1 Introduction

Problems related to an inference rule called condensed detachment (CD)—also known as substitution and detachment—have piqued the attention of mathematicians ([13, 7]) and researchers in the field of automated deduction ([10, 14, 8, 11, 15]) alike. CD is particularly interesting for automated deduction because the basic inference rule is very simple and therefore does not necessitate complex programs. Nevertheless, CD offers problems of the same magnitude of difficulty as problems encountered by resolution-based theorem provers and equational reasoning systems. These properties make CD especially suitable for studying new search-guiding heuristics (cp. [14, 8]), without having to deal with difficulties caused by a large set of complicated inference rules.

Originally, CoDE was created for this very purpose, namely as an experimental prover for problems of CD that can be used as an environment for testing new search-guiding heuristics without having to spend much effort on implementation issues. During the two-year period of development, however, CODE has reached a degree of performance that goes beyond that of a purely experimental program. As a matter of fact, we believe that CODE is currently the most powerful proving system for problems of CD. We shall substantiate this claim by presenting ample experimental evidence in form of a comparison with OTTER ([9]) as the probably only serious competitor in the area of CD.

2 Condensed Detachment with CODE

The inference rule CD is the central part in the study of different logic calculi. This inference rule manipulates first-order terms which we shall also call facts. CD (in its basic form) is defined for a distinguished binary function symbol \( f \in \mathcal{F} \), allowing for the deduction of the fact \( \sigma(t) \) from two given facts \( f(s,t) \) and \( s' \), where \( \sigma \) is the most general unifier (mgu) of \( s \) and \( s' \). (CD can consequently be seen as a generalized version of the well-known modus ponens.) The (proof) problems consist of deducing a certain given fact \( \lambda_G \) (the goal) from a given set \( Ax \) of facts (the axioms) by applying CD.

* This author was supported by the Deutsche Forschungsgemeinschaft (DFG).
A very common principle used to solve such proof problems algorithmically constitutes the core of CoDE. Essentially, CoDE maintains a set \( F^P \) of so-called potential facts from which it selects and removes one fact \( \lambda \) at a time. \( \lambda \) is put into the set \( F^A \) of activated facts or discarded if it is subsumed by an already existing activated fact \( \lambda' \) (forward subsumption). Only activated facts \( \lambda \in F^A \) are allowed to produce new facts via CD, which then are put into \( F^P \). At the beginning, \( F^A = \emptyset \) and \( F^P = Ax \). The indeterministic selection or activation step is realized with a heuristic weighting function \( \mathcal{H} \) that activates that \( \lambda \in F^P \) with the smallest weight \( \mathcal{H}(\lambda) \in \mathbb{N} \).

CoDE has available a variety of such search-guiding heuristics. The experimental results reported here were obtained using heuristic \( \varpi \) described in [3] and [4], heuristic \( \varpi_{on} \) that is similar to heuristic \( occnest \) introduced in [2], and heuristics with learning capabilities (see [4]).

3 Implementation Details

CoDE is written in C. It is obvious that a simple transformation of the algorithm above into a C program does not lead to an efficient theorem prover. Because of the fact that even powerful heuristics cannot guarantee short proof runs often a lot of potential facts have to be generated and activated entailing the execution of huge numbers of basic operations (matching and unification) and a high demand for memory. Therefore it is imperative to develop fast algorithms for basic operations and sophisticated memory management, in particular techniques to reduce the amount of data.

In order to deal with the first problem our newest version of CoDE employs discrimination-tree indexing ([5]) for the matching routines, as implemented in the equational prover WALDMEISTER ([6]). Moreover, CoDE uses “flatterm”s ([1]) for term representation to speed up the routines for unification. In order to reduce the amount of data during a proof run CoDE utilizes a technique for memory management reminiscent of the technique presented in [8]: If a given memory threshold of \( \tau_{\text{max}} \) bytes (the memory limit) is exceeded, the potential facts with the highest (heuristic) weight are eliminated until the allocated memory falls short of another threshold of \( \tau_{\text{min}} \) bytes (memory quota). Note that this can lead to (theoretical) incompleteness. But since only facts having a high weight are discarded—i.e. facts that would not be activated in an acceptable period of time—this did not lead to practical incompleteness in our experiments.

CoDE employs a particular realization of the inference rule CD. Because of the fact that in addition to the rule presented in section 2 also variants of the rule are used, e.g. from \( o(n(s), t) \) and \( s' \) derive \( \sigma(t) \), where \( \sigma = mgu(s, s') \), \( o, n \) are distinguished function symbols, CoDE uses a more general approach in order to avoid the implementation of new routines for different variants. Our rule is defined as follows: Let \( \vartheta \) be a special so-called pattern term and \( p_1, p_2 \) positions in \( \vartheta \). Then the general form of CD is: from \( s, s' \) derive \( t' \), where \( \mu = mgu(s, \vartheta) \), \( \sigma = mgu(\mu(s)[p_1, s']), t' \equiv \sigma(\mu(s)[p_2]) \). The rule presented in section 2 is specified by \( \vartheta \equiv f(x, y), p_1 = 1, p_2 = 2 \), the rule above by \( \vartheta \equiv o(n(x), y), p_1 = 1, p_2 = 2 \).