# AProVE 1.2: Automatic Termination Proofs in the Dependency Pair Framework[⋆]

Jürgen Giesl, Peter Schneider-Kamp, René Thiemann

LuFG Informatik II, RWTH Aachen, Ahornstr. 55, 52074 Aachen, Germany
{giesl|thiemann|psk}@informatik.rwth-aachen.de

**Abstract.** AProVE 1.2 is one of the most powerful systems for automated termination proofs of term rewrite systems (TRSs). It is the first tool which automates the new *dependency pair framework* [8] and therefore permits a completely flexible combination of different termination proof techniques. Due to this framework, AProVE 1.2 is also the first termination prover which can be fully configured by the user.

## 1 Introduction

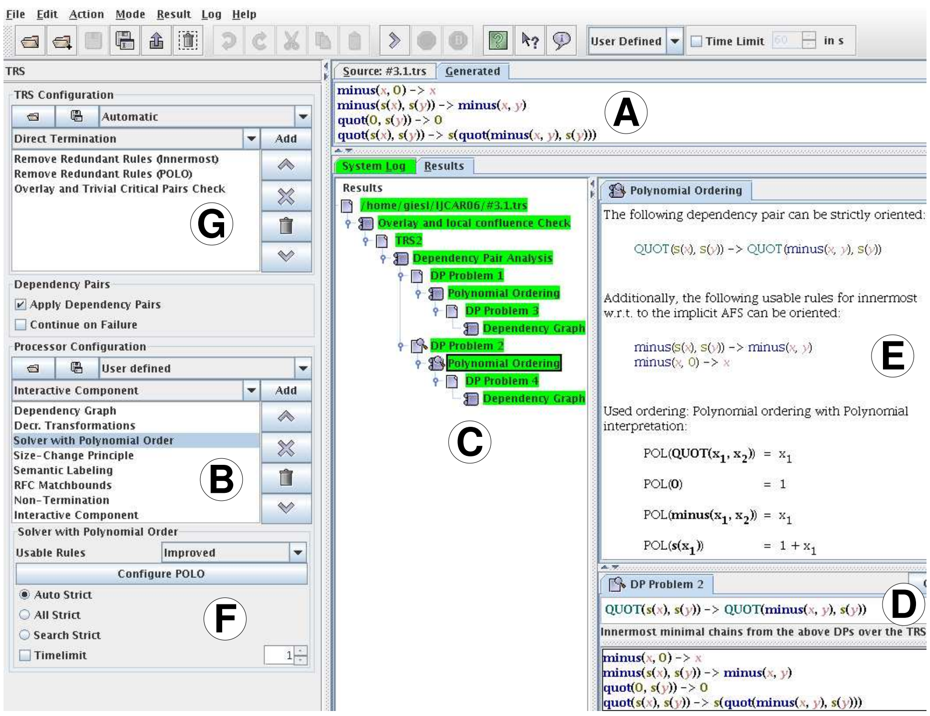
AProVE 1.2 (Automated Program Verification Environment) is a system for automated termination and innermost termination proofs of TRSs. Its predecessor AProVE 1.0 [7] already offered a variety of termination proof techniques. However, there the techniques were applied in a fixed order which could not be influenced by the user. AProVE 1.2 has been totally re-structured (and partly re-implemented) to permit a completely modular combination of the available termination techniques. This increase in modularity of the termination techniques also increases the power of AProVE substantially. The theoretical basis for this re-design is the new *dependency pair (DP) framework* which is briefly recapitulated in Sect. 2. Sect. 3 explains AProVE's structure and shows how the user can configure the tool in order to experiment with self-defined strategies. We conclude in Sect. 4 and describe how to use AProVE in a fully automatic way.

## 2 The Dependency Pair Framework

The DP framework [8] (which was inspired by the cycle analysis algorithm of [12] and which is related to the constraint-based approach of [2, Chapter 7]) is a modular reformulation and improvement of Arts and Giesl's dependency pair approach [1, 5]. Here, root symbols of left-hand sides of rules are called *defined* and all other symbols are *constructors*. For each defined symbol $f$ we introduce a fresh *tuple symbol* $F$. Then for each rule $f(s_1, \ldots, s_n) \to r$ and each subterm $g(t_1, \ldots, t_m)$ of $r$ with defined root $g$, we build a dependency pair $F(s_1, \ldots, s_n) \to G(t_1, \ldots, t_m)$. $DP(\mathcal{R})$ denotes the set of dependency pairs of a TRS $\mathcal{R}$.

In the following screenshot, the `Source` window **(A)** contains the TRS $\mathcal{R}$ under consideration. Here, minus and quot are defined symbols and s and 0 are constructors. Therefore, we have $DP(\mathcal{R}) = \{\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(x, y), \mathsf{QUOT}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(x, y), \mathsf{QUOT}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{QUOT}(\mathsf{minus}(x, y), \mathsf{s}(y))\}$.

⋆ Supported by the Deutsche Forschungsgemeinschaft DFG under grant GI 274/5-1.



```
minus(x, 0) -> x
minus(s(x), s(y)) -> minus(x, y)
quot(0, s(y)) -> 0
quot(s(x), s(y)) -> s(quot(minus(x, y), s(y)))
```

User Defined · Time Limit 60 in s

TRS Configuration: Automatic
Direct Termination
Remove Redundant Rules (Innermost)
Remove Redundant Rules (POLO)
Overlay and Trivial Critical Pairs Check

Dependency Pairs
[✓] Apply Dependency Pairs
[ ] Continue on Failure

Processor Configuration: User defined
Interactive Component
Dependency Graph
Decr. Transformations
Solver with Polynomial Order
Size-Change Principle
Semantic Labeling
RFC Matchbounds
Non-Termination
Interactive Component

Solver with Polynomial Order
Usable Rules: Improved
Configure POLO
(●) Auto Strict
( ) All Strict
( ) Search Strict
[ ] Timelimit 1

```
Results
/home/giesl/IJCAR06/#3.1.trs
  Overlay and local confluence Check
    TRS2
      Dependency Pair Analysis
        DP Problem 1
          Polynomial Ordering
            DP Problem 3
              Dependency Graph
        DP Problem 2
          Polynomial Ordering
            DP Problem 4
              Dependency Graph
```

```
Polynomial Ordering
The following dependency pair can be strictly oriented:

    QUOT(s(x), s(y)) -> QUOT(minus(x, y), s(y))

Additionally, the following usable rules for innermost
w.r.t. to the implicit AFS can be oriented:

    minus(s(x), s(y)) -> minus(x, y)
    minus(x, 0) -> x

Used ordering: Polynomial ordering with Polynomial
interpretation:

    POL(QUOT(x1, x2))  =  x1
    POL(0)             =  1
    POL(minus(x1, x2)) =  x1
    POL(s(x1))         =  1 + x1
```

```
DP Problem 2
QUOT(s(x), s(y)) -> QUOT(minus(x, y), s(y))
Innermost minimal chains from the above DPs over the TRS
minus(x, 0) -> x
minus(s(x), s(y)) -> minus(x, y)
quot(0, s(y)) -> 0
quot(s(x), s(y)) -> s(quot(minus(x, y), s(y)))
```

The DP framework operates on *DP problems* $(\mathcal{P}, \mathcal{R})$ where initially, $\mathcal{P} = DP(\mathcal{R})$.[1] A DP problem $(\mathcal{P}, \mathcal{R})$ is called *finite* if there is no infinite $(\mathcal{P}, \mathcal{R})$-chain, i.e., no infinite sequence of pairs $s_1 \to t_1, s_2 \to t_2, \ldots$ from $\mathcal{P}$ with substitutions $\sigma_i$ such that $t_i\sigma_i$ is terminating w.r.t. $\mathcal{R}$ and such that $t_i\sigma_i \to_{\mathcal{R}}^* s_{i+1}\sigma_{i+1}$ for all $i$. As shown in [1], a TRS $\mathcal{R}$ is terminating iff there is no infinite *chain* of its dependency pairs. So our goal is to prove that the problem $(DP(\mathcal{R}), \mathcal{R})$ is finite.

Termination techniques now operate on DP problems instead of TRSs and are called *DP processors*. Formally, a DP processor *Proc* takes a DP problem as input and returns a new set of DP problems which then have to be solved instead. Alternatively, it can also return "no". A processor *Proc* is *sound* if for all DP problems $d$, $d$ is finite whenever *Proc*$(d)$ is not "no" and all DP problems in *Proc*$(d)$ are finite. *Proc* is *complete* if for all DP problems $d$, $d$ is infinite whenever *Proc*$(d)$ is "no" or when *Proc*$(d)$ contains an infinite DP problem.

Soundness of a DP processor *Proc* is required to prove termination (in particular, to conclude that $d$ is finite if *Proc*$(d) = \varnothing$). Completeness is needed to prove non-termination (in particular, to conclude that $d$ is infinite if *Proc*$(d) =$ no).

So termination proofs in the DP framework start with the initial DP problem $(DP(\mathcal{R}), \mathcal{R})$. Then this problem is transformed repeatedly by sound DP processors. If the final processors return empty sets of DP problems, then termination is proved. If one of the processors returns "no" and all processors used before were complete, then one has disproved termination of the TRS $\mathcal{R}$. So in contrast to AProVE 1.0, AProVE 1.2 can also prove *non-termination*, cf. [9]

[1] For efficiency, AProVE uses a slightly simpler notion of DP problems than [8].

## 3 Structure of AProVE 1.2

Our description of AProVE's structure is based on the windows **(A)** – **(G)** in the screenshot. AProVE 1.2 offers 22 different DP processors. These include virtually all recent techniques and improvements for termination analysis with dependency pairs [6, 8–10, 12, 17] (whereas no other tool implements all of these refinements) as well as processors based on other termination techniques like the *size-change principle* [15, 16], *semantic labeling* [20], and *match-bounds* [4].

In the `Processor Configuration` window **(B)**, the user can select which processors should be used in which order. Whenever AProVE has to solve a DP problem, it first tries the first processor from the list in this window. So in the screenshot, one first applies the `Dependency Graph` processor. Only if a processor does not modify the current problem (i.e., if $Proc(\mathcal{P}, \mathcal{R}) = \{(\mathcal{P}, \mathcal{R})\}$), then AProVE tries the next processor in the list.

In our example, the dependency graph processor determines that any potentially infinite chain either contains infinitely many occurrences of the MINUS- or of the QUOT-dependency pair. Therefore, it transforms the initial DP problem $(DP(\mathcal{R}), \mathcal{R})$ into two new problems (1) $(\{\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(x, y)\}, \mathcal{R})$ and (2) $(\{\mathsf{QUOT}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{QUOT}(\mathsf{minus}(x, y), \mathsf{s}(y))\}, \mathcal{R})$. Now finiteness of the problems (1) and (2) can be proved separately.

This is reflected in the `Results` window **(C)** which depicts the corresponding proof tree. Nodes in the tree (marked with 📄) represent proof obligations. Edges (marked with 📒) represent proof techniques that transform a proof obligation into new proof obligations. In the screenshot, the node "`TRS2`" is the proof obligation which corresponds to the TRS $\mathcal{R}$ and the edge "`Dependency Pair Analysis`" is the proof technique which transforms $\mathcal{R}$ into the initial DP problem $(DP(\mathcal{R}), \mathcal{R})$ and immediately applies the dependency graph processor. All further nodes in the resulting subtrees are DP problems and all further edges are applications of DP processors. So "`DP Problem 1`" and "`DP Problem 2`" are the MINUS- and QUOT-problems (1) and (2) above.

If one clicks on a node or on an edge of the proof tree, then more information on the respective proof obligation or proof technique is displayed in the windows on the right. In the screenshot, the `Proof Obligation` window **(D)** depicts `DP Problem 2` and the `Proof Technique` window **(E)** provides details on the DP processor which was used to transform `DP Problem 2` further. Here, a *reduction pair processor based on polynomial orders* was applied (called "`Solver with Polynomial Order`").[2] For a DP problem $(\mathcal{P}, \mathcal{R})$, this processor tries to find a polynomial order such that all rules in $\mathcal{P}$ and $\mathcal{R}$ are at least weakly decreasing (i.e., $l \succsim r$ for all $l \to r \in \mathcal{P} \cup \mathcal{R}$) and it removes all pairs from $\mathcal{P}$ which are strictly decreasing (i.e., all $l \to r \in \mathcal{P}$ with $l \succ r$). Moreover, under some conditions, it is sufficient if just certain "usable" rules in $\mathcal{R}$ are weakly decreasing. In the screenshot, AProVE found a polynomial order where the only dependency pair of `DP Problem 2` is strictly decreasing. Hence, applying this processor results in `DP Problem 4`, which is $(\varnothing, \mathcal{R})$. Finally, another application of the dependency graph processor to `DP Problem 4` results in no remaining proof obligations. `DP`

[2] AProVE also offers RPOS, KBO, or polynomial orders with negative coefficients [11].

`Problem 1` can be solved in a similar way. Therefore, termination of this example is proved. The generated proof can then be exported as an `html`- or `LaTeX`-file.

AProVE 1.2 is indeed fully configurable by the user, since the user can compose the list of processors in the `Processor Configuration` window **(B)**. Moreover, for each processor, the user can determine its parameters in window **(F)**. So for the `Solver with Polynomial Order`, the user can impose a timeout, choose the method to compute the usable rules and the algorithm for finding strictly decreasing dependency pairs, and determine the degree of the polynomials and the range for their coefficients (by clicking on "`Configure POLO`").

For particularly challenging examples and to develop new heuristics, one can include an "`Interactive Component`" processor in the `Processor Configuration` window **(B)**. The interactive component displays the current DP problem together with all available DP processors. Then the user can select a processor manually and apply it. Afterwards, the list of processors in the `Processor Configuration` window is applied again on the resulting DP problems. Thus, to use the interactive component only if all other DP processors fail, this component should be at the end of the list in the `Processor Configuration` window.

For efficiency, it is often recommendable to simplify the initial TRS before transforming it into a DP problem. Suitable simplification techniques can be chosen in the `TRS Configuration` window **(G)**. Here, the user can select which simplifications should be applied in which order. AProVE starts with applying the first technique in the list to the given TRS. In contrast to the application of DP processors, AProVE does not start with the first technique in the list again when the TRS has been modified by one of the simplifications. Instead, then the second technique is applied to the modified TRS, etc.

One of the most important simplifications is the `Overlay and Trivial Critical Pairs Check`. Under certain conditions, the obligation to prove termination of a TRS can be relaxed to prove only *innermost* termination. The advantage is that innermost termination is often easier to show than termination. Therefore, DP problems also have a flag which indicates whether one wants to prove full or just innermost termination. Depending on this flag, the DP processors behave differently and they are often more powerful for innermost termination.

Finally, AProVE has an extensive online `Help` (by clicking on ?) and a context-dependent help (by clicking on ↖? and selecting any item in the GUI).

## 4 Using AProVE 1.2

For users who do not want to configure AProVE themselves, the "`User Defined Mode`" in the top right corner can be changed into a fully "`Automatic Mode`", where AProVE runs with a fixed list of DP processors. In this setting, processors are even applied in parallel. This mode of AProVE 1.2 corresponds to the one used in the *International Competition of Termination Tools* 2005. In this competition, AProVE 1.2 was the most powerful system for termination analysis of TRSs.[3] The reason is that AProVE is the only tool which features most

[3] The other termination provers for TRSs were CiME [3], Matchbox [19], Teparla [18], TPA [14], TTT [13], cf. `http://www.lri.fr/~marche/termination-competition/`.

modern termination techniques for TRSs and which permits to combine them in a completely flexible way. This combination can even be determined and configured by the user. In addition to ordinary TRSs, AProVE 1.2 also analyzes the termination of several other formalisms, e.g., of conditional TRSs and logic programs. In contrast to AProVE 1.0 it also handles TRSs modulo AC and context-sensitive TRSs. Its power in these areas is again demonstrated by the respective competitions. AProVE 1.2 is written in Java and can be downloaded from `http://aprove.informatik.rwth-aachen.de/`. At this URL one can also run AProVE in fully "`Automatic Mode`" directly via the web on a parallel computer.

## References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
2. C. Borralleras. *Ordering-based methods for proving termination automatically.* PhD thesis, Universitat Politècnica de Catalunya, 2003.
3. E. Contejean, C. Marché, B. Monate, and X. Urbain. CiME. `http://cime.lri.fr`.
4. A. Geser, D. Hofbauer, and J. Waldmann. Match-bounded string rewriting systems. *Applicable Algebra in Eng., Comm. and Computing*, 15(3,4):149–171, 2004.
5. J. Giesl, T. Arts, and E. Ohlebusch. Modular termination proofs for rewriting using dependency pairs. *Journal of Symbolic Computation*, 34(1):21–58, 2002.
6. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Improving dependency pairs. In *Proc. 10th LPAR*, LNAI 2850, pages 165–179, 2003.
7. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with AProVE. In *Proc. 15th RTA*, LNCS 3091, pages 210–220, 2004.
8. J. Giesl, R. Thiemann, and P. Schneider-Kamp. The DP framework: Combining techn. for aut. termination proofs. *Proc. 11th LPAR*, LNAI 3452, p. 301-331, 2005.
9. J. Giesl, R. Thiemann, and P. Schneider-Kamp. Proving and disproving termination of higher-order functions. *Proc. 5th FroCoS*, LNAI 3717, pp. 216–231, 2005.
10. N. Hirokawa and A. Middeldorp. Dependency pairs revisited. In *Proc. 15th RTA*, LNCS 3091, pages 249–268, 2004.
11. N. Hirokawa and A. Middeldorp. Polynomial interpretations with negative coefficients. In *Proc. 7th AISC*, LNAI 3249, pages 185–198, 2004.
12. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1,2):172–199, 2005.
13. N. Hirokawa and A. Middeldorp. Tyrolean Termination Tool. In *Proc. RTA '05*, LNCS 3467, pages 175–184, 2005.
14. A. Koprowski. TPA: Termination proved automatically. In *Proc. 17th RTA*, LNCS, 2006. To appear.
15. C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. In *Proc. 28th POPL*, pages 81–92, 2001.
16. R. Thiemann and J. Giesl. The size-change principle and dependency pairs for termination of term rewriting. *AAECC*, 16(4):229–270, 2005.
17. R. Thiemann, J. Giesl, and P. Schneider-Kamp. Improved modular termination proofs using dependency pairs. *Proc. 2nd IJCAR*, LNAI 3097, pp. 75–90, 2004.
18. J. v. d. Wulp. Teparla. `http://www.win.tue.nl/~hzantema/torpa.html`
19. J. Waldmann. Matchbox: A tool for match-bounded string rewriting. In *Proc. 15th RTA*, LNCS 3091, pages 85–94, 2004.
20. H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995.