in order to save memory by deleting critical pairs whenever they can be joined will not lead to a significantly smaller size of the proof process. Compared with the huge number of

critical pairs touched, only few of them could be removed during *IRP*. More space can be saved by appropriate data structures for representing *SUE*.

Result: Intermediate re–processing and re–weighting of *SUE* in general has negative influence on run times but can lead to faster proofs in some cases (see also Sect. 10). The sets of rules and equations, R and $E_{selected}$, do not get much smaller either, thus for the add–weight heuristic in general the lazy variant that executes no *IRP* at all performs best.

7.3 A Subconnectedness Criterion

Kapur et al. [KMN88] introduced a simple criterion that detects superpositions which can be ignored without losing completeness: If an inner subterm of the current superposition can be reduced w.r.t. the current rewrite relation the prover can ignore the resulting critical pair. This criterion requires an efficient matching algorithm as supported by WALDMEISTER. Although the number of critical pairs generated by the system can be reduced significantly the additional time spent in the matching routine makes the runs a little slower (Table 14)[11]. Nevertheless we found that the application of the criterion with maximum strength is absolutely necessary for proving jar10.2.1 in a reasonable amount of time.

Result: Application of the criterion generally saves critical pairs but should be restricted to additional runs on problems that could not be solved otherwise. A reason for this behavior might be that proof runs do not rely on the success of the completion process and thus on the generation of the most general facts. Proofs might be found as well by applying more special results that would be rejected when applying the criterion.

7.4 Strategies for the Computation of Normal Forms

The task of a normalization procedure is to compute an irreducible successor of the given term. As at an arbitrary stage of the proof process the sets of rules and equations are usually not convergent, different realizations will result in different normal forms. Looking at normalization algorithms one will find three fields where differences may appear:

- the term traversal strategy,
- the way rules and equations are combined, and
- the organization of backtracking.

[11] The columns stating the number of calls to the matching algorithm do *not* include the calls required by the criterion

Criterion	Crit. pairs	R +E	Match Queries	Reductions	Time
mv2.pr					
—	1 169 725	750	22 956 309	2 091 877	225.7
r	1 108 523	750	22 359 204	2 012 990	278.4
re	1 077 750	750	22 201 391	1 993 321	422.4
mv4.pr					
—	2 164 334	1 236	53 572 074	4 589 631	> 1000
r	1 998 683	1 222	50 679 609	4 266 704	> 1000
re	1 605 932	1 050	38 319 210	3 166 792	> 1000
p9a.pr					
—	43 200	693	2 403 964	67 260	9.3
r	36 872	684	2 279 295	56 640	9.3
re	34 771	684	2 262 609	54 883	10.3
ra2.pr					
—	12 753	92	628 461	24 125	17.2
r	12 131	92	611 197	22 480	17.1
re	12 094	93	610 731	22 342	18.4
ra3.pr					
—	127 758	260	9 510 124	240 468	720.2
r	126 365	260	9 430 200	235 111	724.7
re	125 966	260	9 415 150	234 242	758.0
Lusk6.pr					
—	111 878	460	6 096 733	391 980	96.0
r	80 529	430	4 610 917	239 329	75.8
re	80 274	430	4 604 075	238 648	85.9
Luka10.pr					
—	146 281	394	2 196 103	260 502	12.1
r	137 709	394	2 118 209	238 826	16.3
re	137 709	394	2 118 209	238 826	16.2
P16kbo.cp					
—	217 196	3 796	18 459 354	405 342	33.5
r	215 540	3 796	18 446 415	401 632	34.8
re	215 540	3 796	18 446 415	401 632	33.9
P16lpo.cp					
—	84 290	2 306	9 839 451	104 344	16.4
r	82 639	2 306	9 826 527	100 579	16.7
re	82 639	2 306	9 826 527	100 579	16.5
jar10.2.1.pr					
—	230 347	281	14 181 991	266 046	> 1000
r	226 089	279	13 939 999	260 139	> 1000
re	143 147	349	8 330 082	140 210	303.5
jar10.3.1a.pr					
—	6 558	106	227 142	5 579	1.9
r	5 880	104	214 502	4 654	1.9
re	5 610	104	208 200	4 242	2.3

Table 14: Application of the subconnectedness criterion

7.4.1 Term Traversal and Discrimination of Equations

Different term traversal strategies are top–down which applies a breadth–first term traversal, leftmost–innermost that follows the post–ordering of the terms, and leftmost–outermost which follows the pre–ordering.

From a certain point of view those different strategies lead to similiar normalization procedures (cf. Fig. 17): all positions in the given term will be visited following that very traversal strategy. If the subterm at the current position could be reduced, backtracking is invoked. Otherwise the traversal continues until the next reducible subterm is encountered, or the traversal is finished.

R–Normalize(term)

```
position= top–position(term);
repeat
    if (position is reducible w.r.t. rules)
        execute one rewrite step w.r.t. rules;
        position = next(position, backtrack);
    else
        position = next(position, continue);
    fi
until traversal is finished
```

Figure 17: Normalization w.r.t. rules

Bringing equations into the game raises the question how rules and equations should be combined. Different solutions at this point discriminate equations as far as possible, treat them equally, or prefer them. In the case of top–down and leftmost–inermost we only realized the first variant following Fig. 18: After computation of the normal form w.r.t. rules the search for a position reducible with equations starts. If such a position can be found, the reduction takes place and the term is normalized with rules before the treatment with equations starts again. In order to evaluate the different approaches, we implemented three different versions of the leftmost–outermost strategy: leftmost–outermost which behaves according to Fig. 18 as well, leftmost–outermost–RE behaving according to Fig. 19, and leftmost–outermost–ER following the same algorithm but applying equations first.

The effort we spent on realizing backtracking for the various traversal strategies differs as leftmost–outermost showed the most promising behaviour. For that reason, we did not employ marking of irreducible subterms for leftmost–innermost. Furthermore, leftmost–innermost and top–down apply R–Normalize on the top position of the given term each time an E–reduction took place. In contrast, leftmost–outermost applies the R–Normalization on the affected *sub*term only.

Normalize(term)

```
R–Normalize(term);
while (term is reducible with equations)
        execute one rewrite step w.r.t. equations;
        R–normalize(term);
end
```

Figure 18: Normalization w.r.t. rules and equations

Normalize–RE(term)

```
position= top–position(term);
repeat
    if (position is reducible w.r.t. rules)
        execute one rewrite step w.r.t. rules;
        position = next(position, backtrack);
    else if (position is reducible w.r.t. equations)
        execute one rewrite step w.r.t. equations;
        position = next(position, backtrack);
    else
        position = next(position, continue);
    fi
until traversal is finished
```

Figure 19: The algorithm for normalization leftmost–outermost–RE

Result: Table 15 shows how the choice of the normal form strategy affects the behavior of the runs.

- Top–down in general behaves similar to leftmost–outermost but performs worse due to higher effort managing term traversal and backtracking.
- Leftmost–innermost: as the term structure used by the system does not support marking of irreducible subterms every reduction leads to a restart of the procedure on the actual subterm. This is not the only reason for the poor performance of the innermost strategy: it usually produces larger sets R, $E_{selected}$, and SUE. However, this strategy is clearly the best one in one case (jar10.2.1).
- The strategies *outermost–RE* and *outermost–ER* do not discriminate equations

Strategy	Crit. Pairs	Rules	Equs	CoM R	R R	CoM E	R E	Time
mv2.pr								
top-down	1 077 750	744	6	14 927 678	1 915 401	5 877 106	65 992	448.3
innermost	3 065 407	4 156	1	86 492 249	2 628 878	10 869 042	2 601	> 1000
outermost	1 077 750	744	6	16 337 174	1 927 308	5 864 217	66 013	420.2
outermost-ER	1 077 750	744	6	15 290 240	1 873 520	12 736 612	609 272	646.1
outermost-RE	1 077 750	744	6	15 440 228	1 918 432	10 358 945	105 388	510.1
mv4.pr								
top-down	1 574 360	1 010	21	25 020 880	2 914 077	10 341 976	145 221	> 1000
innermost	3 089 467	4 174	1	87 293 086	2 640 042	10 948 212	2 612	> 1000
outermost	1 610 781	1 032	21	27 845 732	3 029 833	10 587 282	149 874	> 1000
outermost-ER	1 404 616	881	19	21 131 897	2 424 673	17 856 025	787 671	> 1000
outermost-RE	1 478 279	947	21	22 867 463	2 658 905	15 917 688	211 732	> 1000
p9a.pr								
top-down	34 771	672	12	1 972 681	45 213	294 515	9 603	11.4
innermost	53 933	1 015	47	3 857 286	59 582	496 262	12 847	18.4
outermost	34 771	672	12	1 969 214	45 270	293 395	9 613	10.3
outermost-ER	32 982	655	12	1 819 746	40 207	338 703	9 289	11.8
outermost-RE	34 745	671	12	1 912 375	44 819	312 591	10 337	10.6
ra2.pr								
top-down	12 094	72	21	386 960	16 519	244 173	5 782	20.6
innermost	273 504	1 044	182	8 961 292	214 697	3 161 933	38 810	571.5
outermost	12 094	72	21	373 943	16 536	236 788	5 806	18.8
outermost-ER	12 469	77	16	348 801	15 893	327 832	8 451	26.2
outermost-RE	12 461	77	16	356 424	17 371	309 663	7 852	24.6
ra3.pr								
top-down	125 966	184	76	5 958 052	173 610	3 901 486	59 644	802.7
innermost	154 678	306	134	2 022 965	116 679	1 703 491	30	> 1000
outermost	125 966	184	76	5 664 008	173 638	3 751 142	60 604	756.7
outermost-ER	102 181	136	68	3 948 864	138 978	3 900 416	46 963	> 1000
outermost-RE	107 530	142	68	4 234 616	150 449	3 974 375	46 260	> 1000
Lusk6.pr								
top-down	79 664	381	47	3 000 421	217 836	1 584 931	17 724	96.9
innermost	266 298	6 600	46	184 903 208	231 978	2 083 767	22 093	> 1000
outermost	80 274	383	47	3 032 064	220 601	1 572 011	18 047	84.3
outermost-ER	79 009	379	41	2 728 536	184 890	2 226 150	77 531	129.0
outermost-RE	81 914	381	45	2 852 775	214 008	2 070 277	35 269	124.0
Luka10.pr								
top-down	137 709	394		1 977 188	236 054			17.6
innermost	137 709	394		3 330 096	251 169			18.1
outermost	137 709	394		2 118 209	238 826			16.3
P16kbo.cp								
top-down	215 540	3 796		18 444 914	401 632			39.4
innermost	215 540	3 796		19 921 844	417 420			36.4
outermost	215 540	3 796		18 446 415	401 632			33.9
P16lpo.cp								
top-down	82 639	2 306		9 827 228	100 579			17.0
innermost	82 639	2 306		10 084 710	100 492			16.9
outermost	82 639	2 306		9 826 527	100 579			16.5
jar10.2.1.pr								
top-down	143 147	164	185	4 847 225	76 195	3 747 288	63 893	346.3
innermost	13 954	257	28	208 232	4 089	64 138	494	4.9
outermost	143 147	164	185	4 614 037	76 248	3 716 045	63 962	306.8
outermost-ER	143 423	165	185	3 900 019	75 200	4 002 174	72 324	314.1
outermost-RE	143 423	165	185	3 971 554	75 322	3 926 352	71 858	315.6
jar10.3.1a.pr								
top-down	5 610	81	23	124 339	2 448	91 284	1 792	3.1
innermost	12 059	246	33	166 978	5 608	60 737	1 487	3.1
outermost	5 610	81	23	118 148	2 450	90 052	1 792	2.3
outermost-ER	5 610	81	23	106 172	2 369	97 076	2 402	2.3
outermost-RE	5 610	81	23	107 333	2 433	93 493	2 142	2.3

Table 15: Strategies for computing normal forms

during normalization as leftmost–outermost does. In agreement with the results from above we find similar runs but significantly higher time consumption.

- The strategy corresponding best with flatterms is *(leftmost–)outermost.*

Another important result to be taken from this table is that shifting the priority towards the application of equations usually slows down the runs significantly. The reasons for this have been mentioned earlier and correspond with Tables 11 and 12.

7.4.2 Organizing Backtracking

Having identified leftmost–outermost as the one normalization strategy corresponding best with flatterms, we will now analyze the choice points remaining after the traversal strategy has been fixed. In the first part of this section we will name the candidates for follow–up reductions and introduce different realizations of the backtracking management. Afterwards we demonstrate the influence of different backtracking directions on the behavior of the prover.

Limitation of backtracking. Avoiding superfluous reduction attempts ought to be a primary goal of any normal form algorithm. After the execution of a rewrite step some previously irreducible positions in the term might have become reducible. Given a traversal strategy these positions can be identified.

In the case of leftmost–outermost one has to test only the path from the currently reduced position up to the root. Only those positions up to the level given by the maximum depth of a left hand side (dashed line in Fig. 20) have to be tested one by one. Above, only reduction attempts have to be repeated which failed due to indirect clashes or >–test (cf. Fig. 20).

As an example, consider the term $t = f(a, b)$ and the rewrite rules $r_1 : f(a, c) \rightarrow c$, $r_2 : f(x, x) \rightarrow x$, where x is a variable, f a function symbol, and a,b,c are constants. Trying to apply r_1 to t fails as a direct clash appears in the second argument ($c \neq b$), whereas the application of r_2 fails because x has already been bound to a when it should be bound to b (indirect clash).

When realizing a backtracking mechanism for the leftmost–outermost reduction procedure we can limit backtracking to the current branch of the term and do not have to traverse the whole term again. Direct clashes make reductions possible up to a certain position above the current one which depends on the maximum depth of a left hand rule side (dashed line in Fig. 20). Indirect clashes encountered with nonlinear rules on certain positions above make further examination necessary. We have compared three different solutions of this problem:

FLAGS manages indirect clashes by flags. Backtracking touches all positions but calls the matching algorithm only if the flag is set.

POINTERS manages indirect clashes by flags and pointers that indicate the next position of an indirect clash in the direction of backtracking. Thus backtracking simply has to follow the pointers.

NONE does not manage indirect clashes at all. Backtracking invokes the matching algorithm everywhere.

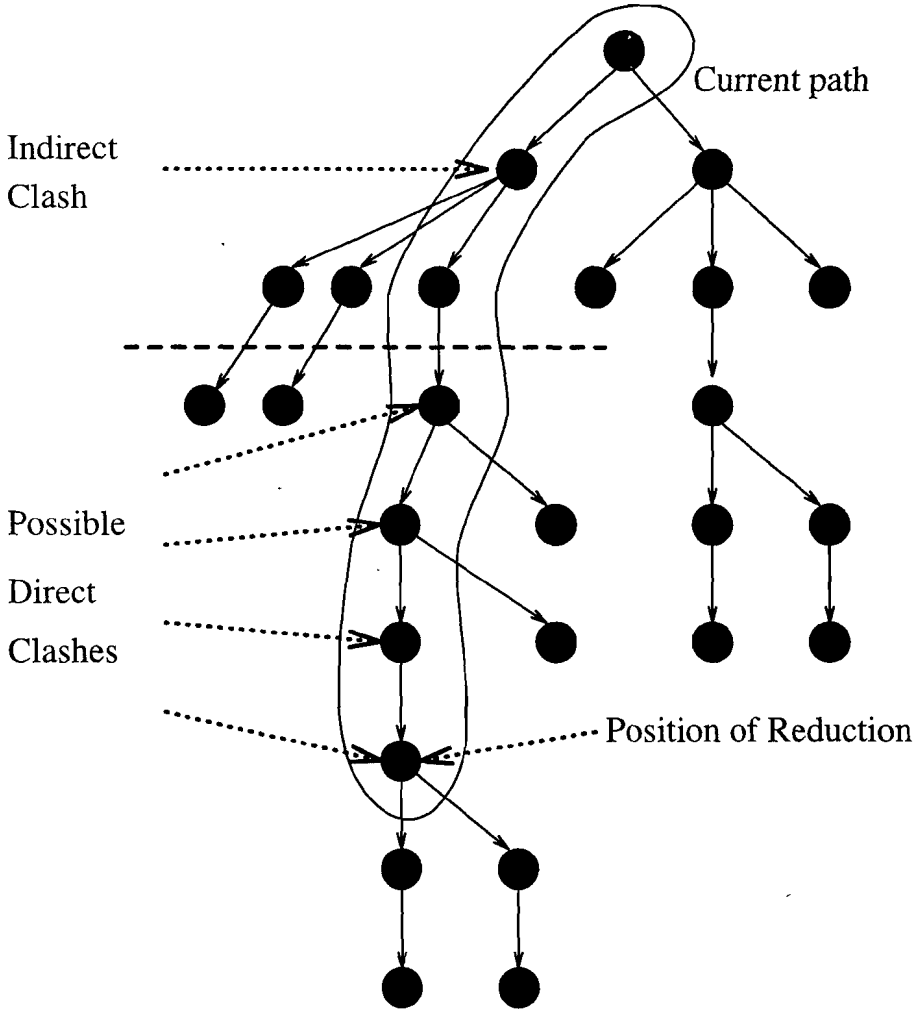


Figure 20: Indirect and direct clahes

All the three variations touch every position in the given term up to the one that can be derived from the maximum depth of the left–hand sides of the rewrite rules. As one should expect the number of reductions done are identical in the three versions whereas the numbers of calls to the matching–routine differ enormously (comparing FLAGS and POINTERS with NONE) — or not at all (comparing FLAGS and POINTERS). The following table shows how the way the limitation is realized affects the overall run time.

Strategy	Rules	Match Queries	Reductions	Time
dt1.rd				
FLAGS	11	98 543 228	205 429	252.5
POINTERS	11	98 543 228	205 429	260.5
NONE	11	122 682 087	205 429	276.9
fac8.rd				
FLAGS	6	254 909	58 078	135.3
POINTERS	6	254 909	58 078	1.1
NONE	6	932 382 091	58 078	1322.8
braid4.cf				
FLAGS	862	2 415 586	657 396	9.6
POINTERS	862	2 415 586	657 396	9.7
NONE	862	5 336 071	657 396	15.2

Table 16: Limitation of backtracking

Result: Our experiments have shown that in general the NONE version performs worst as it spends too much time in the superfluous calls of the matching routine (e.g. 30% superfluous match queries and thus more time on dt1.pr). The POINTERS version usually performs a little worse than the FLAGS version because of its additional overhead. When the terms become very deep the number of calls of the matching routine increases: 30% more match queries and thus 13% more time on dt1.pr. During the computation of $fac(8)$ in peano arithmetics which produces the term $s^{40320}(0)$ the management overhead of the POINTER version is clearly outweighed by the saved flag-lookups. Backtracking recognizes the last position of an indirect clash and hence stops much earlier than if on every position the flag was tested. Nevertheless, in general one should choose the flag version.

Varying backtracking. Table 17 shows how even the different backtracking strategies can affect the performance if the comparison of equally weighted critical pairs is left open (and not fixed by age as in the default configuration). The table shows variations in two aspects: first, a reduction can be followed by further reduction attempts until the current position in the term has become irreducible (*repeat*), or backtracking starts immediately (*backtrack*). Second, backtracking may start with the highest candidate and descent stepwise (*down*), or ascend stepwise until the highest candidate has been reached (*up*). *lo* resp. *li* mean leftmost-outermost resp. leftmost-innermost.

Strategy	Crit. pairs	R + E	Match Queries	Reductions	Time
Lusk6.pr					
lo — repeat — up	73 466	447	4 126 077	223 408	72.2
lo — repeat — down	84 197	471	4 813 307	262 885	89.1
juxta.rd					
li — repeat — up	459	459	21 453	21 447	MEMORY
li — backtrack — up	459	459	14 728 963	141 425	69.0
braid4.cf					
lo — repeat — up	20 813	862	2 008 452	657 396	8.2
lo — repeat — down	20 813	862	2 415 586	657 396	9.6

Table 17: Variations of fixed strategies

Proving Lusk6 took 14.6% more critical pairs and 23.5% more time changing the direction of backtracking from upwards to downwards.[12] This effect is induced by the rough granularity of the add–weight selection strategy.

The example juxta runs out of memory when attempting recently reduced positions in the term again instead of starting backtracking immediately. This is because termination holds only under a certain normalization strategy (innermost combined with a special backtracking mechanism; cf. [WZ95]).

Result: The parameter setting of the backtracking behavior is worthwhile although we could not recognize a tendency. Once again it turns out that even slight changes of the inference machine can influence the behavior of the prover.

7.5 Changing the Default Parameter Setting

The previous evaluations suggest that the system's parameter setting used for the test runs can be significantly improved by turning off the subconnectedness criterion and simplifying critical pairs only with rules on generation. Table 18 shows that the new parameter setting manages to prove even mv4 in less than 1000 seconds which failed when only changing one of the two parameters. The other problems could be solved quicker than before. This suggests that combining the results of different experiments does not suffer from unpredictable overlay–effects in this case — if blurring phenomena like the comparison of equally weighted critical pairs and the backtracking mechanism are recognized and hence suppressed.

Example	Crit. pairs	R+E	Match Queries	Reductions	new Time	old Time
P16kbo.cp	217 196	3 796	18 459 354	405 342	32.2	33.3
P16lpo.cp	84 290	2 306	9 839 451	104 344	16.0	16.2
sims2.cp	55 065	576	7 146 127	250 496	40.4	46.7
z22.cp	3 005	166	131 766	6 451	0.6	0.7
mv2.pr	1 433 662	829	21 279 744	2 555 459	176.9	417.0
mv4.pr	2 817 911	1 467	55 226 948	6 311 862	560.8	> 1000
p9a.pr	42 523	677	2 030 489	54 326	5.5	10.3
ra2.pr	16 997	106	522 896	25 232	4.6	18.3
ra3.pr	118 627	256	4 556 491	166 146	72.4	758.1
Lusk4.pr	1 956	68	44 204	4 754	0.2	0.3
Lusk5.pr	14 068	89	349 957	27 693	2.5	5.5
Lusk6.pr	215 556	595	8 271 919	767 548	66.7	85.5
Luka10.pr	146 281	394	2 196 103	260 502	12.1	16.1
jar10.2.1.pr	35 856	222	1 208 744	24 798	9.4	303.8
jar10.3.1a.pr	2 956	70	73 026	1 293	0.4	2.3
jar10.3.7b.pr	938	33	15 192	746	0.1	0.3

Table 18: Performance of the new parameter setting

Result: The new default parameter setting outperforms the old one significantly.

[12] Remember that the selection among equally weighted critical pairs was left open to the priority queue here.

8 Towards an Efficient Implementation

Now we have designed suitable data structures and algorithms. The next step is to transfer efficient algorithms into an efficient program.

First of all, why do we consider the imperative programming paradigm appropriate? Our notion of "appropriate" is to use the underlying hardware as profitable as possible in order to get a high performance deduction system. As a matter of fact, only controlling the processor explicitly allows taking best advantage of the machine power. Consequently, the programming language should be an imperative one with its concepts close to the machine but yet offering means of abstraction. This leads to the C family and arises the question: C or C++?

Object–oriented concepts are a useful tool for achieving well-structured designs. However, implementing them does not necessarily require an object–oriented programming language. In our case, one cannot benefit by strict encapsulation. Let us take the flat term structure as an example: Gaining efficiency in copying, memory management (deletion in constant time), or generalization retrieval (discrimination tree traversal) bases on knowledge of the low–level data structure. For these reasons, we decided to use pure ANSI C, knowing that this choice imposes humble work such as memory management, elimination of recursions, which is needed at special points for efficiency, but C compilers are not able to do automatically, and explicit construction of generic types such as lists, stacks, and trees. In comparison with higher–level languages or libraries, the resulting data structures are not overspecified[13] but specialized for our application.

From our implementation experience, we can state a few guidelines that fit for programming in C in the deduction area. First of all, one should strive for as much genericity as possible. At the open choice points we have mentioned above, the use of function variables simplifies altering or enlarging the specific functionality. Macros can be employed to create generic types and functions. They also allow to force inlining of tiny functional units that are used very frequently, which is valuable even on SPARC stations. For appropriate evaluation of settings for open choice points, doing careful measurements is inevitable. The more detailed these statistics are, the more our picture of the deduction process becomes precise — and the more effort has to be spent on counting, comparing, etc. Using conditional compilation allows to derive — from that very source code — executables with different focusses: on efficiency, on statistics, on testing.

The last of these three items should not be underestimated, as the algorithms in the deduction domain are not the simplest ones. We recommend that developing complex data structures — Waldmeister discrimination trees for example — should go hand in hand with the design of both visualization routines and internal consistency predicates. In statistics mode, our free–list based memory management logs the number of allocate and free calls for each storage class. This shows up exactly where the system's memory consumption arises from. Tidying up after a program run reveals whether objects have

[13] For example attractive but expensive operations on lists or boundary–checked arrays.

been "lost" intermediately or not. For the case of term cells, this is even possible at run–time. Additionally, special marking operations are able to detect if any cell erroneously is employed by several terms. This is an excellent debugging aid for a low–level language such as C.

Following these guidelines does not automatically result in efficient programs, but we believe that they do help anyone who wants to implement a system with focus on efficiency. Admittedly, many of them have become obvious to us not before the moment when we wished we *had* followed them.

Examples

Macros forcing inlining and for executables with different focusses

The following lines of code show the declaration of the dynamic memory allocation macro `S_Alloc` realizing free–list based memory management (parameterized with an organization structure, of which there is one for each object class). The first element of the free–list is returned if the list is non–empty. Otherwise, a new block of memory is fetched, inscribed a singly–linked list, and the first element returned.

Working with dynamic data structures, one has to pay attention that they are not corrupted by erroneous use of the allocation and freeing routines (e.g. freeing a termcell that has already been freed) which causes failures that are very hard to trace back to their origin. In such a case, memory test routines are included setting the `_TEST_MODE` flag; and they attempt to determine whether there are inconsistencies in the memory management.

On a higher level, there is another source of displeasure: As the management is done without garbage collection, every object that is no longer used has to be freed explicitly. If this is forgotten for one or another case somewhere deep in the system, a considerable amount of memory may be wasted without notice. Having introduced optional statistics of allocating and freeing calls separately for each object class, which is triggered by the `_STATISTICS_MODE`, we are able to determine whether objects have been lost during a system run, and if so which object they belong to.

All in all, there are four different realizations of `S_Alloc` that are derived from the very same source code. Moreover, calls of `S_Alloc` are always inlined at the location of calling, which saves a function call whenever a new object is created.

```
#ifdef _TEST_MODE
  #define if_Test(Action) (Action, 0)
#else
  #define if_Test(Action) 0
#endif

#ifdef _STATISTICS_MODE
  #define if_Stat(Action) (Action,0)
```

```
#else
  #define if_Stat(Action) 0
#endif

#define /*void **/ S_Alloc(/*S_MemoryManagerT*/ mbm)                    \
(                                                                       \
  if_Test(malloc_verify()),                                             \
  if_Stat(MaximizeLong(&mbm.MaxUsedElements,                            \
                       ++mbm.Requests-mbm.Free)),                       \
  mbm.next_free ? (       /* Are there cells left in free-list? */\
    ReturnValue = (void*) mbm.next_free,                                \
    if_Test(check_address(&mbm,ReturnValue)),                           \
    mbm.next_free = mbm.next_free->next_free,                           \
    ReturnValue                                                         \
  ) : (                    /* else function call for allocation */\
   (void*) S_AllocWithNewBlock(&mbm);                                   \
  )                                                                     \
)

/* Usage: NewTermcell = S_Alloc(TermcellManager);
          NewTermcell->Symbol = ...;                                    */
```

Macros for genericity

Aiming at code readability, macros are inevitable if many similiar functions are required. In our case we need a set of functions computing weight tupels from termpairs. The first component returned is a weight that stems from the maximum or the sum of the weights of the two terms. These single weights may be computed via `TermWeight_A` or `TermWeight_B`. Furthermore, the weights computed may be collected by some recording function. Additionally the second component sometimes may hold an age entry. All this functionality must be realized within separate functions for performance reasons, as a one function realization would cause interpreting overhead. A pattern of these functions can be found at the end of the code.

Now, the question is how to generate these $2 \cdot 2 \cdot 2 \cdot 3 = 24$ functions in a piece of code that is readable, changeable, and as short as possible. In situations like this, we employ generic macros both on the encoding and the meta level. The first block of those macros employs the needed functionality: For each variant, there is one macro (or function in the case of TermWeight_A resp. TermWeight_B) that is named according to its behaviour.

The second block of macros does the construction of the functions. First, we find the declaration of a single one of these functions. This is done via `SINGLE_DECLARATION`, which is given the primary (sum, or max) and secondary (elder, younger, none) identifier, an identifier for the computation of term weights (A or B), and finally an identifier that establishes the weight recording function. The following macros each

vary one or two of these identifiers, and finally via BLOCK_DECLARATION(Max) and BLOCK_DECLARATION(Sum) we declare all the 24 functions at once.

```
/* ------------ Functionality ---------------------------------------*/

#define WeightIs_Stat(_weight)   RecordWeight(_weight);
#define WeightIs_NoStat(_weight)

#define PreferElder_BODY(ComputationBody, /*{Stat, NoStat}*/ STAT)    \
{  ComputationBody;                                                   \
   (wt->w2) = ++CPNumber;                                             \
   WeightIs_ ## STAT (wt->w1);}

#define PreferYounger_BODY(ComputationBody, /*{Stat, NoStat}*/ STAT)  \
{  ComputationBody;                                                   \
   (wt->w2) = --CPNumber;                                             \
   WeightIs_ ## STAT (wt->w1);}

#define PreferNone_BODY(ComputationBody, /*{Stat, NoStat}*/ STAT)     \
{  ComputationBody; WeightIs_ ## STAT (wt->w1);}

#define Max_BODY(LHS, RHS, /*{A,B}*/ WEIGHT)                         \
{  long int weight2 = 0;                                              \
   wt->w1 = TermWeight_## WEIGHT(LHS);                                \
   weight2 = TermWeight_## WEIGHT(RHS);                               \
   wt->w1 = (wt->w1>weight2)? (wt->w1) : weight2;}

#define Sum_BODY(LHS, RHS, /*{A,B}*/ WEIGHT)                         \
{  wt->w1 = TermWeight_## WEIGHT(LHS);                                \
   wt->w1 += TermWeight_## WEIGHT(RHS);}

/* ------------ Declarations ----------------------------------------*/

#define SINGLE_DECLARATION(_PRIM_, _SEC_, _WEIGHT_, _STAT_)           \
void _PRIM_##_SEC_##_WEIGHT_##_STAT_                                  \
           (WeightEntryT *wt, TermT lhs, TermT rhs)                   \
Prefer##_SEC_##_BODY(_PRIM_##_BODY(lhs,rhs,_WEIGHT_),_STAT_)

#define DECLARATION(_PRIM_, _SEC_)                                    \
SINGLE_DECLARATION(_PRIM_, _SEC_, A, Stat)                            \
SINGLE_DECLARATION(_PRIM_, _SEC_, A, NoStat)                          \
SINGLE_DECLARATION(_PRIM_, _SEC_, B, Stat)                            \
SINGLE_DECLARATION(_PRIM_, _SEC_, B, NoStat)

#define BLOCK_DECLARATION(_PRIM_)                                     \
```

```
DECLARATION(_PRIM_, None)                                                \
DECLARATION(_PRIM_, Elder)                                               \
DECLARATION(_PRIM_, Younger)

BLOCK_DECLARATION(Max)

BLOCK_DECLARATION(Sum)

/* ------------- Function Pattern ---------------------------------- */
/*                                                                     */
/* void MaxElderAStat(WeightEntryT *wt, TermT lhs, TermT rhs)         */
/* {                                                                   */
/*   long int weight2 = 0;                                             */
/*   wt->w1  = TermWeight_A(LHS);                                      */
/*   weight2 = TermWeight_B(RHS);                                      */
/*   wt->w1  = (wt->w1>weight2)?  (wt->w1):weight2;                    */
/*   wt->w2  = ++CPNr;                                                 */
/*   RecordWeight(wt->w1);                                             */
/* }                                                                   */
/*                                                                     */
```

9 Comparison with the DISCOUNT Inference Machine

The main goal during the development of WALDMEISTER was to build a high performance inference machine later to be topped by sophisticated selection strategies. With the data structures and algorithms we introduced earlier in this report, furthermore with the flexible control in the mid level, we believe to have reached this goal: Table 19 compares the throughputs of the DISCOUNT inference machine and of the one of WALDMEISTER (it does not compare the whole systems in general, since DISCOUNT follows different research goals).

Problem	TPTP or Reference	WALDMEISTER/Inf. Mach. Time	Space	Red/sec	DISCOUNT/Inf. Mach. Time	Space	Red/sec
Luka10.pr	[Tar56]	15.2	5.5	1 773	127.6	31.5	1 629
Lusk3.pr	RNG008-7	0.2	2.4	3 040	4.4	1.2	678
Lusk4.pr	GRP002-4	0.4	2.7	2 750	2.7	3.5	3 487
Lusk5.pr	BOO002-1	3.3	2.9	1 807	31.2	10.6	983
Lusk6.pr	RNG009-7	89.7	8.1	1 772	1713.2	77.0	178
P16lpo.cp	[Chr93]	23.1	8.1	1 661	58.1	42.1	1 721
gt4.3.pr	GRP014-1	0.7	2.4	3 623	6.5	4.2	196
jar10.2.1.pr	GRP051-1	47.3	3.1	1 140	>6000	161.0	3
mv2.pr	LCL111-2	225.8	16.5	1 487	>6000	136.5	274
p9a.pr	GRP178-1	7.3	7.1	2 253	90.2	25.9	807
ra2.pr	ROB023-1	6.3	2.8	2 356	>6000	118.0	21
ra3.pr	ROB005-14	96.1	5.6	1 174	>6000	100.4	16
sims2.cp	[Sim91]	50.8	3.8	807	365.5	61.6	443
z22.cp	[Den93]	0.9	2.5	2 726	3.7	3.9	1 743

Table 19: Comparison of Inference Machines

To that purpose, both provers were run with the same primitive search strategy. In this table, one can find not only a huge span between the timings (given in seconds of wall–clock–time) but also in the space needed (stated in MBytes). The column 'Red/sec' holds the number of reductions that have been performed per second.

10 Cracking Hard Nuts

In the previous sections, we have outlined how to achieve highly efficient deduction systems realizing a given inference system, which was demonstrated using unfailing Knuth–Bendix completion as an example. Our work has yielded a new prover for equational theories. Although the inference machine is not topped by intricate search strategies yet, we are already able to succeed in areas unconquered before. In [LW92], Lusk and Wos propose a set of "Benchmark Problems in Which Equality Plays the Major Role". Two of them characterized as never proved completely in single runs unaided will be tackled now.

10.1 A Problem from the Group Theory Domain

As a consequence of the chase for shorter and shorter single axioms for groups, the inverse problem of deriving the standard group axioms from a single equation arises. Problem gt6 is the single axiom

$$-(-(-(x+y)+(y+x))+(-(z+u)+(z-((v-w)-u))))+w=v$$

which even implies Abelian.

We started a first run using a standard selection strategy, namely picking that unselected equation with shortest left– and right–hand side. If stringterms had not been activated, the system performance would have decayed rapidly due to enormous memory consumption. We noticed that our prover found three of the group axioms, right inverse, Abelian and right identity, which caused the system of rules and equations to collapse. However, those sets grew steadily and aimlessly afterwards again; and the associativity law was not in sight.

Therefore we decided to derive profit from the collapse of R and E. In fact, invoking normalization of the whole set of unselected equations with respect to the now much stronger rewrite relation and subsequent reweighting — thus updating the selection strategy's assessment — helped to complete the proof immediately.

A careful analysis revealed that the last piece in the puzzle had been produced quite early in the run. Due to the length of the terms involved it was kind of hidden in the set *SUE*. The reduction procedure shortened the fact a lot and forced the selection strategy to pick it out.

10.2 A Problem from the Robbins Algebra Domain

A Robbins algebra is made up of an associative–commutative function $+$ ("or") and a unary one $\bar{\cdot}$ ("not") satisfying $\overline{\overline{x+y}+\overline{x+\overline{y}}}=y$ for every x and y. The algebra is a Boolean one as soon as one of the Boolean axioms is added. The problem ra4 is to prove this property for the addition of $\exists x,y : x+y=y$.

Problem TPTP or Reference	gt6.pr GRP084–1	ra4.pr ROB006–1
CPs generated	1 182 208	1 089 411
Rules + Equations	604	1 542
Maximum Size of CP Set	986 741	969 858
Total Match Queries	57 069 361	318 843 055
Total Reductions	175 423	5 551 423
Time (wall clock in sec)	255	3 062
Process–Size (Mbytes)	20.5	34.2
Generated CPs/sec	4 641	356
Reductions/sec	689	1 813
Match Queries/sec	224 055	104 119

Table 20: Cracking hard nuts (on SPARC–Ultra 1, 167 MHz)

Just like above, we did a first run with standard selection strategy and packing. Because of the AC–property of +, all the corresponding equivalence classes are enumerated, arbitrarily mixed up with negation symbols. Consequently it is worthwhile discriminating unselected equations the more disjunction symbols they contain.

The next proof run periodically produced a few rules and equations that simplified some elder ones. We expected these simplifying equalities to contribute to the proof problem. However, it was not before 1500 rules and equations had been produced that the *IRP* led to a success. All in all, the proof was found in a little more than one hour of computation time. See App. A for a complete proof–extract.

11 Concluding Remarks and Future Prospects

We described the design process of a completion–based prover for equational theories. Starting from a logical three–level model for saturation–based provers, we have followed an engineering approach throughout this design. Beginning with the inference system for unfailing Knuth–Bendix completion, we crystalized experiences into a sketch of algorithm and revealed the crucial points responsible for overall efficiency. We introduced sophisticated data structures and algorithms that avoid overindulgence in time and space. We determined an approriate aggregation of the inference steps into an inference machine. To that purpose we evaluated different parameter settings influencing this aggregation. Finally, we demonstrated the power of WALDMEISTER by a comparison with the DISCOUNT inference machine, and showed how two very hard problems have been solved by WALDMEISTER.

Our results suggest that the yet neglected level of the inference machine can influence the performance of the prover so far that comparisons between different search strategies or improved low–level operations can be completely blurred by unrecognized effects on the mid level (remember e.g. the comparison of equally weighted critical pairs or the realization of the normalization routine). Therefore one should never assess results without having these effects under strict control!

Having built a prover that is equivalently optimized on the two lower levels, i.e. basic operations and inference machine, we now can look for promising approaches on the top level. WALDMEISTER is able to do an increased number of derivations in given space and time bounds. Less effort has to be spent on focussing on those parts of the search space that are absolutely necessary for finding the proof. Hence, more general strategies can be employed, and hereby less adjusting has to be done by the user. Learning from past proof experience, and taking domain knowledge into account remain promising approaches. In the future, we want to provide automatic generation of appropriate reduction orderings at runtime. Furthermore, the treatment of goals (which are currently only reduced "on the fly") will be improved. Distributing automated theorem proving on a high level (e.g. employing the TEAMWORK method [Den93]) is equally encouraging, whereas parallelizing inference step execution on the lowest level will hardly outperform specialized indexing routines like those used in WALDMEISTER — for it requires adaption of the basic algorithms, and a lot of communication effort between processes.

As the normalization of critical pairs after generation has the main purpose to put the selection strategy in focus, we are currently working on strategies which base on the sets of different irreducible successors of each term. This will even lead to a stronger notion of 'joinable'.

Analyzing machine proofs by hand often suffers from the way these proofs are presented. Especially proofs found via completion can be very spacious and unwieldy. Schulz and Denzinger introduced a protocol language, along with methods and tools for extracting and presenting proofs from those protocols (cf. [DS93]). We are currently adapting this method for WALDMEISTER and are working on space saving protocol

mechanisms since the original language produces protocol files that exhaust even large hard disk capacities.

Summing up, our system WALDMEISTER, which was designed according to a systematic stepwise design process, does not only show an outstanding performance. Distinguishing three levels in the system's open architecture leaves a broad variety of additional choice points up to the user. By this means, the statistically ascertained default parameterization is adjustable to special proof tasks. Combined with the high performance on the level of inference step execution, even standard selection strategies succeed in problems known as very hard.

WALDMEISTER and further publications will soon be available via internet at:
`http://www.uni-kl.de/AG-AvenhausMadlener/waldmeister.html`

A Proof of RA4

Axioms

Axiom 1: $x_2 + x_1 = x_1 + x_2$

Axiom 2: $c + d = d$

Axiom 3: $(x_3 + x_2) + x_1 = x_3 + (x_2 + x_1)$

Axiom 4: $\overline{\overline{x_2 + x_1} + \overline{x_2 + \overline{x_1}}} = x_2$

Theorem

Theorem: $\overline{\overline{a} + b} + \overline{\overline{a} + \overline{b}} = a$

Proof:

Lemma 1: $d + c = d$

$$\underbrace{d + c}$$

$=$ by Axiom 1 RL with $\{x_1 \leftarrow d, x_2 \leftarrow c\}$

$$\underbrace{c + d}$$

$=$ by Axiom 2 LR with $\{\}$

$$\overbrace{d}$$

Lemma 2: $d + c + x_1 = d + x_1$

$$\underbrace{d + c} + x_1$$

$=$ by Lemma 1 LR with $\{\}$

$$\overbrace{d} + x_1$$

Lemma 3: $\overline{\overline{z + u} + \overline{u + \overline{z}}} = u$

$$\underbrace{\overline{\overline{z + u} + \overline{u + \overline{z}}}}$$

$=$ by Axiom 1 RL with $\{x_1 \leftarrow z, x_2 \leftarrow u\}$

$$\underbrace{\overline{\overbrace{u + z} + \overline{u + \overline{z}}}}$$

$=$ by Axiom 4 LR with $\{x_1 \leftarrow z, x_2 \leftarrow u\}$

$$\overbrace{u}$$

Lemma 4: $\overline{\overline{z + x_1} + \overline{\overline{x_1} + z}} = z$

$$\overline{\overline{z + x_1} + \underbrace{\overline{x_1} + z}}$$

$=$ by Axiom 1 RL with $\{x_1 \leftarrow \overline{x_1}, x_2 \leftarrow z\}$

$$\underbrace{\overline{z + x_1} + \overbrace{z + \overline{x_1}}}$$

$=$ by Axiom 4 LR with $\{x_1 \leftarrow x_1, x_2 \leftarrow z\}$

$\overbrace{z}$

Lemma 5: $\overline{\overline{z+v}+\overline{\overline{z}+v}} = v$

$\overline{\overline{z+v}+\underbrace{\overline{\overline{z}+v}}}$

$=$ by Axiom 1 RL with $\{x_1 \leftarrow \overline{z}, x_2 \leftarrow v\}$

$\overline{\underbrace{\overline{z+v}+\overbrace{v+\overline{z}}}}$

$=$ by Lemma 3 LR with $\{u \leftarrow v, z \leftarrow z\}$

$\overbrace{v}$

Lemma 6: $u+z+x_1 = z+u+x_1$

$\underbrace{u+z}+x_1$

$=$ by Axiom 1 LR with $\{x_1 \leftarrow z, x_2 \leftarrow u\}$

$\overbrace{z+u}+x_1$

Lemma 7: $\overline{c+\overline{d}} = \overline{\overline{c+\overline{d}+d+c}}$

$\underbrace{\overline{c+\overline{d}}}$

$=$ by Lemma 4 RL with $\{x_1 \leftarrow d, z \leftarrow \overline{c+\overline{d}}\}$

$\overbrace{\overline{\overline{c+\overline{d}}+d+\underbrace{\overline{d}}+\overline{c+\overline{d}}}}$

$=$ by Lemma 1 RL with $\{\}$

$\overline{\overline{\overline{c+\overline{d}}+d+\overbrace{\overline{d+c}+\overline{c+\overline{d}}}}}$

$=$ by Lemma 3 LR with $\{u \leftarrow c, z \leftarrow d\}$

$\overline{\overline{\overline{c+\overline{d}}+d+\overbrace{c}}}$

Lemma 8: $v+u+z = z+u+v$

$\underbrace{v+u+z}$

$=$ by Axiom 1 LR with $\{x_1 \leftarrow z, x_2 \leftarrow v+u\}$

$\overbrace{z+\underbrace{v+u}}$

$=$ by Axiom 1 LR with $\{x_1 \leftarrow u, x_2 \leftarrow v\}$

$z+\overbrace{u+v}$

Lemma 9: $\overline{\overline{w+u}+\overline{\overline{y+u}+w+\overline{u+\overline{y}}}} = w$

$\overline{\overline{w+u}+\underbrace{\overline{\overline{y+u}+w+\overline{u+\overline{y}}}}}$

$=$ by Lemma 8 RL with $\{z \leftarrow \overline{y+u}, u \leftarrow w, v \leftarrow \overline{u+\overline{y}}\}$

$\underbrace{\overline{\overline{w+u}+\overbrace{\overline{u+\overline{y}}+w+\overline{y+u}}}}$

$$
\begin{aligned}
&= \quad \text{by Axiom 1 RL with } \{x_1 \leftarrow \overline{w+u}, x_2 \leftarrow \overline{\overline{u+\overline{y}}+w+\overline{y+u}}\} \\
&\overbrace{\overline{\overline{u+\overline{y}}+w+\overline{y+u}}+\overline{w+\underbrace{u}}} \\
&= \quad \text{by Lemma 3 RL with } \{u \leftarrow u, z \leftarrow y\} \\
&\underbrace{\overline{\overline{u+\overline{y}}+w+\overline{y+u}}}+w+\overbrace{\overline{\overline{y+u}+\overline{u+\overline{y}}}} \\
&= \quad \text{by Axiom 1 RL with } \{x_1 \leftarrow \overline{u+\overline{y}}, x_2 \leftarrow w+\overline{y+u}\} \\
&\underbrace{\overbrace{\overline{w+\overline{y+u}+\overline{u+\overline{y}}}}+\overline{w+\overline{\overline{y+u}+\overline{u+\overline{y}}}}} \\
&= \quad \text{by Axiom 4 LR with } \{x_1 \leftarrow \overline{y+u}+\overline{u+\overline{y}}, x_2 \leftarrow w\} \\
&\overbrace{w}
\end{aligned}
$$

Lemma 10: $\overline{d+\overline{c+\overline{d}}} = \overline{d}$

$$
\begin{aligned}
&\underbrace{\overline{d+\overline{c+\overline{d}}}} \\
&= \quad \text{by Lemma 5 RL with } \{v \leftarrow \overline{d+\overline{c+\overline{d}}}, z \leftarrow c+\overline{d}\} \\
&\overbrace{\overline{c+\overline{d}+d+\overline{c+\overline{d}}}+\overline{c+\overline{d}}+\underbrace{d+\overline{c+\overline{d}}}} \\
&= \quad \text{by Lemma 2 RL with } \{x_1 \leftarrow \overline{c+\overline{d}}\} \\
&\overline{\overline{c+\overline{d}+d+\overline{c+\overline{d}}}+\overline{c+\overline{d}}+\underbrace{\overbrace{d+c+\overline{c+\overline{d}}}}} \\
&= \quad \text{by Lemma 6 RL with } \{x_1 \leftarrow \overline{c+\overline{d}}, z \leftarrow d, u \leftarrow c\} \\
&\overline{\overline{c+\overline{d}+d+\overline{c+\overline{d}}}+\underbrace{\overline{c+\overline{d}}+\overbrace{c+d+\overline{c+\overline{d}}}}} \\
&= \quad \text{by Axiom 1 RL with } \{x_1 \leftarrow \overline{c+\overline{d}}, x_2 \leftarrow \overline{c+d+\overline{c+\overline{d}}}\} \\
&\overline{\overline{c+\overline{d}+d+\overline{c+\overline{d}}}+\overbrace{\overline{c+d+\overline{c+\overline{d}}}+\underbrace{\overline{c+\overline{d}}}}} \\
&= \quad \text{by Lemma 7 LR with } \{\} \\
&\overline{\overline{c+\overline{d}+d+\overline{c+\overline{d}}}+\overline{c+d+\overline{c+\overline{d}}}+\overbrace{\underbrace{c+\overline{d}+d+c}}} \\
&= \quad \text{by Axiom 1 LR with } \{x_1 \leftarrow c, x_2 \leftarrow \overline{c+\overline{d}+d}\} \\
&\overline{\overline{c+\overline{d}+d+\overline{c+\overline{d}}}+\overline{c+d+\overline{c+\overline{d}}}, \overbrace{c+\underbrace{c+\overline{d}+d}}} \\
&= \quad \text{by Axiom 1 LR with } \{x_1 \leftarrow d, x_2 \leftarrow \overline{c+\overline{d}}\} \\
&\overline{\overline{c+\overline{d}+d+\overline{c+\overline{d}}}+\underbrace{\overline{c+d+\overline{c+\overline{d}}}+\overbrace{c+d+\overline{c+\overline{d}}}}} \\
&= \quad \text{by Axiom 4 LR with } \{x_1 \leftarrow d+\overline{c+\overline{d}}, x_2 \leftarrow c\} \\
&\underbrace{\overline{c+\overline{d}+d+\overline{c+\overline{d}}}+\overbrace{c}}
\end{aligned}
$$

$=$ by Axiom 1 LR with $\{x_1 \leftarrow c, x_2 \leftarrow \overline{c+\overline{d}+d+c+\overline{d}}\}$

$$\overline{c+\underbrace{c+\overline{d}+d+c+\overline{d}}}$$

$=$ by Lemma 6 RL with $\{x_1 \leftarrow \overline{d+\overline{c+\overline{d}}}, z \leftarrow c, u \leftarrow \overline{d}\}$

$$\overline{c+\underbrace{\overline{d}}+c+\overline{d+\overline{c+\overline{d}}}}$$

$=$ by Lemma 1 RL with $\{\}$

$$\overline{\underbrace{c}+\overbrace{\overline{d+c}}+c+\overline{d+\overline{c+\overline{d}}}}$$

$=$ by Lemma 3 RL with $\{u \leftarrow c, z \leftarrow d\}$

$$\underbrace{\overbrace{\overline{d+c}+c+\overline{d}}+\overline{d+c+c+d+\overline{c+\overline{d}}}}$$

$=$ by Axiom 1 RL with $\{x_1 \leftarrow \overline{\overline{d+c}+c+\overline{d}}, x_2 \leftarrow \overline{\overline{d+c}+c+d+\overline{c+\overline{d}}}\}$

$$\overline{\overbrace{\overline{d+c}+c+d+\overline{c+\overline{d}}}+\overline{d+c}+\underbrace{c+\overline{d}}}$$

$=$ by Lemma 7 LR with $\{\}$

$$\overline{\overline{d+c}+c+d+\overline{c+\overline{d}}+\overline{d+c}+\overbrace{\overbrace{c+\overline{d}+d}+c}}$$

$=$ by Axiom 1 LR with $\{x_1 \leftarrow c, x_2 \leftarrow \overline{c+\overline{d}+d}\}$

$$\overline{\overline{d+c}+c+d+\overline{c+\overline{d}}+\overline{d+c}+c+\overbrace{\overbrace{c+\overline{d}+d}}}$$

$=$ by Axiom 1 LR with $\{x_1 \leftarrow d, x_2 \leftarrow c+\overline{d}\}$

$$\underbrace{\overline{\overline{d+c}+c+d+\overline{c+\overline{d}}+\overline{d+c}+c+\overbrace{d+\overline{c+\overline{d}}}}}$$

$=$ by Axiom 4 LR with $\{x_1 \leftarrow c+\overline{d+\overline{c+\overline{d}}}, x_2 \leftarrow \overline{d+c}\}$

$$\underbrace{\overbrace{d+c}}$$

$=$ by Lemma 1 LR with $\{\}$

$$\overbrace{\overline{d}}$$

Lemma 11: $d = \overline{\overline{d+\overline{d}}+d+\overline{d}+d+\overline{d}}$

$$\underbrace{d}$$

$=$ by Axiom 4 RL with $\{x_1 \leftarrow \overline{d+c+\overline{d}}+\overline{d+c+\overline{d}}, x_2 \leftarrow d\}$

$$\overline{d+\overline{d+c+\overline{d}}+d+c+\overline{d}+\underbrace{d}+\overbrace{d+c+\overline{d}+d+c+\overline{d}}}$$

$=$ by Axiom 4 RL with $\{x_1 \leftarrow c+\overline{d}, x_2 \leftarrow d\}$

$$\overline{d+\overline{d+c+\overline{d}}+d+c+\overline{d}+\underbrace{\overbrace{d+c+\overline{d}+d+c+\overline{d}}+\overline{d+c+\overline{d}+d+c+\overline{d}}}}$$

$$= \quad \text{by Axiom 4 LR with } \{x_1 \leftarrow \overline{d+c+\overline{\overline{d}}}, x_2 \leftarrow \overline{d+c+\overline{d}}\}$$

$$\overline{\underbrace{d+\overline{d+c+\overline{\overline{d}}}+d+c+\overline{d}}+\overbrace{d+c+\overline{d}}}$$

$$= \quad \text{by Axiom 1 LR with } \{x_1 \leftarrow \overline{d+c+\overline{\overline{d}}}, x_2 \leftarrow \overline{d+\overline{d+c+\overline{\overline{d}}}+d+c+\overline{d}}\}$$

$$\overline{\overbrace{d+c+\overline{\overline{d}}+d+\underbrace{d+c+\overline{\overline{d}}}}+d+c+\overline{d}}$$

$$= \quad \text{by Axiom 1 LR with } \{x_1 \leftarrow c+\overline{\overline{d}}, x_2 \leftarrow d\}$$

$$\overline{d+c+\overline{\overline{d}}+d+\overbrace{c+\overline{\overline{d}}+d}+\underbrace{d+c+\overline{\overline{d}}}}$$

$$= \quad \text{by Lemma 10 LR with } \{\}$$

$$\overline{\underbrace{d+c+\overline{\overline{d}}}+d+\overline{c+\overline{\overline{d}}+d}+\overbrace{\overline{\overline{d}}}}$$

$$= \quad \text{by Lemma 2 LR with } \{x_1 \leftarrow \overline{\overline{d}}\}$$

$$\overline{\overbrace{d+\overline{\overline{d}}}+d+\underbrace{c+\overline{\overline{d}}+d+\overline{\overline{d}}}}$$

$$= \quad \text{by Axiom 1 LR with } \{x_1 \leftarrow \overline{\overline{d}}, x_2 \leftarrow \overline{c+\overline{\overline{d}}+d}\}$$

$$\overline{d+\overline{\overline{d}}+d+\overbrace{\overline{\overline{d}}+\underbrace{c+\overline{\overline{d}}+d}}}$$

$$= \quad \text{by Lemma 8 LR with } \{z \leftarrow d, u \leftarrow \overline{\overline{d}}, v \leftarrow c\}$$

$$\overline{d+\overline{\overline{d}}+d+\overline{\overline{d}}+\overbrace{d+\underbrace{\overline{\overline{d}}+c}}}$$

$$= \quad \text{by Axiom 1 RL with } \{x_1 \leftarrow \overline{\overline{d}}, x_2 \leftarrow c\}$$

$$\overline{d+\overline{\overline{d}}+d+\overline{\overline{d}}+\underbrace{d+\overbrace{c+\overline{d}}}}$$

$$= \quad \text{by Lemma 2 LR with } \{x_1 \leftarrow \overline{\overline{d}}\}$$

$$\overline{d+\overline{\overline{d}}+d+\overline{\overline{d}}+\overbrace{d+\overline{\overline{d}}}}$$

Lemma 12: $\overline{\overline{d}} = \overline{d+d+\overline{d}+d+\overline{\overline{d}}+d+\overline{\overline{d}}}$

$$\underbrace{\overline{\overline{d}}}$$

$$= \quad \text{by Lemma 9 RL with } \{y \leftarrow \overline{\overline{d}}+d, u \leftarrow \overline{d+\overline{\overline{d}}}, w \leftarrow \overline{\overline{d}}\}$$

$$\overline{\underbrace{\overline{d}+d+\overline{\overline{d}}}+\overline{\overline{d}+d+d+\overline{d}+\overline{d}+d+\overline{d}+\overline{\overline{d}}+\overline{\overline{d}}+d}}$$

$$= \quad \text{by Axiom 1 RL with } \{x_1 \leftarrow \overline{\overline{d}}, x_2 \leftarrow \overline{d+\overline{\overline{d}}}\}$$

$$\overline{\overbrace{d+\overline{\overline{d}}}+\overline{\overline{d}}+\overline{\overline{d}+d+d+\overline{d}+\overline{d}+d+\overline{d}+\overline{\overline{d}}+\overline{\overline{d}}+d}}$$

$$= \quad \text{by Lemma 2 RL with } \{x_1 \leftarrow \overline{\overline{d}}\}$$

$$\overline{\overbrace{d+c+\overline{\overline{d}}}+\underbrace{\overline{\overline{d}}}+\overline{\overline{d}+d+d+\overline{d}+\overline{d}+d+\overline{d}+\overline{\overline{d}}+\overline{\overline{d}}+d}}$$

$=$ by Lemma 10 RL with $\{\}$

$$\underbrace{\overline{d+c+\overline{d}}}+\overbrace{d+\overline{c+\overline{d}}}+\overline{\overline{d}+d+\overline{d+\overline{d}}+\overline{d}+\overline{d+\overline{d}}+\overline{\overline{d}}+d}$$

$=$ by Axiom 4 LR with $\{x_1 \leftarrow c+\overline{d}, x_2 \leftarrow d\}$

$$\overbrace{d}+\underbrace{\overline{\overline{d}+d+d+\overline{d}}+\overline{d}+\overline{d+\overline{d}}+\overline{\overline{d}}+d}$$

$=$ by Lemma 6 LR with $\{x_1 \leftarrow \overline{d+\overline{d}+\overline{\overline{d}}+d}, z \leftarrow \overline{d}, u \leftarrow \overline{\overline{d}+d+d+\overline{d}}\}$

$$\overbrace{d+\overline{d}+\overline{\overline{d}+d+d+\overline{d}}+\underbrace{\overline{d+\overline{d}+\overline{\overline{d}}+d}}}$$

$=$ by Lemma 4 LR with $\{x_1 \leftarrow \overline{d}, z \leftarrow d\}$

$$d+\underbrace{\overline{d}+\overline{\overline{d}+d+d+\overline{d}}+\overbrace{d}}$$

$=$ by Lemma 8 LR with $\{z \leftarrow d, u \leftarrow \overline{\overline{d}+d+\overline{d+\overline{d}}}, v \leftarrow \overline{d}\}$

$$\overbrace{d+d+\underbrace{\overline{\overline{d}+d+d+\overline{d}}+\overline{d}}}$$

$=$ by Axiom 1 LR with $\{x_1 \leftarrow \overline{d}, x_2 \leftarrow \overline{\overline{d}+d+\overline{d+\overline{d}}}\}$

$$d+d+\overbrace{\overline{d}+\underbrace{\overline{\overline{d}+d+d+\overline{d}}}}$$

$=$ by Lemma 6 LR with $\{x_1 \leftarrow \overline{d+\overline{d}}, z \leftarrow d, u \leftarrow \overline{\overline{d}}\}$

$$d+d+\overline{d}+\overbrace{d+\overline{\overline{d}+d+\overline{d}}}$$

Lemma 13: $\overline{\overline{c+z+c}+z+\overline{d}+c+c+\overline{\overline{d}}} = z$

$$\underbrace{\overline{\overline{c+z+c}+z+\overline{d}+\overline{c+c+\overline{d}}}}$$

$=$ by Lemma 6 RL with $\{x_1 \leftarrow c, z \leftarrow c, u \leftarrow z\}$

$$\underbrace{\overbrace{\overline{z+c+c}+\overline{z+\overline{d}+c+c+\overline{d}}}}$$

$=$ by Axiom 1 RL with $\{x_1 \leftarrow \overline{z+c+c}, x_2 \leftarrow \overline{z+\overline{d}+c+c+\overline{d}}\}$

$$\overbrace{z+\overline{d}+\overline{c+c+\overline{d}}+\overline{z+\underbrace{c+c}}}$$

$=$ by Lemma 3 RL with $\{u \leftarrow c+c, z \leftarrow d\}$

$$\overline{z+\overline{d}+\overline{c+c+\overline{d}}}+z+\overbrace{\underbrace{\overline{d+c+c}}+\overline{c+c+\overline{d}}}$$

$=$ by Lemma 2 LR with $\{x_1 \leftarrow c\}$

$$\overline{z+\overline{d}+\overline{c+c+\overline{d}}}+z+\overbrace{\underbrace{\overbrace{d+c}}+\overline{c+c+\overline{d}}}$$

$=$ by Lemma 1 LR with $\{\}$

$$\underbrace{\overline{z+\overline{d}+\overline{c+c+\overline{d}}}+z+\overbrace{d}+\overline{c+c+\overline{d}}}$$

$=$ by Axiom 4 LR with $\{x_1 \leftarrow \overline{d}+\overline{c+c+\overline{d}}, x_2 \leftarrow z\}$

$$\overbrace{z}$$

Lemma 14: $c + \overline{d + \overline{\overline{d}}} = \overline{\overline{\overline{d} + c + \overline{\overline{d}} + d + \overline{d + \overline{d}}}}$

$\underbrace{c + \overline{d + \overline{\overline{d}}}}$

$=$ by Lemma 4 RL with $\{x_1 \leftarrow \overline{\overline{d}} + d, z \leftarrow c + \overline{d + \overline{\overline{d}}}\}$

$\overbrace{\underbrace{c + \overline{d + \overline{\overline{d}}} + \overline{\overline{d}} + d}} + \overline{\overline{d} + d + c + \overline{d + \overline{\overline{d}}}}}$

$=$ by Lemma 8 LR with $\{z \leftarrow d, u \leftarrow \overline{\overline{d}}, v \leftarrow c + \overline{d + \overline{\overline{d}}}\}$

$\overbrace{d + \underbrace{\overline{\overline{d}} + c} + \overline{d + \overline{\overline{d}}}} + \overline{\overline{\overline{d} + d + c + \overline{d + \overline{\overline{d}}}}}$

$=$ by Axiom 1 RL with $\{x_1 \leftarrow \overline{\overline{d}}, x_2 \leftarrow c\}$

$\underbrace{d + \overbrace{c + \overline{\overline{d}}}} + \overline{d + \overline{\overline{d}}} + \overline{\overline{\overline{d} + d + c + \overline{d + \overline{\overline{d}}}}}$

$=$ by Lemma 2 LR with $\{x_1 \leftarrow \overline{\overline{d}}\}$

$\overbrace{d + \overline{\overline{d}}} + \overline{d + \overline{\overline{d}}} + \overline{\underbrace{\overline{\overline{d}} + d + c + \overline{d + \overline{\overline{d}}}}}$

$=$ by Lemma 6 LR with $\{x_1 \leftarrow \overline{d + \overline{\overline{d}}}, z \leftarrow c, u \leftarrow \overline{\overline{\overline{d}} + d}\}$

$\overline{\underbrace{d + \overline{\overline{d}} + \overline{d + \overline{\overline{d}}}}} + \overbrace{c + \overline{\overline{\overline{d}} + d + \overline{d + \overline{\overline{d}}}}}$

$=$ by Lemma 5 RL with $\{v \leftarrow \overline{d + \overline{\overline{d}} + d + \overline{\overline{d}}}, z \leftarrow d + \overline{\overline{d}}\}$

$\overline{\overbrace{d + \overline{\overline{d}} + d + \overline{\overline{d}} + d + \overline{\overline{d}}} + \underbrace{\overline{d + \overline{\overline{d}} + d + \overline{\overline{d}} + d + \overline{\overline{d}}}} + c + \overline{\overline{\overline{d}} + d + \overline{d + \overline{\overline{d}}}}}$

$=$ by Lemma 11 RL with $\{\}$

$\overline{\underbrace{d + \overline{\overline{d}} + d + \overline{\overline{d}} + d + \overline{\overline{d}} + \overbrace{d}}} + \overline{c + \overline{\overline{d}} + d + \overline{d + \overline{\overline{d}}}}$

$=$ by Axiom 1 LR with $\{x_1 \leftarrow d, x_2 \leftarrow \overline{d + \overline{\overline{d}} + d + \overline{\overline{d}} + d + \overline{\overline{d}}}\}$

$\overline{\overbrace{\underbrace{d + d + \overline{\overline{d}} + d + \overline{\overline{d}} + d + \overline{\overline{d}}}}} + c + \overline{\overline{\overline{d}} + d + \overline{d + \overline{\overline{d}}}}$

$=$ by Lemma 12 RL with $\{\}$

$\overbrace{\overline{\overline{d}}} + c + \overline{\overline{\overline{d}} + d + \overline{d + \overline{\overline{d}}}}$

Lemma 15: $p + \overline{d + \overline{\overline{d}}} = \overline{\overline{p + \overline{d} + \overline{\overline{d}} + p + \overline{d + \overline{\overline{d}}}}}$

$\underbrace{p + \overline{d + \overline{\overline{d}}}}$

$=$ by Lemma 5 RL with $\{v \leftarrow p + \overline{d + \overline{\overline{d}}}, z \leftarrow d\}$

$\overbrace{\underbrace{d + p + \overline{d + \overline{\overline{d}}}} + \overline{\overline{d} + p + \overline{d + \overline{\overline{d}}}}}$

$=$ by Lemma 6 LR with $\{x_1 \leftarrow \overline{d + \overline{\overline{d}}}, z \leftarrow p, u \leftarrow d\}$

$\overbrace{p + \underbrace{d + \overline{d + \overline{\overline{d}}}}} + \overline{\overline{d} + p + \overline{d + \overline{\overline{d}}}}$

$$
\begin{array}{ll}
= & \text{by Lemma 2 RL with } \{x_1 \leftarrow \overline{d+\overline{d}}\} \\
& \overline{\overbrace{p+d+\underbrace{c+\overline{d+\overline{d}}}}+\overline{\overline{d}+p+\overline{d+\overline{d}}}} \\
= & \text{by Lemma 14 LR with } \{\} \\
& \overline{\overbrace{p+d+\overline{d}+c+\underbrace{\overline{\overline{d}+d}+\overline{d+\overline{d}}}}+\overline{\overline{d}+p+\overline{d+\overline{d}}}} \\
= & \text{by Axiom 1 LR with } \{x_1 \leftarrow \overline{d+\overline{d}}, x_2 \leftarrow \overline{\overline{d}+d}\} \\
& \overline{\overbrace{p+d+\overline{d}+c+\overline{d+\overline{d}}+\underbrace{\overline{\overline{d}+d}}}+\overline{\overline{d}+p+\overline{d+\overline{d}}}} \\
= & \text{by Axiom 1 LR with } \{x_1 \leftarrow d, x_2 \leftarrow \overline{\overline{d}}\} \\
& \overline{p+d+\underbrace{\overline{d}}+c+\overline{d+\overline{d}}+\overbrace{d+\overline{\overline{d}}}+\overline{\overline{d}+p+\overline{d+\overline{d}}}} \\
= & \text{by Lemma 1 RL with } \{\} \\
& \overline{p+d+\overbrace{\underbrace{\overline{d+c}}+c+\overline{d+\overline{d}}+d+\overline{\overline{d}}}+\overline{\overline{d}+p+\overline{d+\overline{d}}}} \\
= & \text{by Axiom 1 RL with } \{x_1 \leftarrow \overline{d+c}, x_2 \leftarrow \overline{c+\overline{d+\overline{d}}+d+\overline{\overline{d}}}\} \\
& \overline{\overbrace{p+d+c+\overline{d+\overline{d}}+d+\overline{\overline{d}}+\underbrace{\overline{d}+c}}+\overline{\overline{d}+p+\overline{d+\overline{d}}}} \\
= & \text{by Axiom 4 RL with } \{x_1 \leftarrow \overline{d}, x_2 \leftarrow d\} \\
& \overline{p+d+\underbrace{\overline{c+\overline{d+\overline{d}}+d+\overline{\overline{d}}}+\overbrace{\overline{d+\overline{d}}+d+\overline{\overline{d}}}+c}+\overline{\overline{d}+p+\overline{d+\overline{d}}}} \\
= & \text{by Lemma 4 LR with } \{x_1 \leftarrow \overline{d+\overline{d}}+\overline{d+\overline{\overline{d}}}, z \leftarrow c\} \\
& \overline{p+\underbrace{d+\overbrace{c}}+\overline{\overline{d}+p+\overline{d+\overline{d}}}} \\
= & \text{by Lemma 1 LR with } \{\} \\
& \overline{p+\overbrace{d}+\overline{\overline{d}+p+\overline{d+\overline{d}}}}
\end{array}
$$

Lemma 16: $p+\overline{d+\overline{d}}=p$

$$
\begin{array}{ll}
& \underbrace{p+\overline{d+\overline{d}}} \\
= & \text{by Lemma 15 LR with } \{p \leftarrow p\} \\
& \overbrace{\overline{p+d}+\overline{\overline{d}+p+\underbrace{d+\overline{d}}}} \\
= & \text{by Lemma 2 RL with } \{x_1 \leftarrow \overline{d}\} \\
& \overline{\overline{p+d}+\overline{\overline{d}+p+\overbrace{d+\underbrace{c+\overline{d}}}}} \\
= & \text{by Axiom 1 LR with } \{x_1 \leftarrow \overline{d}, x_2 \leftarrow c\} \\
& \overline{\overline{p+d}+\overline{\overline{d}+p+\underbrace{d+\overbrace{\overline{d}+c}}}}
\end{array}
$$

$=$ by Lemma 8 RL with $\{z \leftarrow d, u \leftarrow \overline{d}, v \leftarrow c\}$

$$\overline{\overline{p+d}+\underbrace{\overline{d}+p+\overbrace{c+\overline{d}+d}}}$$

$=$ by Lemma 8 RL with $\{z \leftarrow \overline{d}, u \leftarrow p, v \leftarrow \overline{c+\overline{d}+d}\}$

$$\overline{\overline{p+d}+\overbrace{\overline{c+\overline{d}+d}+p+\underbrace{\overline{d}}}}$$

$=$ by Lemma 10 RL with $\{\}$

$$\underbrace{\overline{p+d}+\overline{c+\overline{d}+d}+p+\overbrace{\overline{d+c+\overline{d}}}}$$

$=$ by Lemma 9 LR with $\{y \leftarrow c+\overline{d}, u \leftarrow d, w \leftarrow p\}$

$$\overbrace{p}$$

Lemma 17: $\overline{\overline{\overline{u}}} = \overline{u}$

$$\underbrace{\overline{\overline{\overline{u}}}}$$

$=$ by Lemma 16 RL with $\{p \leftarrow \overline{\overline{u}}\}$

$$\underbrace{\overbrace{\overline{\overline{u}}+\overline{d+\overline{d}}}}$$

$=$ by Lemma 5 RL with $\{v \leftarrow \overline{\overline{\overline{u}}+d+\overline{\overline{d}}}, z \leftarrow \overline{d+\overline{d}}+\overline{u}\}$

$$\overbrace{\overline{d+\overline{d}+\overline{u}+\overline{\overline{u}}+d+\overline{\overline{d}}}+\underbrace{\overbrace{\overline{d+\overline{d}+\overline{u}+\overline{\overline{u}}+d+\overline{d}}}}}$$

$=$ by Lemma 4 LR with $\{x_1 \leftarrow \overline{u}, z \leftarrow \overline{d+\overline{d}}\}$

$$\underbrace{\overline{\overline{d+\overline{d}+\overline{u}+\overline{\overline{u}}+d+\overline{d}}}+\overbrace{\overline{d+\overline{d}}}}$$

$=$ by Lemma 16 LR with $\{p \leftarrow \overline{d+\overline{d}+\overline{u}+\overline{\overline{u}}+d+\overline{d}}\}$

$$\underbrace{\overbrace{\overline{d+\overline{d}+\overline{u}+\overline{\overline{u}}+d+\overline{d}}}}$$

$=$ by Axiom 1 RL with $\{x_1 \leftarrow \overline{d+\overline{d}}, x_2 \leftarrow \overline{u}+\overline{\overline{u}}+\overline{d+\overline{d}}\}$

$$\underbrace{\overbrace{\overline{u}+\overline{\overline{u}}+\overline{d+\overline{d}}+\overline{d+\overline{d}}}}$$

$=$ by Lemma 16 LR with $\{p \leftarrow \overline{u}+\overline{\overline{u}}+\overline{d+\overline{d}}\}$

$$\overbrace{\overline{u}+\underbrace{\overline{\overline{u}}+\overline{d+\overline{d}}}}$$

$=$ by Lemma 16 LR with $\{p \leftarrow \overline{\overline{u}}\}$

$$\overline{\overline{u}}+\underbrace{\overbrace{\overline{\overline{u}}}}$$

$=$ by Lemma 16 RL with $\{p \leftarrow \overline{u}\}$

$$\overline{\underbrace{\overline{u}}+\overbrace{\overline{u}+\overline{d+\overline{d}}}}$$

$$
\begin{aligned}
&= \quad \text{by Lemma 16 RL with } \{p \leftarrow u\} \\
&\overbrace{\overbrace{\overline{\overline{u}+d+\overline{d}}}^{}+\overline{\overline{\overline{u}+d+\overline{d}}}}^{} \\
&= \quad \text{by Lemma 16 RL with } \{p \leftarrow \overline{\overline{u}+d+\overline{d}}+\overline{\overline{\overline{u}+d+\overline{d}}}\} \\
&\overbrace{\overbrace{\overline{\overline{u}+d+\overline{d}}+\overline{\overline{\overline{u}+d+\overline{d}}}}^{}+\overline{\overline{\overline{u}+d+\overline{d}}+\overline{\overline{\overline{u}+d+\overline{d}}}}}^{} \\
&= \quad \text{by Axiom 1 LR with } \{x_1 \leftarrow \overline{d+\overline{d}}, x_2 \leftarrow \overline{\overline{u}+d+\overline{d}}+\overline{\overline{\overline{u}+d+\overline{d}}}\} \\
&\overbrace{\overbrace{\overline{d+\overline{d}}+\overline{\overline{u}+d+\overline{d}}}^{}+\overline{\overline{\overline{u}+d+\overline{d}}}}^{} \\
&= \quad \text{by Lemma 5 RL with } \{v \leftarrow \overline{d+\overline{d}}, z \leftarrow u\} \\
&\overbrace{\overbrace{\overline{\overline{u}+d+\overline{d}}+\overline{\overline{\overline{u}+d+\overline{d}}}}^{}+\overline{\overline{u}+d+\overline{d}}+\overline{\overline{\overline{u}+d+\overline{d}}}}^{} \\
&= \quad \text{by Axiom 4 LR with } \{x_1 \leftarrow \overline{\overline{u}+d+\overline{d}}, x_2 \leftarrow \overline{u+d+\overline{d}}\} \\
&\overbrace{\overline{\overline{u}+d+\overline{d}}}^{} \\
&= \quad \text{by Lemma 16 LR with } \{p \leftarrow u\} \\
&\overbrace{\overline{u}}^{}
\end{aligned}
$$

Lemma 18: $y = \overline{\overline{y}}$

$$
\begin{aligned}
&\overbrace{y}^{} \\
&= \quad \text{by Lemma 13 RL with } \{z \leftarrow y\} \\
&\overbrace{\overline{\overline{c+y+c}+y+\overline{d}+c+c+\overline{d}}}^{} \\
&= \quad \text{by Lemma 17 RL with } \{u \leftarrow \overline{\overline{c+y+c}+y+\overline{d}+c+c+\overline{d}}\} \\
&\overbrace{\overline{\overline{\overline{c+y+c}+y+\overline{d}+c+c+\overline{d}}}}^{} \\
&= \quad \text{by Lemma 13 LR with } \{z \leftarrow y\} \\
&\overbrace{\overline{\overline{y}}}^{}
\end{aligned}
$$

Theorem: $\overline{\overline{a}+b}+\overline{\overline{a}+\overline{b}}=a$

$\overbrace{\overline{\overline{a}+b}}+\overline{\overline{a}+\overline{b}}$

$=$ by Axiom 1 LR with $\{x_1 \leftarrow b, x_2 \leftarrow \overline{a}\}$

$\overline{\overbrace{b+\overline{a}}}+\overbrace{\overline{\overline{a}+\overline{b}}}$

$=$ by Axiom 1 LR with $\{x_1 \leftarrow \overline{b}, x_2 \leftarrow \overline{a}\}$

$\overbrace{\overline{b+\overline{a}}+\overline{\overbrace{\overline{b}+\overline{a}}}}$

$=$ by Lemma 18 LR with $\{y \leftarrow \overline{b+\overline{a}}+\overline{\overline{b}+\overline{a}}\}$

$\overbrace{\overline{\overline{b+\overline{a}}+\overline{\overline{b}+\overline{a}}}}$

$=$ by Lemma 5 LR with $\{v \leftarrow \overline{a}, z \leftarrow b\}$

$\overbrace{\overline{\overbrace{\overline{a}}}}$

$=$ by Lemma 18 RL with $\{y \leftarrow a\}$

$\overbrace{a}$

The proof–extract was generated from a WALDMEISTER protocol using the PCL tools of J. Denzinger and S. Schulz (cf. [DS93]).

B Examples

The specification format is straightforward.

- `NAME` `foo`

  A (hopefully meaningful) name for the subsequent specification.

- `MODE` `(COMPLETION | CONFLUENCE | PROOF | REDUCTION)`

  What is the system expected to do with this specification?
  - COMPLETION: Derive a convergent set of rules and equations.
  - CONFLUENCE: Is the given set of equations (oriented from left to right) convergent?
  - PROOF: Do the given conclusions hold in the specified equational theory?
  - REDUCTION: Normalize the given conclusions with the set of equations (oriented from left to right).

- `SORTS` `Sort1, Sort2`

  Declares the sorts the operators are expected to work on.

- `SIGNATURE`
  ```
  Op1, Op2: Sort1 Sort2 -> Sort2
  Const: -> Sort2
  ```

  States the signature of the operators (which also includes constants).

- `ORDERING`
  ```
  ((LPO
    Const > Op2
    Const > Op1) |
   (KBO
    Op1 = 3, Op2 = 2, Const = 1
    Const > Op2
    Const > Op1))
  ```

  For modes COMPLETION and PROOF, a reduction ordering has to be specified: either linear path ordering, based on an operator precedence, or extended Knuth–Bendix ordering with a weight for each operator and an additional operator precedence for the comparison of equally weighted terms.
  For mode PROOF, the reduction ordering has to be complete on ground terms; henceforth, a total precedence is required.

- `VARIABLES`
  ```
  x, y: Sort1
  z: Sort2
  ```

  Declares the variables along with their sorts. There is one common name space for all sorts.

- `EQUATIONS` `Op1(x,Op1(y,Op2(y,z))) = Op1(x,Op2(x,Const))`

  Set of equations defining the equational theory. The equations are considered as implicitely all–quantified.

- CONCLUSION `Op1(x,Const) = Op2(x,Const)`
  - For mode REDUCTION, a set of equations to be normalized.
  - For mode PROOF, a set of hypotheses to be proved. Variables are implicitely existentially quantified. All–quantification is expressed via Skolem constants.

braid4.cf [Art47], for the equations see [Cho48]

```
NAME            Braid4
MODE            CONFLUENCE
SORTS           ANY
SIGNATURE       p,q,r,s,t,u,P,Q,R,S,T,U: ANY -> ANY
                305,306,307,308,315,316,317,318,325: ANY -> ANY
                326,327,328,405,406,407,408,415,416: ANY -> ANY
                417,418,425,426,427,428: ANY -> ANY
                a: -> ANY
VARIABLES       x: ANY
EQUATIONS       305(x) = x  % The complete set contains 862 equations.

                P(p(x)) = x
                p(P(x)) = x
                Q(q(x)) = x
                q(Q(x)) = x
                ...
                426(p(x)) = S(t(s(426(x))))
                307(p(x)) = t(307(x))
                407(p(x)) = T(u(t(407(x))))
                317(p(x)) = s(317(x))
                ...
                307(Q(x)) = R(T(U(t(u(307(x))))))
                407(Q(x)) = R(T(U(T(u(t(t(407(x))))))))
                317(Q(x)) = Q(S(U(s(u(317(x))))))
                ...
                316(U(x)) = R(P(Q(p(q(S(T(s(U(S(t(s(u(316(x))))))))))))))
                326(U(x)) = P(S(T(S(t(s(s(326(x))))))))
                426(U(x)) = P(Q(p(S(T(U(t(S(T(u(t(s(s(426(x))))))))))))))
                307(U(x)) = Q(S(U(s(u(307(x))))))
                407(U(x)) = P(S(T(s(t(407(x))))))
                ...
```

dt1.rd [OKW]

```
NAME            Fibonacci
MODE            REDUCTION
SORTS           ANY
SIGNATURE       0: -> ANY
                s, p: ANY -> ANY
```

```
                fib, dfib: ANY -> ANY
                +, -: ANY ANY -> ANY
VARIABLES       x,y: ANY
EQUATIONS       fib(0) = s(0)
                fib(s(0)) = s(0)
                fib(s(s(x))) = +(fib(s(x)),fib(x))
                dfib(0) = s(0)
                dfib(s(0)) = s(0)
                dfib(s(s(x))) = +(dfib(s(x)),+(dfib(x),dfib(x)))
                +(x,s(y)) = s(+(x,y))
                +(x,0) = x
                -(s(x),s(y)) = -(x,y)
                -(x,0) = x
                p(x) = -(x, s(0))
CONCLUSION      -(-(fib(p(dfib(dfib(s(s(s(0)))))))),
                dfib(s(fib(s(dfib(s(s(s(0))))))))),
                dfib(dfib(s(s(s(0))))))) = s(s(0))
```

fac8.rd (Peano arithmetics)

```
NAME            faculty_of_8
MODE            REDUCTION
SORTS           NAT
SIGNATURE       0 : -> NAT
                s : NAT -> NAT
                + : NAT NAT -> NAT
                * : NAT NAT -> NAT
                fac : NAT -> NAT
VARIABLES       x,y : NAT
EQUATIONS       +(x,0) = x
                +(x,s(y)) = s(+(x,y))
                *(x,0) = 0
                *(x,s(y)) = +(*(x,y),x)
                fac(0) = s(0)
                fac(s(x)) = *(s(x),fac(x))
CONCLUSION      fac(s(s(s(s(s(s(s(s(s(0))))))))) = 0
```

gt4.3.pr [LW92]

```
NAME            gt4-3
MODE            PROOF
SORTS           ANY
SIGNATURE       g: ANY -> ANY
                f: ANY ANY -> ANY
                a, b, c: -> ANY
ORDERING        LPO
                g > f > a > b > c
VARIABLES       x,y,z,u: ANY
```

```
EQUATIONS       f(x,g(f(y,f(f(f(z,g(z)),g(f(u,y))),x)))) = u
CONCLUSION      f(f(a,b),c) = f(a,f(b,c))
```

gt6.pr [LW92]

```
NAME            gt6
MODE            PROOF
SORTS           ANY
SIGNATURE       i: ANY -> ANY
                f: ANY ANY -> ANY
                a, b, c: -> ANY
ORDERING        LPO
                i > f > a > b > c
VARIABLES       x,y,z,u,v,w: ANY
EQUATIONS       f(i(f(i(f(i(f(x,y)),f(y,x))),
                      f(i(f(z,u)),f(z,i(f(f(v,i(w)),i(u)))))))),
                  w) = v
CONCLUSION      f(f(a,b),c) = f(a,f(b,c))
                f(a,f(a,i(a))) = a
                f(a,i(a)) = f(b,i(b))
                f(a,b) = f(b,a)
```

jar10.2.1.pr [McC93]

```
NAME            jar10-2-1
MODE            PROOF
SORTS           ANY
SIGNATURE       i: ANY -> ANY
                f: ANY ANY -> ANY
                a, b, c: -> ANY
ORDERING        LPO
                i > f > a > b > c
VARIABLES       x,y,z: ANY
EQUATIONS       f(f(i(f(x,i(f(y,z)))),f(x,i(z))),i(f(i(z),z))) = y
CONCLUSION      f(f(a,b),c) = f(a,f(b,c))
```

jar10.3.1a.pr [McC93]

```
NAME            jar10-3-1a
MODE            PROOF
SORTS           ANY
SIGNATURE       i:       ANY -> ANY
                f:       ANY ANY -> ANY
                a, b, c: -> ANY
ORDERING        LPO
                i > f > a > b > c
VARIABLES       x,y,z,u: ANY
```

```
EQUATIONS       f(x,i(f(y,f(f(f(z,i(z)),i(f(u,y))),x)))) = u
CONCLUSION      f(f(a,b),c) = f(a,f(b,c))
```

juxta.rd [WZ95]

```
NAME            JuxtaPositionIntegers
MODE            REDUCTION
SORTS           INT
SIGNATURE       _:       INT INT -> INT     % Juxtaposition operator
                0,1,2,3,
                4,5,6,7,
                8,9 :    -> INT             % digits
                +, -, *: INT INT -> INT     % arithmetic function
                neg:     INT -> INT         % negative inverse
                zfac:    INT INT -> INT     % auxiliary function for fac
                fac :    INT -> INT         % factorial
VARIABLES       x,y,z : INT
EQUATIONS       % All in all, more than 500 equations are needed.
                _(0,x) = x
                _(x,_(y,z)) = _(+(x,y),z)
                +(0,x) = x
                +(x,0) = x
                _(x,neg(_(y,z))) = neg(_(-(y,x),z))
                _(neg(x),y) = neg(_(x,neg(y)))
                neg(neg(x)) = x
                neg(0) = 0
                +(x,_(y,z)) = _(y,+(x,z))
                +(_(x,y),z) = _(x,+(y,z))
                +(x,neg(y)) = -(x,y)
                +(neg(x),y) = -(y,x)
                -(0,x) = neg(x)
                -(x,0) = x
                -(_(x,y),z) = _(x,-(y,z))
                -(x,_(y,z)) = neg(_(y,-(z,x)))
                -(x,neg(y)) = +(x,y)
                -(neg(x),y) = neg(+(x,y))
                *(0,x) = 0
                *(x,0) = 0
                *(x,_(y,z)) = _(*(x,y),*(x,z))
                *(_(x,y),z) = _(*(x,z),*(y,z))
                *(x,neg(y)) = neg(*(x,y))
                *(neg(x),y) = neg(*(x,y))

                % From each of the following schemata, all ground
                % equations with d and e replaced with any of the
                % digits from 1 to 9 has to be build.
                  _(d,neg(e)) = _(d^,e-)
                  _(_(x,0),neg(d)) = _(_(x,neg(1)),d-)
```

```
                _(_(x,d),neg(e)) = _(_(x,d^),e-)
                +(d,e) = ++(d,e)
                -(d,e) = --(d,e)
                *(d,e) = **(d,e)

              zfac(0,z) = z
              ...
              zfac(9,z) = zfac(8,*(9,z))

              % fac(_(x,y)) = *(_(x,y),fac(-(_(x,y),1)))
              zfac(_(x,0),z) = zfac(_(-(x,1),9),*(_(x,0),z))
              zfac(_(x,9),z) = zfac(_(x,8),*(_(x,9),z))
              ...
              zfac(_(x,1),z) = zfac(_(x,0),*(_(x,1),z))

              fac(x) = zfac(x,1)

CONCLUSION    fac(_(_(1,1),9)) = 0        % calculate 119!
```

Luka10.pr [Tar56]

```
NAME          Luka10
MODE          PROOF
SORTS         ANY
SIGNATURE     C:  ANY ANY -> ANY
              N:  ANY -> ANY
              T, Ap, Aq:  ->ANY
ORDERING      LPO
              C > N > T > Ap > Aq
VARIABLES     p,q,r: ANY
EQUATIONS     C(T,p) = p
              C(p,C(q,p)) = T
              C(C(p,C(q,r)),C(C(p,q),C(p,r))) = T
              C(C(p,C(q,r)),C(q,C(p,r))) = T
              C(C(p,q),C(N(q),N(p))) = T
              C(N(N(p)),p) = T
              C(p,N(N(p))) = T
CONCLUSION    C(N(Ap),C(Ap,Aq)) = T
```

Lusk5.pr [LO85]

```
NAME          Lusk5
MODE          PROOF
SORTS         ANY
SIGNATURE     f: ANY ANY ANY -> ANY
              g: ANY -> ANY
              a, b: ->ANY
ORDERING      LPO
```

```
                g > f > b > a
VARIABLES       v,w,x,y,z: ANY
EQUATIONS       f(f(v,w,x),y,f(v,w,z)) = f(v,w,f(x,y,z))
                f(y,x,x)     = x
                f(x,x,y)     = x
                f(g(y),y,x)  = x
CONCLUSION      f(a,g(a),b) = b
```

Lusk6.pr [LO85]

```
NAME            Lusk6
MODE            PROOF
SORTS           ANY
SIGNATURE       *, +: ANY ANY -> ANY
                -: ANY -> ANY
                0: -> ANY
                a, b: -> ANY
ORDERING        KBO
                * = 5, + = 4, - = 3, 0 = 1, a = 1, b = 1
                * > + > - > 0 > a > b
VARIABLES       x,y,z: ANY
EQUATIONS       +(0,x)      = x
                +(x,0)      = x
                +(-(x),x)   = 0
                +(x,-(x))   = 0
                +(+(x,y),z) = +(x,+(y,z))
                +(x,y)      = +(y,x)
                *(*(x,y),z) = *(x,*(y,z))
                *(x,+(y,z)) = +(*(x,y),*(x,z))
                *(+(x,y),z) = +(*(x,z),*(y,z))
                *(*(x,x),x) = x
CONCLUSION      +(+(+(a,a),a),+(+(a,a),a)) = 0
                *(a,b) = *(b,a)
```

mv2.pr [LW92]

```
NAME            mv2
MODE            PROOF
SORTS           ANY
SIGNATURE       i: ANY ANY -> ANY
                n: ANY -> ANY
                a, b, c, T: -> ANY
ORDERING        LPO
                i > n > T > a > b > c
VARIABLES       x,y,z: ANY
EQUATIONS       i(T,x) = x
                i(i(x,y),y) = i(i(y,x),x)
                i(i(n(x),n(y)),i(y,x)) = T
```

```
                i(i(x,y),i(i(y,z),i(x,z))) = T
CONCLUSION      i(i(a,b),i(i(c,a),i(c,b))) = T
```

mv4.pr [LW92]

```
NAME            mv4
MODE            PROOF
SORTS           ANY
SIGNATURE       i: ANY ANY -> ANY
                n: ANY -> ANY
                a, b, T: -> ANY
ORDERING        LPO
                i > n > T > a > b
VARIABLES       x,y,z: ANY
EQUATIONS       i(T,x) = x
                i(i(x,y),y) = i(i(y,x),x)
                i(i(n(x),n(y)),i(y,x)) = T
                i(i(x,y),i(i(y,z),i(x,z))) = T
CONCLUSION      i(i(i(a,b),i(b,a)),i(b,a)) = T
```

p9a.pr [Fuc94]

```
NAME            p9a
MODE            PROOF
SORTS           ANY
SIGNATURE       i: ANY -> ANY
                f, n, u: ANY ANY -> ANY
                1, a, b, c: -> ANY
ORDERING        LPO
                i > f > n > u > 1 > a > b > c
VARIABLES       x,y,z: ANY
EQUATIONS       u(x,x) = x
                n(x,x) = x
                f(1,x) = x
                n(x,y) = n(y,x)
                u(x,y) = u(y,x)
                u(x,n(x,y)) = x
                n(x,u(x,y)) = x
                f(i(x),x) = 1
                u(1,a) = a
                u(1,b) = b
                u(1,c) = c
                n(a,b) = 1
                n(x,n(y,z)) = n(n(x,y),z)
                u(x,u(y,z)) = u(u(x,y),z)
                f(x,f(y,z)) = f(f(x,y),z)
                f(x,u(y,z)) = u(f(x,y),f(x,z))
                f(x,n(y,z)) = n(f(x,y),f(x,z))
```

```
                f(u(x,y),z) = u(f(x,z),f(y,z))
                f(n(x,y),z) = n(f(x,z),f(y,z))
CONCLUSION      n(a,f(b,c)) = n(a,c)
```

P16lpo.cp [Chr93]

```
NAME            P16lpo
MODE            COMPLETION
SORTS           ANY
SIGNATURE       e32, e31, ..., e1: -> ANY
                i32, i31, ..., i1: ANY -> ANY
                f: ANY,ANY -> ANY
ORDERING        LPO
                i32 > i31 > ... > i1 > f >
                e32 > e31 > ... > e1
VARIABLES       x, y, z  : ANY
EQUATIONS       f(f(x,y),z)     = f(x,f(y,z))
                f(e1,x)         = x
                ...
                f(e32,x)        = x
                f(x,i1(x))      = e1
                ...
                f(x,i32(x))     = e32
```

P16kbo.cp [Chr93]

```
NAME            P16kbo
MODE            COMPLETION
SORTS           ANY
SIGNATURE       e32, e31, ..., e1: -> ANY
                i32, i31, ..., i1: ANY -> ANY
                f: ANY,ANY -> ANY
ORDERING        KBO
                f=0,
                i32=0, i31=2, i30=4, ..., i1=62,
                e32=2, e31=1, e30=1, ..., e1=1
                i32 > i31 > ... > i1 > f >
                e32 > e31 > ... > e1
VARIABLES       x, y, z  : ANY
EQUATIONS       f(f(x,y),z)     = f(x,f(y,z))
                f(e1,x)         = x
                ...
                f(e32,x)        = x
                f(x,i1(x))      = e1
                ...
                f(x,i32(x))     = e32
```

ra2.pr [LW92]

```
NAME            ra2
MODE            PROOF
SORTS           ANY
SIGNATURE       n: ANY -> ANY
                o: ANY ANY -> ANY
                a, b: -> ANY
ORDERING        LPO
                n > o > a > b
VARIABLES       x,y,z: ANY
EQUATIONS       o(x,x) = x
                o(x,y) = o(y,x)
                o(o(x,y),z) = o(x,o(y,z))
                n(o(n(o(x,y)),n(o(x,n(y))))) = x
CONCLUSION      o(n(o(n(a),b)),n(o(n(a),n(b)))) = a
```

ra3.pr [LW92]

```
NAME            ra3
MODE            PROOF
SORTS           ANY
SIGNATURE       n: ANY -> ANY
                o: ANY ANY -> ANY
                a, b, c: -> ANY
ORDERING        KBO
                c = 1, n = 1, o = 1, a = 1, b = 1
                c > n > o > a > b
VARIABLES       x,y,z: ANY
EQUATIONS       o(x,y) = o(y,x)
                o(c,c) = c
                o(o(x,y),z) = o(x,o(y,z))
                n(o(n(o(x,y)),n(o(x,n(y))))) = x
CONCLUSION      o(n(o(n(a),b)),n(o(n(a),n(b)))) = a
```

ra4.pr [LW92]

```
NAME            ra4
MODE            PROOF
SORTS           ANY
SIGNATURE       n: ANY -> ANY
                o: ANY ANY -> ANY
                a, b, c, d: -> ANY
ORDERING        KBO
                c = 1, d = 1, n = 1, o = 1, a = 1, b = 1
                c > d > n > o > a > b
VARIABLES       x,y,z: ANY
EQUATIONS       o(x,y) = o(y,x)
```

```
                o(o(x,y),z) = o(x,o(y,z))
                n(o(n(o(x,y)),n(o(x,n(y))))) = x
                o(c,d) = d
CONCLUSION      o(n(o(n(a),b)),n(o(n(a),n(b)))) = a
                # Huntington's Axiom
```

sims2.cp [Sim91]

```
NAME            sims2
MODE            COMPLETION
SORTS           ANY
SIGNATURE       T: ANY -> ANY
                t: ANY -> ANY
                S: ANY -> ANY
                s: ANY -> ANY
                R: ANY -> ANY
                r: ANY -> ANY
ORDERING        KBO
                T = 1, t = 1, S = 1, s = 1, R = 1, r = 1
                T > t > S > s > R > r
VARIABLES       x : ANY
EQUATIONS       %  Definition of inverses
                T(t(x)) = x
                t(T(x)) = x
                S(s(x)) = x
                s(S(x)) = x
                R(r(x)) = x
                r(R(x)) = x
                % Definition of the group
                t(t(S(T(s(T(r(t(R(R(S(t(s(
                  T(T(r(r(T(R(t(r(r(T(R(t(x))...) = x
                r(r(T(R(t(R(s(r(S(S(T(r(t(
                  R(R(s(s(R(S(r(s(s(R(S(r(x))...) = x
                s(s(R(S(r(S(t(s(T(T(R(s(r(
                  S(S(t(t(S(T(s(t(t(S(T(s(x))...) = x
```

z22.cp [AD93]

```
NAME            Z22
MODE            COMPLETION
SORTS           ANY
SIGNATURE       a, a1, b, b1, c, c1, d, d1, e, e1: ANY -> ANY
ORDERING        LPO
                e1 > e > d1 > d > c1 > c > b1 > b > a1 > a
VARIABLES       x: ANY
EQUATIONS       a(b(c(x))) = d(x)
                b(c(d(x))) = e(x)
                c(d(e(x))) = a(x)
```

```
d(e(a(x))) = b(x)
e(a(b(x))) = c(x)
a(a1(x)) = x
a1(a(x)) = x
b(b1(x)) = x
b1(b(x)) = x
c(c1(x)) = x
c1(c(x)) = x
d(d1(x)) = x
d1(d(x)) = x
e(e1(x)) = x
e1(e(x)) = x
```

References

[AD93] J. Avenhaus and J. Denzinger. Distributing equational theorem proving. In *Proceedings of the 5th International Conference on Rewriting Techniques and Applications*, volume 690 of *LNCS*, pages 62–76. Springer Verlag, 1993.

[Art47] E. Artin. Theory of braids. *Annals of Mathematics*, 48(1):101–126, 1947.

[Ave95] J. Avenhaus. *Reduktionssysteme.* Springer Verlag, 1995.

[BDP89] L. Bachmair, N. Dershowitz, and D.A. Plaisted. Completion without failure. In *Resolution of Equations in Algebraic Structures*, volume 2, pages 1–30. Academic Press, 1989.

[BGK96] R. Bündgen, M. Göbel, and W. Küchlin. A master–slave approach to parallel term rewriting on a hierarchical multiprocessor. In *Proceedings of the 4th International Symposium on Design and Implementation of Symbolic Computation Systems*, volume 1126 of *LNCS*, pages 183–194. Springer Verlag, 1996.

[Cho48] W.L. Chow. On the algebraical braid group. *Annals of Mathematics*, 49(3):654–658, 1948.

[Chr89] J. Christian. Fast Knuth–Bendix completion: A summary. In *Proceedings of the 3rd International Conference on Rewriting Techniques and Applications*, volume 355 of *LNCS*, pages 551–555. Springer Verlag, 1989.

[Chr93] J. Christian. Flatterms, discrimination nets and fast term rewriting. *Journal of Automated Reasoning*, 10(1):95–113, 1993.

[CMM90] S.E. Conry, D.J. MacIntosh, and R.A. Meyer. Dares: A distributed automated reasoning system. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 78–85. AAAI Press, 1990.

[Den93] J. Denzinger. TEAMWORK: *eine Methode zum Entwurf verteilter, wissensbasierter Theorembeweiser.* PhD thesis, Universität Kaiserslautern, 1993.

[DS93] J. Denzinger and S. Schulz. Analysis and Representation of Equational Proofs Generated by a Distributed Completion–Based Proof System. SEKI–Report 94–05, Universität Kaiserslautern, 1993.

[DS96] J. Denzinger and S. Schulz. Learning domain knowledge to improve theorem proving. In *Proceedings of the 13th International Conference on Automated Deduction*, volume 1104 of *LNAI*, pages 76–62. Springer Verlag, 1996.

[FSST86] M.L. Fredman, R. Sedgewick, D.D. Sleator, and R.E. Tarjan. The pairing–heap: a new form of self–adjusting heap. *Algorithmica*, 1(1):111–129, 1986.

[Fuc94] M. Fuchs. Goal-Oriented Heuristics for Proving Equational Theorems. SEKI-Report SR 94-02, Universität Kaiserslautern, 1994.

[Fuc96a] D. Fuchs. TEAMWORK-Varianten für die verteilte wissensbasierte Suche am Beispiel eines Condensed-Detachement-Beweisers. Master's thesis, Universität Kaiserslautern, August 1996.

[Fuc96b] M. Fuchs. Experiments in the heuristic use of past proof experience. In *Proceedings of the 13th International Conference on Automated Deduction*, volume 1104 of *LNAI*, pages 523–537. Springer Verlag, 1996.

[Gra94] P. Graf. ACID: A Collection of Indexing Datastructures. Available via internet at `http://www.mpi-sb.mpg.de/tools/deduction/ACID`, 1994.

[Gra96] P. Graf. *Term Indexing*, volume 1053 of *LNAI*. Springer Verlag, 1996.

[KB70] D.E. Knuth and P.B. Bendix. Simple word problems in universal algebra. In *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[KMN88] D. Kapur, D.R. Musser, and P. Narendran. Only prime superpositions need be considered in the Knuth-Bendix completion procedure. *Journal of Symbolic Computation*, 6(1):19–36, 1988.

[LO85] E. Lusk and R.A. Overbeek. Reasoning about equality. *Journal of Automated Reasoning*, 1(2):209–228, 1985.

[Luk70] J. Lukasiewicz. *Selected Works*. North-Holland, 1970.

[LW92] E. Lusk and L. Wos. Benchmark problems in which equality plays the major role. In *Proceedings of the 11th International Conference on Automated Deduction*, volume 607 of *LNCS*, pages 781–785. Springer Verlag, 1992.

[McC92] W. McCune. Experiments with discrimination-tree indexing and path indexing for term retrieval. *Journal of Automated Reasoning*, 9(3):147–167, 1992.

[McC93] W. McCune. Single axioms for groups and abelian groups with various operations. *Journal of Automated Reasoning*, 10(1):1–13, 1993.

[OKW] H.J. Ohlbach, M. Kohlhase, and C. Weidenbach. Deduktionstreffen 1995. `http://js-sfbsun.cs.uni-sb.de:80/pub/www/deduktion/dedtreff95`. For proof task `dt1.pr` follow links `Beweiserhappening` and `EQ`.

[Sim91] C.C. Sims. The Knuth-Bendix procedure for strings as a substitute for coset enumeration. *Journal of Symbolic Computation*, 12(4/5):439–442, 1991.

[Sti84] M.E. Stickel. A case study of theorem proving by the Knuth-Bendix method: Discovering that $x^3 = x$ implies ring commutativity. In *Proceedings of the 7th International Conference on Automated Deduction*, volume 170 of *LNCS*, pages 248–258. Springer Verlag, 1984.

[SV87] J.T. Stasko and J.S. Vitter. Pairing–heaps: Experiments and analysis. *Communications of the ACM*, 30(3):234–249, 1987.

[Tar56] A. Tarski. *Logic, Semantics, Metamathematics.* Oxford University Press, 1956.

[WZ95] H.R. Walters and H. Zantema. Rewrite systems for integer arithmetic. In *Proceedings of the 6th International Conference on Rewriting Techniques and Applications*, volume 914 of *LNCS*, pages 324–338. Springer Verlag, 1995.

Acknowledgements: We would like to thank our tutors Roland Vogt and Bernd Löchner for their many ideas and their permanent support, Prof. Jürgen Avenhaus for his competent guidance, Thomas Deiss, Jörg Denzinger, Dirk Fuchs, Marc Fuchs, Matthias Fuchs, Peter Graf, Bernhard Gramlich, Andreas Jaeger, Christoph Kögl, Martin Kronburg, Carlo Reiffers, Andrea Sattler–Klein, Stephan Schulz, and Dirk Zeckzer for many fruitful suggestions and discussions. Finally, special thanks apply to C.-P. Wirth, whose choleric temper promoted the development of space saving techniques.