Strategy Parallelism and Lemma Evaluation

Joachim Draeger & Andreas Wolf
Institut für Informatik der Technischen Universität München
D–80290 München
Germany
{draeger,wolfa}@informatik.tu-muenchen.de

Abstract
Automated Deduction offers no unique strategy which is uniformly successful on all problems. Hence a parallel combination of several different strategies increases the chances of success. The efficiency of this approach can be increased even more by the exchange of suitable intermediate results. The paper offers a cooperative solution approach to this task. As a special kind of cooperation we present here the selection of suitable lemmata together with a model of a cooperative parallel theorem prover which combines different lemma selection techniques within a strategy parallel prover environment. We give a short assessment of the results of first experiments and an outline of the future work.

Introduction
Up to now, sequential automated theorem provers (ATPs) have set a high standard; but when dealing with difficult problems they are still inferior to a human mathematician. An important technique to increase the performance is to employ parallelism. Possible parallelization concepts vary from the parallel use of different configurations of the employed provers to the partitioning of the proof task into subtasks that are tackled in parallel. Another promising technique is the use of lemmata for reducing the search space, which has to be processed for obtaining a solution. The serious drawback of a lemmatization is the introduced additional redundancy originating from the fact that during the ongoing search both lemma applications and lemma reproductions are possible. The amount of redundancy increases rapidly with the number of lemmata added to the original axiomatization. Hence, the lemmata supporting the proof process have to be selected very carefully.

The parallelism concept can be improved by interactions between the different parallel provers (cooperation) by exchanging intermediate results like lemmata. The advantages of such a combination of parallelism and lemmatization are twofold. At first, the exchange of intermediate results will reduce the amount of redundancy contained in the parallel computations. Secondly, if the different strategies include different lemma selection schemes, the lemma selection treatment will profit, too. In many cases the specific effects of different lemma selection strategies are not known in advance; hence, a reliable assessment of these strategies requires practical experiences.

As we have seen, both parallelism and lemmatization can profit from the combination of several lemma selection strategies in a competitive manner. Thus our aim will be the realization of such a combination. For the evaluation of a combination of different lemma selection strategies, we will restrict our considerations to the model elimination calculus (Loveland 1968), which is based on problem decomposition. A given query is recursively decomposed into new sub-queries until each of them can be solved either by a unit-lemma of the axiomatization, or by one of the assumptions made during the decomposition. In this way possible decompositions are enumerated until a proof can be constructed.

The paper is organized as follows. In the next section we give a selection of related work. Then follow three sections dealing with our main topics: lemma generation and evaluation, lemma selection, and combination of lemma generation techniques. We conclude with a short assessment of first experimental results and with an outlook.

Related Work
In the past years there have been several approaches for cooperation among bottom-up theorem provers. Some of them have been based on superposition, unfailing completion, or resolution. Methods which partition the clauses among several sub-provers are used in DARES (Conry et al. 1990). The Teamwork (Denzinger 1995) method of DISCOUNT provides cooperation among complete competitive provers by periodically exchanging evaluated and selected results. Here, we also find a similarity based deduction guidance similar to our lemma similarity strategy. An approach for lemma generation and application by sequential cooperation among a saturating component and a top-down prover has been developed in DELTA (Schumann 1994).
**Lemma Evaluation**

The generation and use of lemmata is a special kind of learning on the domain level. Their application yields a result while avoiding the search needed for their deduction. Hence, its effect is a reordering of the search space, supporting some solution constructions while limiting others. In the introduction this was explained by the ability of lemmata to guide the search as intermediate results. Now we discuss this effect in a more formal way. By separating parts of an original proof \( p \) as lemmata a modularization of both the proof and the search process is achieved. Technically a simple version of such a modularization can be realized as a procedure which generates unit-lemmata \( f_1, f_2, \ldots \) and uses them for constructing the proof \( p' \) of the actual problem. The use of unit-lemmata \( f_1, f_2, \ldots \) makes the proof \( p' \) smaller and hence easier to find than \( p \). If the lemmata \( f_1, f_2, \ldots \) are 'useful' with respect to the modularization of the actual proof task, the desired restriction of the search space is performed.

The above considerations yield a more abstract point of view. Lemmata can be considered as pieces of knowledge which are starting points for future exploration of the search space. If these starting points are chosen in a suitable way, e.g., if they represent intermediate results of the proof \( p \), the necessary effort for finding a solution can be reduced dramatically.

Which lemmata can be considered to be useful? The lemmatization has to reduce the necessary effort for finding a proof by modularization as described before. Consequently, a lemma may only be considered as useful if it enables a separation of a significant part of the original proof. This is only possible, if the lemma itself requires a significantly complex proof. A suitable way to measure the proof complexity of a lemma \( f \) is the proof length \( p(f) \), the number of inferences contained in the proof of \( f \). Unfortunately, the proof length is no invariant. For example, the insertion of equivalence transformations into the proof can enlarge \( p(f) \) without limit making this parameter ill-defined. However, it is fair to say that additional inferences blowing up a proof are irrelevant. This leads to the use of the minimal proof length. In the following, we always use the term 'proof length' for the minimal proof length. A comparatively large value of this parameter will be our first selection criterion.

The proof length alone is not sufficient for an efficient lemma selection; in most cases an overwhelming number of lemmata requiring non-trivial proofs exist. Hence an additional selection criterion is needed. It can be based on the observation that the potential of separating a significant part of some proofs is not sufficient; the separation must actually happen in a proof of the actual problem. Consequently, we choose the relevance \( r(f) \) of a lemma \( f \) with respect to the actual proof task as second selection criterion. Contrary to the proof complexity, there is no obvious way for measuring the relevance of a lemma. Consequently, many different methods for the relevance estimation are possible. Each one leads to a specific selection strategy. In this paper, we will discuss the following two main strategies.

**Strategy 1. Lemma Size.** In this strategy we assume that a lemma \( f \) with low syntactic complexity is more relevant than a lemma \( f \) with a high syntactic complexity. One can argue in the following way. The proof of a problem (if there is one) is finite; hence the number of useful lemmata is finite, too. On the other hand, the total number of lemmata, which are valid in the underlying theory, is typically not finite or exceeds at least the number of useful lemmata by far. In other words, the syntactic complexity of an arbitrary lemma is much larger than the syntactic complexity of a useful lemma on the average. Therefore limiting the syntactic complexity of a selected lemma will raise the probability that this lemma is useful for the construction of the actual proof.

The measurement of the syntactic complexity of a unit-lemma \( f \) can be performed in different ways. In this paper, we use the symbol size \( s(f) \), i.e. the number of constant and function symbols contained in the assertion of \( f \), and the symbol depth \( d(f) \), i.e. the length of the longest path in the assertion of \( f \) represented as symbol tree. Both criteria symbol size \( s(f) \) and symbol depth \( d(f) \) are used as reciprocal values, i.e. \( r(f) = (s(f))^{-1} \) and \( r(f) = (d(f))^{-1} \), respectively.

**Strategy 2. Lemma Similarity.** The second strategy principle is the relevance measurement based on the similarity of a considered unit-lemma \( f \) to the query \( q \). The main idea of this strategy is the identification of lemmata, which are useful with respect to a step-by-step construction of the query \( q \). Following this idea it is suggestive to consider a unit-lemma \( f \) to be the more useful, the stronger the similarity of \( f \) and \( q \) is. Hence we will demand a strong similarity of \( f \) and \( q \) in order to make the relevance of \( f \) for the construction of the desired proof as large as possible. This should also enable the prover system aiming at the query more explicitly.

For measuring the similarity of two literals \( a \) and \( b \) we use two methods. The first one measures the structural similarity \( u_q(f) \), the second one the signature similarity \( g_q(f) \). Let \( w_1, \ldots, w_n \) be the maximal sub-terms contained both in \( f \) and \( q \). The function \( u_q(f) \) is defined to be

\[
u_q(f) = s(q) + s(f) - 2 \cdot s(w_1) - \ldots - 2 \cdot s(w_n).
\]

Similarly, the function \( g_q(f) \) counts the numbers \( n_q(a_1), n_f(a_1) \) of occurrences of each function, constant, and predicate symbol \( a_1, \ldots, a_m \) contained in \( q \) and \( f \). The value of \( g_q(f) \) is determined by

\[
g_q(f) = |n_q(a_1) - n_f(a_1)| + \ldots + |n_q(a_m) - n_f(a_m)|.
\]
The relevance of a lemma \( f \) is once more \( r(f) = u_q(f) \) or \( r(f) = g_q(f) \), respectively.

As an example, let us consider a problem with the query \( q = p(g(a), f(g(c)), b) \). We assume that the unit-lemmata \( f_1 = p(f(g(a)), f(g(b)), f(g(c))) \) and \( f_2 = p(g(a), g(g(c)), c) \) are produced. Then the different relevance measures have the following values:

<table>
<thead>
<tr>
<th></th>
<th>( 1/s )</th>
<th>( 1/d )</th>
<th>( u_q )</th>
<th>( g_q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_1 )</td>
<td>1/10</td>
<td>1/4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>( f_2 )</td>
<td>1/7</td>
<td>1/4</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

**Information Measure.** The information measure \( I \) introduced in (Draeger 1998a) evaluates a lemma \( f \) with respect to one of these two strategies. This is done by using the product \( I(f) = p(f) \cdot r(f) \) of the proof complexity \( p(f) \) and the relevance \( r(f) \). A lemma \( f \) is considered to be suitable if \( I(f) \) has a large value. In this way both uninteresting lemmata, which have a small relevance value, and trivialities, which have a small complexity, are excluded. These considerations establish an argument for the naming of “information measure”, too. The value of \( I(f) \) is large, if \( f \) seems to be of great value for the construction of the final proof.

The four different relevance measurement methods of the two main strategies have different areas of application. This is an immediate consequence of the individual qualification criteria. So for example, if we consider the symbol size criteria, the prover must be able to discriminate lemmata with different degrees of relevance. In other words, the assertions of the lemmata should strongly vary in their symbol size; otherwise, no discrimination of suitable and unsuitable lemmata is possible.

It is useful to develop a refined information measure for general use. One possibility for such a refinement is the combination of the different relevance criteria in an evaluation vector. However, this approach is not always desirable. Lemma knowledge can be used for different purposes which all need their own selection strategies. As mentioned before, the applicability of these strategies is often restricted and requires a significant degree of specialization. Hence different strategies for lemma selection will perform with different success.

The ideas described in this section are an improvement of some topics from (Draeger 1998a) and (Draeger 1998b).

**Dynamic Lemma Selection**

In the previous section, we have introduced four possibilities for the evaluation of generated lemmata. In this section, we present a prover model, which allows a dynamic selection of sets of high valuable lemmata to enrich the original proof task.

Our prover is based on the cooperation of a pair \((RG, LG)\) of a request generation component and a lemma generation component. In our implementation, both generators use the SETHEO model elimination prover (Moser et al. 1997). The lemma generator works similar to the DELTA iterator (Schumann 1994). It constructs a new existentially quantified query for every predicate contained in the original formula. Processing possible substitutions of these queries produces lemmata over the corresponding predicates valid in the considered theory. These lemmata are evaluated by the strategies given in the previous section.

In order to prove a given set of input clauses, the request generator \( RG \) tries to enumerate a closed tableau and generates proof requests. Proof requests in this model are subgoals which fail because of the lack of resources during the proof attempt. The lemma generator \( LG \) produces unit lemmata. To achieve cooperation between \( RG \) and \( LG \) we repeatedly choose during the proof search a subset of the lemmata generated by \( LG \) with a lemma selection component \( LS \) which knows all lemmata and requests generated so far by \( LG \) and \( RG \). Each time such a set of lemmata has been selected, a new dependent sub-prover is started which tries to refute the input clauses augmented by the selected lemmata. Thus, the whole system consists of a triple \((RG, LG, LS)\) which starts dependent competitive provers. We call such a triple a cooperative cell. The whole system stops after a time-out is exceeded or one of the involved provers is able to refute the given input clauses.

It is useful to combine different cooperative cells using different generation and evaluation strategies in a strategy parallel environment (Wolf & Letz 1998) as described in the next section.

In detail, our implementation works as follows. The \( LG \) component sends a data stream of generated lemmata to the \( LS \) component enriched with the value of the information measure of this lemma. In order to support the lemma selection the \( RG \) component adds data similar to the information measure to the generated proof requests and sends them to the \( LS \) component. If a lemma is unifiable with a request it may be useful for the proof search. When additionally using the lemma selected that way, the proof procedure would succeed at this position and possibly complete the proof attempt. In the lemma selection component \( LS \) the received data is ranked with respect of the attached evaluation value. So, the \( LS \) component administers two dynamically ranked data bases of evaluated formulae. At any time, the best \( k \) lemmata of the lemma pool in the \( LS \) component represent possibly well-suited lemmata and form a lemma set \( \mathcal{L} \). Every time this set \( \mathcal{L} \) has “significantly” changed, a new dependent sub-prover is started. This prover tries to refute the original input clauses which have been augmented with \( \mathcal{L} \).

In the following figure, the data flow of a cooperative cell is illustrated. The width of the arrows indicates the amount of data transmitted between the components. The scheme additionally shows that not all generated lemmata and requests will be transmitted. Those for-
mulae which get a very low evaluation value do not enter the LS component.

---

**Strategy Parallelism**

A search problem is typically solved by applying a uniform search procedure. In automated deduction, different search strategies may have a strongly different behavior on a given problem. This especially holds considering cooperative strategies. In general, it cannot be decided in advance which strategy is the best for a given problem. This motivates the competitive use of different strategies. In our approach, we employ the paradigm of **strategy parallelism**. An application of strategy parallelism to automated deduction can be found in (Wolf & Letz 1998). Strategy parallelism is, roughly spoken, the selection of more than one search strategy in combination with techniques to partition the available resources depending on the actual task.

A combination of strategies increases the chances of success. Limitations of resources such as time or processors enforce efficient use of these resources by partitioning them adequately among the involved strategies. Here, we concentrate on the problem of determining an optimal selection of strategies from a fixed set of given strategies. Such a method could be useful in practice, as follows. A non-specialist user of a theorem proving system can adapt the system to his specific domain of problems by selecting a representative training set of problems and computing an optimal set of competitive strategies $S$ for the training set. If the training set was representative for the domain, then the computed strategies will also perform well on the whole domain.

When trying to determine an optimal selection of strategies for a given training set, we are faced with the following strategy allocation problem. Given a set $A = \{a_1, \ldots, a_n\}$ of training problems, a set $B = \{b_1, \ldots, b_m\}$ of strategies $b_i : A \to \mathbb{N}^* \cup \{\infty\}$, and nonnegative integers $t_i$ (time limit) and $n_j$ (number of processors). Find ordered pairs $(t_1, p_1), \ldots, (t_m, p_m)$ (strategy $b_i$ will be scheduled for time $t_i$ on processor $p_i$) of nonnegative integers such that

$$\sum_{j=1}^n t_j \leq t$$

for $j = 1, \ldots, n$, and $| \bigcup_{i=1}^m \{a : b_i(a) \leq t_i\} |$ is maximal².

The decision variant of the problem is in NP: a given satisfying allocation can be verified in polynomial time. Unfortunately, the decision variant of the problem is already strongly NP-complete³ for a single processor. Therefore, in practice the determination of an optimal solution will be not possible, at least not on larger sets and with classical methods. One reasonable possibility is to use a gradient procedure as we do it in our implementation (Wolf 1998). This procedure has been used to determine the schedule for the following experiments.

**Experiments**

To determine the influence of the cooperation on the proof process, we compare the results of our lemma selection strategies with a successful conventional prover strategy of SETHEO. We show the results of all four suggested strategy variants on some selected problems from the TPTP (Sutcliffe et al. 1994).

<table>
<thead>
<tr>
<th>Problem</th>
<th>Conventional</th>
<th>Strategy 1</th>
<th>Strategy 2</th>
<th>Strategy 3</th>
<th>Strategy 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAT008-1</td>
<td>-</td>
<td>13</td>
<td>7</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>GEO004-1</td>
<td>-</td>
<td>108</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>GEO008-1</td>
<td>-</td>
<td>14</td>
<td>121</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>GRP008-1</td>
<td>-</td>
<td>14</td>
<td>20</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>HEN006-3</td>
<td>-</td>
<td>-</td>
<td>138</td>
<td>-14</td>
<td></td>
</tr>
<tr>
<td>LCI090-1</td>
<td>-</td>
<td>27</td>
<td>68</td>
<td>67</td>
<td>151</td>
</tr>
<tr>
<td>PUD010-1</td>
<td>-</td>
<td>-</td>
<td>122</td>
<td>137</td>
<td>127</td>
</tr>
<tr>
<td>RNG038-1</td>
<td>-</td>
<td>-</td>
<td>178</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>ROB016-1</td>
<td>92</td>
<td>-</td>
<td>16</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>SYN310-1</td>
<td>187</td>
<td>202</td>
<td>55</td>
<td>202</td>
<td>51</td>
</tr>
</tbody>
</table>

In the table above we depict for each of these problems the (in seconds) needed using each lemma evalu-
tion strategy and the conventional strategy. The time limit in this experiment was 300 seconds. Then we give the time needed for a strategy parallel competition of the four strategies on one processor and the speed-up compared with the reference strategy. The next table shows the summarized results on a subset of 92 problems\textsuperscript{4} taken from the eligibles of the CADE-15 ATP Competition. 92 of these problems can be solved by one of the cooperative strategies or by the conventional strategy. We measure the time needed by our four lemma evaluation strategies to treat all problems and count the proofs. Then we do the same with the conventional reference strategy and a strategy parallel and count the proofs. Then we do the same with the four lemma evaluation strategies to treat all problems. We measure the time needed by one of the cooperative strategies or by the conventional strategy. The time for each attempt (even the strategy parallel) is 300 seconds.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Proofs</th>
<th>Time (s)</th>
<th>Time/Proof (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy 1 size s</td>
<td>46</td>
<td>15825</td>
<td>344</td>
</tr>
<tr>
<td>Strategy 1 depth d</td>
<td>41</td>
<td>16416</td>
<td>400</td>
</tr>
<tr>
<td>Strategy 2 structure u_q</td>
<td>53</td>
<td>14217</td>
<td>268</td>
</tr>
<tr>
<td>Strategy 2 signature g_q</td>
<td>40</td>
<td>17923</td>
<td>448</td>
</tr>
<tr>
<td>Conventional</td>
<td>34</td>
<td>21181</td>
<td>622</td>
</tr>
<tr>
<td>Strategy parallel</td>
<td>71</td>
<td>15689</td>
<td>220</td>
</tr>
</tbody>
</table>

This experiment shows that the cooperative strategies are able to prove many more problems than the conventional strategy. But the sets of problems solved by different lemma selection strategies differ very strongly. This makes lemma generation strategies very convenient for strategy parallelism\textsuperscript{5}. We see that it is not sufficient to use only one cooperative strategy. The combination strongly increases the number of solved problems. The strategy parallel combination of conventional and lemma selection based strategies combines the high number of solved problems with comparatively low response times.

Assessment and Future Work

Despite of the prototypical implementation of the presented algorithm, the results of the lemma application are very promising. Although the lemmatization methods proposed in this paper have been tested using a model elimination prover, the concepts presented in this paper are more general and can be applied to other calculi, too.

The experimental results show that the cooperation among provers by lemma exchange can achieve very high speed-ups. Our work once more confirms that the main problem for cooperative theorem provers is the intelligent selection of relevant data from a growing amount of information. Our evaluation and selection techniques were successful in order to solve problems which are unreachable with conventional search methods. Nevertheless, the methods and techniques for information assessment and selection still need further research. Note that our cooperation approach can be combined with other parallelization paradigms like search space partitioning (Suttner & Schumann 1994). Thus, the good scalability of these models can easily be incorporated into our prover.

A second advantage of our prover model is the adaptability of our approach to the difficulty of the actual proof task. The lemmata are generated step by step, and so we get new sets of selected lemmata during the whole run time of the generators. So a simple proof task may be proved even without starting a sub-prover with a lemma enriched clause set, and difficult problems with a long run-time will employ a large amount of these sub-provers.

Acknowledgments

This work is supported by the Deutsche Forschungsgemeinschaft within the Sonderforschungsbereich 342, subproject A5 (PARIS).

References


