TPLP: Page 1–8. © The Author(s), 2021. Published by Cambridge University Press 2021
doi:10.1017/xxxxx

*Abstract Environment Trimming**

DANIEL JURJO-RIVAS† JOSE F. MORALES†
PEDRO LÓPEZ-GARCÍA†† MANUEL V. HERMENEGILDO†

† *Universidad Politécnica de Madrid (UPM) & IMDEA Software Institute, Spain*
†† *Spanish Council for Scientific Research & IMDEA Software Institute, Spain*

submitted xx xx xxxx; revised xx xx xxxx; accepted xx xx xxxx

Abstract

Variable sharing is a fundamental property in the static analysis of logic programs, since it is instrumental for ensuring correctness and increasing precision while inferring many useful program properties. Such properties include modes, determinacy, non-failure, cost, etc. This has motivated significant work on developing abstract domains to improve the precision and performance of sharing analyses. Much of this work has centered around the family of *set-sharing* domains, because of the high precision they offer. However, this comes at a price: their scalability to a wide set of realistic programs remains challenging and this hinders their wider adoption. In this work, rather than defining new sharing abstract domains, we focus instead on developing techniques which can be incorporated in the analyzers to address aspects that are known to affect the efficiency of these domains, such as the number of variables, without affecting precision. These techniques are inspired in others used in the context of compiler optimizations, such as expression reassociation and variable trimming. We present several such techniques and provide an extensive experimental evaluation of over 1100 program modules taken from both production code and classical benchmarks. This includes the Spectector cache analyzer, the s(CASP) system, the libraries of the `Ciao` system, the `LPdoc` documenter, the PLAI analyzer itself, etc. The experimental results are quite encouraging: we have obtained significant speed-ups, and, more importantly, the number of modules that require a timeout was cut in half. As a result, many more programs can be analyzed precisely in reasonable times.

1 Introduction

Abstract Interpretation (Cousot and Cousot 1977) allows constructing sound program analysis tools that can extract properties of a program by safely approximating its semantics. Abstract interpretation-based analysis was proved practical and effective in the context of (Constraint) Logic Programming ((C)LP) (García de la Banda et al. 1996; Kelly et al. 1998; Le Charlier and Van Hentenryck 1994; Muthukumar and Hermenegildo 1990; 1992; Van Roy and Despain 1990; Warren et al. 1988), which was also one of its very first application areas (Giacobazzi and Ranzato 2022), and the techniques developed for CLP have also proved useful in the analysis and verification of other programming languages by using semantic translation into Constraint Horn Clauses (CHCs) (De Angelis et al. 2021; Henriksen and Gallagher 2006; Méndez-Lojo et al. 2007a). In the context of static analysis of (C)LP programs, variable sharing soon emerged as a fundamental property, which has led to very active and continuous development of variable sharing analysis domains. Sharing

* Partially funded by MICINN projects PID2019-108528RB-C21 *ProCode*, TED2021-132464B-I00 *PRODIGY*, and by the Tezos foundation.
We also thank the anonymous reviewers for their very useful feedback.
The authors declare that they have no competing interests.

proved immediately necessary for ensuring correctness and precision while inferring most other useful program properties such as modes, determinacy, non-failure, and cost, among others. In fact, some early LP analyses were actually incorrect because variable sharing was not taken into account. Sharing analyses have also proven fundamental in the optimization of unification (Søndergaard 1986) and in automatic (and-)parallelization (Hermenegildo and Rossi 1995; Cabeza and Hermenegildo 1994; Bueno et al. 1999; Pontelli et al. 1997; García de la Banda et al. 2000). E.g., in parallelization it is crucial to precisely detect whether two variables are independent (i.e., they do not *share*), a variable is ground, etc. Sharing has also been used as the basis for more complex analyses for related properties such as linearity, paths, or freeness (King and Soper 1994; Amato and Scozzari 2014; Amato et al. 2022; Bruynooghe and Codish 1993; Muthukumar and Hermenegildo 1991). Furthermore, beyond (C)LP, sharing analysis is directly related to *aliasing* and *points-to* analyses in imperative programming, widely used in the context of languages with pointers and dynamic memory (Aiken et al. 2003; Landi and Ryder 1992; Steensgaard 1996; Bravenboer and Smaragdakis 2009; Navas et al. 2009; Rountev et al. 2001; Whaley and Lam 2002), sometimes applying directly domains stemming from (C)LP (Zanardini 2018; Méndez-Lojo and Hermenegildo 2008). In fact, (C)LP sharing analyses constituted some of the very first abstract interpretation-based pointer aliasing analyses for any programming language.

In this paper we concentrate on a popular abstract domain for variable sharing analysis: *set-sharing* analysis (Jacobs and Langen 1989; Muthukumar and Hermenegildo 1989). This domain captures which *sets* of program variables share run-time variables, encoding this information in *sharing sets*. For example, assume X, Y, and Z are the syntactic program variables that we need to track, and consider the substitution (run-time store) $\{X/f(M,K,a), Y/g(b,M), Z/g(a,b)\}$. This substitution is abstracted to the sharing set $\{\{X\},\{X,Y\}\}$, where $\{X,Y\}$ represents that there is (or, more precisely, may be) one or more variables shared in the terms to which X and Y are bound, and $\{X\}$ represents that there is one or more variables that appear only in X. Additionally, Z not appearing in any set means that it contains no variables, i.e., Z is bound to a ground term. Set-sharing encodes not only variable independence, i.e., the fact that substitutions that affect a given variable will not affect another one, but also groundness, grounding dependencies (e.g., the fact that if Y becomes ground then X becomes ground, but not the other way around), independence relationships, etc. This representation richness does come, however, at a price: the scalability of set-sharing domains to a wide set of realistic programs is challenging, since the number of sharing sets carried by the abstraction can be exponential in the number of variables of the clause being analyzed. This has prompted much work in improvements and alternative representations of set-sharing abstractions, with the objective of improving performance while maintaining precision as much as possible. In contrast, in this work, rather than defining new sharing abstract domains or modifying existing ones, we concentrate on developing techniques that can be incorporated in the analysis framework to address the root causes of the performance issues faced by the set-sharing domains, such as the number of variables, without affecting precision. We draw inspiration from techniques used in the context of compiler optimizations, which significantly reduce the number of variables presented in the abstractions.

The rest of the paper is structured as follows: First, Section 2 provides the necessary background, covering set-sharing abstract domains and top-down analysis. Section 3 presents our approach, offering first a program transformation that can provide an optimal solution; and second, an alternative solution based on *variable trimming* that can be applied during analysis without modifying the program. Section 4 reports our experimental evaluation and finally, Section 5 summarizes our conclusions and discusses some lines of future work.

2 Notation and Preliminaries

We represent variables by uppercase letters (for example: X, Y, Z, etc.) and atoms by lowercase letters (for example: a, b, c, etc.). $\mathcal{P}(S)$ denotes the powerset of set S and $\mathcal{P}^0(S)$ the *proper* powerset of set S, i.e., $\mathcal{P}^0(S) = \mathcal{P}(S) \backslash \{\emptyset\}$. Given a term `T`, `vars(T)` denotes the set of its variables. A Constraint Logic Program (CLP) is a set of *clauses* of the form H :- $A_1, \ldots, A_n$ where $A_1, \ldots, A_n$ are *literals* that form the *body* and H is a positive literal said to be the *head* of the clause. A *substitution* is a set $\theta = \{V_1/t_1, \ldots, V_n/t_n\}$ with V_i distinct variables and t_i terms. We say that t_i is the *value* of V_i in θ. The set $\{V_1, \ldots, V_n\}$ is the *domain* of θ (`dom`(θ)).
The main idea behind *Abstract Interpretation* (Cousot and Cousot 1977) is to interpret the program over a special, abstract domain whose elements are finite representations of possibly infinite sets of actual substitutions in the concrete domain. We denote the concrete domain as D_γ, the abstract domain as D_α, and the functions that relate sets of concrete substitutions with abstract substitutions as the *abstraction* function $\alpha : D_\gamma \to D_\alpha$ and the *concretization* function $\gamma : D_\alpha \to D_\gamma$. The concrete domain is typically a complete lattice with the set inclusion order which induces an ordering relation in the abstract domain represented by $\sqsubseteq$. Under this relation the abstract domain is usually a complete lattice and $(D_\gamma, \alpha, D_\alpha, \gamma)$ is a Galois insertion/connection (Cousot and Cousot 1977). Several frameworks for abstract interpretation exist; this work focuses on *top-down frameworks*, discussed in Section 2.

Set-Sharing Analyses. As mentioned in the introduction, in static analysis of logic programs, tracking of variables shared among terms is essential. A set of program variables $V_1, \ldots, V_m$ *share* if, in some execution of the program, they may respectively be bound to terms $\mathtt{T}_1, \ldots, \mathtt{T}_m$ such that $vars(\mathtt{T}_1) \cap \ldots \cap vars(\mathtt{T}_m) \neq \emptyset$. One of the most accurate abstract domains defined for tracking sharing information is *set-sharing* (Jacobs and Langen 1989; Muthukumar and Hermenegildo 1989). This domain captures whether at run-time there are variables sharing, condensing this information in a concise set representation. As an example, consider program variables `X`, `Y`, `Z`, `W`, `T`, and assume they are bound at run-time as follows: θ = {`X/f(M,a), Y=g(b,M,c), Z/[1,M,3], W/g(b), T/h(K,m)`}. This substitution (run-time state) is represented in the *set-sharing* domain by the *sharing* abstraction {{`X,Y,Z`}, {`T`}}. The first element, {`X,Y,Z`}, represents the fact that there is (at least one) variable (i.e., `M`) that appears in all of `X`, `Y`, `Z`, but not in `T` or `W`; and the second element, {`T`}, represents that there is (at least one) variable that appears in `T` (i.e., `K`) but not in any of the others. We say that `X`, `Y` and `Z` "share", and that `T` does not "share" with `X`, `Y`, or `Z`. The fact that `W` does not appear in any set means it contains no variables and thus, it is ground. This representation also captures that there are no other variables in `X`, `Y`, or `Z` in addition to the one(s) they share, which has important implications with respect to grounding: after a program statement that grounds one of them (e.g., `Z=[1,2,3]`), we know both `X` and `Y` will also be grounded. However, grounding `T` does not ground any of `X`, `Y`, or `Z`. Other abstract domains have also been studied, notably the *pair-sharing*, which keeps track of *pairs* of variables that share. E.g., for the example above, its pair-sharing abstraction is: {(`X`, `Y`), (`Y`, `Z`), (`X`, `Z`)}. The intricacies of the relation and trade-offs between set-sharing and pair-sharing are beyond the scope of this paper; the reader is referred to, for example, Bagnara et al. (1997) and Bueno and García de la Banda (2004) for further discussion of this topic. However, our subject of study in this work is set-sharing analyses.

The set-sharing domain has attracted a lot of attention in the literature and has been enhanced in different ways and extended with other kinds of information such as *freeness* or *linearity* (Muthukumar and Hermenegildo 1991; Bruynooghe et al. 1994; Filé 1994; King

and Soper 1994; Codish et al. 1996; Fecht 1996; Zaffanella et al. 1999; Hill et al. 2004; Navas et al. 2006; Trias et al. 2008; Amato and Scozzari 2009; Amato et al. 2022).

However, the set representation, which allows the sharing domain to offer high precision, is also one of the reasons why this family of domains presents scalability challenges. A set-sharing abstraction is presented as a set of sets, each of them capturing a *possible* sharing that occurs at runtime. In cases where there is not much (or any) sharing information at runtime, more (or all the) sharing sets are possible. Given a set of variables appearing in a program being analyzed (V), the size of a set-sharing abstraction has an upper bound given by the abstraction which captures all the possible non-empty sharing sets ($\mathcal{P}^0(V)$). Thus, the size of an abstraction is, in the worst case, exponential in terms of the number of variables that appear in the program. To overcome these problems, various representations have been proposed, such as collapsing subsets of the abstraction into "cliques" (sets of variables that represent the proper powerset of those variables). These representations allow for a reduction in the size of the sharing abstraction and can improve performance, even more so when equipped with widenings (albeit then at the cost of losing precision) (Zaffanella et al. 1999; Navas et al. 2006). For example, the set-sharing abstraction $\{\{\texttt{X}\},\{\texttt{X,Y}\},\{\texttt{Y}\},\{\texttt{Z}\}\}$ can be represented using cliques as the tuple $(\{\{\texttt{X,Y}\}\},\{\{\texttt{Z}\}\})$ where the clique $\{\texttt{X,Y}\}$ represents $\mathcal{P}^0(\{\texttt{X,Y}\})$.

The PLAI Top-Down Analyzer. *Top-down* analyses are a family of static analyses that build an *analysis graph* starting from a series of program *entry points*. This approach was first used in analyzers such as MA3 and Ms (Warren et al. 1988), and matured in the PLAI analyzer (Muthukumar and Hermenegildo 1990; 1992) using an optimized fixpoint algorithm now also referred to as the *top-down algorithm* or *solver*. This algorithm was later applied to the analysis of CLP/CHCs (García de la Banda et al. 1996) and imperative programs (De Angelis et al. 2021; Henriksen and Gallagher 2006; Méndez-Lojo et al. 2007a;b), and used in analyzers such as GAIA (Le Charlier and Van Hentenryck 1994), the CLP($\mathcal{R}$) analyzer (Kelly et al. 1998), or Goblint (Seidl and Vogler 2021; Tilscher et al. 2023).

The graph constructed by the PLAI algorithm during analysis is a finite, abstract object whose concretization approximates the (possibly infinite) set of (possibly infinite) maximal AND-trees of the concrete semantics. This approach separates the abstraction of the structure of the concrete trees (the paths through the program) from the abstraction of the *substitutions* at the nodes in those concrete trees (the program states in those paths). The first abstraction, T_α, is typically built-in, as an abstract domain of *analysis graphs*. The framework is *parametric* on a second abstract domain, D_α, whose elements appear as labels in the nodes of the analysis graph. A more detailed recent discussion can be found in (De Angelis et al. 2021). Assume we are analyzing a literal `Goal` in the body of some clause in the program, and that `Head :- Body` is a clause in a predicate whose head unifies with `Goal`. Assume also that the substitution affecting `Goal` at the time of this call is approximated by the abstract substitution `Call` such that $\texttt{vars}(\texttt{Goal})\subseteq\texttt{dom}(\texttt{Call})$ and $\texttt{vars}(\texttt{Call})\cap(\texttt{vars}(\texttt{Head})\cup\texttt{vars}(\texttt{Body}))=\emptyset$. The success (exit state) of `Goal` after having executed the above clause is represented by the abstract substitution `Success` given by:

$$\begin{aligned}\texttt{Success} &= \texttt{extend}(\texttt{Call},\texttt{Goal},\texttt{Prime})\\ \texttt{Prime} &= \texttt{exitToPrime}(\texttt{project}(\texttt{vars}(\texttt{Head}),\texttt{Exit}),\texttt{Head},\texttt{Goal})\\ \texttt{Exit} &= \texttt{entryToExit}(\texttt{Entry},\texttt{Head},\texttt{Body})\\ \texttt{Entry} &= \texttt{augment}(\texttt{vars}(\texttt{Body})\backslash\texttt{vars}(\texttt{Head}),\texttt{callToEntry}(\texttt{Proj},\texttt{Goal},\texttt{Head}))\\ \texttt{Proj} &= \texttt{project}(\texttt{vars}(\texttt{Goal}),\texttt{Call})\end{aligned}$$

As an example, let `Goal` = `p(A,f(B),E)`, `Call` = $\{\{\texttt{A}\},\{\texttt{B,C}\},\{\texttt{A,C,D}\}\}$ (notice that `E` is ground, since it does not appear in any sharing set) and `Head :- Body` be the clause

`p(f(X),f(Y),Z) :- [X|T1]=[X,Y|T2]`, whose `Head` unifies with `Goal`. The success abstraction is computed as follows:

i) First, the abstraction `Call` is *projected* onto the variables in `Goal` by means of the `project` function, obtaining `Proj` $= \{\{\texttt{A}\}, \{\texttt{B}\}\}$.

ii) Then, `callToEntry`(`Proj`, `Goal`, `Head`) yields an abstract substitution that represents the unification `p(A,f(B),E)`=`p(f(X),f(Y),Z)` (i.e., `Goal`=`Head`) in the context represented by `Proj`. In our example, such abstraction is $\{\{\texttt{X}\}, \{\texttt{Y}\}\}$, where `Z` is becomes ground because it is unified with `E`, which is ground.

iii) Now, the domain of such abstraction is extended with the variables in `Body` that do not appear in `Head` (i.e., `T1` and `T2`), and the abstraction is updated accordingly by including safely approximated information. This is done by operation `augment`, which returns the `Entry` abstraction $\{\{\texttt{X}\}, \{\texttt{Y}\}, \{\texttt{T1}\}, \{\texttt{T2}\}\}$.

iii) Then, `Body` is traversed so that each of its literals are (recursively) analyzed by procedure entryToExit, described in Algorithm 1.
In our example, entryToExit(`Entry`, `Head`, `Body`) proceeds as follows:

a) First, the `Exit` abstraction is initialized with the current `Entry` abstraction (Line 2), and the first literal of the body is selected (Line 3), which in this case corresponds to the only literal in the body: `[X|T1]=[Z,Y|T2]`.

b) The PLAI framework proceeds differently depending on the kind of literal being analyzed (see Lines 4–15). Since the literal `[X|T1]=[Z,Y|T2]` is neither a recursive call nor a call to a predicate in the analysis scope, the steps in Lines 9–15 are performed as explained below.

c) First, the abstraction is projected onto the variables in the literal, returning $\{\{\texttt{X}\}, \{\texttt{Y}\}, \{\texttt{T1}\}, \{\texttt{T2}\}\}$) and the operation `abstractLiteral` is invoked, which captures the abstract behavior of the literal. In our example, it performs the abstract unification `[X|T1]=[Z,Y|T2]`. Since `Z` is ground, and the unification induces `X=Z`, the groundness information is propagated to `X`. Such unification also induces `T1=[Y|T2]`, which results in the creation of new sharing sets accordingly. The $\texttt{Abs}_{\texttt{exit}}$ abstraction obtained after these operations is $\{\{\texttt{Y}, \texttt{T1}\}, \{\texttt{Y}, \texttt{T1}, \texttt{T2}\}, \{\texttt{Y}, \texttt{T2}\}\}$.

d) Finally, such abstraction is used to update the previous exit abstraction by the `extend` operation (Line 15), yielding `Exit`$=\{\{\texttt{Y}, \texttt{T1}\}, \{\texttt{Y}, \texttt{T1}, \texttt{T2}\}, \{\texttt{Y}, \texttt{T2}\}\}$.

iv) After the execution of entryToExit, the obtained `Exit` abstraction is projected over `vars`(`Head`) and represented in the context of the variables of `Call`. This is done by operation `exitToPrime`(`project`(`vars`(`Head`), `Exit`), `Head`, `Goal`), which captures the effects of the unification `Head`=`Goal`. In our example, it yields `Prime`$=\{\{\texttt{B}\}\}$, since the groundness information has been propagated from `X` to `A`.

v) The analysis concludes with the `extend`(`Call`, `Prime`) operation, which *updates* the initial `Call` abstraction with the new inferred information in the `Prime` abstraction, obtaining the success abstraction: `Success` $= \{\{\texttt{B,C}\}\}$, where the sharing-set `{A,C,D}` has been deleted because `A` is ground, and such information is propagated to `D`.

As some final remarks, if no predicate head can be unified with the goal under analysis, a bottom abstraction ($\bot$) is returned (representing that the exit state is unreachable). If several clauses are available, all of them are analyzed, and a collection of *prime* abstractions $\texttt{Prime}_1, \ldots, \texttt{Prime}_m$ is obtained, one abstraction per clause, where m is the number of clauses. Then, the success abstraction is computed as `Success`=`extend`(`Call`,`computeLub`($\texttt{Prime}_1, \ldots, \texttt{Prime}_m$)), where `computeLub` yields the

Algorithm 1 A schematic description of `entryToExit`

```
1:  function ENTRYTOEXIT(Entry, Head, Body)
2:      Exit ← Entry
3:      for Literal ∈ Body do
4:          if RECURSIVE-CALL(Literal, Head) then
5:              Exit ← COMPUTE-FIXPOINT(Exit, Literal)
6:          else if PREDICATE-IN-SCOPE(Literal) then
7:              Exit ← ANALYZE-PRED(Exit, Literal)
8:          else
9:              Proj ← project(vars(Literal), Exit)
10:             MaybeAbs ← abstractLiteral(Literal, Proj)
11:             if MaybeAbs = fail then
12:                 Abs_exit ← topmost(vars(Literal), Proj)
13:             else
14:                 Abs_exit ← MaybeAbs
15:             Exit ← extend(Exit, Abs_exit)
16:     return Exit
```

least upper bound of the collection of abstractions (other operators, including disjunction and widenings, are possible).

In the ENTRYTOEXIT loop (Lines 3-15), when the current literal under analysis corresponds to a recursive call (Lines 4-5), the analyzer computes a fixpoint for the associated call pattern. Such call pattern is determined by the current literal `Goal` and the abstraction `Call` representing the environment under which `Goal` is executed (this may require the use of a widening operation to ensure termination). A detailed discussion on the different fixpoint computation methods is outside the scope of this work and we believe that it is not necessary for understanding our approach and contribution. The reader is referred to, e.g., Muthukumar and Hermenegildo (1990; 1992) for more details.

In the case that the current literal is not recursive but corresponds to a call to a predicate within the analysis scope, the associated call pattern is analyzed (Lines 6-7) following steps i) to v) illustrated above using our example, with the clauses that unify with the literal.

Finally, if the literal to be analyzed does not correspond to any of the two cases discussed above and the analysis domain does not know how to abstract it either (Lines 9–11), the invocation to the `abstractLiteral` operation returns a `fail` atom. The analyzer then computes the *top-most* information for `vars(Literal)` by calling the `topmost` function (Line 12). Then, the original `Call` abstraction is extended with such top-most information. In our example, if the abstract domain did not implement how to abstract the unification `[X|T1]=[Z,Y|T2]`, the top-most abstraction would be $\mathcal{P}^0(\{\texttt{X,T1,T2,Y,Z}\})$. Notice that this can be quite common when, for example, the analyzer has to process a call to a predicate which is imported from a library whose source code is not available, is not in the analysis scope, etc. In this case, the top-most abstraction has to be assumed to ensure correctness.

3 Environment Reassociation and Abstract Environment Trimming

Given a clause H :- $B_1, \ldots, B_n$, its *environment* is the set of all the variables appearing in the clause, defined as $\texttt{vars}(H) \cup \bigcup_{i=1}^{n} \texttt{vars}(B_i)$. As mentioned in Section 2, a set-sharing abstraction at any analysis point contains, at most, $2^V - 1$ sharing sets, where V is the size of the environment. Intuitively, a clause should be faster to analyze if it has fewer variables.

Since it is not possible to artificially reduce the number of variables present in a clause, we propose two techniques: rearranging the literals of the body into new predicates, and modifying the domain of the abstraction without altering the clause being analyzed.

3.1 Environment Reassociation

Expression reassociation (Briggs and Cooper 1994), also known as reordering or restructuring, is a technique that involves changing the grouping of terms in an expression without altering its overall value. It is used for optimization purposes, such as improving performance, reducing floating-point errors, eliminating redundancies, etc.

Given a clause H :- $B_1, \ldots, B_n$, a *partition* is a collection $P_1, \ldots, P_s$ such that each P_j is a subsequence of *consecutive* literals, $P_j = B_i, B_{i+1}, \ldots, B_k$ with $1 \leq i < k \leq n$, such that given P_j,P_{j+1} with $j \in \{1, s-1\}$: a) if the first element of P_{j+1} is B_k, then the last element of P_j is B_{k-1}, b) $B_1 \in P_1$ and c) $B_n \in P_s$.

Folding each of the literals encapsulated by each P_i generates new auxiliary predicates whose environments are smaller (or equal) than the environment of the original predicate. This procedure can be repeated recursively over the auxiliary predicates obtaining a new collection of predicates with reduced environments. Finally, our focus is to find an *optimal* partition. An optimal partition is a (possibly recursive) partition where the number of variables in the environments of each of the auxiliary predicates generated is minimal.

Consider the clause of predicate `qplan/3` shown in Figure 1a (the meaning of the comments will be explained later). We refer to body literals by $\mathtt{L}_i$, where i is a position in the body of the clause. For example, $\mathtt{L}_4$=`mark(P0,L,0,V1)`.

The collections P_1=$\mathtt{L}_1, \ldots, \mathtt{L}_3$, P_2=$\mathtt{L}_4, \ldots, \mathtt{L}_6$, and P_3=$\mathtt{L}_7, \ldots, \mathtt{L}_9$ define a partition of the predicate `qplan`. This partition, when folded, generates three auxiliary predicates: `aux1(P0,X0,Vg,N)` :- $\mathtt{L}_1, \ldots, \mathtt{L}_3$, `aux2(P0,Vg,P2)` :- $\mathtt{L}_4, \ldots, \mathtt{L}_6$, and `aux3(N,X0,P2,X,P)` :- $\mathtt{L}_7, \ldots, \mathtt{L}_9$, with environments containing 5, 6, and 6 variables respectively. Finally, Figure 1b presents a transformation of `qplan` obtained by recursively reassociating the predicate. Each auxiliary clause is annotated with the worst case size for any set-sharing abstraction traversing it. While in the original program the maximum size is 2^{12}-1, it is reduced to 2^6-1 in the transformed program.

3.2 Abstract Environment Trimming

The technique of environment reassociation described before, allows obtaining, given a clause, a collection of clauses where the number of variables appearing in each one is reduced with respect to the original one. However, applying transformations over the program under analysis may not always be desirable. In this section we provide an alternative approach, where the domain of abstractions is dynamically modified as the analysis of a clause processes each body literal. Although the resulting abstraction domains might not be optimal, this technique avoids the transformations, since such dynamic domain modifications are performed as part of the abstract operations. Given a clause `Head` :- $\mathtt{B}_1, \ldots, \mathtt{B}_n$ a variable `X` is a *live variable* while analyzing $\mathtt{B}_i$ (that we refer to as the analysis point $\mathtt{B}_i$) if `X` $\in$ `vars(Head)` or there exists j, k $1 \leq j \leq i \leq k \leq n$ such that `X` belongs to both `vars`($\mathtt{B}_j$) and `vars`($\mathtt{B}_k$). Conversely, a variable `X` is a *dead variable* at analysis point $\mathtt{B}_i$ if it does not appear in the body after such point, i.e., `X` $\notin \bigcup_{j=i}^{n}$ `vars`($\mathtt{B}_j$). Our definition of liveness is similar to imperative programming, with the difference that variables are not reassigned and that variables become live on the first appearance, since logic variables do not need to be declared in the body of a clause (Aho et al. 2006, p. 608-610). Figure 1a shows, at each program point, which body variable lives or dies. In that sense it is reminiscent of the concept of *environment trimming* used in the Warren Abstract Machine (Warren 1987;

Figure 1: `qplan/3` predicate and its environment trimming-based transformation.

1a `qplan/3` definition (with annotations on live variables).

```
1  qplan(X0,P0,X,P) :-            % #Abs < 2^12
2     numbervars(X0,0,I),         % I alive
3     variables(X0,0,Vg),         % I,Vg alive
4     numbervars(P0,I,N),         % I,Vg,N alive
5     mark(P0,L,0,Vl),            % Vg,N,L,Vl alive and I dies
6     schedule(L,Vg,P1),          % Vg,N,L,Vl,P1 alive
7     quantificate(Vl,0,P1,P2),   % N,Vl,P1,P2 alive and Vg,L die
8     functor(VA,f,N),            % N,P2,VA alive and Vl,P1 die
9     melt(X0,VA,X),              % P2,VA alive and N dies
10    melt(P2,VA,P).              % P2,VA alive
```

```
1  qplan(X0,P0,X,P) :- % #Abs < 2^6
2      qplan_aux1(X0,P0,P2,VA),
3      qplan_aux2(X0,X,P,P2,VA).
4
5  qplan_aux1(X0,P0,P2,VA) :- % #Abs < 2^5
6      qplan_aux11(X0,P0,P2,N),
7      functor(VA,f,N).
8
9  qplan_aux11(X0,P0,P2,N) :- % #Abs < 2^5
10     qplan_aux111(X0,P0,N,Vg),
11     qplan_aux112(P0,P2,Vg).
12
13 qplan_aux111(X0,P0,N,Vg) :- % #Abs < 2^5
14     qplan_aux1111(X0,Vg,I),
15     numbervars(P0,I,N).
```

```
16 qplan_aux1111(X0,Vg,I) :- % #Abs < 2^3
17     numbervars(X0,0,I),
18     variables(X0,0,Vg).
19
20 qplan_aux112(P0,P2,Vg) :- % #Abs < 2^5
21     mark(P0,L,0,Vl),
22     qplan_aux1121(P2,Vg,L,Vl).
23
24 qplan_aux1121(P2,Vg,L,Vl) :- % #Abs < 2^5
25     schedule(L,Vg,P1),
26     quantificate(Vl,0,P1,P2).
27
28 qplan_aux2(X0,X,P,P2,VA) :- % #Abs < 2^5
29     melt(X0,VA,X),
30     melt(P2,VA,P).
```

1b Optimal transformation of `qplan/3` with the minimal number of non-existential variables in environments. The maximum size of the set-sharing abstractions is included as a comment.

Ait-Kaci 1991), but more general, since variables also come in instead of only out, and the objective is of course different: optimizing abstract operations vs. saving stack space.

Algorithm 2 presents the operations LIVE-VARS and DEAD-VARS to obtain the variables becoming alive and the ones which are dead at a given analysis point. LIVE-VARS receives the set of current live variables (`LiveVars`), and the literal that is going to be analyzed (`B`), and returns the set of new live variables at that analysis point, i.e., the variables that appear in the literal but were not in `LiveVars`. The other operation, DEAD-VARS, takes as input the set of current live variables (`LiveVars`), the variables of the head (`HVars`), and the set of literals that have not been analyzed yet ($\{\mathtt{B}_i, \ldots, \mathtt{B}_n\}$), and produces as output a set containing the variables of `LiveVars` that do not appear in any of the literals nor belong to the clause head. More efficient procedures to determine the liveness of variables are possible. However, we checked experimentally that they do not offer substantial improvements, and thus we decided to keep these simpler definitions. With these auxiliary operations, the analyzer can determine, at each analysis point, whether a variable is live or dead. With this information, it is possible to restrict the domain of the abstractions to the set of live variables. To do so, the computation of the `Success` abstraction is modified slightly:

Algorithm 2 Functions to detect live and dead variables.

1: **function** LIVE-VARS($\mathtt{LiveVars}, \mathtt{B}$)
2: $\quad\mathtt{LitVars} \leftarrow \mathtt{vars}(\mathtt{B})$
3: $\quad$**return** $\{X \in \mathtt{LitVars}\ s.t.\ X \notin \mathtt{LiveVars}\}$
4: **function** DEAD-VARS($\mathtt{LiveVars}, \mathtt{HVars}, \{\mathtt{B}_i, \ldots, \mathtt{B}_n\}$)
5: $\quad\mathtt{FutVars} \leftarrow \bigcup_{j=1}^{n} \mathtt{vars}(\mathtt{B}_j)$
6: $\quad$**return** $\{X \in \mathtt{LiveVars}\ s.t.\ X \notin \mathtt{FutVars} \wedge X \notin \mathtt{HVars}\}$

Algorithm 3 Version of `entryToExit` dynamically modifying the abstraction domain.

```
1:  function ENTRYTOEXIT(Entry, Head, {B_1, ..., B_m})
2:      HVars ← vars(Head)                                   ▷ Obtain the Head Variables
3:      LiveVars ← HVars                                     ▷ Initialize the Live Variables
4:      Exit ← Entry
5:      for i ∈ {1, ..., m} do
6:          NLiveVars ← LIVE-VARS(dom(Entry), B_i)           ▷ Get the New Live Vars.
7:          LiveVars ← LiveVars ∪ NLiveVars                  ▷ Update the Live Variables
8:          Entry^aug ← augment(NLiveVars, Entry)            ▷ Augment the Abstraction
9:          if RECURSIVE-CALL(Literal, Head) then
10:             Exit ← COMPUTE-FIXPOINT(Exit, B_i)
11:         else if PREDICATE-IN-SCOPE(Literal) then
12:             Exit ← ANALYZE-PRED(Exit, B_i)
13:         else
14:             Proj ← project(vars(B_i), Exit)
15:             MaybeAbs ← abstractLiteral(B_i, Proj)
16:             if MaybeAbs ≠ fail then
17:                 Abs_exit ← topmost(vars(B_i), Proj)
18:             else
19:                 Abs_exit ← MaybeAbs
20:             Exit ← extend(Exit, Abs_exit)
21:             DeadVars ← DEAD-VARS(LiveVars, HVars, {B_j}_{j=i+1}^m)   ▷ Get Dead Vars.
22:             LiveVars ← LiveVars\DeadVars                 ▷ Update Live Variables
23:             Exit ← project(LiveVars, Exit)               ▷ Restrict to the Live Variables
24:     return Exit
```

$$
\begin{aligned}
\texttt{Success} &= \texttt{extend}(\texttt{Call}, \texttt{Goal}, \texttt{Prime}) \\
\texttt{Prime} &= \texttt{exitToPrime}(\texttt{project}(\texttt{vars}(\texttt{Head}), \texttt{Exit}), \texttt{Head}, \texttt{Goal}) \\
\texttt{Exit} &= \texttt{entryToExit}(\texttt{Entry}, \texttt{Head}, \texttt{Body}) \\
\texttt{Entry} &= \texttt{callToEntry}(\texttt{Proj}, \texttt{Goal}, \texttt{Head}) \\
\texttt{Proj} &= \texttt{project}(\texttt{vars}(\texttt{Goal}), \texttt{Call}).
\end{aligned}
$$

In this case, the `Entry` abstraction is obtained by directly computing `callToEntry`, instead of by augmenting the result of the `callToEntry` invocation, as was done in the schema presented in Section 2. Thus, in this approach, the domain of the `Entry` abstraction is exactly the set of head variables (which are the only variables alive at that analysis point). Finally, the function `entryToExit` presented in Algorithm 1 is modified so that it keeps the abstraction defined only over the live variables. Such a modified version is described by Algorithm 3. There, a set containing the live variables is carried around while analyzing a clause body ($\{\texttt{B}_1, \ldots, \texttt{B}_m\}$). Such set is initialized with the variables of the clause head (Line 3), and updated by adding new variables to it as they become alive (Lines 6-7) and by removing the dead variables (Lines 21-22). Accordingly, the abstraction is augmented with the new live variables (Line 8) and reduced when some of them die (Line 23).

4 Experimental Evaluation

We have conducted an extensive experimental study to assess the benefits of the techniques proposed, to which we will refer to here as *reassociation* and *trimming* for short. In particular, we measured, for a variety of programs, the effects of applying both techniques

to a number of abstract domains that use set sharing: the classical set-sharing domain, `share` (Muthukumar and Hermenegildo 1989), the combination of sharing and freeness, `shfr` (Muthukumar and Hermenegildo 1991), and the combination of sharing and freeness including cliques, `shfr-clique` (Navas et al. 2006); the latter is the most efficient of the three, while `share` and `shfr` are, in general, more precise but slower. All experiments were performed on Debian/GNU Linux 12 (bookworm) 64bit (amd64) with 128GB RAM, and 800 GB of disk space. We focus on analysis times since precision is unchanged.

We start by showing in Table 1 the detailed results for one of our benchmarks, the `LPdoc` documenter (Hermenegildo and Morales 2011; Hermenegildo 2000), which is a relatively large, real-world application, and whose results are typical. The `LPdoc` source code is composed of 26 modules that exhibit different challenges: for some of them analysis terminates in times that range from a few milliseconds to several minutes, while others cannot be analyzed, either because of a timeout, set for these experiments at 5 minutes per module, or by running out of memory. In either of these cases we will say that the analysis fails. For each abstract domain, Column **TC** shows the *total* time that the classical domain requires to analyze the modules, columns **TR** and **TT** show the *total* analysis times when applying reassociation and trimming respectively to the corresponding classical domain, and columns ρ**R** and ρ**T** present the relative speed-up computed as **TC**/**TT** and **TC**/**TR**. *Total* analysis times are the addition of the times for the loading of the module, the pre-processing (including the transformation required by the reassociation method), and the actual analysis time. Some statistics are included at the bottom of the table: the total number of modules (**Mods**) and the number of modules for which the different analyses fail, for the classical approach (**FC**), when applying reassociation (**FR**), and when applying trimming (**FT**). Finally, μ**T** and μ**R** show the mean of the speed-ups obtained. We can see that applying reassociation and trimming allows analyzing a significant number of modules that could not be analyzed with the classical techniques. In particular, when applied to the `shfr-clique` domain they achieve a full analysis of the `LPdoc` source code. The mean speed-ups show that trimming outperforms reassociation in this application. This is due to the significant improvements that trimming brings to the modules **images** and **refsdb**. Specifically, in the case of **images**, the trimming approach significantly reduces analysis times for the `share` and `shfr` domains, resulting in speed-ups comparable to those achieved by `shfr-clique`. However, trimming introduces a slow-down when applied to the `shfr-clique` domain. Conversely, reassociation does not provide any significant changes. In the **texinfo** module, reassociation performs 10 times faster than trimming, while in the remaining modules, both methods behave similarly. Our hypothesis is that the overhead introduced by the transformation is what causes reassociation to slightly under-perform in some cases.

The proposed methods were also evaluated across a set of classic benchmarks (see Table C 1 in the appendices). The benchmarks include a variety of examples, ranging from simple predicates that feature only direct recursions, such as **qsort** and **append**, to more complex cases with mutual recursions and elaborate aliasing. Some benchmarks are extracted from real-world programs. For example, **aikal** is part of an analyzer for the AKL language, while **read** and **rdtok** are Prolog parsers. Additionally, parts of actual programs are also included, such as **ann** (the &-Prolog parallelizer), **qplan** (the core of the Chat-80 application), and **witt** (a conceptual clustering application). As expected, in these comparatively less challenging programs the advantages obtained are smaller, but they are still significant. For instance, using trimming and reassociation bring mean relative speed-ups of 1.1 and 1.06 respectively for the `share` domain; 0.95 and 1.5 for `shfr`; and 2.2 and 1.97 for `shfr-clique`.

Table 1: Analysis times and statistics for the source code of `LPdoc`. Times are in mS.

	share					**shfr**					**shfr-clique**				
module	**TC**	**TR**	**ρR**	**TT**	**ρT**	**TC**	**TR**	**ρR**	**TT**	**ρT**	**TC**	**TR**	**ρR**	**TT**	**ρT**
html	23485	1377	17.04	2646	8.87	20858	1373	15.19	2516	8.29	1522	1293	1.18	1532	0.99
html_assets	6.20	8.20	0.76	6.75	0.92	7.25	6.46	1.12	7.79	0.93	13.70	12.42	1.10	14.25	0.96
aux	4.12	4.60	0.90	4.09	1.01	4.51	5.17	0.87	4.58	0.98	5.57	6.75	0.83	5.82	0.96
man	timeout	32944	-	64490	-	timeout	29235	-	60521	-	23.74	27.23	0.87	23.32	1.02
doctree	10617	7095	1.50	5807	1.83	6426	5443	1.18	1898	3.39	4485	555.77	8.07	568.98	7.88
docmod	0.82	0.76	1.07	0.85	0.97	0.75	0.88	0.85	0.77	0.97	0.75	0.97	0.87	0.70	1.08
images	132364	134906	0.98	134.10	987.03	136011	130448	1.04	123.82	1098	16.95	14.83	1.14	52.66	0.32
messages	3.75	3.58	1.05	4.29	0.87	4.22	4.39	0.96	4.32	0.98	6.08	5.47	1.06	5.96	1.02
structure	15.59	12.63	1.23	11.54	1.35	17.77	14.07	1.26	13.27	1.34	21.44	16.34	1.31	18.24	1.18
lpdoc_sing.	342.13	17.53	19.51	12.58	27.19	331.32	19.85	16.96	14.21	23.31	533.04	29.56	18.03	22.26	23.94
docmaker	40.49	34.69	1.17	33.97	1.19	41.71	46.96	0.89	33.44	1.25	60.92	76.91	0.79	69.32	0.88
bibrefs	119737	64348	1.86	91029	1.32	74779	12024	6.22	40949	1.83	1560	590.09	2.64	780.31	2.00
html_templ.	124.43	34.46	3.61	26.27	4.74	121.22	25.99	4.66	22.03	5.50	41.23	42.65	0.97	46.61	0.88
lpdoc_help	3.01	3.56	0.85	2.73	1.10	3.84	4.22	0.91	3.01	1.28	5.34	5.82	0.92	4.43	1.21
texinfo	timeout	timeout	-	timeout	-	timeout	timeout	-	timeout	-	timeout	171.50	-	1440	-
comments	35.68	38.58	0.93	22.50	1.59	25.90	24.91	1.04	23.46	1.10	44.29	49.59	0.89	36.56	1.21
errors	0.79	0.95	0.83	0.74	1.07	0.97	0.76	1.29	0.76	1.29	0.00	1.06	0.09	1.14	0.09
parse	55513	54284	1.02	18022	3.08	49511	49577	1.00	15353	3.22	2381	2404	0.99	2492	0.96
autodoc	timeout	timeout	-	timeout	-	timeout	timeout	-	timeout	-	timeout	6242	-	6864	-
index	1856	395.24	4.70	685.04	2.71	1298	344.32	3.77	562.93	2.31	93.31	66.97	1.39	95.55	0.98
filesystem	80.38	41.11	1.96	53.51	1.50	80.32	51.71	1.55	51.93	1.55	59.12	65.75	0.90	63.98	0.92
doccfg	3.06	3.53	0.87	2.18	1.40	4.03	3.75	1.08	2.49	1.62	3.03	3.54	0.85	0.44	6.82
refsdb	timeout	1507	-	2991	-	97871	1093	89.54	572.88	170.84	26853	1433	18.73	393.76	68.20
lookup	11.68	12.04	0.97	6.98	1.67	14.44	12.54	1.15	4.01	3.60	25.30	18.14	1.39	13.09	1.93
version	0.28	0.23	1.23	0.30	0.92	0.32	0.30	1.05	0.31	1.04	0.28	0.32	0.88	0.30	0.93
settings	15.36	17.97	0.85	17.19	0.89	19.17	22.26	0.86	20.03	0.96	37.77	42.99	0.88	41.01	0.92
nil	2.11	2.60	0.81	2.27	0.93	3.52	2.26	1.56	2.43	1.45	1.56	1.63	0.96	1.69	0.92
state	38271	4745	8.06	4782	8.00	38006	4647	8.25	4733	8.03	1177	1103	1.07	1103	1.07

	share	**shfr**	**shfr-clique**
Mods	27	27	27
FC	4	5	2
FR	2	2	0
FT	2	2	0
μR	3.07	8.13	2.64
μT	44.26	53.82	4.97

Most of these modules already required small analysis times (less than 0.1 seconds, which may indicate they offer fewer opportunities for optimization). The improvements are most significant for `qplan`, which had the largest analysis times to begin with: 5.7 seconds (`share`), 1 second (`shfr`), and 5 seconds (`shfr-clique`). In this case, trimming and reassociation bring analysis times of 1.1 and 0.5 seconds, respectively, with `share`; and 0.3 and 0.2 seconds with `shfr`. Moreover, with `shfr-clique` both techniques bring the time down to 0.1 seconds (50× speed-up, the largest one). The full experimental results can be found in Appendix C.1.

Given the positive results, in a second phase of our experimentation we decided to greatly expand the code base used to measure the impact of applying abstract environment trimming, in order to explore whether the improvements obtained for the classic benchmarks and the `LPdoc` application would translate much more widely. All in all we have analyzed around 1200 program modules. These include, in addition to the previously mentioned `LPdoc` documenter (28 modules) and classic benchmarks (26 modules): all the libraries distributed with the `Ciao` system (Hermenegildo et al. 2012) (comprising more than 1000 modules), `CiaoPP`'s program analyzer (Hermenegildo et al. 2003) PLAI (56 modules), s(CASP) (Arias et al. 2018), a top-down interpreter for ASP programs with constraints (44 modules), and Spectector (Guarnieri et al. 2020), a tool for automatically detecting leaks caused by speculatively executed instructions in x64 assembly programs (15 modules). The ≈1000 library modules in the standard `Ciao` distribution come from many sources including libraries ported from SICStus (Carlsson and Mildner 2012); libraries common with Yap (Santos Costa et al. 2012), XSB (Swift and Warren 2012), SWI (Wielemaker et al. 2012), etc., including those from the Prolog Commons project (`https://prolog-commons.org/`); the classic libraries from O'Keefe and those for Constraint Logic Programming over Rationals and Reals (Holzbaur 1995); libraries for implementing language extensions such as functional syntax or the `Ciao` assertion language; etc., developed by around 100 different programmers. We argue that this selection of benchmarks constitutes a good representation of real-life code written in Prolog. The detailed results of these experiments can be found in Appendix C, while here we present them in a more manageable aggregated form.

We first present in Figure 2 a *cactus plot* of the results. Cactus plots display, on the X-axis, the number of benchmarks that can be analyzed, i.e., those for which the analysis does not time out or run out of memory, and, on the Y-axis, the *accumulated* analysis time. The plots for each abstract domain and analysis technique are generated by taking all the analysis times, *sorting them in increasing order*, and then plotting them following the formula (i, t_i) where t_i is the *accumulated* time in the i-th position. This way, the analysis time of the benchmark corresponding to the $i+1$-th position is $t_{i+1} - t_i$. Darker colors represent abstract domains with trimming, while lighter colors represent domains with classic analyses. Empty circles correspond to the classic set-sharing domain, crosses to `shfr`, and stars to `shfr-clique`. The module compilation times are given for comparison, represented by gray triangles –resulting in the gray vertical line. The plot has been zoomed in to exclude points with X-values (numbers of benchmarks) less than 800, as the most interesting information is in the more challenging benchmarks beyond, that require the highest analysis times. Also, plots corresponding to modules for which the analysis fails are excluded. This figure allows us to observe that when the domains are used with abstract trimming, they perform better than their classical counterparts. Specifically, `share` and `shfr` reduce the number of modules they are unable to analyze from 134 and 130 to 81 and 74, respectively, corresponding to a 39.55% and 43.07% improvement. For `shfr-clique` applying abstract environment trimming results in failure to analyze only 21 modules versus the 47 that the classical approach failed to analyze, while reducing the accumulated time

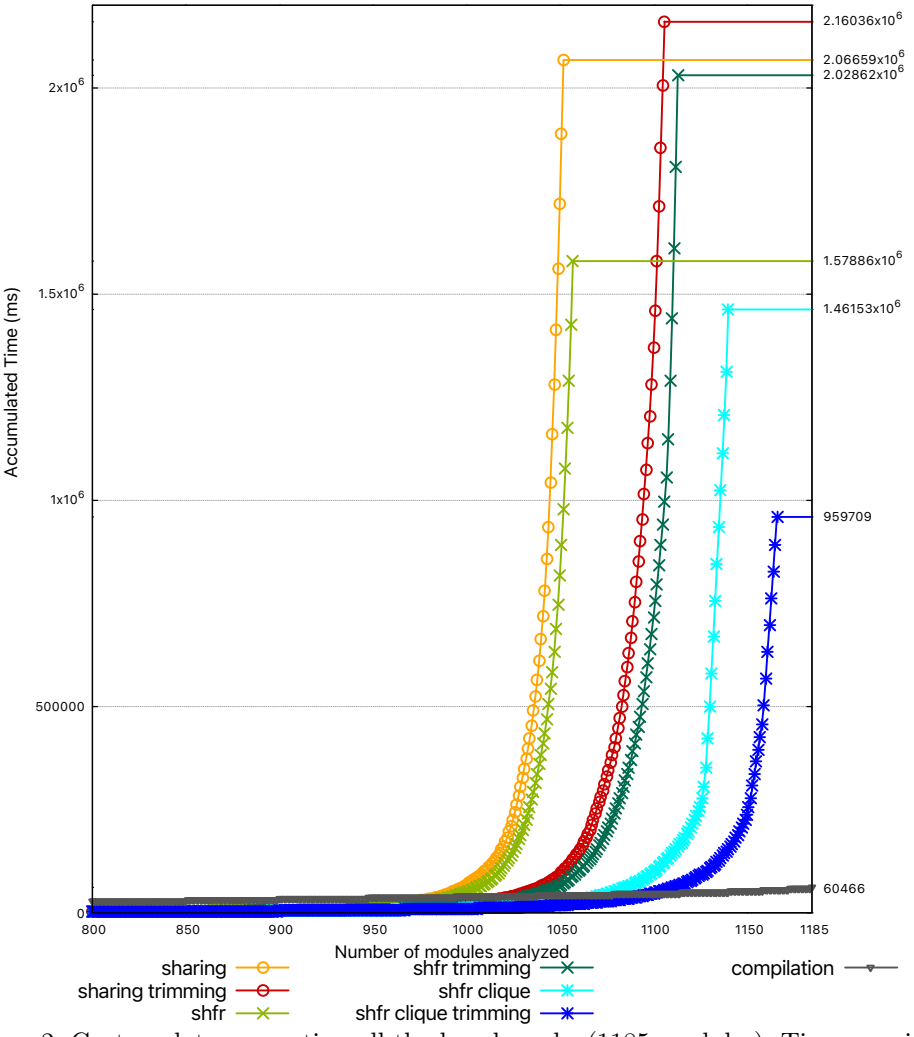

Figure 2: Cactus plot aggregating all the benchmarks (1185 modules). Times are in mS.

by 8.36 minutes. This translates to a reduction in timeouts by 55.32% while reducing the accumulated time by 34.3%. We have also obtained mean speed-ups of 5.82, 5.91, and 22.08 when analyzing with `share`, `shfr` and `shfr-clique` respectively. Moreover, the number of modules that can be analyzed in a time lower than the compilation time also increases.

The cactus plot shows how the analysis results accumulate. In order to show how these results relate individually, Figure 3 presents a scatter plot displaying the time required to analyze each module. Given a point (x, y), the value in x corresponds to the time required by the classical analysis to analyze a given module, while y corresponds to the time required to analyze that same module using abstract trimming. Modules that are not analyzed with the classic approach or with both are not displayed. If the points are close to the orange line, it means that the times are very similar; if they are above the line, it means that abstract trimming introduces an overhead; if they are below, abstract trimming speeds up the analysis. To complement this information, Figure 4 presents a scatter plot displaying a comparison between the time required to analyze each module with the classical analysis (X-axis) and the speedup obtained by applying the abstract trimming technique (Y-axis).

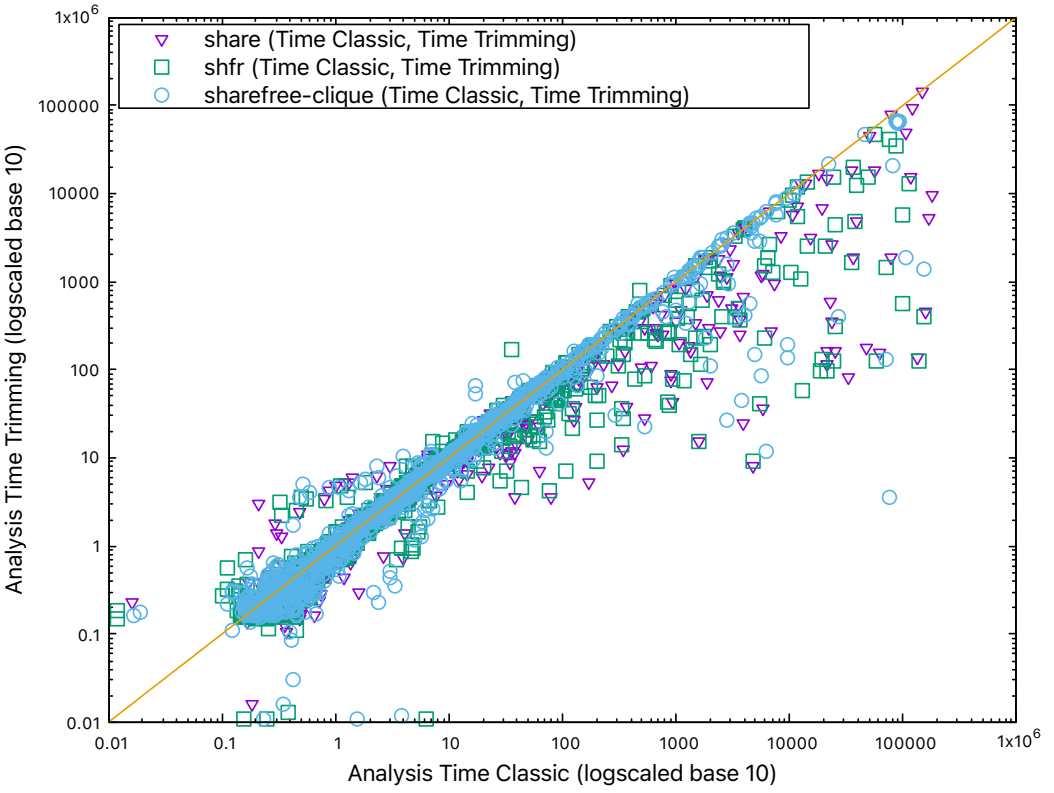

Figure 3: Scatter plot comparing absolute analysis times (in mS).

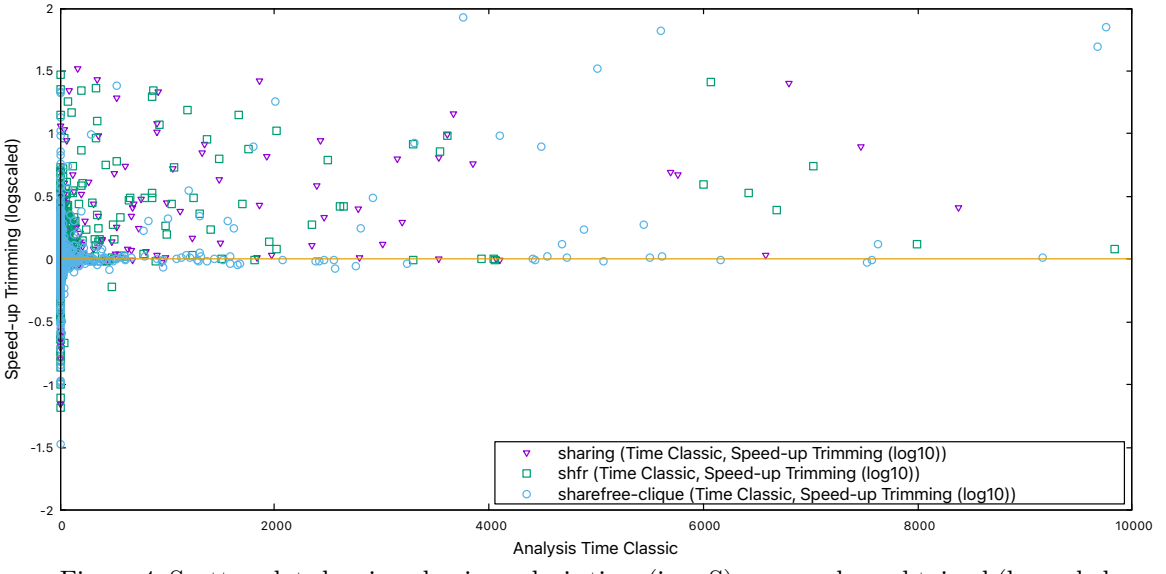

Figure 4: Scatter plot showing classic analysis time (in mS) vs. speed-up obtained (logscaled base 10).

The speedup values range from very close to zero to significantly larger numbers (see the 0.09 and the 1098 speedup obtained by abstract trimming when analyzing the "errors" mod-

Table 2: Complete statistics.

	share	**shfr**	**shfr-clique**
Modules	1186	1186	1186
FC	134	130	47
FT	81	74	21
μ**T**	5.82	5.91	22.08
μ**T** $>$ 1s (70-74-160)	62.13	57.64	146.15
μ**T** $>$ 0.5s (93-97-180)	47.86	45.05	130.66
μ**T** $>$ 0.1s (143-147-230)	32.11	30.60	102.65
μ**T** $<$ 0.1s (909-913-999)	1.68	2.10	24.12

ule with sharefree-clique and the "images" module with shfr, as shown in Table 1). To better represent these values, we have applied a base 10 logarithm to the resulting speedups. Thus, values between 0 and 1 become negative (with larger absolute values the closer they are to 0), while values greater than 1 are scaled in the positive plane. The most significant performance improvements are observed in the right-most side of the figures, corresponding to the modules where the classical approach takes more time. Conversely, in cases where the classical approach is very fast (left-most part of the figures), the technique of abstract trimming does not yield many speedups but introduces some overheads, which are however small. Another observation is that in most benchmarks, the analysis times are quite low, but of course our target has been the rest that present significant challenges. To concentrate on this set we have collected the speed-up results considering only the benchmarks taking more than 0.1, 0.5, and 1 second when analyzed with `share` (which represents the slower domain). These results are presented in Table 2. For example, the row starting with "μ**T** $>$ 1s (70-74-160)" shows the mean speedup for the benchmarks successfully analyzed such that the time required to analyze them with `share` is greater than 1 second or `share` times out. The results show that in the case of `share`, 70 of these more challenging modules are successfully analyzed with both trimming and reassociation (note that both approaches need to succeed in order to compute the means), 74 in the case of `shfr`, and 160 in the case of `shfr-clique`. Similar very positive results are obtained for the other cases.

5 Conclusions

We have proposed a number of techniques for addressing the scalability problems inherent to set-sharing analyses. We have focused on the root of the problem: the potentially exponential dependency of the size of the abstractions on the number of variables. We have cast this problem as an instance of expression reassociation and provided an optimal solution using program transformations. Additionally, we have proposed a practical solution that can be integrated into top-down analyzers, based on the liveness of variables in the body of the clause being analyzed. We have conducted an extensive experimental evaluation of over 1100 program modules taken from both production code and classical benchmarks. We have obtained significant speed-ups, and, more importantly, the number of modules that require a timeout was cut in half. As a result, many more programs can be analyzed precisely in reasonable times. We believe that the results obtained suggest that the proposed local technique improves significantly the scalability of set-sharing analyses, and can thus enhance the practicality of top-down set sharing analysis for production code. As a possible avenue for future work, note that the definition of live variables used in this work is local to each clause of the predicate being analyzed. Future lines of work could explore a more global notion that also considers the calls to predicates within the clause under analysis. This will

presumably incur additional cost but could also possibly allow further reduction in the size of the domains of the sharing abstractions.

References

Aho, A. V., Lam, M. S., Sethi, R., and Ullman, J. D. 2006. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA.

Aiken, A., Foster, J. S., Kodumal, J., and Terauchi, T. Checking and inferring local non-aliasing. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation 2003, San Diego, California, USA, June 9-11, 2003* 2003, pp. 129–140. ACM.

Ait-Kaci, H. 1991. *Warren's Abstract Machine, A Tutorial Reconstruction*. MIT Press.

Amato, G., Meo, M. C., and Scozzari, F. 2022. The role of linearity in sharing analysis. *Math. Struct. Comput. Sci.*, *32*, 1, 44–110.

Amato, G. and Scozzari, F. 2009. Optimality in goal-dependent analysis of sharing. *Theory Pract. Log. Program.*, *9*, 5, 617–689.

Amato, G. and Scozzari, F. 2014. Optimal multibinding unification for sharing and linearity analysis. *Theory Pract. Log. Program.*, *14*, 3, 379–400.

Arias, J., Carro, M., Salazar, E., Marple, K., and Gupta, G. 2018. Constraint Answer Set Programming without Grounding. *Theory and Practice of Logic Programming*, *18*, 3-4, 337–354.

Bagnara, R., Hill, P. M., and Zaffanella, E. Set-sharing is redundant for pair-sharing. In *Static Analysis Symposium* 1997, pp. 53–67. Springer-Verlag.

Bravenboer, M. and Smaragdakis, Y. 2009. Strictly declarative specification of sophisticated points-to analyses. *SIGPLAN Not.*, *44*, 10, 243–262.

Briggs, P. and Cooper, K. D. Effective partial redundancy elimination. In *ACM-SIGPLAN Symposium on Programming Language Design and Implementation* 1994.

Bruynooghe, M. and Codish, M. Freeness, sharing, linearity and correctness–all at once. In *Proc. Third International Workshop on Static Analysis* 1993, number 724 in LNCS, pp. 153–164. Springer.

Bruynooghe, M., Codish, M., and Mulkers, A. Abstract unification for a composite domain deriving sharing and freeness properties of program variables. In de Boer, F. and Gabbrielli, M., editors, *Verification and Analysis of Logic Languages* 1994, pp. 213–230.

Bueno, F. and García de la Banda, M. Set-Sharing is not always redundant for Pair-Sharing. In *7th International Symposium on Functional and Logic Programming (FLOPS 2004)* 2004, number 2998 in LNCS, Heidelberg, Germany. Springer-Verlag.

Bueno, F., García de la Banda, M., and Hermenegildo, M. V. 1999. Effectiveness of Abstract Interpretation in Automatic Parallelization: A Case Study in Logic Programming. *ACM TOPLAS*, *21*, 2, 189–238.

Cabeza, D. and Hermenegildo, M. Extracting Non-strict Independent And-parallelism Using Sharing and Freeness Information. In *1994 International Static Analysis Symposium* 1994, number 864 in LNCS, pp. 297–313, Namur, Belgium. Springer-Verlag.

Carlsson, M. and Mildner, P. 2012. SICStus Prolog – the First 25 Years. *Theory and Practice of Logic Programming*, *12*, 1-2, 35–66.

Codish, M., Dams, D., Filé, G., and Bruynooghe, M. 1996. On the design of a correct freeness analysis for logic programs. *The Journal of Logic Programming*, *28*, 3, 181–206.

Cousot, P. and Cousot, R. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proc. of POPL'77* 1977, pp. 238–252. ACM Press.

De Angelis, E., Fioravanti, F., Gallagher, J. P., Hermenegildo, M. V., Pettorossi, A., and Proietti, M. 2021. Analysis and Transformation of Constrained Horn Clauses for Program Verification. *TPLP*, *22*, 6, 1–69.

Fecht, C. An efficient and precise sharing domain for logic programs. In Kuchen, H. and Swierstra, S. D., editors, *PLILP* 1996, volume 1140 of *Lecture Notes in Computer Science*, pp. 469–470. Springer.

Filé, G. 1994. Share x Free: Simple and correct. Technical Report 15, Dipartamento di Matematica, Universita di Padova.

García de la Banda, M., Hermenegildo, M. V., Bruynooghe, M., Dumortier, V., Janssens, G., and Simoens, W. 1996. Global Analysis of Constraint Logic Programs. *ACM Trans. on Programming Languages and Systems*, *18*, 5, 564–615.

García de la Banda, M., Hermenegildo, M. V., and Marriott, K. 2000. Independence in CLP Languages. *ACM Transactions on Programming Languages and Systems*, *22*, 2, 269–339.

Giacobazzi, R. and Ranzato, F. 2022. History of abstract interpretation. *IEEE Ann. Hist. Comput.*, *44*, 2, 33–43.

Guarnieri, M., Köpf, B., Morales, J. F., Reineke, J., and Sánchez, A. Spectector: Principled Detection of Speculative Information Flows. In *2020 IEEE Symposium on Security and Privacy (SP)* 2020, pp. 1–19.

Henriksen, K. S. and Gallagher, J. P. Abstract Interpretation of PIC Programs through Logic Programming. In *SCAM '06* 2006, pp. 184–196. IEEE Computer Society.

Hermenegildo, M. and Rossi, F. 1995. Strict and Non-Strict Independent And-Parallelism in Logic Programs: Correctness, Efficiency, and Compile-Time Conditions. *Journal of Logic Programming*, *22*, 1, 1–45.

Hermenegildo, M. V. A Documentation Generator for (C)LP Systems. In *Int'l. Conf. CL 2000* 2000, volume 1861 of *LNAI*, pp. 1345–1361. Springer-Verlag.

Hermenegildo, M. V., Bueno, F., Carro, M., Lopez-Garcia, P., Mera, E., Morales, J., and Puebla, G. 2012. An Overview of Ciao and its Design Philosophy. *Theory and Practice of Logic Programming*, *12*, 1–2, 219–252.

Hermenegildo, M. V. and Morales, J. 2011. The LPdoc Documentation Generator. Ref. Manual (v3.0). Technical report, UPM. Available at `http://ciao-lang.org`.

Hermenegildo, M. V., Puebla, G., Bueno, F., and Lopez-Garcia, P. Program Development Using Abstract Interpretation (and The Ciao System Preprocessor). In *10th International Static Analysis Symposium (SAS'03)* 2003, number 2694 in LNCS, pp. 127–152. Springer-Verlag.

Hill, P. M., Zaffanella, E., and Bagnara, R. 2004. A correct, precise and efficient integration of set-sharing, freeness and linearity for the analysis of finite and rational tree languages. *Theory and Practice of Logic Programming*, *4*, 3, 289–323.

Holzbaur, C. 1995. OFAI CLP(Q,R) Manual, Edition 1.3.3. Technical Report TR-95-09, Austrian Research Institute for Artificial Intelligence, Vienna.

Jacobs, D. and Langen, A. Accurate and Efficient Approximation of Variable Aliasing in Logic Programs. In *North American Conference on Logic Programming* 1989.

Kelly, A., Marriott, K., Søndergaard, H., and Stuckey, P. 1998. A Practical Object-Oriented Analysis Engine for CLP. *Software: Practice and Experience*, *28*, 2, 188–224.

King, A. and Soper, P. Depth-k Sharing and Freeness. In *International Conference on Logic Programming* 1994. MIT Press.

Landi, W. and Ryder, B. G. A safe approximate algorithm for interprocedural pointer aliasing (with retrospective). In McKinley, K. S., editor, *Best of PLDI* 1992, pp. 473–489. ACM.

Le Charlier, B. and Van Hentenryck, P. 1994. Experimental Evaluation of a Generic Abstract Interpretation Algorithm for Prolog. *ACM TOPLAS*, *16*, 1, 35–101.

Méndez-Lojo, M. and Hermenegildo, M. Precise Set Sharing Analysis for Java-style Programs. In *9th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'08)* 2008, number 4905 in LNCS, pp. 172–187. Springer-Verlag.

Méndez-Lojo, M., Navas, J., and Hermenegildo, M. A Flexible (C)LP-Based Approach to the Analysis of Object-Oriented Programs. In *LOPSTR* 2007a, volume 4915 of *LNCS*, pp. 154–168. Springer-Verlag.

Méndez-Lojo, M., Navas, J., and Hermenegildo, M. V. An Efficient, Parametric Fixpoint Algorithm for Analysis of Java Bytecode. In *ETAPS Workshop on Bytecode Semantics, Verification, Analysis and Transformation (BYTECODE'07)* 2007b.

Muthukumar, K. and Hermenegildo, M. Determination of Variable Dependence Information at Compile-Time Through Abstract Interpretation. In *NACLP'89* 1989, pp. 166–189. MIT Press.

Muthukumar, K. and Hermenegildo, M. 1990. Deriving A Fixpoint Computation Algorithm for Top-down Abstract Interpretation of Logic Programs. Technical Report ACT-DC-153-90, Microelectronics and Comp. Tech. Corp. (MCC).

Muthukumar, K. and Hermenegildo, M. Combined Determination of Sharing and Freeness of Program Variables Through Abstract Interpretation. In *8th Int'l. Conference on Logic Programming* 1991, pp. 49–63. MIT Press.

Muthukumar, K. and Hermenegildo, M. 1992. Compile-time Derivation of Variable Dependency Using Abstract Interpretation. *JLP*, *13*, 2/3, 315–347.

Navas, J., Bueno, F., and Hermenegildo, M. V. Efficient Top-Down Set-Sharing Analysis Using Cliques. In *8th Int'l. Symp. on Practical Aspects of Declarative Languages (PADL'06)* 2006, number 2819 in LNCS, pp. 183–198. Springer.

Navas, J., Méndez-Lojo, M., and Hermenegildo, M. V. User-Definable Resource Usage Bounds Analysis for Java Bytecode. In *Proceedings of the Workshop on Bytecode Semantics, Verification, Analysis and Transformation (BYTECODE'09)* 2009, volume 253 of *Electronic Notes in Theoretical Computer Science*, pp. 65–82. Elsevier - North Holland.

Pontelli, E., Gupta, G., Pulvirenti, F., and Ferro, A. Automatic Compile-time Parallelization of Prolog Programs for Dependent And-Parallelism. In Naish, L., editor, *Proc. of the Fourteenth International Conference on Logic Programming* 1997, pp. 108–122. MIT Press.

Rountev, A., Milanova, A., and Ryder, B. G. Points-to analysis for Java using annotated constraints. In *Conference on Object-Oriented* 2001, pp. 43–55.

Santos Costa, V., Rocha, R., and Damas, L. 2012. The YAP Prolog system. *Theory and Practice of Logic Programming*,, 1-2, 5–34.

Seidl, H. and Vogler, R. 2021. Three improvements to the top-down solver. *Math. Struct. Comput. Sci.*, *31*, 9, 1090–1134.

Søndergaard, H. An Application of Abstract Interpretation of Logic Programs: Occur Check Reduction. In *European Symposium on Programming, LNCS 123* 1986, pp. 327–338. Springer.

Steensgaard, B. Points-to analysis in almost linear time. In *Symposium on Principles of Programming Languages* 1996, pp. 32–41.

Swift, T. and Warren, D. 2012. XSB: Extending Prolog with Tabled Logic Programming. *TPLP*, *12*, 1-2, 157–187.

Tilscher, S., Stade, Y., Schwarz, M., Vogler, R., and Seidl, H. The Top-Down Solver—An Exercise in A^{2}I. In Arceri, V., Cortesi, A., Ferrara, P., and Olliaro, M., editors, *Challenges of Software Verification* 2023, volume ISRL 238, chapter 9, pp. 157–179. Springer, Singapore.

Trias, E., Navas, J., Ackley, E. S., Forrest, S., and Hermenegildo, M. V. Negative Ternary Set-Sharing. In *International Conference on Logic Programming, ICLP* 2008, number 5366 in LNCS, pp. 301–316, Udine (Italy). Springer-Verlag.

Van Roy, P. and Despain, A. M. The Benefits of Global Dataflow Analysis for an Optimizing Prolog Compiler. In *North American Conf. on Logic Programming* 1990, pp. 501–515. MIT Press.

Warren, D. H. D. The SRI Model for OR-Parallel Execution of Prolog—Abstract Design and Implementation. In *Symp. on Logic Prog.* 1987, pp. 92–102.

Warren, R., Hermenegildo, M., and Debray, S. K. On the Practicality of Global Flow Analysis of Logic Programs. In *JICSLP* 1988, pp. 684–699. MIT Press.

Whaley, J. and Lam, M. S. An efficient inclusion-based points-to analysis for strictly-typed languages. In *SAS* 2002, volume 2477 of *Lecture Notes in Computer Science*, pp. 180–195. Springer.

Wielemaker, J., Schrijvers, T., Triska, M., and Lager, T. 2012. SWI-Prolog. *TPLP*, *12*, 1-2, 67–96.

Zaffanella, E., Bagnara, R., and Hill, P. M. Widening Sharing. In Nadathur, G., editor, *PPDP* 1999, volume 1702 of *LNCS*, pp. 414–432. Springer-Verlag, Berlin.

Zanardini, D. 2018. Field-sensitive sharing. *Journal of Logical and Algebraic Methods in Programming*, *95*, 103–127.

```
1 app([],L,L).
2 app([H|T],L0,L2) :-
3     app(T,L0,L1),
4     L2 = [H|L1].
```

Figure A 1

Appendix A An Additional Top-down Sharing Analysis Example

The example provided in Section 2 is chosen to be simple in order to make it easy to follow. To complete the discussion about how the analyzer proceeds, we provide a richer example showing how a complex sharing pattern is computed. This example will also allow us to show how sharing-set representations easily grow. Let `app/3` be a predicate that succeeds if its third argument is the list resulting from appending the first two arguments as showed in Figure A 1. Let this predicate be called with `Goal` = `app([A],[B,C],[A,B,D])`, and calling abstraction `Call`$=\{\{$`A,B`$\}, \{$`C`$\}, \{$`D,E`$\}\}$. Thus, the success abstraction is computed as follows:

i) `Proj`=`project`$(\{$`A,B,C,D`$\},$ `Call`$) = \{\{$`A,B`$\}, \{$`C`$\}, \{$`D`$\}\}$.

ii) $\texttt{Abs}_{\texttt{entry}}$=`callToEntry`$($`Proj`$,$ `Goal`$,$ `app([H|T],L0,L2)`$) = \{\{$`L0`$\}, \{$`L0,L2`$\}, \{$`L0,L2,H`$\}, \{$`L2`$\}\}$
Notice how, in this case, the head of the second clause has been selected instead of the head of the first clause (`app([],L,L)`). This is because the analyzer only selects the clauses which can unify with the goal.

iii) `Entry`=`augment`$(\{$`L1`$\},$ $\texttt{Abs}_{\texttt{entry}}) = \{\{$`L0`$\}, \{$`L0,L2`$\}, \{$`L0,L2,H`$\}, \{$`L1`$\}, \{$`L2`$\}\}$

iv) Now the abstracion `Exit` is computed by calling ENTRYTOEXIT(`Entry`,`Head`,`Body`) as follows:

   a) Since the first literal (`app(T,L0,L1)`) is a recursive call the fixpoint has to be computed. In a very simplified way: both clauses would be analyzed with `Goal`=`app(T,L0,L1)` and $\texttt{Call}_1$=`project`$(\{$`T,L0,L1`$\},$ `Entry`$) = \{\{$`L0`$\}, \{$`L1`$\}\}$.

   b) The analysis of the first clause (`app([],L,L)`), creates a sharing between `L0` and `L1` since the unification of `Goal`=`Head` induces `L0=L1` . Then, analyzing this clause outputs the prime abstraction $\texttt{Prime}_1 = \{\{$`L0,L1`$\}\}$.

   c) Then, the second clause is analyzed. The entry is computed by computing the abstract unification of `app(T,L0,L1)` and `app(`$\texttt{H}_r$`|`$\texttt{T}_r$`], L0`$_r$`, L2`$_r$. We have renamed the variables of the clause being analyzed (adding a subscript$_r$) to avoid confusion. Now, the entry abstraction obtained is $\texttt{Entry}_{\texttt{fxp}} = \{\{$`L0`$_r\}, \{$`L2`$_r\}, \{$`L1`$_r\}\}$. Notice that, since the abstract unification induces the unification `T=[`$\texttt{H}_r$`|`$\texttt{T}_r$`]` and `T` is ground in the call, both $\texttt{H}_r$ and $\texttt{T}_r$ become ground. In the next invocation to ENTRYTOEXIT, the literal `app(`$\texttt{T}_r$`,`$\texttt{L0}_r$`,`$\texttt{L1}_r$`)` has to be analyzed with call $\{\{$`L0`$_r\}, \{$`L1`$_r\}\}$. This call is the same as $\texttt{Call}_1$ (up to variable renaming). Thus, a fixpoint is reached and the literal is analyzed yielding the abstraction $\{\{$`L0`$_r$`,L1`$_r\}\}$. This is extended with $\texttt{Entry}_{\texttt{fxp}}$ returning: $\{\{$`L0`$_r\}, \{$`L2`$_r\}, \{$`L1`$_r\}\}$. Then, the next literal (`L2`$_r$`=[`$\texttt{H}_r$`|`$\texttt{L1}_r$`]`) is analyzed and the abstraction $\{\{$`L0`$_r$`,L1`$_r$`,L2`$_r\}\}$ obtained. Finally, the `exitToPrime` invocation returns: $\texttt{Prime}_2 = \{\{$`L0,L1`$\}\}$

   d) Then, the least upper bound of $\texttt{Prime}_1$ and $\texttt{Prime}_2$ is computed. Since they are the same the obtained abstraction is $\{\{$`L0,L1`$\}\}$ as well.

   e) Such abstraction is used to update the exit abstraction carried during the execution of ENTRYTOEXIT. Thus, the current `Exit` abstraction ($\{\{$`L0`$\}, \{$`L0,L2`$\}, \{$`L0,L2,H`$\}, \{$`L1`$\}, \{$`L2`$\}\}$) is updated to `Exit`$=\{\{$`L0,L2,H,L1`$\}, \{$`L0,L2,L1`$\}, \{$`L0,L1`$\}, \{$`L2`$\}\}$ propagating the new sharing created between `L0` and `L1`.

   f) The next literal (`L2=[H|L1]`) is analyzed with abstraction `project`$(\{$`L1,H,L2`$\},$ `Exit`$)$ $=\{\{$`L2,H,L1`$\}, \{$`L2,L1`$\}, \{$`L1`$\}, \{$`L2`$\}\}$. This literal creates sharing between `L2` and `H` and between `L2` and `L1` respectively. Thus, the abstraction obtained is $\{\{$`L2,H,L1`$\}, \{$`L2,L1`$\}\}$ and, after extending, `Exit`$=\{\{$`L0,H,L1,L2`$\}, \{$`L0,L1,L2`$\}\}$.

v) Then,
`Prime`=`exitToPrime`$($`Exit`$,$ `Head`$,$ `Goal`$)=\{\{$`A,B`$\}, \{$`A,B,C`$\}, \{$`A,B,C,D`$\}, \{$`A,B,D`$\}, \{$`C,D`$\}\}$. Note how, the unification `app([H|T],L0,L2)=app([A],[B,C],[A,B,D])` propagates the sharings between the variables.

vi) Finally, the success abstraction is computed by updating the call: `Success`=`extend`$($`Call`$,$`Goal`$,$`Prime`$)=\{\{$`A,B`$\}, \{$`A,B,C`$\}, \{$`A,B,C,D,E`$\}, \{$`A,B,D,E`$\}, \{$`C,D,E`$\}\}$.

Appendix B Plots by Set of Benchmarks

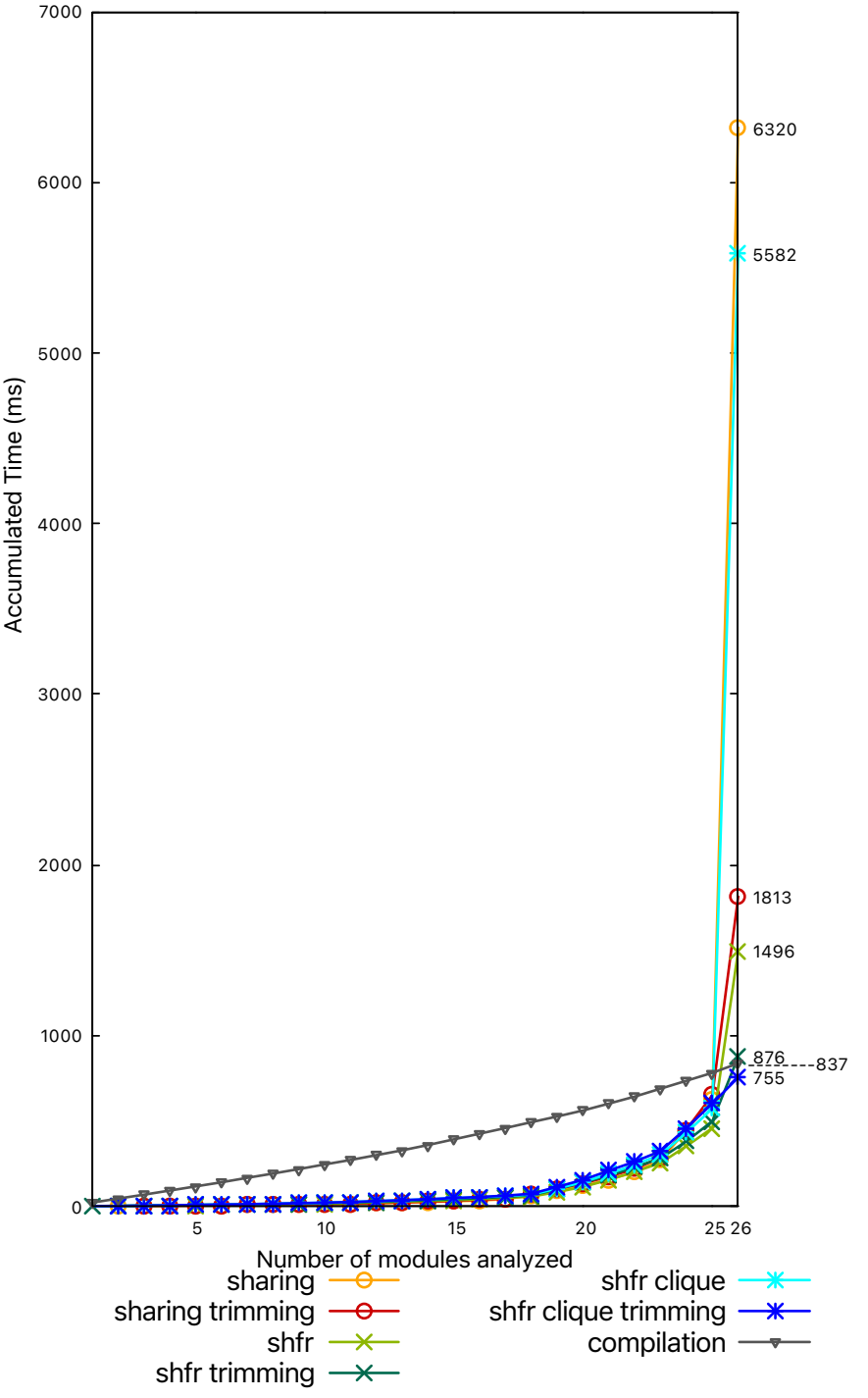

Figure B 1: Cactus plots aggregating results of analyzing the classic benchmarks set.

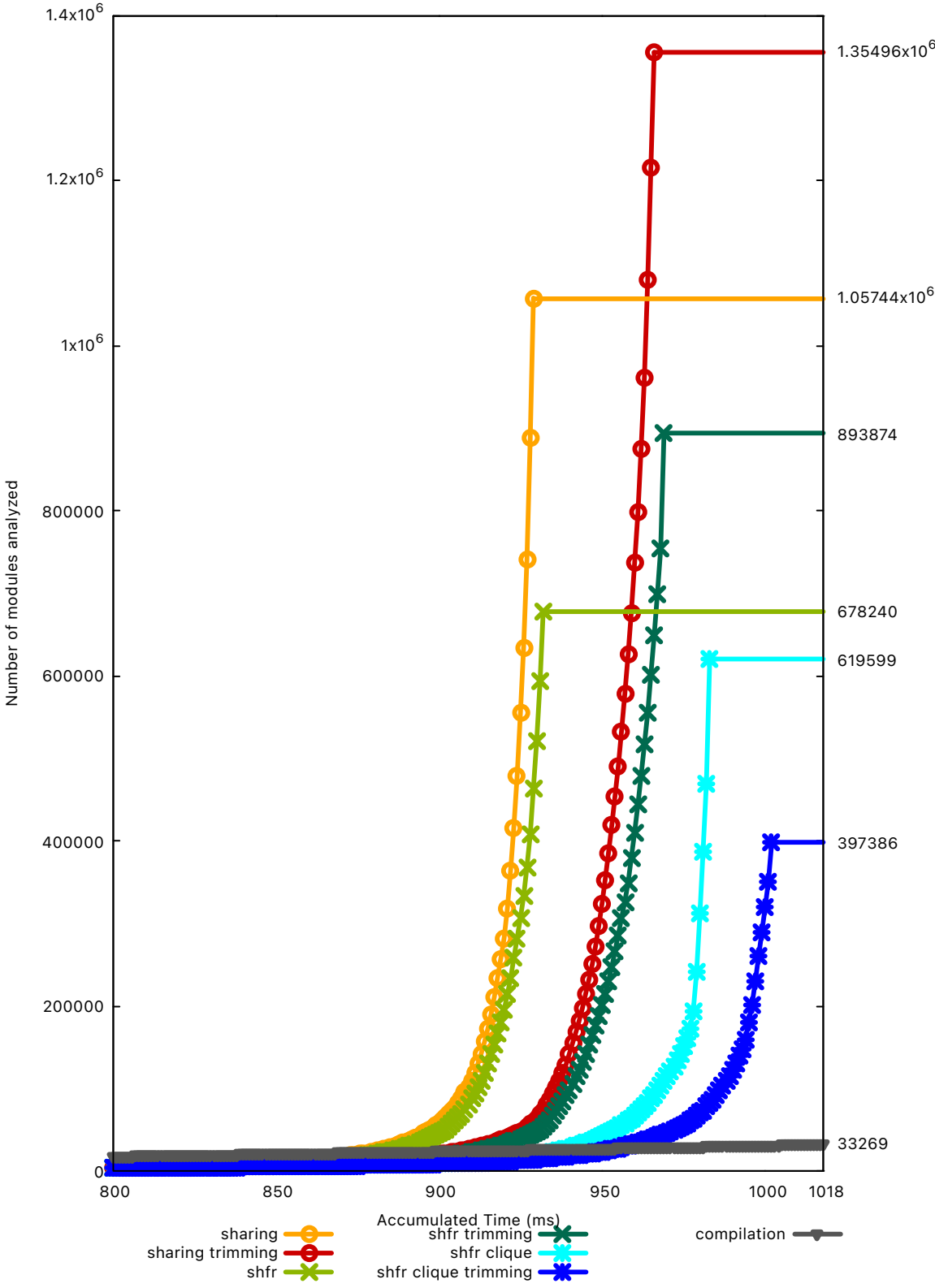

Figure B 2: Cactus plots aggregating results of analyzing all libraries in the `Ciao` system.

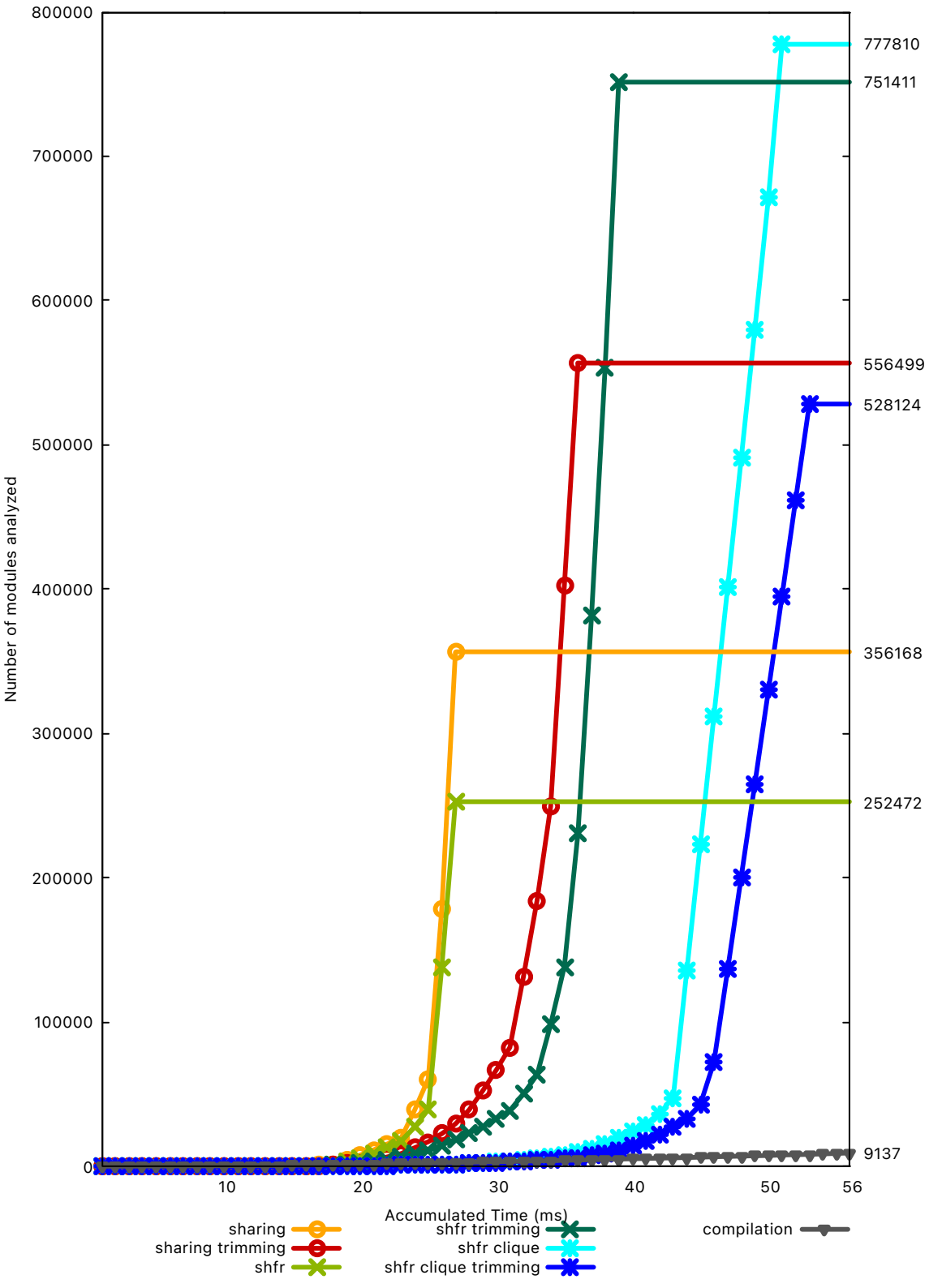

Figure B 3: Cactus plots aggregating results of analyzing the source code of PLAI.

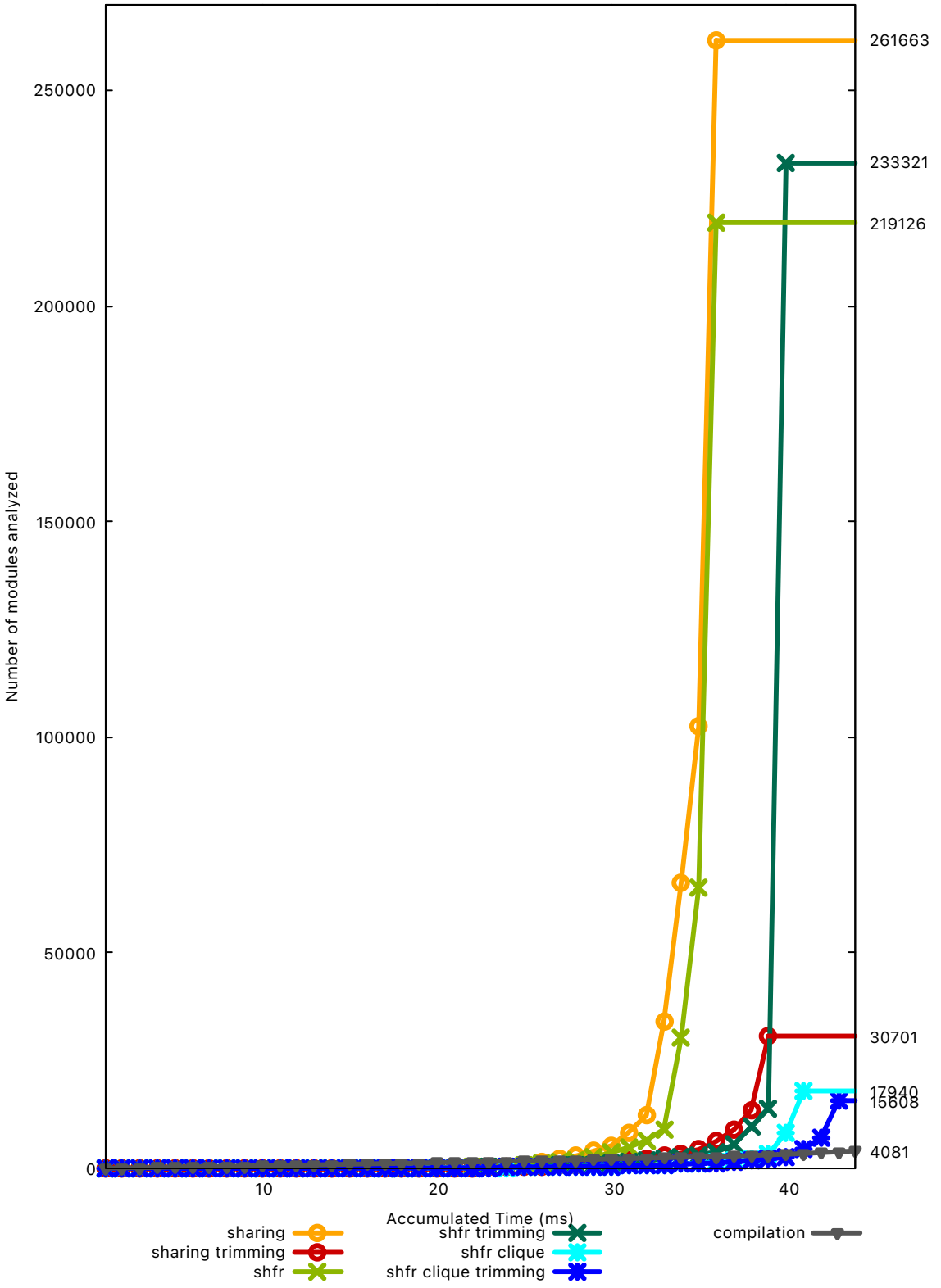

Figure B 4: Cactus plots aggregating results of analyzing the source code of s(CASP).

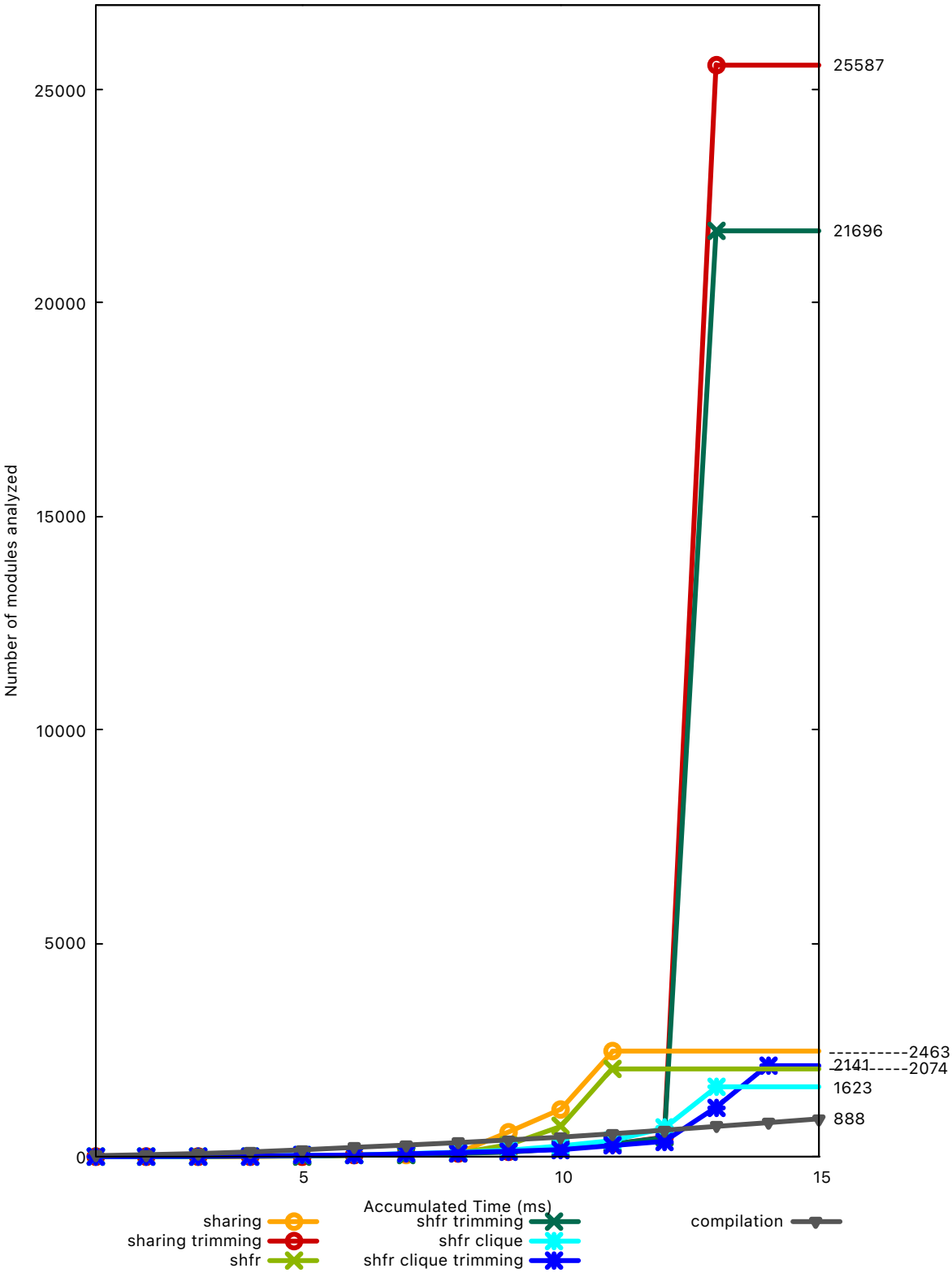

Figure B 5: Cactus plots aggregating results of analyzing the source code of Spectector.

Appendix C Analysis Times by Sets of Benchmarks.

This section contains all the analysis times obtained while running the experiments described in Section 4. The **TC** columns show the time that the classical analysis requires to analyze the modules. The **TT** columns show the time required by abstract trimming, and ρ**T** shows the relative speedup given by the formula ρT = TC/TT.

At the end of each table, some statistics are collected. The number of timeouts with the classic approach for each analysis, as well as the timeouts obtained while applying abstract trimming are shown. Then, some statistics related to the speedups are gathered. First, the mean speedup is provided. Then, we provide the mean speedups considering only programs which take more than a time T to be analyzed by the slowest domain (`share`).

We show the number of modules of each kind that are analyzed by each domain. For example, a row starting with "μ**T** > 1s (44-47-101)" shows the mean speedup for the benchmarks successfully analyzed such that the time of analyzing them with `share` is greater than 1 second or `share` times out. It also shows that in the case of `share`, 44 of those modules are analyzed successfully with both approaches, 47 in the case of `shfr`, and 101 in the case of `shfr-clique`. Notice that both of the approaches (classic and trimming) need to succeed in order to compute the means.

These results support the conclusion that the speedups obtained are greater when the benchmarks are harder, and also that in most cases, the benchmarks are computed in less than 0.1 seconds.

Finally, please note that, even though some module names may appear multiple times, this does not mean that they are duplicated benchmarks, but rather files with the same name in different folders and whose code is typically different. To make the comparisons, the full paths where the modules being analyzed are placed are considered; however in the final table we only present the module names for readability.

C.1 Classic Benchmarks

Table C 1: Analysis times and statistics for analyzing the classic benchmarks set.

	share					shfr					shfr-clique				
M	**TC**	**TT**	**ρT**	**TR**	**ρR**	**TC**	**TT**	**ρT**	**TR**	**ρR**	**TC**	**TT**	**ρT**	**TR**	**ρR**
query	2.46	3.50	0.70	4.20	0.59	2.40	2.63	0.91	4.30	0.56	3.82	3.65	1.05	5.48	0.70
zebra	0.89	4.17	0.21	76.20	0.01	4.01	4.59	0.87	78.41	0.05	6.48	6.71	0.97	19848.91	0.00
boyer	42.04	44.91	0.94	49.82	0.84	47.12	50.96	0.92	53.10	0.89	38.56	40.09	0.96	44.30	0.87
tak	1.37	1.63	0.84	3.19	0.43	1.45	1.83	0.79	3.56	0.41	1.95	2.08	0.94	4.24	0.46
witt	38.81	40.18	0.97	68.53	0.57	41.88	40.19	1.04	68.53	0.61	53.54	53.10	1.01	84.32	0.63
aiakl	3.94	4.69	0.84	4.71	0.84	4.51	5.06	0.89	4.95	0.91	6.22	6.47	0.96	7.00	0.89
serialize	25.66	26.47	0.97	38.00	0.68	7.70	9.20	0.84	14.31	0.54	7.64	7.85	0.97	11.05	0.69
mmatrix	5.16	5.69	0.91	6.84	0.75	6.21	6.65	0.93	7.84	0.79	5.50	5.63	0.98	6.96	0.79
qsort	2.24	2.09	1.07	2.03	1.11	1.98	2.58	0.77	2.60	0.76	2.77	2.73	1.01	2.98	0.93
occur	1.43	1.71	0.83	1.80	0.79	1.80	2.11	0.85	2.04	0.88	2.56	2.28	1.12	2.23	1.15
browse	4.04	1.42	2.85	6.12	0.66	4.05	4.54	0.89	6.77	0.60	5.25	5.72	0.92	7.67	0.69
hanoiapp	1.34	1.72	0.78	2.69	0.50	1.43	1.85	0.77	2.96	0.48	2.03	1.93	1.05	3.17	0.64
bid	6.68	7.15	0.93	9.00	0.74	7.18	8.19	0.88	9.95	0.72	6.95	10.76	0.65	13.50	0.51
fib	0.78	0.91	0.85	1.81	0.43	0.99	1.14	0.87	2.36	0.42	1.37	1.46	0.94	2.94	0.47
qsortapp	1.48	1.64	0.90	2.28	0.65	1.51	1.75	0.86	2.35	0.64	2.03	2.14	0.95	2.76	0.74
deriv	1.62	2.37	0.68	3.10	0.52	2.04	2.22	0.92	3.36	0.61	2.54	2.69	0.94	0.95	2.68
append	1.11	1.41	0.78	1.20	0.92	1.91	2.01	0.95	0.00	19.11	1.57	1.40	1.12	1.56	1.01
progeom	3.95	3.95	1.00	4.82	0.82	3.88	4.87	0.80	5.38	0.72	5.60	5.34	1.05	6.53	0.86
grammar	1.96	1.60	1.23	2.74	0.72	2.66	2.73	0.98	3.14	0.85	3.14	3.62	0.87	3.88	0.81
read	28.14	31.12	0.90	55.34	0.51	30.52	38.70	0.79	59.29	0.51	47.51	53.84	0.88	80.43	0.59
ann	174.83	162.28	1.08	192.04	0.91	99.34	99.37	1.00	135.61	0.73	129.27	135.36	0.96	155.42	0.83
rdtok	23.40	31.23	0.75	49.70	0.47	26.59	35.17	0.76	52.75	0.50	35.62	41.88	0.85	66.82	0.53
entry	0.76	0.90	0.84	0.84	0.90	0.89	1.04	0.85	1.00	0.88	0.97	0.93	1.05	1.14	0.86
warplan	181.21	202.36	0.90	203.12	0.89	106.12	117.87	0.90	140.63	0.75	141.17	144.20	0.98	154.49	0.91
peephole	71.08	68.75	1.03	75.04	0.95	50.79	51.53	0.99	54.93	0.92	58.22	60.74	0.96	62.35	0.93
qplan	5693	1161	4.90	547.39	10.40	1037	377.31	2.75	249.25	4.16	5010	152.00	32.96	161.69	30.99
Mods			26					26					26		
μT			1.10					0.95					2.20		
μR			1.06					1.50					1.97		

C.2 Source code of the PLAI Analyzer.

Table C 2: Analysis times and statistics for analyzing the source code of PLAI.

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
fixpo_di	timeout	timeout	-	timeout	timeout	-	88817	66530	1.33
domains	timeout	49167	-	timeout	34882	-	5063	5203	0.97
intermod	21486	14406	1.49	11839	5512	2.15	10968	9839	1.11
fixpo_dx_check_common	0.19	0.02	11.63	0.16	0.18	0.87	0.18	0.18	1.00
intermod_punit	timeout	13342	-	timeout	11593	-	775.79	459.35	1.69
trace_fixp	188.43	137.64	1.37	164.88	122.81	1.34	37.86	71.38	0.53
intermod_options	0.25	0.32	0.78	0.29	0.27	1.07	0.00	0.28	0.36
domains_hooks	4063	4079	1.00	4055	4091	0.99	5.72	5.68	1.01
fixpo_bu	timeout	5566	-	timeout	4195	-	570.52	556.03	1.03
fixpo_plai	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-
fixpo_ops	timeout	timeout	-	timeout	timeout	-	790.88	746.41	1.06
intermod_db	154.23	86.24	1.79	96.69	46.40	2.08	40.75	41.76	0.98
plai_errors	1.19	5.15	0.23	5.87	5.69	1.03	5.82	5.94	0.98
fixpo_check_di2	timeout	timeout	-	timeout	timeout	-	87526	66501	1.32
fixpo_dx_common	0.17	0.18	0.95	0.17	0.16	1.02	0.17	0.22	0.79
apply_assertions_old	177347	9618	18.44	98954	5622	17.60	4402	4372	1.01
fixpoint_options	0.16	0.21	0.79	0.17	0.17	1.01	0.17	0.19	0.93
fixpo_check_di4	timeout	timeout	-	timeout	timeout	-	89499	64305	1.39
re_analysis	timeout	timeout	-	timeout	timeout	-	1279	1301.53	0.98
fixpo_check_di3	timeout	timeout	-	timeout	timeout	-	91042	65200	1.40
plai_db	timeout	2172.26	-	timeout	1971.52	-	161.38	160.44	1.01
intermod_entry	timeout	timeout	-	timeout	timeout	-	4685	3568	1.31
transform	3012	2270	1.33	1946	1420	1.37	218.21	219.24	1.00
apply_assertions	timeout	timeout	-	timeout	timeout	-	2422	2537	0.95
auxinfo_dump	91.23	91.57	1.00	143.37	138.49	1.04	42.00	42.99	0.98
psets	10.87	11.12	0.98	9.71	10.61	0.91	3.78	3.95	0.96
fixpo_plai_gfp	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-
intermod_schedule	13.67	15.12	0.90	15.80	18.35	0.86	23.79	24.08	0.99
notrace	0.26	0.26	1.03	0.33	0.25	1.32	0.28	0.29	0.96
tarjan	timeout	153251	-	timeout	40190	-	2457	2498	0.98
notrace_tr	2.88	2.82	1.02	2.69	2.55	1.05	1.93	1.93	1.00

	share			shfr			shfr-clique		
module	**TC**	**TT**	ρ**T**	**TC**	**TT**	ρ**T**	**TC**	**TT**	ρ**T**
acc_ops	3536	549.14	6.44	3298	401.85	8.21	493.97	511.58	0.97
program_tarjan	114.05	111.78	1.02	106.62	107.19	0.99	30.70	31.65	0.97
plai	timeout	702.09	-	timeout	637.55	-	190.46	179.71	1.06
fixpo_check_reduced_di	timeout	timeout	-	timeout	timeout	-	88247	64173	1.38
apply_assertions_inc	timeout	timeout	-	timeout	timeout	-	217.79	196.61	1.11
plai_domain	4104	4182	0.98	4043	4015	1.01	3.94	3.62	1.09
fixpo_ops_gfp	timeout	timeout	-	timeout	timeout	-	393.43	400.72	0.98
fixpo_dd	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-
normalize_args	980.98	942.73	1.04	892.09	918.40	0.97	17.63	23.11	0.76
fixpo_check_di	timeout	timeout	-	timeout	timeout	-	89032	64926	1.37
fixpo_check_di5	timeout	timeout	-	timeout	timeout	-	89487	64330	1.39
intermod_success	timeout	154006	-	timeout	150781	-	7624	5813	1.31
intermod_ops	timeout	timeout	-	timeout	timeout	-	timeout	29295	-
view_fixp	timeout	timeout	-	timeout	197829	-	106547	1843	57.81
domain_hooks_unfinished	4.13	4.17	0.99	4.10	4.13	0.99	4.43	4.32	1.02
plai_db_instances	3193	1606	1.99	2352	1239.10	1.90	1008	480.83	2.10
incanal_options	0.19	0.16	1.18	0.17	0.18	0.94	0.19	0.17	1.07
plai_db_comparator	118355	15626	7.57	114378	13088	8.74	228.80	232.58	0.98
incanal_experiments	12.26	5.80	2.11	17.48	9.20	1.90	9.73	8.84	1.10
tarjan_inc	timeout	timeout	-	timeout	172030	-	1296	590.74	2.20
incanal_persistent_db	8.88	8.15	1.09	9.53	9.49	1.00	16.33	13.09	1.25
incanal_driver	timeout	timeout	-	timeout	92415	-	timeout	344.13	-
incanal_deletion	19485	6777	2.88	10132	1263	8.02	72.97	58.69	1.24
incanal_db	timeout	52733	-	timeout	4536	-	237.39	232.69	1.02
incanal	timeout	65018	-	timeout	2296	-	1805	226.57	7.97

	share	shfr	shfr-clique
Mods	56	56	56
FC	29	29	5
FT	20	17	3
μ**T**	2.67	2.61	2.36
μ**T** $>$ 1s (9-9-33)	4.68	5.55	3.14
μ**T** $>$ 0.5s (10-10-34)	4.32	5.10	3.07
μ**T** $>$ 0.1s (13-13-37)	3.64	4.26	2.89

C.3 Libraries in the `Ciao` *system.*

Table C 3: Analysis times and statistics for analyzing the `Ciao` libraries.

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
initial	0.25	0.24	1.05	0.25	0.35	0.70	0.29	0.42	0.69
operators	16.06	17.93	0.90	16.60	13.70	1.21	14.58	19.98	0.73
unittestdecls	0.25	0.28	0.92	0.24	0.33	0.74	0.25	0.26	0.96
system	2795.54	2710.89	1.03	3288.71	3316.89	0.99	4436.77	4552.79	0.97
default	0.26	0.47	0.55	0.26	0.24	1.06	0.26	0.30	0.88
stream_utils	18.21	13.45	1.35	19.93	17.21	1.16	24.32	24.71	0.98
dynmod_holder	0.63	0.63	0.99	0.71	0.67	1.06	0.69	1.01	0.69
dict	3534.96	3545.48	1.00	1819.39	1867.04	0.97	171.06	194.10	0.88
aggregates	47.63	51.00	0.93	50.99	57.91	0.88	37.20	37.02	1.01
sort	610.39	109.84	5.56	523.88	86.12	6.08	5599.79	83.88	66.76
llists	7.32	7.95	0.92	7.45	7.48	1.00	9.48	9.98	0.95
goal_trans	169.94	5.18	32.80	204.02	9.28	21.98	6.85	10.53	0.65
strings	2.71	2.75	0.99	0.32	3.17	0.10	6.15	2.49	2.47
trace	0.26	0.29	0.92	0.26	0.30	0.88	0.28	0.38	0.74
prelude	0.26	0.24	1.05	0.26	0.12	2.24	0.29	0.42	0.68
miscprops	1.36	1.36	1.00	1.45	1.59	0.91	2.16	0.30	7.17
io_port_reify	27.17	29.13	0.93	37.86	41.87	0.90	27.04	36.47	0.74
debug	0.29	0.27	1.05	0.27	0.28	0.98	0.28	0.18	1.53
lists.test	1.62	1.69	0.96	1.99	2.17	0.92	3.58	3.95	0.91
messages	37.65	37.65	1.00	34.18	35.68	0.96	54.69	57.73	0.95
libpaths	3.56	3.50	1.02	4.38	4.32	1.01	10.29	9.84	1.05
streams	2.31	6.12	0.38	6.38	0.01	580.00	1.94	2.39	0.81
write	429.88	442.67	0.97	486.75	477.78	1.02	1592.24	1724.51	0.92
between	1.04	1.18	0.88	1.05	1.47	0.71	1.33	1.50	0.89
pathnames	35.18	34.22	1.03	36.10	33.51	1.08	54.30	52.52	1.03
lists	63.54	66.12	0.96	78.73	82.69	0.95	150.94	159.78	0.94
tokenize	225.49	211.82	1.06	276.90	267.87	1.03	192.19	193.84	0.99
nortchecks	0.25	0.25	0.98	0.26	0.24	1.05	0.28	0.45	0.62
read	34894.61	18094.06	1.93	35972.50	19784.24	1.82	7527.64	7932.97	0.95
attrdump	2431.19	274.00	8.87	2022.07	192.86	10.48	41.79	50.08	0.83
define_flag	0.42	0.50	0.85	0.51	0.57	0.90	0.88	0.96	0.92
terms	7.32	8.08	0.91	8.16	9.32	0.88	11.50	9.21	1.25

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
errhandle	4828.63	7.98	605.17	4704.53	9.29	506.41	6153.10	12.02	511.86
nodebug	0.31	0.26	1.22	0.28	0.27	1.02	0.28	0.30	0.94
format	384.59	387.01	0.99	81.12	91.92	0.88	132.88	141.31	0.94
dec10_io	16.27	16.32	1.00	18.52	14.68	1.26	27.73	26.99	1.03
unittestdecls_doc	0.16	0.17	0.94	0.16	0.18	0.90	0.21	0.19	1.12
read_from_string	2396.51	617.13	3.88	1703.07	613.87	2.77	420.40	407.83	1.03
old_database	2.96	3.08	0.96	4.03	3.94	1.02	5.95	8.12	0.73
default_iso_strict	0.26	0.29	0.90	0.25	0.27	0.92	0.26	0.38	0.68
cyclic_terms	9.19	9.39	0.98	11.54	12.34	0.94	11.32	12.34	0.92
fastrw	1.59	0.30	5.24	1.92	2.05	0.94	3.04	3.15	0.96
foreign_compilation	1.37	1.32	1.04	1.53	1.52	1.01	3.02	2.95	1.03
pathnames_boot	0.45	0.50	0.89	0.45	0.45	0.99	0.50	0.57	0.88
port_reify	2.03	2.14	0.94	2.23	2.30	0.97	3.09	3.08	1.00
odd	1.35	1.29	1.04	1.43	1.56	0.92	2.12	2.19	0.97
sets	673.25	680.44	0.99	442.94	462.05	0.96	72.03	80.09	0.90
format_to_string	949.27	999.36	0.95	777.84	710.52	1.09	timeout	1682.76	-
ctrlcclean	1.95	2.16	0.90	2.23	2.41	0.92	2.66	2.75	0.97
pure	0.28	0.25	1.09	0.31	0.25	1.24	0.27	0.33	0.83
pure_doc	0.19	0.16	1.19	0.22	0.23	0.96	0.17	0.18	0.92
dynamic_doc	0.18	0.15	1.14	0.15	0.15	0.99	0.18	0.46	0.39
dynamic_rt	37.90	38.38	0.99	44.21	47.69	0.93	41.38	47.34	0.87
dynamic	0.25	0.25	1.02	0.25	0.25	0.99	0.26	0.33	0.80
native_props_cost_doc	10.41	11.05	0.94	11.69	11.93	0.98	13.23	13.86	0.95
native_props_sideff_doc	0.64	0.66	0.97	0.63	0.65	0.96	0.68	0.78	0.88
native_props_exceptions	0.18	0.16	1.13	0.16	0.16	1.06	0.17	0.27	0.64
assertions_basic	0.25	0.25	0.99	0.25	0.27	0.92	0.26	0.40	0.63
native_props_polyhedral	0.16	0.18	0.93	0.15	0.16	0.90	0.16	0.16	0.99
native_props_usage_doc	0.28	0.26	1.05	0.26	0.25	1.05	0.26	0.48	0.54
native_props_polyhedral_doc	2.03	2.23	0.91	2.29	2.42	0.95	2.67	2.85	0.94
assrt_lib_extra	40.19	38.15	1.05	43.98	45.35	0.97	47.53	51.93	0.92
native_props_sideff	0.16	0.16	0.99	0.20	0.16	1.21	0.19	0.19	0.96
assrt_norm_common	0.17	0.17	0.98	0.18	0.17	1.04	0.18	0.28	0.64
native_props_nfdet	0.17	0.16	1.06	0.20	0.15	1.30	0.15	0.22	0.72
c_itf_props	1.06	1.09	0.97	1.23	1.11	1.11	1.61	1.74	0.93
native_props_cardinality_doc	0.86	0.81	1.07	0.88	0.84	1.05	0.89	0.96	0.94

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
assertions_doc	0.29	0.26	1.12	0.28	0.27	1.04	0.27	0.44	0.62
native_props_nfdet_doc	1.45	1.69	0.86	1.63	1.53	1.06	1.80	1.88	0.96
native_props_cardinality	0.18	0.16	1.14	0.19	0.16	1.15	0.16	0.25	0.66
assertions	0.24	0.25	0.97	0.27	0.26	1.04	0.24	0.26	0.93
assrt_lib	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-
doc_props	0.26	0.26	1.03	0.34	0.34	1.01	0.36	0.45	0.80
native_props_cost	0.20	0.16	1.20	0.18	0.00	1.79	0.17	0.21	0.84
native_props	32.75	34.85	0.94	32.88	35.08	0.94	45.36	50.17	0.90
assertions_props	916.76	42.70	21.47	856.91	43.11	19.88	23.13	24.22	0.96
native_props_shfrg	0.15	0.16	0.96	0.23	0.16	1.42	0.16	0.19	0.84
native_props_exceptions_doc	0.98	0.99	0.99	1.03	1.13	0.91	1.16	1.22	0.95
native_props_shfrg_doc	4.82	5.44	0.89	6.55	7.56	0.87	7.86	8.72	0.90
native_props_rtc	2457.61	1147.48	2.14	498.71	265.17	1.88	51.86	48.81	1.06
assrt_write	2779.05	1104.50	2.52	2643.96	1005.78	2.63	487.23	493.61	0.99
tester_aux1	0.34	0.25	1.38	0.00	0.29	0.35	0.45	0.33	1.38
foomain	0.19	0.20	0.96	0.18	0.16	1.13	0.18	0.38	0.49
foo	0.55	0.57	0.96	0.11	0.57	0.19	0.64	0.67	0.95
testdisj	0.53	0.54	0.99	0.31	0.47	0.65	0.49	0.68	0.73
arith	4.52	4.51	1.00	4.65	5.07	0.92	5.66	6.61	0.86
tester_aux2	0.34	0.35	0.96	0.38	0.37	1.05	0.45	0.60	0.75
tester	0.23	0.24	0.96	0.25	0.23	1.11	0.26	0.26	1.02
main	1.21	1.18	1.02	1.38	1.32	1.05	1.64	1.97	0.83
foo_main	0.18	0.18	1.04	0.18	0.15	1.15	0.17	0.55	0.30
bar	0.16	0.16	0.96	0.18	0.16	1.10	0.18	0.24	0.74
tester_alt	0.31	0.32	0.97	0.39	0.33	1.17	0.38	0.51	0.75
resolve	0.63	0.60	1.05	0.71	0.61	1.15	0.68	0.69	0.98
arithmetic	15.34	15.68	0.98	16.29	16.69	0.98	17.82	18.64	0.96
basic_props_test	0.18	0.17	1.08	0.19	0.20	0.92	0.19	0.19	0.99
aux1	0.35	0.35	1.01	0.41	0.37	1.09	0.44	0.59	0.75
part1	0.19	0.17	1.12	0.18	0.16	1.16	0.16	0.30	0.53
modules_user	0.28	0.29	1.00	0.27	0.27	1.00	0.29	0.44	0.66
tester_aux	0.34	0.33	1.05	0.38	0.40	0.97	0.41	0.49	0.83
tester_aux3	0.35	0.28	1.24	0.30	0.30	1.00	0.29	0.16	1.78
main	0.56	0.50	1.12	0.60	0.63	0.96	0.71	0.65	1.09
my_native_props	3.23	2.66	1.22	3.07	2.96	1.04	3.34	3.38	0.99

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
SETTINGS	3.21	3.17	1.01	3.42	3.37	1.02	4.90	4.63	1.06
dummy_end_of_file	0.32	0.27	1.18	0.30	0.30	0.99	0.29	0.64	0.45
pp_tr	11.59	12.81	0.90	13.50	14.54	0.93	15.60	19.91	0.78
pp	0.24	0.28	0.87	0.25	0.32	0.79	0.27	0.32	0.85
ex3_ana	0.42	0.36	1.17	0.46	0.44	1.05	0.46	0.59	0.79
ex2_check	0.35	0.37	0.95	0.37	0.41	0.90	0.38	0.54	0.70
ex1_check	0.35	0.34	1.04	0.40	0.35	1.15	0.42	0.46	0.92
rtc_old	0.16	0.17	0.95	0.22	0.17	1.36	0.19	0.20	0.96
assrt_write0	59.04	60.45	0.98	61.07	65.01	0.94	50.31	52.31	0.96
assrt_synchk	0.18	0.18	1.01	0.01	0.18	0.07	0.16	0.22	0.75
assrt_mpp	4.72	4.90	0.96	5.17	5.68	0.91	2.29	8.10	0.28
embedded_tr	timeout	timeout	-	timeout	timeout	-	25.35	14.74	1.72
debugger_lib	timeout	timeout	-	timeout	timeout	-	2757.33	3109.76	0.89
debugger	186.88	197.17	0.95	190.39	185.00	1.03	11.98	12.01	1.00
embedded_rt	167.14	167.10	1.00	155.95	156.75	0.99	6.44	2.84	2.27
debugger_doc	0.17	0.19	0.91	0.16	0.17	0.95	0.17	0.23	0.75
debugger_tr	timeout	59742.26	-	timeout	38265.80	-	171.88	190.72	0.90
embedded	0.29	0.29	1.00	0.29	0.26	1.13	0.28	0.55	0.51
debug_srcdbg	0.30	0.27	1.12	0.31	0.29	1.09	0.35	0.40	0.87
debug_raw	0.27	0.26	1.03	0.24	0.25	0.98	0.28	0.50	0.56
basicmodes_doc	0.79	0.86	0.92	0.89	1.13	0.79	1.05	1.00	1.05
basicmodes	0.18	0.16	1.10	0.20	0.18	1.12	0.18	0.20	0.90
isomodes_doc	0.87	0.86	1.02	0.93	0.97	0.97	1.09	1.30	0.84
isomodes	0.17	0.17	1.04	0.16	0.19	0.86	0.18	0.20	0.89
attr_tr	14.56	9.21	1.58	12.70	7.46	1.70	4.86	5.53	0.88
attr_mono_compat	0.53	0.55	0.96	0.54	0.62	0.86	0.55	0.55	1.00
attr	0.61	0.61	1.00	0.59	0.69	0.86	0.62	0.59	1.05
attr_rt	107.66	85.83	1.25	97.70	82.26	1.19	105.00	91.87	1.14
attr_doc	0.20	0.16	1.21	0.17	0.16	1.08	0.19	0.32	0.60
attr_bench	4.29	5.04	0.85	4.59	5.50	0.84	5.37	6.13	0.88
myfreeze	2.76	2.92	0.95	0.56	3.38	0.17	3.00	3.24	0.93
mattr_global	6.95	7.49	0.93	5.91	6.25	0.95	7.34	8.15	0.90
swi_mattr_global_tr	0.83	0.89	0.93	0.95	0.93	1.03	0.86	1.03	0.83
mattr_local_common	39.89	41.71	0.96	40.90	43.85	0.93	17.26	18.80	0.92
mattr_sicstus_doc	1.43	1.34	1.07	1.60	1.67	0.96	1.94	2.74	0.71

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
mattr_global_trans	23.16	19.56	1.18	20.48	12.83	1.60	11.02	7.85	1.40
mattr_global_doc	0.18	0.19	0.98	0.16	0.17	0.95	0.20	0.25	0.78
mattr_global_code	11.75	12.15	0.97	9.26	10.37	0.89	16.35	18.16	0.90
noprelude	0.26	0.26	0.98	0.25	0.25	1.02	0.25	0.32	0.78
noprelude_doc	0.16	0.18	0.92	0.17	0.18	0.93	0.16	0.20	0.81
opt_rt_safe_ctrt_poly	0.24	0.26	0.94	0.25	0.26	0.97	0.27	0.42	0.64
opt_rt_safe_ctrt_blind	0.25	0.26	0.95	0.32	0.25	1.27	0.00	0.69	0.14
opt_rt_safe_rt	0.24	0.26	0.93	0.38	0.25	1.53	0.28	0.26	1.05
opt_rt_unsafe	0.28	0.24	1.15	0.25	0.26	0.97	0.25	0.25	1.01
opt_rt_client_safe	0.27	0.25	1.10	0.26	0.30	0.89	0.33	0.31	1.08
opt_rt_safe_ctrt	0.28	0.25	1.11	0.24	0.27	0.87	0.28	0.30	0.91
fsimp	17.51	19.01	0.92	17.12	19.13	0.89	21.14	26.17	0.81
autoshallow	128.22	88.20	1.45	99.47	58.51	1.70	46.44	51.33	0.90
rtchecks_shallow_doc	0.17	0.18	0.91	0.15	0.18	0.85	0.19	0.20	0.93
rtchecks_shallow	0.46	0.27	1.69	0.28	0.30	0.95	0.30	0.39	0.76
rtchecks_shallow_tr	timeout	118624.81	-	timeout	47278.43	-	149970.29	1379.15	108.74
rtchecks_shallow_rt	11.46	11.67	0.98	12.20	13.56	0.90	13.61	14.74	0.92
example	3.63	3.42	1.06	3.76	0.96	3.90	4.64	4.79	0.97
foo	3.62	3.68	0.99	3.86	4.41	0.88	4.58	4.93	0.93
cost_ana	2.59	2.70	0.96	3.10	3.17	0.98	2.87	3.48	0.82
foo_4modes	1.22	1.19	1.03	1.34	1.32	1.02	1.61	1.65	0.98
write_tokens	6.69	7.69	0.87	7.73	9.74	0.79	9.96	11.53	0.86
write_c	5758.57	1231.92	4.67	7024.12	1271.55	5.52	955.62	1109.17	0.86
process	169754.06	5337.71	31.80	24943.13	4427.60	5.63	1433.98	1415.68	1.01
process_channel	45.19	43.88	1.03	45.58	50.38	0.90	46.30	51.64	0.90
restricted_syntax	0.25	0.26	0.99	0.24	0.31	0.77	0.26	0.27	0.97
restricted_syntax_tr	1.56	1.75	0.89	1.70	1.92	0.88	2.48	2.74	0.91
string_type	0.28	0.26	1.08	0.25	0.27	0.93	0.24	0.30	0.80
string_type_rt	1.25	1.23	1.01	1.50	1.54	0.97	2.65	2.61	1.02
bundle_flags	9.11	8.83	1.03	10.45	9.65	1.08	17.06	18.00	0.95
bundle_paths	3663.25	252.21	14.52	3541.74	494.74	7.16	62.23	71.55	0.87
bundlereg_db	0.68	0.68	1.00	0.73	0.72	1.02	0.70	0.73	0.96
bundlereg_load	0.17	0.16	1.01	0.01	0.15	0.08	0.16	0.20	0.79
bundle_info	2.53	2.72	0.93	2.77	2.93	0.94	3.32	3.82	0.87
dynamic_bundle	0.18	0.17	1.09	0.20	0.18	1.11	0.18	0.19	0.97

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
dynamic_clauses_doc	0.22	0.17	1.26	0.19	0.18	1.02	0.16	0.22	0.71
dynamic_clauses_rt	30.43	32.72	0.93	34.34	35.75	0.96	25.92	27.43	0.94
dynamic_clauses	0.51	0.46	1.12	0.53	0.50	1.07	0.48	0.54	0.88
selfmodif	0.33	1.25	0.27	1.32	1.49	0.89	1.25	1.09	1.14
metatypes	0.27	0.26	1.07	0.26	0.29	0.91	0.11	0.22	0.49
metatypes_tr	1.35	1.43	0.94	1.44	1.62	0.89	1.63	2.01	0.81
cmdline_parser	1478.87	342.18	4.32	858.17	257.31	3.34	125.41	80.04	1.57
cmdline_help	0.18	0.17	1.03	0.16	0.19	0.85	0.17	0.19	0.87
term_list	16.03	16.94	0.95	17.16	18.98	0.90	23.24	24.41	0.95
rtchecks_send	130.19	37.85	3.44	122.20	36.53	3.35	16.75	9.53	1.76
rtchecks_pretty	timeout	2114.75	-	timeout	1717.84	-	65.84	68.23	0.97
rtchecks_meta	timeout	timeout	-	timeout	timeout	-	36.03	24.01	1.50
rtchecks_rt	278.59	284.14	0.98	262.80	267.92	0.98	262.97	274.57	0.96
rtchecks_basic	timeout	15539.41	-	timeout	13396.82	-	79.93	62.14	1.29
rtchecks	0.28	0.24	1.17	0.27	0.25	1.11	0.26	0.32	0.83
rtchecks_tr	timeout	timeout	-	timeout	timeout	-	timeout	5237.39	-
rtchecks_doc	0.18	0.18	1.01	0.16	0.18	0.92	0.18	0.22	0.79
rtchecks_rt_propimpl	0.29	0.24	1.20	0.31	0.26	1.18	0.26	0.33	0.78
rtchecks_tr_library	0.30	0.25	1.18	0.26	0.26	1.00	0.27	0.27	0.99
rtchecks_example3	timeout	1520.07	-	timeout	1295.97	-	612.98	618.56	0.99
rtc_external	0.49	0.39	1.25	0.50	0.46	1.09	0.43	0.61	0.70
rtchecks_example	timeout	timeout	-	timeout	timeout	-	368.63	379.73	0.97
rtchecks_disc	14.77	14.88	0.99	15.91	14.06	1.13	2.99	0.51	5.80
example_fullasr	1600.22	15.33	104.42	1564.79	15.05	103.97	8.48	5.28	1.61
rtchecks_constraints	2.52	3.11	0.81	3.05	3.27	0.93	3.11	3.75	0.83
rtchecks_example2	timeout	1529.99	-	timeout	1364.42	-	402.53	397.81	1.01
rtchecks_inline_tr	1.78	1.70	1.05	1.94	2.00	0.97	1.28	1.48	0.87
rtchecks_tr	1.96	1.91	1.03	2.28	2.34	0.98	2.53	2.62	0.96
rtchecks_doc	0.16	0.16	1.01	0.17	0.37	0.46	0.16	0.21	0.76
api_internal_dec	0.16	0.17	0.98	0.18	0.16	1.09	0.18	0.20	0.94
rtchecks_inline	49.22	42.40	1.16	44.68	34.61	1.29	7.07	8.75	0.81
apply_dict	30.50	31.40	0.97	29.68	30.57	0.97	21.24	23.30	0.91
pretty_names	1.68	1.66	1.01	2.42	2.11	1.15	3.80	3.86	0.98
complete_dict	53.95	48.38	1.12	52.92	50.13	1.06	23.17	26.03	0.89
dict_types	1.43	1.42	1.01	1.66	1.67	1.00	3.38	0.35	9.73

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
metaprops	0.19	0.17	1.11	0.16	0.16	1.01	0.01	0.30	0.03
meta_props	2.13	2.21	0.97	2.45	2.49	0.98	2.94	2.99	0.98
clause_check	1.40	1.41	0.99	1.43	1.52	0.94	1.47	1.61	0.91
global_module_options	9.17	9.58	0.96	11.51	12.94	0.89	23.46	26.85	0.87
pl2wam	timeout	29302.76	-	timeout	7552.63	-	2076.58	2107.85	0.99
callback	0.30	0.30	1.01	0.30	0.31	0.96	0.30	0.36	0.82
foreign__gluecode	timeout	4660.36	-	timeout	4508.20	-	59.72	48.84	1.22
basic_compilation_modules	0.28	0.26	1.10	0.24	0.26	0.94	0.26	0.29	0.89
c_itf	timeout	timeout	-	timeout	timeout	-	21775.41	21199.19	1.03
shell_itf	369.85	290.04	1.28	338.34	260.89	1.30	667.66	652.17	1.02
srcdbg	timeout	timeout	-	timeout	timeout	-	37.11	37.57	0.99
up_to_date	0.64	0.74	0.87	0.76	0.91	0.84	0.92	1.06	0.86
pl2wam_tables	18.83	18.34	1.03	17.37	16.47	1.05	8.28	8.27	1.00
add_goal_trans	0.99	1.02	0.97	0.99	0.91	1.08	1.05	1.02	1.03
frontend_condcomp	34.08	37.13	0.92	36.26	39.71	0.91	21.42	22.59	0.95
translation	531.36	297.01	1.79	557.96	261.18	2.14	49.74	53.87	0.92
file_buffer	4.79	4.57	1.05	5.02	4.57	1.10	6.39	6.83	0.94
gauge_aux	4.83	4.51	1.07	4.83	4.78	1.01	4.51	4.54	0.99
build_foreign_interface	timeout	135449.89	-	timeout	timeout	-	1624.92	923.83	1.76
mexpand	7.96	7.19	1.11	7.12	7.14	1.00	7.83	7.26	1.08
compiler	6.88	7.61	0.90	8.13	8.89	0.91	9.67	10.60	0.91
emulator_data	0.29	0.27	1.04	0.30	0.29	1.03	0.12	0.30	0.42
exemaker	timeout	104.21	-	timeout	87.67	-	221.91	179.96	1.23
frontend_core	6.55	6.65	0.99	6.83	6.89	0.99	6.66	7.55	0.88
engine_path	3.86	3.82	1.01	4.23	4.42	0.96	6.96	6.63	1.05
win_exec_ext	0.48	0.31	1.57	0.28	0.30	0.90	0.29	0.41	0.71
wamql	30.16	28.21	1.07	29.30	29.63	0.99	28.69	30.14	0.95
pwamql	0.16	0.16	1.04	0.15	0.01	14.09	0.00	0.17	0.57
module	0.28	0.25	1.09	0.25	0.26	0.95	0.31	0.29	1.06
plpwam	0.17	0.19	0.88	0.15	0.17	0.87	0.18	0.18	0.99
builtins_pwam	0.19	0.16	1.14	0.17	0.17	1.01	0.18	0.20	0.92
make_actmod	0.84	0.73	1.15	0.77	0.78	0.99	0.78	0.85	0.92
clause_print	63.66	7.24	8.79	105.28	7.10	14.83	5.55	6.44	0.86
program_keys	156.08	154.51	1.01	177.48	176.94	1.00	224.78	233.73	0.96
p_unit	timeout	timeout	-	timeout	timeout	-	46538.76	46974.10	0.99

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
p_unit_hooks	1.00	0.98	1.03	1.16	1.33	0.87	0.87	1.37	0.64
aux_filenames	18.15	16.88	1.08	19.78	18.28	1.08	35.57	33.66	1.06
tr_syntax	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-
unexpand	215.79	212.00	1.02	167.68	165.80	1.01	89.43	105.23	0.85
p_unit_db	12639.96	13034.07	0.97	11689.59	11906.34	0.98	9166.68	8934.33	1.03
p_asr_package_info	0.74	0.48	1.54	0.45	0.52	0.87	0.47	0.48	0.97
native	13.69	9.07	1.51	15.81	15.26	1.04	17.04	18.64	0.91
p_canonical	timeout	timeout	-	timeout	34127.38	-	6161.06	6261.99	0.98
assrt_norm	timeout	timeout	-	timeout	timeout	-	5443.27	2884.65	1.89
p_unit_argnames	0.27	0.24	1.14	0.33	0.25	1.32	0.27	0.26	1.02
p_unit_basic	903.45	74.28	12.16	867.01	38.98	22.24	70.74	68.63	1.03
p_printer	timeout	timeout	-	timeout	timeout	-	timeout	8472.19	-
prednames	1.67	2.02	0.83	2.09	2.11	0.99	2.57	3.73	0.69
p_asr_cache	0.17	0.16	1.01	0.15	0.21	0.69	0.18	0.19	0.94
emugen_errors	72.54	81.92	0.89	49.09	55.39	0.89	56.70	65.98	0.86
emugen	0.24	0.24	1.01	0.26	0.27	0.97	0.27	0.39	0.69
emugen_tr	timeout	2190.45	-	timeout	546.70	-	342.85	380.89	0.90
emugen_ops	0.43	0.26	1.67	0.29	0.26	1.10	0.26	0.31	0.84
emugen_doc	0.16	0.16	0.96	0.19	0.18	1.06	0.18	0.17	1.05
condcomp_doc	0.17	0.16	1.01	0.17	0.24	0.69	0.19	0.35	0.54
condcomp	0.25	0.25	0.99	0.26	0.25	1.04	0.26	0.38	0.68
test4	0.27	0.32	0.85	0.28	0.27	1.02	0.28	0.29	0.95
test2	0.55	0.54	1.01	0.61	0.70	0.87	0.83	0.68	1.23
test1	0.45	0.48	0.94	0.42	0.52	0.80	0.63	0.93	0.68
test3	1.80	1.75	1.03	1.79	2.19	0.82	2.20	2.49	0.88
datafacts	0.24	0.26	0.93	0.24	0.25	0.98	0.31	0.27	1.14
datafacts_rt	8.38	8.93	0.94	10.61	11.37	0.93	13.81	16.11	0.86
datafacts_doc	0.19	0.15	1.24	0.17	0.20	0.86	0.17	0.24	0.72
foreign_interface_doc	0.16	0.20	0.78	0.17	0.17	1.03	0.20	0.17	1.16
foreign_interface_properties	7.03	7.01	1.00	7.53	7.88	0.96	10.41	10.11	1.03
foreign_interface_tr	3.14	2.74	1.15	3.33	3.48	0.96	3.04	4.66	0.65
foreign_interface	0.25	0.28	0.88	0.28	0.25	1.12	0.29	0.31	0.92
foreign_interface_properties_OC	5.65	5.69	0.99	6.37	6.85	0.93	8.41	8.19	1.03
test_all	7.36	10.45	0.70	8.67	14.78	0.59	17.33	20.37	0.85
objects	0.38	0.43	0.88	0.49	0.48	1.03	0.84	0.76	1.10

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
strings_and_atoms	0.73	0.58	1.27	0.80	0.78	1.03	1.49	1.50	1.00
byte_lists	0.41	0.42	0.98	0.52	0.52	1.00	0.87	0.90	0.97
int_lists	0.44	0.45	0.97	0.51	0.54	0.95	0.88	1.05	0.84
newtype	0.70	0.75	0.94	0.75	0.82	0.92	1.11	1.13	0.98
random	0.91	0.77	1.19	0.89	0.93	0.95	1.42	2.29	0.62
bigints	0.82	0.85	0.96	0.86	1.06	0.81	1.49	1.49	1.00
foreign_init	0.48	0.46	1.04	0.45	0.64	0.69	0.56	0.84	0.66
any_term	0.39	0.38	1.02	0.47	0.60	0.78	0.79	0.91	0.87
math	0.54	0.48	1.11	0.57	0.85	0.67	1.28	1.40	0.91
math_user	0.30	0.31	0.97	0.33	0.30	1.09	0.31	0.58	0.53
cc_stack	2.30	2.21	1.04	2.67	3.17	0.84	3.42	3.70	0.92
exceptions_example	1.07	1.15	0.93	1.28	1.23	1.04	1.85	1.86	1.00
test	1.38	1.42	0.97	1.48	1.50	0.99	1.62	1.60	1.01
test	1.04	1.09	0.95	1.13	1.40	0.80	1.31	1.49	0.88
fsmemo_doc	0.16	0.16	0.98	0.18	0.18	0.96	0.16	0.17	0.93
fsmemo_defs	0.40	0.40	1.02	0.44	0.56	0.79	0.45	0.79	0.56
fsmemo_rt	30.96	19.92	1.55	29.93	22.31	1.34	17.29	17.98	0.96
fsmemo	0.40	0.42	0.96	0.41	0.47	0.87	0.44	0.54	0.82
hiord	0.30	0.29	1.01	0.26	0.39	0.67	0.28	0.29	0.96
fsimp	15.27	17.04	0.90	16.26	17.72	0.92	19.81	21.79	0.91
ops	0.29	0.27	1.06	0.25	0.26	0.97	0.26	0.34	0.77
fastchecks	0.18	0.16	1.09	0.16	0.24	0.67	0.18	0.24	0.76
rtchecks_cached_doc	0.16	0.15	1.06	0.16	0.18	0.90	0.18	0.20	0.93
rtchecks_cached_rt	49.58	16.20	3.06	51.76	17.62	2.94	17.29	19.16	0.90
cond_undo_cache	1.44	0.76	1.89	1.78	1.68	1.06	1.47	0.82	1.78
rtchecks_cached	0.25	0.27	0.92	0.29	0.28	1.06	0.28	0.41	0.68
rtchecks_cached_tr	timeout	86793.74	-	39230.18	12543.39	3.13	2802.31	1576.73	1.78
optparse	0.33	0.31	1.08	0.32	0.43	0.75	0.34	0.34	0.98
optparse_rt	901.31	87.36	10.32	923.85	78.27	11.80	4.58	5.40	0.85
optparse_tr	timeout	timeout	-	timeout	timeout	-	32.18	29.68	1.08
dcg_tr	78871.35	77198.84	1.02	10548.53	9484.39	1.11	462.53	564.54	0.82
dcg_phrase_doc	0.23	0.16	1.41	0.14	0.17	0.88	0.19	0.23	0.83
dcg	0.26	0.25	1.04	0.26	0.30	0.86	0.00	0.39	0.26
dcg_phrase	0.28	0.36	0.78	0.32	0.56	0.57	0.32	0.61	0.52
dcg_phrase_rt	1.28	1.65	0.77	1.38	1.48	0.93	1.30	2.19	0.59

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
dcg_ops	0.26	0.34	0.76	0.10	0.27	0.37	0.26	0.40	0.65
dcg_doc	0.16	0.16	1.00	0.35	0.19	1.88	0.17	0.21	0.78
phrase_test	2.84	3.55	0.80	3.43	3.55	0.97	4.54	5.19	0.88
regtypes_doc	0.00	0.16	0.61	0.16	0.20	0.80	0.17	0.20	0.82
regtypes_tr	0.62	0.63	0.98	0.63	0.75	0.85	0.69	0.70	0.98
regtypes	0.25	0.26	0.98	0.25	0.42	0.61	0.28	0.40	0.68
concurrency	12.82	12.93	0.99	13.65	13.26	1.03	16.26	18.48	0.88
examples	13.04	16.49	0.79	14.81	18.14	0.82	20.20	24.06	0.84
e	0.57	0.64	0.89	0.64	0.81	0.80	0.79	1.06	0.75
foo	1.97	2.23	0.88	2.06	2.27	0.91	2.43	2.60	0.93
goo	2.64	0.77	3.41	2.74	2.76	0.99	5.32	3.93	1.35
modes	0.26	0.26	1.03	0.28	0.27	1.03	0.25	0.42	0.60
modes_doc	0.89	0.90	0.99	1.00	1.43	0.70	1.00	1.07	0.93
argnames_fsyntax	0.21	0.17	1.23	0.23	0.23	1.03	0.26	0.20	1.33
nativeprops	0.37	0.27	1.35	0.32	0.26	1.27	0.38	0.29	1.31
counters	1.67	1.72	0.97	2.91	2.09	1.39	2.40	2.20	1.09
queues	6.09	6.05	1.01	6.48	6.51	1.00	3.40	3.83	0.89
arithpreds	6.70	6.55	1.02	8.09	7.76	1.04	8.40	7.74	1.09
listing	4.21	3.34	1.26	4.33	2.97	1.46	5.34	4.59	1.16
formulae	30.80	30.08	1.02	43.19	49.72	0.87	50.09	52.80	0.95
ttyout	3.42	3.40	1.00	3.86	3.65	1.06	4.73	4.70	1.01
version_strings	20.47	17.23	1.19	19.24	15.38	1.25	40.21	38.39	1.05
symlink_locks	38.68	3.55	10.90	42.77	4.63	9.24	6.33	6.05	1.05
io_alias_redirection	1.11	1.26	0.88	1.40	0.96	1.46	1.41	1.37	1.03
assoc	timeout	timeout	-	timeout	timeout	-	408.18	402.11	1.02
functions	0.39	0.28	1.40	0.31	0.24	1.30	0.32	0.34	0.95
sortnums	32.97	19.27	1.71	33.26	20.16	1.65	16.88	17.19	0.98
terms_io	1.30	5.28	0.25	6.03	6.72	0.90	7.81	8.16	0.96
iso_incomplete	55.33	58.31	0.95	39.12	31.39	1.25	71.98	65.31	1.10
pretty_print	35.49	36.51	0.97	41.83	47.79	0.88	44.17	49.25	0.90
prompt	1.87	2.07	0.90	2.45	2.18	1.13	2.80	2.87	0.98
text_template	17.29	13.63	1.27	18.71	13.88	1.35	41.25	35.46	1.16
lsets	548.53	493.58	1.11	502.44	348.90	1.44	592.38	549.06	1.08
terms_check	90.48	91.95	0.98	29.80	31.81	0.94	26.24	26.09	1.01
idlists	13.83	14.77	0.94	21.36	21.76	0.98	20.94	19.04	1.10

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
archive_files	101.92	99.71	1.02	94.14	87.84	1.07	44.69	39.05	1.14
parse_spec	66.03	58.59	1.13	60.12	54.88	1.10	26.31	26.35	1.00
keys	33.67	33.65	1.00	27.97	27.56	1.01	2.99	2.87	1.04
glob	362.49	37.42	9.69	335.48	36.61	9.16	134.14	66.79	2.01
iso_char	8.20	7.90	1.04	8.65	11.70	0.74	16.83	15.64	1.08
conc_aggregates	4.93	5.82	0.85	5.56	5.87	0.95	6.58	6.76	0.97
system_extra	13835.06	12712.88	1.09	9838.64	8227.61	1.20	1292.18	1328.85	0.97
hiordlib	timeout	timeout	-	timeout	timeout	-	316.70	311.69	1.02
opendoc	8.79	8.99	0.98	9.29	9.62	0.97	13.29	13.56	0.98
logged_process	24899.57	165.57	150.39	24208.86	128.20	188.84	71.20	19.48	3.66
numlists	5.21	5.43	0.96	5.73	6.00	0.96	7.54	8.05	0.94
use_url	11.08	9.66	1.15	11.13	9.23	1.21	7.83	7.65	1.02
sh_process	1.12	1.15	0.98	1.47	1.33	1.10	1.47	1.45	1.01
terms_vars	30.79	36.89	0.83	42.90	48.12	0.89	34.75	36.12	0.96
datetime	timeout	742.76	-	timeout	886.23	-	1127.60	1107.31	1.02
bitcodesets	456.65	472.57	0.97	447.47	457.93	0.98	21.02	22.64	0.93
vndict	2349.65	1814.99	1.29	2609.38	994.63	2.62	71.54	72.84	0.98
functional	0.43	0.33	1.31	0.54	0.51	1.06	0.12	0.11	1.11
arrays	77.84	77.30	1.01	81.82	77.51	1.06	25.87	28.00	0.92
getopts	50.93	53.58	0.95	32.93	34.85	0.94	44.78	75.98	0.59
parse_shell_args	7.89	8.17	0.97	7.30	7.08	1.03	11.42	10.17	1.12
mutables	2.40	2.71	0.89	3.11	2.80	1.11	4.86	4.47	1.09
clp	0.23	0.18	1.31	0.25	0.01	22.45	0.27	0.16	1.62
version_strings.test	1.96	1.60	1.22	2.23	1.77	1.26	3.35	2.86	1.17
mtarjan	34.94	19.57	1.79	46.30	20.72	2.23	26.81	25.94	1.03
diff	8384.49	3232.31	2.59	6005.52	1511.38	3.97	38.91	43.94	0.89
loops	0.84	0.61	1.36	1.21	0.73	1.66	1.15	0.80	1.43
iso_misc	9.55	9.83	0.97	11.10	11.51	0.96	14.89	10.56	1.41
random_aggregates	5.57	4.18	1.33	4.54	4.71	0.96	5.42	6.29	0.86
global_vars	1.98	1.95	1.02	2.28	1.97	1.15	3.89	2.76	1.41
argnames_doc	0.52	0.36	1.42	0.30	0.22	1.35	0.65	0.28	2.31
argnames_trans	796.59	695.07	1.15	329.08	309.71	1.06	154.17	137.73	1.12
argnames	0.40	0.42	0.96	0.45	0.11	4.05	0.14	0.30	0.49
zebra_argnames	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-
simple_db	4.34	4.03	1.08	4.72	5.57	0.85	2.64	2.73	0.97

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
argnames_test	3.05	2.42	1.26	2.56	2.50	1.02	3.51	2.77	1.27
md5sum	2.75	2.86	0.96	3.31	3.24	1.02	5.78	6.05	0.96
andorra_builtins	392.83	401.55	0.98	385.20	397.91	0.97	26.66	34.82	0.77
andorraops	0.42	0.26	1.61	0.37	0.29	1.25	0.42	0.28	1.51
andorra_tr	timeout	13582.33	-	timeout	5899.48	-	179.42	187.86	0.96
andorra	0.39	0.27	1.43	0.58	0.32	1.81	0.45	0.28	1.63
andorra_rt	310.01	113.39	2.73	306.28	109.19	2.81	60.77	56.92	1.07
andorra_doc	0.57	0.55	1.03	0.63	0.55	1.15	0.69	0.48	1.43
andorra_builtins_exports	1.57	1.64	0.96	1.50	1.54	0.98	1.74	1.54	1.13
map_and	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-
mqu_and	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-
crypt_and	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-
map_and_user	timeout	36475.42	-	timeout	29992.22	-	timeout	56.22	-
map	3.56	3.81	0.93	4.21	4.10	1.03	5.33	5.93	0.90
money_and	timeout	timeout	-	timeout	timeout	-	timeout	30587.79	-
mqu	6.52	11.00	0.59	8.87	8.68	1.02	12.32	13.29	0.93
crypt	7.13	7.42	0.96	7.65	8.71	0.88	15.78	16.52	0.95
mqu_and_user	timeout	timeout	-	timeout	timeout	-	timeout	287.15	-
money_and_user	timeout	timeout	-	timeout	timeout	-	timeout	28571.91	-
money	timeout	timeout	-	timeout	timeout	-	timeout	30225.02	-
crypt_and_user	timeout	timeout	-	timeout	timeout	-	4546.50	4314.11	1.05
qu_vitor	3.46	3.17	1.09	3.85	3.50	1.10	3.84	3.94	0.97
qu_evan	2.96	2.99	0.99	3.92	3.67	1.07	4.75	4.46	1.06
fib_and1p1	62004.11	154.42	401.54	57838.01	124.60	464.20	19.35	19.13	1.01
ejem1	timeout	25072.54	-	timeout	23125.39	-	91.82	94.83	0.97
mutest_and1p1	6580.54	6126.35	1.07	7989.77	6115.32	1.31	1089.27	1076.18	1.01
map_and	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-
mqu_and1p1	timeout	timeout	-	timeout	timeout	-	5495.03	5281.44	1.04
dia_sums_and_user	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-
fib_and	timeout	9111.44	-	timeout	8555.17	-	25.69	26.88	0.96
foo	0.84	0.46	1.84	0.97	0.77	1.26	0.74	0.65	1.14
mqu_and	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-
crypt_and	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-
crypt_and1p1	timeout	27421.26	-	timeout	11434.72	-	824.83	405.33	2.03
map_and_user	timeout	35234.97	-	timeout	30715.34	-	timeout	50.83	-

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
dia_sums	2.99	8.08	0.37	4.15	7.99	0.52	5.73	8.52	0.67
qu_vitor_and_user	timeout	10752.35	-	timeout	9981.60	-	150.32	155.43	0.97
mutest_and_user	timeout	timeout	-	timeout	timeout	-	1660.39	1835.75	0.90
money_and1p1	timeout	timeout	-	timeout	140205.56	-	timeout	232.55	-
map	3.54	3.81	0.93	3.78	4.15	0.91	5.89	5.69	1.03
qu_vitor_and	timeout	timeout	-	timeout	timeout	-	7573.45	7758.25	0.98
qu_evan_and	timeout	timeout	-	timeout	timeout	-	3240.71	3530.90	0.92
money_and	timeout	timeout	-	timeout	timeout	-	timeout	29215.79	-
mutest	5.08	5.34	0.95	5.62	5.78	0.97	7.59	7.38	1.03
mutest_and	timeout	timeout	-	timeout	timeout	-	1681.39	1842.46	0.91
dia_sums_and1p1	timeout	timeout	-	timeout	timeout	-	527.25	528.17	1.00
mqu	10.84	10.84	1.00	9.07	10.20	0.89	12.12	11.72	1.03
crypt	6.35	7.91	0.80	6.05	8.43	0.72	15.97	12.75	1.25
fib	1.02	0.99	1.03	1.00	1.08	0.92	1.24	1.27	0.98
fib_and_user	timeout	946.24	-	timeout	759.66	-	11.92	12.42	0.96
dia_sums_and	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-
mqu_and_user	timeout	timeout	-	timeout	timeout	-	timeout	286.96	-
defineq	106.50	106.66	1.00	103.66	103.78	1.00	6.25	6.18	1.01
qu_evan_and1p1	timeout	6935.39	-	timeout	6873.94	-	439.93	451.79	0.97
money_and_user	timeout	timeout	-	timeout	timeout	-	timeout	29961.15	-
qu_vitor_and1p1	timeout	687.21	-	timeout	644.94	-	49.96	52.01	0.96
money	3.84	5.35	0.72	4.85	6.28	0.77	9.37	9.81	0.96
map_and1p1	77328.44	1835.06	42.14	71241.36	1424.91	50.00	32.75	34.17	0.96
crypt_and_user	timeout	timeout	-	timeout	timeout	-	4723.34	4545.38	1.04
bench2	0.27	0.18	1.51	0.27	0.21	1.30	0.41	0.19	2.10
qu_evan_and_user	timeout	17729.40	-	timeout	16623.39	-	104.41	111.77	0.93
qe_andorra_lib	13.03	9.38	1.39	12.75	12.82	0.99	12.29	11.03	1.11
visandor	2.13	1.96	1.09	2.85	2.17	1.32	2.53	2.55	0.99
apll	14.73	14.34	1.03	15.86	15.32	1.04	19.55	18.88	1.04
backtr_join	85.92	90.66	0.95	86.86	90.81	0.96	85.91	82.72	1.04
layout_dcg	0.32	0.35	0.93	0.42	0.30	1.40	0.65	0.30	2.16
layout_dcg_rt	18.96	21.55	0.88	22.33	25.25	0.88	22.46	23.07	0.97
runtime_ops_doc	0.21	0.16	1.27	0.38	0.01	29.38	0.26	0.21	1.25
runtime_ops	0.62	0.25	2.45	0.36	0.31	1.17	0.35	0.32	1.11
runtime_ops_tr	0.85	0.77	1.09	1.12	0.69	1.63	0.78	0.67	1.16

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
regexp_code	1969.40	1837.79	1.07	2020.86	1686.20	1.20	1307.30	1157.49	1.13
regexp_doc	0.23	0.18	1.29	0.19	0.17	1.10	0.22	0.16	1.40
regexp_flags	0.75	0.55	1.37	0.88	0.67	1.30	1.43	1.05	1.36
regexp_trans	15.69	14.99	1.05	16.86	16.42	1.03	16.11	14.12	1.14
regexp	0.53	0.27	1.94	0.33	0.29	1.15	0.51	0.30	1.73
classic_predicates	15.27	18.88	0.81	15.78	15.02	1.05	21.51	13.03	1.65
classic_doc	0.31	0.33	0.96	0.59	0.27	2.19	0.67	0.25	2.66
classic	0.38	0.31	1.25	0.47	0.30	1.58	0.46	0.25	1.83
service_registry	9.71	9.99	0.97	10.82	11.16	0.97	14.68	14.83	0.99
serve_http	11.65	10.70	1.09	12.88	11.85	1.09	12.19	11.73	1.04
service_loader	33.08	33.46	0.99	33.81	32.33	1.05	12.55	11.47	1.09
actmod_ctl	32.33	30.34	1.07	31.38	31.22	1.01	7.24	2.68	2.70
expander_tr	48.64	21.81	2.23	58.99	25.17	2.34	23.28	27.40	0.85
mio	0.59	0.50	1.17	0.66	0.61	1.09	1.08	0.70	1.54
det_hook_tr	1.16	1.11	1.05	1.35	1.28	1.06	1.48	0.97	1.53
det_hook_rt	2.85	2.73	1.05	0.61	0.61	1.00	4.16	3.41	1.22
det_hook_doc	0.22	0.17	1.35	0.17	0.16	1.05	0.44	0.19	2.32
det_hook	0.34	0.26	1.32	0.33	0.27	1.23	0.41	0.30	1.35
example	1.01	4.90	0.21	6.02	6.74	0.89	7.55	6.95	1.09
enumerate	0.30	1.37	0.22	2.09	1.72	1.21	2.39	0.23	10.61
example2	3.38	4.21	0.80	3.80	4.61	0.82	6.34	6.18	1.03
error_templates	1.49	1.74	0.86	1.61	1.26	1.27	1.95	1.89	1.03
ops	0.33	0.28	1.18	0.48	0.24	2.01	0.32	0.57	0.56
color_space	12.37	13.50	0.92	12.54	13.72	0.91	23.70	23.84	0.99
json	40.48	49.59	0.82	47.35	52.14	0.91	59.64	64.43	0.93
pillow	0.35	0.34	1.03	0.30	0.37	0.80	0.31	0.24	1.28
pillow_doc	0.46	0.18	2.49	0.35	0.18	1.97	0.46	0.17	2.75
html	907.83	844.37	1.08	1406.85	824.95	1.71	timeout	817.05	-
xmlterm	9.60	10.28	0.93	6.83	6.83	1.00	5.48	5.55	0.99
html_demo	0.47	0.26	1.84	0.22	0.00	2.20	0.49	0.36	1.35
cookies	4.18	4.69	0.89	4.51	4.66	0.97	8.70	7.90	1.10
phones	0.78	0.54	1.46	0.79	0.55	1.43	0.77	0.56	1.37
check_links	9.84	10.32	0.95	12.68	12.36	1.03	15.71	14.94	1.05
html_forms	1.31	1.11	1.18	1.38	1.12	1.23	1.33	0.95	1.39
pillow.hooks	0.29	0.24	1.23	0.34	0.18	1.93	0.24	0.18	1.31

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
Make_aux	0.35	0.29	1.21	0.26	0.25	1.01	0.58	0.25	2.29
adds_eclipse	2.39	2.11	1.13	2.02	2.02	1.00	2.11	1.97	1.07
adds_yap	2.15	2.23	0.96	2.18	2.04	1.07	2.29	2.21	1.04
adds_sicstus	2.20	1.85	1.19	2.26	1.96	1.15	2.31	2.18	1.06
icon_address	0.49	0.27	1.86	0.42	0.34	1.25	0.66	0.46	1.44
ops_eclipse	0.19	0.22	0.89	0.25	0.16	1.59	0.20	0.17	1.18
SETTINGS	4.36	4.10	1.06	4.75	4.29	1.11	5.74	5.81	0.99
freeze__old	0.32	0.16	1.96	0.42	0.18	2.35	0.23	0.16	1.37
freeze	4.28	3.60	1.19	4.53	4.15	1.09	12.83	10.68	1.20
sendmoney_freeze	23573.34	349.44	67.46	25560.43	307.94	83.00	3763.67	44.27	85.03
sort_freeze	9.63	10.85	0.89	12.15	12.96	0.94	18.39	20.55	0.89
queens_freeze	12.74	11.49	1.11	14.97	14.08	1.06	14.91	15.25	0.98
nreverse_freeze	8.89	10.92	0.81	11.69	12.03	0.97	14.20	14.11	1.01
actmod_doc	0.39	0.29	1.34	0.41	0.26	1.57	0.37	0.29	1.27
actmod_rt	210.10	168.20	1.25	207.02	146.46	1.41	101.21	103.80	0.98
actmod_process	46.15	42.09	1.10	43.68	45.90	0.95	26.99	25.63	1.05
actmod_hooks	27.52	26.39	1.04	27.22	26.04	1.05	1.72	1.93	0.89
regp_platformserver	29.07	28.25	1.03	29.46	28.88	1.02	3.76	3.70	1.02
actmod_dist	89.69	71.41	1.26	84.57	69.49	1.22	51.83	60.08	0.86
rundaemon	0.68	0.61	1.12	0.79	0.70	1.14	0.80	0.62	1.29
filebased_common	6.68	7.15	0.93	7.08	7.46	0.95	8.47	7.05	1.20
actmod_holder	0.96	0.74	1.31	1.03	0.73	1.41	1.15	0.74	1.54
regp_filebased	36.73	28.85	1.27	34.06	31.60	1.08	7.74	6.48	1.19
regp_webserver	29.64	29.21	1.01	29.02	28.00	1.04	1.32	4.15	0.32
actmod	29.64	24.07	1.23	28.32	27.37	1.03	2.97	2.82	1.05
regp_webbased	30.73	31.83	0.97	31.56	30.07	1.05	1.84	5.84	0.32
actmod_tr	102.93	71.36	1.44	97.44	70.93	1.37	37.39	30.12	1.24
webbased_common	3.80	3.27	1.16	4.14	4.00	1.04	5.25	8.90	0.59
regp_platformbased	31.68	30.95	1.02	31.26	31.64	0.99	7.49	6.55	1.14
simple_server	28.85	29.78	0.97	28.81	29.23	0.99	4.37	3.62	1.21
simple_client	29.84	29.15	1.02	30.34	30.18	1.01	5.30	4.98	1.07
simple_client_with_main	31.78	31.58	1.01	32.26	31.43	1.03	7.00	6.88	1.02
webbased_simple_client	29.49	29.86	0.99	29.18	30.10	0.97	0.52	4.99	0.11
send	29.14	28.19	1.03	30.30	27.72	1.09	3.47	3.28	1.06
agent2	0.29	1.82	0.16	2.19	2.12	1.03	2.18	2.01	1.08

	share			shfr			shfr-clique		
module	**TC**	**TT**	ρ**T**	**TC**	**TT**	ρ**T**	**TC**	**TT**	ρ**T**
agent1	2.11	2.19	0.96	2.11	1.86	1.13	2.09	1.96	1.07
test	29.73	28.66	1.04	28.29	27.69	1.02	3.41	3.27	1.04
nameserver	31.64	31.22	1.01	31.49	28.00	1.12	5.35	5.04	1.06
tmpbased_locate	27.37	27.92	0.98	27.91	28.05	0.99	3.01	2.10	1.43
SETTINGS_actmod	4.77	4.35	1.10	5.26	6.33	0.83	5.95	6.38	0.93
guess_current_exec	0.61	0.51	1.20	0.50	0.50	1.00	0.70	0.40	1.75
tmpbased_publish	24.34	23.18	1.05	23.29	19.76	1.18	3.00	2.09	1.44
emacs_batch	4.79	5.30	0.90	5.70	5.91	0.96	10.97	12.23	0.90
emacs	6.28	6.17	1.02	6.68	3.28	2.04	8.44	8.91	0.95
commons_compat_tr	0.36	0.29	1.21	0.22	0.17	1.30	0.17	0.17	0.97
yap_compat_tr	146.26	113.96	1.28	93.44	89.17	1.05	16.93	65.74	0.26
yap_compat_ops	0.42	0.43	0.98	0.11	0.32	0.34	0.30	0.28	1.07
commons	0.22	0.19	1.18	0.24	0.19	1.30	0.18	0.20	0.92
hprolog	0.39	0.16	2.39	0.29	0.16	1.81	0.67	0.17	3.87
yap_compat	0.41	0.31	1.34	0.32	0.32	1.01	0.42	0.34	1.24
hprolog_compat	0.58	0.29	2.04	0.56	0.32	1.78	0.51	0.25	2.06
yap	0.24	0.19	1.28	0.25	0.17	1.51	0.42	0.03	13.61
commons_compat_ops	0.42	0.28	1.50	0.42	0.35	1.21	0.48	0.26	1.85
commons_modes	0.25	0.17	1.45	0.28	0.17	1.60	0.42	0.20	2.13
commons_compat	0.39	0.32	1.21	0.49	0.28	1.76	0.39	0.10	3.77
ciao_server_rt	4.48	4.26	1.05	4.66	1.05	4.45	5.97	1.84	3.24
ops	0.43	0.29	1.51	0.38	0.33	1.14	0.67	0.33	2.04
ciao_server	8.14	8.09	1.01	10.06	9.54	1.06	6.84	11.82	0.58
socket_info	0.67	0.34	1.98	0.44	0.41	1.08	0.56	0.81	0.69
remote_doc	0.77	0.69	1.11	0.97	0.76	1.27	0.87	0.61	1.43
ciao_client	4.03	3.90	1.03	4.73	4.67	1.01	9.49	5.55	1.71
ciao_client_rt	2.55	3.15	0.81	2.83	2.95	0.96	3.22	3.25	0.99
read_and_write	3.90	0.74	5.27	3.53	3.91	0.90	5.74	4.05	1.42
queens_with_delays	2.59	3.19	0.81	2.76	3.09	0.89	3.96	3.63	1.09
simpleclp	0.92	0.74	1.25	0.65	0.72	0.90	0.91	0.58	1.58
queens	2.60	2.51	1.04	0.46	2.50	0.18	3.30	2.99	1.10
queensclp	28.26	11.58	2.44	29.18	11.75	2.48	6.93	7.22	0.96
ciao_server.bak	3.32	3.47	0.95	4.25	3.72	1.14	5.11	4.40	1.16
ciao_client.works	3.55	3.51	1.01	4.00	3.81	1.05	4.32	3.82	1.13
ciao_server.works	4.97	6.56	0.76	5.96	5.10	1.17	1.63	5.26	0.31

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
ciao_server.bak.old	4.38	4.76	0.92	1.47	1.15	1.28	5.32	4.91	1.08
ciao_client.keep_stream	3.46	3.71	0.93	4.57	3.61	1.27	5.24	4.38	1.20
ciao_client.bak	3.39	3.12	1.09	0.83	3.27	0.25	3.84	3.25	1.18
ciao_server.keep_stream	3.69	4.13	0.89	3.89	3.69	1.05	4.84	4.68	1.03
ciao_client.bak.old	4.72	3.12	1.52	3.33	3.33	1.00	4.96	3.90	1.27
unif_constr	5.08	4.98	1.02	5.46	1.62	3.38	8.02	6.52	1.23
measure_size	6.11	6.56	0.93	7.66	7.25	1.06	8.32	7.63	1.09
xrefs	1.83	2.21	0.83	2.27	2.23	1.02	2.43	2.33	1.04
xrefsread	1050.11	197.54	5.32	1061.41	194.96	5.44	158.83	95.42	1.66
xrefsbuild	201.00	153.77	1.31	142.78	112.45	1.27	29.07	33.69	0.86
callgraph	14.90	15.10	0.99	28.20	30.79	0.92	25.94	26.14	0.99
xrefs2graph	321.90	335.25	0.96	299.69	308.59	0.97	9.73	8.65	1.12
pxrefs	3.34	3.11	1.07	3.35	3.35	1.00	4.28	4.09	1.05
mrefs	1.80	1.67	1.07	2.17	1.97	1.10	2.93	2.75	1.06
xrefs_doc	0.42	0.27	1.54	0.44	0.19	2.35	0.00	0.19	0.52
SETTINGS	4.51	3.97	1.14	4.40	4.37	1.01	5.53	5.40	1.02
symfnames	2.32	1.90	1.22	2.17	2.31	0.94	2.78	2.58	1.08
myfiles	0.38	0.00	3.81	0.40	0.37	1.07	0.31	0.31	1.01
main	0.67	0.40	1.67	0.26	0.40	0.64	0.78	0.76	1.02
mm	0.21	0.87	0.24	1.23	1.16	1.06	1.40	1.12	1.25
stream_wait	2.85	2.63	1.08	3.96	3.00	1.32	4.50	3.87	1.16
file_locks	1.76	1.27	1.38	1.84	1.56	1.18	4.01	3.03	1.32
http_forms	71.02	44.51	1.60	77.99	48.93	1.59	73.67	77.78	0.95
http_date	1925.16	291.76	6.60	1754.56	234.96	7.47	175.86	195.74	0.90
http_service	1.76	1.58	1.11	2.09	1.76	1.19	3.29	2.52	1.30
http_service_rt	3.14	3.08	1.02	3.83	3.67	1.04	5.79	8.16	0.71
multipart_form_data	204.16	61.88	3.30	193.94	50.36	3.85	31.52	26.59	1.19
http_client	9.51	7.74	1.23	10.76	8.33	1.29	12.80	13.13	0.97
mimetypes	1.55	1.69	0.92	1.91	1.69	1.13	2.91	2.69	1.09
http_server_hooks	0.79	0.45	1.77	0.51	0.55	0.94	1.07	0.46	2.33
http_messages	5740.38	36.38	157.78	5375.75	40.64	132.26	25.81	31.01	0.83
http_server	7472.34	938.63	7.96	12446.83	1080.47	11.52	1309.46	1256.63	1.04
cgi	46.57	23.73	1.96	18.50	16.03	1.15	19.75	20.41	0.97
url	13.57	18.61	0.73	17.86	18.72	0.95	24.42	26.90	0.91
http_grammar	79.24	53.26	1.49	83.53	58.26	1.43	56.77	49.11	1.16

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
http_doc	0.20	0.21	0.93	0.25	0.17	1.47	0.23	0.01	21.36
multipart_from_stream	3.84	3.84	1.00	0.50	3.58	0.14	3.68	3.66	1.01
af	0.45	0.38	1.21	0.59	0.28	2.12	0.40	0.35	1.16
ops	0.63	0.53	1.19	0.43	0.50	0.87	0.28	0.26	1.07
sr	0.36	0.31	1.18	0.49	0.27	1.82	0.55	0.45	1.24
sr_tr	13.92	14.26	0.98	15.15	15.29	0.99	15.20	14.12	1.08
bf	0.42	0.29	1.45	0.34	0.31	1.08	0.46	0.31	1.49
bfall	0.75	0.57	1.33	0.44	0.55	0.80	0.55	0.47	1.17
bf_rt	0.77	0.72	1.07	0.70	0.64	1.09	0.72	0.51	1.40
af_rt	0.73	0.65	1.13	0.79	0.65	1.21	0.68	0.60	1.15
bf_doc	0.21	0.17	1.19	0.22	0.20	1.11	0.20	0.17	1.21
afall	0.79	0.64	1.23	0.64	0.60	1.07	0.76	0.61	1.25
chain	25.97	29.38	0.88	31.30	27.71	1.13	24.04	25.43	0.95
chain_bfall	35.42	36.91	0.96	37.65	37.36	1.01	24.26	25.15	0.96
sublistapp	66.57	65.71	1.01	59.68	60.52	0.99	5609.79	5365.11	1.05
toplevel_proc	18.30	12.13	1.51	19.80	13.53	1.46	23.99	21.25	1.13
ugraphs	1836.88	1774.00	1.04	1510.08	1498.18	1.01	70.35	77.99	0.90
graphs	13.84	13.15	1.05	6.45	5.77	1.12	7.52	7.28	1.03
lgraphs	27.29	27.53	0.99	24.53	25.04	0.98	8.31	9.10	0.91
modblobs	6.43	9.88	0.65	11.43	10.26	1.11	15.80	13.00	1.22
vcs_git	1.73	1.72	1.01	1.87	1.63	1.15	2.74	2.62	1.05
vcs_svn	6.89	4.81	1.43	7.91	2.81	2.81	23.11	16.06	1.44
subversion	0.19	0.17	1.12	0.19	0.16	1.14	0.18	0.16	1.12
soap	24.70	17.17	1.44	22.76	16.63	1.37	17.27	10.66	1.62
toplevel	0.37	0.18	2.10	0.35	0.20	1.72	0.30	0.26	1.13
example	0.92	0.57	1.62	0.66	0.54	1.21	0.78	0.69	1.12
soap_server	88.31	89.48	0.99	88.23	91.85	0.96	6.53	6.17	1.06
server	0.65	0.66	0.98	0.76	0.65	1.17	0.77	0.64	1.21
client	0.83	0.63	1.30	0.85	0.54	1.57	0.79	0.68	1.15
soap_client	1.63	1.05	1.55	1.52	1.22	1.25	2.06	1.43	1.45
server	1.58	1.49	1.06	1.92	1.56	1.22	2.06	1.62	1.27
client	0.31	0.23	1.36	0.37	0.21	1.77	0.29	0.19	1.51
block_tr	39.02	40.18	0.97	35.83	166.55	0.22	16.00	24.48	0.65
block_rt	6.17	6.50	0.95	7.26	7.17	1.01	6.77	6.59	1.03
block	0.57	0.29	2.00	0.40	0.27	1.50	0.54	0.32	1.68

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
block_doc	0.33	0.17	1.88	0.33	0.18	1.78	0.45	0.16	2.78
block_example	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-
http_get	2.18	2.01	1.09	2.82	2.29	1.23	2.91	2.86	1.02
factsdb	0.21	0.38	0.56	0.54	0.40	1.36	0.79	0.39	2.01
factsdb_tr	1.59	1.27	1.26	1.61	1.33	1.21	1.65	1.98	0.84
factsdb_rt	14.15	12.90	1.10	15.98	13.88	1.15	11.73	12.32	0.95
factsdb_doc	0.33	0.26	1.27	0.51	0.52	0.99	0.61	0.27	2.28
dfib	0.34	0.62	0.56	0.89	0.86	1.03	0.77	0.92	0.84
fibdb	0.44	0.31	1.45	0.45	0.34	1.31	0.55	0.32	1.71
fib	1.39	1.68	0.82	1.40	1.25	1.12	1.21	0.96	1.26
ffib	0.47	0.40	1.16	0.64	0.57	1.12	0.56	0.39	1.42
fib0_2	2.16	2.17	1.00	2.30	2.11	1.09	2.54	2.10	1.21
fib0_2_ops	0.60	0.38	1.60	0.41	0.29	1.40	0.42	0.43	0.98
p	0.39	0.50	0.77	0.49	0.33	1.45	0.67	0.36	1.85
cache_rt	1.07	0.97	1.10	1.20	1.13	1.06	1.41	1.23	1.15
cache_doc	0.30	0.22	1.35	0.39	0.24	1.61	0.31	0.27	1.16
cache_tr	1.34	1.05	1.28	1.14	1.66	0.68	1.68	1.34	1.25
cache_rt	4.79	5.42	0.88	5.48	5.90	0.93	5.51	6.11	0.90
cache_tr	2.38	2.64	0.90	2.19	2.46	0.89	2.60	3.07	0.85
cache_rt2	3.52	3.74	0.94	4.24	4.19	1.01	4.87	4.32	1.13
functionstr	105919.64	48199.34	2.20	85682.33	34515.50	2.48	606.11	598.50	1.01
ops	0.48	0.38	1.24	0.45	0.31	1.45	0.52	0.27	1.94
fsyntax	0.34	0.34	1.00	0.42	0.29	1.44	0.30	0.35	0.86
functional_basics	1.12	1.05	1.07	1.56	1.06	1.48	1.14	1.43	0.80
fsyntax_doc	0.32	0.19	1.73	0.26	0.16	1.58	0.39	0.19	1.99
functional_defs	0.21	0.20	1.04	0.23	0.23	0.97	0.20	0.17	1.20
factf_fsyntax	1.12	0.75	1.48	1.22	0.87	1.40	1.22	1.35	0.90
lazyex	6.24	6.25	1.00	6.99	6.65	1.05	6.43	2.03	3.17
revf	3.19	2.76	1.15	3.74	3.61	1.04	3.84	3.64	1.05
lazy_simpler	3.55	4.32	0.82	4.20	4.61	0.91	4.97	2.22	2.24
fib_fun	1.37	1.61	0.85	1.95	1.91	1.02	2.32	2.27	1.02
factf_assrt	0.95	0.84	1.12	1.32	1.06	1.24	1.47	1.06	1.38
factf_arithclpq_assrt	1.79	1.54	1.16	1.96	1.65	1.19	2.61	1.88	1.39
revf_assrt_new	5.05	4.65	1.09	6.67	6.13	1.09	17.14	16.18	1.06
arrays_ops	0.19	0.19	1.00	0.38	0.24	1.60	0.20	0.16	1.22

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
arrays_test	2.14	1.78	1.20	2.67	2.12	1.26	6.14	5.70	1.08
open_file_f_wide	4.28	4.14	1.03	4.74	4.52	1.05	7.71	7.99	0.96
hiordfun	timeout	4.04	-	timeout	4.44	-	76221.49	3.53	21580.26
deriv	3.08	3.01	1.02	3.08	3.10	0.99	3.93	3.46	1.14
arrays_rt	13.00	13.74	0.95	15.60	15.95	0.98	28.79	26.56	1.08
arrays	0.53	0.27	1.95	0.47	0.36	1.31	0.48	0.36	1.32
append	0.95	0.72	1.32	0.87	0.77	1.12	1.04	0.79	1.32
clpqf	0.26	0.16	1.60	0.23	0.17	1.36	0.25	0.41	0.61
lazy_simple	2.00	1.80	1.11	2.55	2.34	1.09	2.80	2.69	1.04
factf	0.81	0.83	0.97	1.02	1.03	1.00	1.01	1.38	0.73
factf_clpq_assrt	1.65	1.69	0.98	1.78	1.70	1.05	2.00	2.18	0.92
revf_assrt_bug	3.97	3.62	1.10	4.46	4.55	0.98	0.88	4.54	0.19
propsf	21.02	23.79	0.88	24.61	20.64	1.19	18.00	18.89	0.95
max	2.82	2.65	1.06	3.22	2.96	1.09	3.59	4.06	0.88
when__old	0.36	0.16	2.23	0.19	0.20	0.97	0.19	0.17	1.11
when	47307.62	178.32	265.29	12748.15	58.20	219.04	2833.53	26.68	106.22
bench	1.41	1.17	1.20	1.43	1.24	1.15	1.55	1.38	1.12
basic_stat	14.30	6.39	2.24	4.40	5.07	0.87	6.95	7.48	0.93
test	1.60	1.51	1.06	1.54	1.31	1.18	1.45	1.37	1.06
librowser	26.39	30.01	0.88	21.79	18.43	1.18	37.24	39.09	0.95
syntax_highlight	27.75	15.60	1.78	29.71	16.72	1.78	31.14	29.77	1.05
sockets_io	1500.56	1113.05	1.35	1235.59	1140.49	1.08	15.63	15.25	1.02
sockets	7.96	7.57	1.05	8.79	8.59	1.02	12.95	7.32	1.77
server	4.16	3.96	1.05	4.75	4.80	0.99	3.46	6.17	0.56
client	4.57	5.32	0.86	5.42	6.36	0.85	8.52	8.36	1.02
socket_number	0.71	0.32	2.18	0.54	0.33	1.62	1.01	0.59	1.72
clpr	0.60	0.37	1.60	0.62	0.41	1.52	0.53	0.50	1.06
clpr_src	0.42	0.38	1.12	0.13	0.31	0.42	0.51	0.50	1.01
clpr_compiler	397.32	278.12	1.43	362.15	261.13	1.39	31.07	33.17	0.94
clpr_meta	6.53	6.06	1.08	7.42	7.05	1.05	6.78	7.07	0.96
clpr_dump	timeout	32670.82	-	timeout	19900.37	-	382.87	333.18	1.15
expand_r	5.72	6.44	0.89	4.78	7.62	0.63	4.19	3.83	1.09
solver_r	timeout	timeout	-	timeout	timeout	-	timeout	5750.69	-
clpr_doc	0.22	0.17	1.30	0.36	0.16	2.30	0.47	0.23	2.07
clpr_rt	timeout	timeout	-	timeout	timeout	-	36.77	35.72	1.03

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
clpr_tr	3.88	3.55	1.09	4.26	3.88	1.10	3.17	3.18	1.00
clpr_attr	0.89	0.69	1.30	0.66	0.67	0.99	0.80	0.72	1.12
eval_r	4.87	5.14	0.95	5.44	5.96	0.91	8.01	8.40	0.95
clprtests	2.47	2.53	0.98	2.67	3.32	0.81	4.14	3.44	1.20
iso_strict_doc	0.23	0.19	1.24	0.40	0.17	2.34	0.25	0.35	0.71
iso_strict	0.53	0.26	2.00	0.38	0.26	1.47	0.43	0.48	0.89
fuzzy_search	147.77	125.89	1.17	116.99	88.75	1.32	38.81	38.86	1.00
sha1	0.62	0.33	1.87	0.64	0.53	1.20	0.80	0.90	0.89
example	0.00	0.34	0.30	0.39	0.31	1.24	0.51	0.37	1.36
ctchecks	0.72	0.24	2.98	0.34	0.29	1.16	0.33	0.39	0.83
foo	0.56	0.43	1.29	0.74	0.64	1.16	0.59	0.48	1.24
lazy_doc	0.36	0.18	1.97	0.26	0.16	1.61	0.53	0.18	2.98
lazy_lib	759.52	253.75	2.99	792.30	256.44	3.09	38.69	38.39	1.01
ops	0.29	0.29	0.98	0.40	0.28	1.45	0.46	0.29	1.58
lazy	0.53	0.25	2.12	0.14	0.27	0.52	0.46	0.31	1.52
lazytr	137.72	78.08	1.76	128.80	70.31	1.83	13.10	12.46	1.05
primes_function	7.84	8.28	0.95	8.19	8.62	0.95	8.40	8.06	1.04
module1	0.90	0.97	0.93	1.90	1.61	1.18	2.09	1.89	1.11
module2	2.43	2.04	1.19	1.17	1.00	1.18	1.32	1.10	1.20
ex	14.83	10.99	1.35	17.09	16.42	1.04	11.87	11.03	1.08
tetrahedral_function	1.81	1.90	0.96	2.46	2.19	1.12	2.85	2.73	1.04
naturals_function	2.24	2.21	1.02	3.01	2.40	1.26	3.56	3.90	0.91
naturals	3.15	2.83	1.11	3.88	3.22	1.21	4.61	3.85	1.20
fib	6.44	6.67	0.96	8.04	7.87	1.02	5.60	1.29	4.36
fib_function	18.11	19.68	0.92	18.82	18.12	1.04	16.54	19.13	0.86
lqsort_function	13.03	14.10	0.92	14.78	16.27	0.91	9.61	10.56	0.91
primes	6.11	5.71	1.07	4.89	8.07	0.61	6.03	6.66	0.91
tetrahedral	3.20	2.93	1.09	3.11	3.83	0.81	4.07	3.99	1.02
ex_function	7.82	8.04	0.97	6.65	6.65	1.00	9.26	7.91	1.17
lqsort	6.94	7.16	0.97	7.92	6.97	1.14	15.52	17.12	0.91
foreign_cxx_ops	0.29	0.25	1.17	0.45	0.45	1.01	0.40	0.37	1.07
foreign_cxx_tr	181.62	112.50	1.61	173.89	63.32	2.75	2004.26	111.24	18.02
foreign_cxx	0.41	0.26	1.58	0.58	0.26	2.26	0.33	0.00	3.29
timeout	3.42	3.28	1.04	1.29	3.54	0.36	4.23	4.55	0.93
test_timeout	16.05	16.92	0.95	17.47	19.50	0.90	23.11	24.06	0.96

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
random	1.02	0.87	1.17	1.34	1.12	1.19	2.47	1.85	1.33
clpq_tr	3.43	3.54	0.97	3.74	4.36	0.86	3.31	3.52	0.94
clpq_compiler	394.87	279.07	1.41	368.49	256.70	1.44	37.12	33.62	1.10
eval_q	131.30	134.14	0.98	140.81	147.67	0.95	1368.17	1428.44	0.96
clpq_attr	0.87	0.77	1.13	0.88	0.77	1.14	0.93	0.90	1.03
clpq_meta	6.28	6.63	0.95	7.16	7.58	0.94	6.67	6.70	1.00
clpq_dump	timeout	32972.12	-	timeout	20195.81	-	394.05	327.52	1.20
clpq_rt	timeout	timeout	-	timeout	timeout	-	38.53	37.49	1.03
solver_q	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-
expand_q	4.71	3.97	1.19	5.11	4.70	1.09	4.99	4.16	1.20
clpq_src	0.43	0.25	1.69	0.38	0.28	1.34	0.31	0.37	0.84
clpq	0.51	0.35	1.46	0.59	0.47	1.26	0.46	0.36	1.27
clpq_doc	0.18	0.18	1.03	0.34	0.18	1.89	0.29	0.25	1.16
id	0.48	0.35	1.37	0.66	0.44	1.50	0.42	0.27	1.57
id_doc	0.21	0.16	1.31	0.17	0.19	0.92	0.24	0.23	1.06
id_tr	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-
example_id	4.32	3.97	1.09	4.64	4.28	1.08	5.13	4.85	1.06
exampleq	0.70	0.62	1.14	0.64	0.70	0.91	0.89	0.90	0.99
qsort	8.12	8.39	0.97	6.69	6.40	1.04	6.79	6.53	1.04
examplep	1.64	1.07	1.53	1.37	1.49	0.92	1.33	1.25	1.07
example2	2.12	1.89	1.12	2.21	1.96	1.13	2.75	2.81	0.98
tabling_doc	0.29	0.18	1.62	0.35	0.18	1.97	0.33	0.20	1.62
tabling	0.93	0.84	1.10	1.19	0.89	1.33	1.14	0.95	1.20
forward_trail	0.70	0.64	1.10	1.01	0.69	1.47	0.95	0.81	1.17
tabling_rt	63.51	22.39	2.84	59.34	22.67	2.62	72.58	13.20	5.50
tabling_tr	11567.08	7124.96	1.62	6680.47	2684.62	2.49	2550.66	2595.74	0.98
path_tabling	2.31	1.95	1.18	0.34	2.21	0.15	2.54	2.27	1.12
tab_subsumption	1.17	0.44	2.65	1.26	1.15	1.09	1.07	1.32	0.81
tab_aggregates	1.64	1.58	1.04	1.92	1.82	1.05	1.90	1.87	1.02
edge	38.10	37.30	1.02	40.18	37.48	1.07	39.79	33.66	1.18
swap_control	3.39	3.52	0.96	3.58	4.07	0.88	4.48	4.26	1.05
swap	5.27	4.91	1.07	5.32	1.45	3.68	4.36	3.63	1.20
noswap	4.05	3.86	1.05	4.55	4.15	1.10	4.67	4.43	1.05
CCAT	0.37	0.11	3.48	0.49	0.30	1.63	0.56	0.32	1.76
tabling_rt_CCAT	1.87	1.90	0.98	2.44	2.08	1.17	3.30	2.72	1.21

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
tabling_tr_CCAT	13.16	14.76	0.89	7.29	15.58	0.47	20.67	21.55	0.96
p	0.61	0.46	1.32	0.80	0.56	1.43	0.76	0.00	7.58
pathCCAT	53.50	52.93	1.01	71.45	70.95	1.01	106.93	102.76	1.04
basictypes	0.24	0.16	1.47	0.25	0.17	1.46	0.34	0.02	21.37
fetch	2.36	2.16	1.09	2.73	2.36	1.16	2.84	3.19	0.89
server	timeout	1347.06	-	timeout	1227.03	-	1215.91	1176.63	1.03
client	0.17	0.22	0.78	0.25	0.15	1.65	0.37	0.18	2.04
client_vip	0.22	0.18	1.19	0.21	0.16	1.33	0.20	0.21	0.94
pm-im-pm	0.36	0.18	1.95	0.21	0.23	0.93	0.43	0.20	2.20
server_vip	1.70	1.26	1.35	1.46	1.45	1.00	1.31	1.22	1.07
im-rm-im	0.38	0.17	2.19	0.45	0.16	2.84	0.43	0.18	2.36
server2	1.47	1.32	1.12	1.89	1.44	1.32	1.44	1.30	1.11
lisp_syntax_xml	30.12	29.42	1.02	32.24	31.29	1.03	33.33	35.28	0.94
xml_syntax_fipa	0.48	2.42	0.20	3.11	3.18	0.98	5.36	5.03	1.06
lisp_syntax_fipa	10.31	10.05	1.03	10.64	10.36	1.03	10.63	7.39	1.44
lisp_syntax_term	19.63	19.57	1.00	14.26	19.43	0.73	46.83	35.05	1.34
test	0.89	0.76	1.18	1.18	0.79	1.49	1.10	0.84	1.31
server	0.80	0.65	1.24	0.99	0.66	1.48	1.13	0.69	1.63
clpfd_debug_rt	1.02	0.72	1.41	1.17	0.73	1.61	0.94	0.77	1.22
indexicals_tr	1320.98	187.28	7.05	1487.27	233.42	6.37	4096.13	422.55	9.69
fd_var	4.58	4.02	1.14	5.00	4.44	1.13	5.97	5.65	1.06
clpfd_doc	0.02	0.23	0.07	0.22	0.16	1.34	0.02	0.16	0.10
fd_range_intervals	51281.93	45130.85	1.14	14175.78	13402.61	1.06	184.04	177.11	1.04
fd_optim	6.69	6.26	1.07	7.80	7.12	1.10	10.84	11.87	0.91
clpfd_options	0.36	0.27	1.33	0.39	0.25	1.58	0.31	0.32	0.97
fd_pchains	9.81	9.91	0.99	9.73	10.32	0.94	8.37	8.48	0.99
indexicals	0.17	0.39	0.44	0.48	0.27	1.74	0.44	0.32	1.38
fd_range_finite_intervals	18149.82	16777.68	1.08	3929.33	3900.46	1.01	236.59	258.84	0.91
clpfd_tr	160.65	159.20	1.01	111.66	120.28	0.93	16.79	18.06	0.93
fd_term	13.42	9.55	1.41	14.56	15.47	0.94	17.76	19.06	0.93
clpfd_ops	0.33	0.30	1.09	0.46	0.26	1.76	0.54	0.29	1.84
clpfd_debug	0.48	0.26	1.85	0.41	0.25	1.60	0.42	1.69	0.25
int_extra	0.78	0.66	1.17	0.00	0.69	0.15	0.99	0.92	1.07
fd_constraints	35.98	44.30	0.81	45.50	50.76	0.90	46.93	50.93	0.92
fd_range	4.52	4.04	1.12	4.38	4.16	1.05	0.58	4.09	0.14

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
fd_range_bits_unsafe	9.34	5.02	1.86	10.05	9.88	1.02	10.87	7.36	1.48
fd_ops	0.29	0.27	1.09	0.52	0.32	1.65	0.32	0.28	1.16
indexicals_ops	0.33	0.28	1.18	0.39	0.28	1.37	0.60	0.31	1.89
clpfd_stats	1.04	0.85	1.21	0.99	0.81	1.22	1.03	1.05	0.99
fd_labeling	15432.30	3195.55	4.83	14441.73	2532.69	5.70	130.93	129.02	1.01
clpfd_rt	12.93	13.24	0.98	14.37	7.03	2.04	17.54	15.12	1.16
clpfd	0.33	0.29	1.10	0.29	0.27	1.07	0.34	0.37	0.92
fd_utils	2.22	1.62	1.37	2.10	2.00	1.05	2.10	2.07	1.01
queens_fd	7.21	8.23	0.88	8.68	8.60	1.01	11.78	10.90	1.08
sudoku	timeout	timeout	-	timeout	timeout	-	timeout	1329.45	-
metacall_test	5.72	4.86	1.18	1.81	5.35	0.34	8.57	7.28	1.18
bridge	119.02	69.51	1.71	103.42	57.31	1.80	33.41	32.40	1.03
bench	79.93	3.64	21.93	76.41	4.23	18.05	7.36	7.57	0.97
queens_old	4.75	4.24	1.12	4.47	4.29	1.04	4.81	4.48	1.07
queens	3876.13	24.86	155.93	347.94	27.46	12.67	23.96	26.54	0.90
sudoku_old	4.99	4.97	1.00	5.29	6.35	0.83	5.35	5.03	1.06
fib	4.04	0.00	40.36	4.11	4.66	0.88	4.17	4.54	0.92
money_user_level	timeout	8042.51	-	timeout	6338.26	-	291.60	282.18	1.03
queens-swi	0.22	0.16	1.36	0.23	0.17	1.34	0.20	0.25	0.83
counters_test	2.64	2.25	1.17	2.64	2.57	1.03	4.06	3.85	1.05
nondet_move_top_rt	9.43	9.15	1.03	10.96	11.28	0.97	12.78	5.37	2.38
andprolog_props	1.16	0.93	1.25	1.45	1.02	1.42	1.66	1.21	1.37
andprolog_doc	0.49	0.19	2.63	0.35	0.17	2.09	0.45	0.20	2.22
andprolog_rt	3.69	0.01	368.60	3.75	3.26	1.15	3.80	0.01	316.50
agents_rt	5.21	4.72	1.10	5.18	5.59	0.93	7.07	6.88	1.03
andprolog_sched	0.68	0.41	1.65	0.80	0.45	1.75	0.58	0.62	0.94
callh_rt	3.00	2.67	1.12	3.10	3.39	0.91	0.48	3.49	0.14
andprolog_ops	0.43	0.25	1.70	0.38	0.29	1.34	0.41	0.27	1.49
det_rt	7.68	7.12	1.08	1.12	4.82	0.23	10.04	9.86	1.02
disj_wait_rt	3.86	3.96	0.97	4.63	0.85	5.44	2.42	6.03	0.40
andprolog	0.36	0.34	1.05	0.57	0.33	1.74	0.12	0.30	0.40
hamming	timeout	timeout	-	timeout	timeout	-	timeout	896.59	-
hanoi_dl	997.52	351.37	2.84	856.60	276.02	3.10	287.60	286.27	1.00
boyer	timeout	61042.98	-	timeout	55196.08	-	timeout	2654.32	-
tak	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
hanoi	3613.48	370.14	9.76	3611.10	372.37	9.70	118.71	100.04	1.19
extras	3.51	4.23	0.83	4.47	4.20	1.06	6.52	6.49	1.00
table_performance	232.83	115.76	2.01	246.36	142.13	1.73	126.09	160.58	0.79
mandel	666.58	571.53	1.17	113.27	32.36	3.50	522.28	521.66	1.00
chat_parser	timeout	timeout	-	56035.88	46615.02	1.20	11634.32	11660.70	1.00
aiakl	timeout	1224.63	-	timeout	269.33	-	9750.45	136.48	71.44
pal	42.67	25.02	1.71	72.72	33.41	2.18	93.62	88.68	1.06
queens10	22.58	14.55	1.55	36.36	17.74	2.05	24.82	24.21	1.03
fft	53.59	27.94	1.92	86.77	40.91	2.12	100.79	83.05	1.21
qsort	48.12	24.24	1.99	69.23	28.92	2.39	60.47	60.56	1.00
queens	26.50	13.78	1.92	38.34	18.13	2.11	24.83	23.73	1.05
qsort_dl	109.01	89.45	1.22	86.84	45.14	1.92	959.32	972.40	0.99
fibo	37.81	11.36	3.33	60.07	18.07	3.32	26.63	22.28	1.20
mmat	66.52	41.25	1.61	89.32	48.67	1.84	36.58	34.65	1.06
deriv	582.33	566.19	1.03	601.74	565.85	1.06	120.00	121.11	0.99
progeom	42.35	33.60	1.26	54.22	35.13	1.54	39.59	38.88	1.02
ann	147702.51	139613.53	1.06	150.80	109.17	1.38	188.01	188.75	1.00
numbers	216.79	192.75	1.12	195.57	182.40	1.07	322.43	324.24	0.99
disjwait	4.50	5.54	0.81	5.64	6.01	0.94	7.60	8.41	0.90
check_files	35.53	11.86	3.00	62.69	15.34	4.09	24.58	19.01	1.29
extras	1.99	1.81	1.10	1.99	1.62	1.23	1.86	1.85	1.01
qsort_nd	32.68	10.90	3.00	54.33	16.14	3.37	30.85	26.10	1.18
qsort	22.26	13.52	1.65	34.94	13.46	2.60	38.07	35.96	1.06
fibo	19.39	6.22	3.12	28.34	5.38	5.27	13.81	11.32	1.22
mmat	29.78	23.27	1.28	44.92	23.45	1.92	18.11	19.69	0.92
illumination	42.67	23.21	1.84	72.94	27.15	2.69	38.11	27.57	1.38
check_files	6.80	7.46	0.91	4.74	4.86	0.98	10.82	11.10	0.97
extras	1.45	1.36	1.06	1.80	2.02	0.89	1.67	1.75	0.95
qsort_nd	7.57	7.79	0.97	8.76	8.94	0.98	16.76	15.21	1.10
qsort	3.67	3.59	1.02	4.24	4.97	0.85	10.44	10.35	1.01
fibo	2.17	2.06	1.05	0.62	0.00	6.25	3.68	3.53	1.04
mmat	3.13	3.56	0.88	3.81	4.08	0.93	5.37	5.81	0.92
illumination	0.22	0.20	1.12	0.21	0.18	1.21	0.24	0.17	1.43
andprolog_assrt	0.61	0.34	1.80	0.16	0.70	0.23	0.61	0.48	1.28
andprolog_ops_assrt	0.42	0.28	1.52	0.37	0.28	1.35	0.37	0.29	1.26

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
andprolog_rt_assrt	28.61	32.52	0.88	38.76	35.52	1.09	40.62	39.27	1.03
backtracking	2.73	2.53	1.08	3.02	3.25	0.93	3.39	3.54	0.96
qsort	10.31	7.03	1.47	10.92	12.58	0.87	28.04	23.26	1.21
fibo	4.32	4.76	0.91	5.37	6.21	0.86	8.25	4.49	1.84
fibo_gran	4.68	5.00	0.94	6.29	5.89	1.07	8.52	5.13	1.66
rlimit_rt	2.02	1.87	1.08	2.89	2.56	1.13	6.91	6.17	1.12
rlimit	0.56	0.33	1.71	0.49	0.25	2.00	0.40	0.29	1.38
rlimit_tr	6.04	6.23	0.97	5.56	6.20	0.90	6.68	7.17	0.93
a	1.10	1.06	1.05	1.10	1.07	1.04	1.08	1.15	0.93
bundle_source_tree	18.30	19.02	0.96	17.90	19.68	0.91	38.71	36.18	1.07
source_tree	486.12	356.39	1.36	480.14	802.86	0.60	327.07	258.21	1.27
pmrule	0.30	0.26	1.16	0.50	0.31	1.65	0.44	0.33	1.34
ops	0.40	0.29	1.41	0.36	0.42	0.87	0.38	0.33	1.17
pmrule_tr	115.17	85.24	1.35	89.43	76.82	1.16	35.91	35.14	1.02
pmrule_doc	0.24	0.37	0.66	0.38	0.18	2.09	0.30	0.18	1.73
test1	3.48	3.12	1.12	3.63	3.98	0.91	4.10	4.79	0.86
test1fun	3.49	3.44	1.01	3.56	4.45	0.80	4.27	4.62	0.92
extmod_doc	0.25	0.16	1.58	0.40	0.17	2.33	0.26	0.17	1.51
extmod	0.60	0.29	2.08	0.14	0.35	0.40	0.59	0.32	1.84
doccomments	0.35	0.12	2.83	0.51	0.26	2.01	0.40	0.29	1.41
doccomments_tr	135.83	66.57	2.04	130.32	60.08	2.17	5.38	5.08	1.06
doccomments_doc	0.32	0.16	1.96	0.22	0.18	1.25	0.27	0.17	1.55
markup_test	3.59	2.95	1.21	4.00	3.92	1.02	5.21	5.27	0.99
markdown_test	4.68	4.34	1.08	6.09	5.82	1.04	7.68	6.93	1.11
rbtrees	0.33	0.16	2.04	0.00	0.19	0.54	0.37	0.18	2.01
more_markup	3.47	3.12	1.11	4.80	4.51	1.07	6.50	6.41	1.01
foomodes	0.28	0.16	1.73	0.44	0.18	2.50	0.28	0.17	1.64
usagedoc_modes	1.01	0.88	1.14	0.90	0.86	1.04	1.79	1.30	1.38
usagedoc_types	0.88	0.98	0.90	1.12	0.83	1.36	1.02	1.06	0.96
test	2.95	2.89	1.02	0.00	3.58	0.03	5.04	4.60	1.10
write_alt	516.64	465.52	1.11	508.16	503.37	1.01	1592.10	1526.24	1.04
aux	0.20	0.17	1.17	0.18	0.24	0.75	0.31	0.21	1.48
toplevel_scope	0.66	0.16	4.07	0.48	0.58	0.83	0.40	0.36	1.10
toplevel_doc	0.21	2.98	0.07	3.60	2.98	1.21	3.79	3.66	1.03
prettysols	631.53	526.78	1.20	319.07	223.00	1.43	94.69	87.51	1.08

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
toplevel_debugger	0.33	0.18	1.85	0.44	0.19	2.31	0.31	0.25	1.23
toplevel_prompt	1.75	1.59	1.11	2.02	1.81	1.12	2.25	2.06	1.09
toplevel	93.88	54.94	1.71	89.94	52.85	1.70	31.38	32.47	0.97
toplevel_io	1.51	1.20	1.25	1.47	1.87	0.79	1.68	1.75	0.96
pattern_filler	38.76	35.44	1.09	31.32	31.21	1.00	43.57	41.56	1.05
menu_generator	timeout	21236.87	-	timeout	19005.62	-	3304.27	388.70	8.50
menu_tr	21324.41	159.49	133.70	19573.34	133.45	146.67	103.15	100.27	1.03
menu_common	5.58	5.26	1.06	5.42	5.20	1.04	1.02	0.73	1.40
menu_doc	6.22	5.98	1.04	6.34	5.75	1.10	1.54	0.01	140.09
menu_op	0.50	0.37	1.36	0.43	0.44	0.99	0.25	0.28	0.90
menu	6.59	6.51	1.01	6.41	6.62	0.97	1.47	1.39	1.06
menu_rt	1.76	1.46	1.20	2.05	1.63	1.26	1.01	1.11	0.92
menu_2	5.30	5.32	1.00	6.29	5.73	1.10	4.27	4.30	0.99
menu_1	5.39	5.37	1.00	5.83	6.83	0.85	0.54	0.46	1.17
menu_n	5.10	4.72	1.08	5.12	5.02	1.02	3.79	3.33	1.14
menu_3	6.09	5.87	1.04	6.75	3.33	2.03	4.48	4.93	0.91
menu_0	4.91	5.10	0.96	5.61	5.37	1.04	4.51	3.91	1.15
fibers_blt	7.08	6.14	1.15	7.31	6.99	1.05	7.65	7.55	1.01
fibers	1.64	1.70	0.96	1.84	1.78	1.03	1.62	1.49	1.09
fibers_rt	19.74	19.82	1.00	25.60	24.80	1.03	27.62	28.30	0.98
fnct_rt	0.37	0.18	2.11	0.29	0.19	1.55	0.02	0.18	0.11
fibers_hooks	1.53	1.48	1.03	1.60	1.31	1.22	1.15	1.22	0.94
stream_watchdog	9.04	8.86	1.02	10.19	9.66	1.05	15.82	15.04	1.05
fibers_data	4.71	5.00	0.94	4.90	4.67	1.05	5.48	5.67	0.97
mexpand_extra	2.87	2.32	1.24	2.52	2.41	1.04	2.50	2.30	1.09
fibers_tr	86.21	72.44	1.19	73.74	68.97	1.07	69.13	71.30	0.97
actmod_http	310.09	256.10	1.21	83.10	82.63	1.01	60.04	64.46	0.93
markdown_parser	timeout	50934.09	-	timeout	49696.21	-	82009.35	20948.57	3.91
markdown_syntax	5.05	4.78	1.06	5.45	4.99	1.09	6.45	6.31	1.02
markdown_translate	733.43	412.58	1.78	981.99	533.64	1.84	timeout	496.25	-
fsyntaxplus	0.67	0.32	2.09	0.35	0.35	1.00	0.44	0.00	4.41
statevars_tr	timeout	timeout	-	24167.26	15470.60	1.56	2921.11	958.13	3.05
shpa_tr	143.57	150.04	0.96	82.01	81.40	1.01	67.84	60.68	1.12
statevars_rt	1.34	1.01	1.33	1.19	1.09	1.09	1.04	0.82	1.27
blk_tr	34.91	8.68	4.02	32.16	7.20	4.46	6.55	6.53	1.00

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
test_common	2.43	2.27	1.07	2.43	2.12	1.14	2.55	2.03	1.26
test_fsyntaxplus	timeout	13335.10	-	timeout	6380.37	-	2562.40	3034.83	0.84
nested	2.16	1.52	1.42	1.91	1.55	1.24	1.15	0.92	1.25
nested_defs	1.68	1.54	1.09	2.34	1.91	1.23	1.26	0.87	1.44
nested_rt	39.32	13.83	2.84	21.29	13.84	1.54	3.96	10.61	0.37
nested_tr	timeout	7669.86	-	timeout	10466.20	-	903.56	891.60	1.01
nested_ops	0.55	0.35	1.59	0.33	0.27	1.26	0.45	0.30	1.49
nested_doc	0.49	0.17	2.92	0.39	0.17	2.33	0.26	0.18	1.43
extension	6.55	6.62	0.99	6.84	7.49	0.91	5.02	8.00	0.63
structs	15.16	14.43	1.05	15.89	15.75	1.01	20.00	18.74	1.07
hrtime	1.08	1.08	1.00	1.40	1.05	1.34	1.94	1.88	1.03
noise	2.87	3.37	0.85	3.11	3.62	0.86	4.27	4.38	0.98
hrtimea	1.85	1.69	1.10	2.43	2.11	1.15	1.06	4.61	0.23
persdb_examples	0.37	0.25	1.47	0.55	0.27	2.01	0.41	0.09	4.79
persdb_decl	0.79	0.74	1.07	1.03	0.76	1.36	0.78	0.48	1.62
persdb_tr	1.53	1.60	0.96	1.97	1.40	1.41	1.09	1.01	1.08
persdb_rt	6796.75	269.46	25.22	6069.43	233.08	26.04	43.76	44.52	0.98
datadir	2.34	2.54	0.92	2.76	2.79	0.99	0.00	3.41	0.03
persdb_ll	0.94	0.70	1.35	0.80	0.79	1.02	1.03	0.54	1.91
persdb_cache	timeout	723.96	-	timeout	598.30	-	70869.29	129.64	546.67
persdb	1.38	0.65	2.11	1.32	0.75	1.77	0.71	0.53	1.34
queuedy	3.84	3.56	1.08	4.19	4.16	1.01	4.61	4.71	0.98
example_static	1.73	1.52	1.14	1.80	1.83	0.98	2.20	2.28	0.97
queuell	3.06	3.48	0.88	3.40	0.69	4.93	4.32	4.23	1.02
example_dynamic	1.57	1.53	1.03	1.49	1.44	1.03	1.46	1.77	0.82
queue	3.19	3.19	1.00	3.27	3.26	1.00	4.15	3.70	1.12
example_static	1.57	1.47	1.07	1.63	1.84	0.88	2.40	2.16	1.11
example_dynamic	2.19	1.61	1.37	2.22	2.06	1.08	2.31	2.26	1.02
queue	0.80	3.46	0.23	4.63	3.95	1.17	5.19	5.18	1.00
persistentdb	4.45	5.31	0.84	5.25	6.50	0.81	6.43	7.27	0.88
arith_extra	1.53	1.40	1.09	1.58	1.80	0.87	1.96	1.55	1.27
nf	4.92	4.56	1.08	4.74	4.48	1.06	5.04	4.76	1.06
clpqr_ops	0.75	0.29	2.62	0.50	0.24	2.05	0.40	0.30	1.35
clpqr_dump	10.41	10.63	0.98	11.58	9.89	1.17	11.56	10.97	1.05
clpqr_options	0.30	0.27	1.13	0.32	0.41	0.79	0.54	0.27	1.98

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
clpqr_meta	0.97	1.17	0.83	1.02	1.01	1.01	1.11	1.27	0.87
nl_eval	9.85	9.27	1.06	9.62	9.16	1.05	9.67	10.26	0.94
clpqr_rt	1.32	1.22	1.08	1.26	0.99	1.28	1.38	0.85	1.63
simplex	4.60	4.70	0.98	4.75	4.28	1.11	4.46	4.34	1.03
fourier_motzkin	3.07	2.35	1.31	2.75	2.72	1.01	2.92	2.57	1.13
clpqr_compiler	3.65	3.67	1.00	3.57	3.37	1.06	3.99	3.39	1.18
clpqr_attr_hooks	61.75	63.50	0.97	24.53	28.86	0.85	7.64	7.75	0.99
solve	6.18	6.18	1.00	6.22	5.77	1.08	6.26	5.88	1.06
clpqr_tr	0.83	0.78	1.07	0.69	0.63	1.09	0.84	0.59	1.41
critical	timeout	41848.54	-	timeout	37996.99	-	timeout	timeout	-
mmatrix_q	0.84	0.53	1.60	0.94	0.59	1.61	0.78	0.77	1.01
mmatrix_r	0.78	0.60	1.32	0.73	0.53	1.38	0.64	0.74	0.86
nqueens_q	0.97	0.72	1.36	0.68	0.63	1.08	0.79	0.57	1.40
fib_r	2.54	2.99	0.85	2.59	2.94	0.88	4.59	4.18	1.10
laplace	0.92	0.83	1.11	0.87	0.80	1.09	0.82	0.76	1.07
fib_q	2.10	2.28	0.92	2.40	2.63	0.91	3.07	3.02	1.01
fibpl	0.75	0.38	1.97	0.57	0.47	1.21	0.44	0.41	1.06
fibclpr	0.60	0.40	1.50	0.56	0.38	1.47	0.56	0.40	1.41
mmatrix	0.20	0.18	1.14	0.31	0.16	1.94	0.27	0.18	1.44
fib_example	1.83	2.22	0.82	2.23	2.64	0.85	2.73	2.93	0.93
nqueens_q	0.98	0.75	1.31	0.41	0.75	0.55	1.01	0.69	1.46
mmult	0.19	0.19	0.99	0.19	0.21	0.91	0.30	0.17	1.73
t	0.42	0.40	1.04	0.70	0.41	1.72	0.45	0.46	0.99
power	2.45	2.13	1.15	2.27	1.96	1.16	2.25	1.97	1.14
traits_doc	0.49	0.15	3.23	0.37	0.20	1.88	0.36	0.18	2.02
traits	0.48	0.35	1.39	0.00	0.36	0.28	0.73	0.30	2.44
traits_tr	1862.07	71.23	26.14	1184.31	76.02	15.58	39.10	41.80	0.94
traits_ops	0.40	0.32	1.27	0.44	0.26	1.68	0.36	0.33	1.10
trait_orig	6.96	6.68	1.04	6.47	6.54	0.99	7.63	7.96	0.96
trait_test	6.85	7.46	0.92	6.51	6.66	0.98	8.75	7.69	1.14
trait_test2	6.80	6.99	0.97	6.52	7.41	0.88	7.91	8.03	0.98
goedel	2.45	3.25	0.75	3.99	3.58	1.12	4.98	1.16	4.31
indexer_doc	1.57	1.73	0.90	2.04	1.66	1.23	2.74	2.22	1.23
indexer_tr	22706.71	593.89	38.23	18871.27	98.95	190.71	203.33	83.79	2.43
hash	2.81	4.51	0.62	3.44	3.32	1.04	4.84	4.14	1.17

	share			shfr			shfr-clique		
module	**TC**	**TT**	ρ**T**	**TC**	**TT**	ρ**T**	**TC**	**TT**	ρ**T**
indexer	0.57	0.30	1.89	0.46	0.28	1.64	0.34	0.29	1.19
t	3.25	3.61	0.90	4.20	5.29	0.79	4.77	4.72	1.01
t	3.12	3.93	0.79	4.59	4.99	0.92	4.87	4.81	1.01
t_out	0.84	0.81	1.04	0.76	0.87	0.87	0.71	0.68	1.05
hash_	1.31	1.94	0.68	2.08	1.73	1.20	1.83	1.96	0.94
idx_test	7.76	3.72	2.09	9.14	9.25	0.99	8.66	8.19	1.06
	share			shfr			shfr-clique		
Modules	1018			1018			1018		
Modules	1018	-	-	-	-	-	-	-	-
FC	89			86			35		
FC :	89	-	-	86	-	-	35	-	-
FT	52			49			16		
FT :	52	-	-	49	-	-	16	-	-
μ**T**	4.19			4.37			25.16		
μ**T**	4.19	-	-	4.37	-	-	25.16	-	-
μ**T** $>$ 1s (44-47-101)	52.51			48.03			229.06		
μ**T** $>$ 1s (44-47-101)	52.51	-	-	48.03	-	-	229.06	-	-
μ**T** $>$ 0.5s (61-64-115)	39.00			36.58			201.87		
μ**T** $>$ 0.5s (61-64-115)	39.00	-	-	36.58	-	-	201.87	-	-
μ**T** $>$ 0.1s (102-105-156)	24.27			23.12			149.20		
μ**T** $>$ 0.1s (102-105-156)	24.27	-	-	23.12	-	-	149.20	-	-

C.4 Source code of the s(CASP) System

Table C 4: Analysis times and statistics for analyzing the source code of the s(CASP) system.

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
scasp_load_compiled	7.82	9.24	0.85	8.85	10.00	0.89	11.32	12.50	0.91
clp_call_stack	4.97	5.51	0.90	5.56	5.51	1.01	7.36	7.17	1.03
test_results_c_forall	0.14	0.20	0.73	0.15	0.22	0.68	0.15	0.17	0.87
test_results	0.17	0.14	1.22	0.13	0.18	0.73	0.18	0.14	1.30
clp_disequality_rt	15.29	12.35	1.24	17.29	17.47	0.99	14.24	18.99	0.75
clp_disequality	0.29	0.24	1.20	0.31	0.29	1.08	0.25	0.27	0.94
clp_disequality_test	7.37	8.86	0.83	8.32	9.91	0.84	6.89	10.91	0.63
scasp	timeout	timeout	-	timeout	timeout	-	timeout	8419.57	-
test	20.52	26.53	0.77	25.31	25.73	0.98	44.39	44.63	0.99
scasp_io	1115.74	468.17	2.38	1242.18	401.14	3.10	419.41	410.74	1.02
clp_clpq	13.85	14.79	0.94	11.45	15.89	0.72	14.85	16.04	0.93
scasp_top_level	0.31	0.34	0.92	0.41	0.32	1.26	0.33	0.32	1.01
main_forgetting	1233.61	839.94	1.47	990.29	625.80	1.58	144.41	132.70	1.09
forgetting	21638.05	115.74	186.95	21295.85	98.16	216.95	7.47	7.79	0.96
forgetting_aux	36039.37	1888.67	19.08	34688.66	1655.99	20.95	133.40	103.49	1.29
loops	357.16	162.52	2.20	343.06	150.77	2.28	13.85	15.77	0.88
aux_negation	40.99	41.78	0.98	51.66	58.23	0.89	51.29	57.18	0.90
auxiliary	675.51	260.40	2.59	658.96	212.29	3.10	210.84	158.25	1.33
graph_aux	159221.08	461.40	345.08	154079.84	401.00	384.24	69.04	77.16	0.89
graph	685.98	247.95	2.77	647.51	219.33	2.95	92.92	41.46	2.24
dual_rules_for_graphs	22.43	7.86	2.86	20.24	7.76	2.61	8.21	8.30	0.99
jquery_tree	0.35	0.35	1.00	0.40	0.36	1.11	0.58	0.48	1.20
html_head	0.36	0.38	0.93	0.32	0.39	0.82	0.40	0.44	0.92
html_tail	0.36	0.35	1.04	0.41	0.40	1.02	0.40	0.45	0.87
ciao_auxiliar	86.01	45.45	1.89	85.78	43.96	1.95	63.81	62.34	1.02
tokenizer	269.42	64.96	4.15	210.14	51.97	4.04	63.48	52.87	1.20
interactive	4.19	5.47	0.77	4.38	1.67	2.63	7.57	8.97	0.84
variables	3140.49	497.54	6.31	2496.18	401.37	6.22	79.98	92.35	0.87
chs	timeout	timeout	-	timeout	219507.23	-	1199.49	339.91	3.53
io	timeout	4519.12	-	timeout	4188.43	-	timeout	1736.07	-

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
debug	13.01	8.99	1.45	8.93	8.79	1.02	16.50	17.21	0.96
main	15.25	19.01	0.80	17.25	20.04	0.86	24.51	29.16	0.84
output	timeout	2640.58	-	timeout	614.03	-	206.05	194.17	1.06
rbtrees	timeout	timeout	-	timeout	timeout	-	324.78	339.40	0.96
common	25.29	19.44	1.30	28.26	17.39	1.63	42.37	44.53	0.95
call_graph	3854.71	662.94	5.81	1668.77	117.42	14.21	3.05	3.17	0.96
program	32489.64	80.44	403.90	145.67	72.99	2.00	85.56	78.11	1.10
options	2.82	2.92	0.97	3.25	3.31	0.98	4.08	4.22	0.97
text_dcg	660.13	298.30	2.21	359.78	202.88	1.77	9670.78	196.81	49.14
nmr_check	timeout	timeout	-	timeout	timeout	-	4885.73	2853.24	1.71
solve	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-
doc_gen	0.28	0.24	1.18	0.26	0.28	0.93	0.29	0.31	0.95
comp_duals	timeout	17261.48	-	timeout	4151.69	-	9.19	9.65	0.95
config	0.62	0.61	1.02	0.62	0.70	0.88	0.77	0.83	0.92
	share			shfr			shfr-clique		
Mods	44			44			44		
FC	8			8			3		
FT	5			4			1		
μ**T**	28.07			19.16			2.27		
μ**T** $>$ 1s (8-8-13)	121.37			81.15			1.26		
μ**T** $>$ 0.5s (11-11-16)	88.96			59.73			4.32		
μ**T** $>$ 0.1s (13-13-18)	75.76			51.03			3.95		

C.5 Source Code of the Spectector Tool

Table C 5: Analysis times and statistics for analyzing the source code of the Spectector tool.

	share			shfr			shfr-clique		
module	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**	**TC**	**TT**	**ρT**
spectector_flags	3.71	3.96	0.94	4.08	4.36	0.94	4.87	4.73	1.03
muasm_semantics	1351.25	164.39	8.22	1364.86	150.69	9.06	98.89	94.20	1.05
muasm_print	25.04	26.79	0.93	13.15	15.23	0.86	13.44	10.91	1.23
muasm_syntax	0.25	0.25	0.99	0.27	0.25	1.07	0.31	0.27	1.18
spectector_stats	timeout	71.00	-	timeout	112.20	-	292.50	29.88	9.79
spectector	timeout	timeout	-	timeout	timeout	-	timeout	timeout	-
muasm_program	1.41	5.98	0.24	5.04	5.54	0.91	8.40	9.36	0.90
spectector_noninter	timeout	25132.10	-	timeout	21251.12	-	timeout	801.15	-
x86_table	4.52	4.40	1.03	4.77	0.94	5.08	7.26	6.92	1.05
x86_to_muasm	timeout	timeout	-	timeout	timeout	-	952.86	983.30	0.97
gas_parser	531.04	27.52	19.30	203.68	27.36	7.45	48.20	38.63	1.25
muasm_parser	17.89	18.50	0.97	20.21	20.01	1.01	28.56	27.73	1.03
intel_parser	501.00	104.55	4.79	428.86	76.74	5.59	127.88	98.40	1.30
muasm_dump	3.20	3.57	0.90	3.63	3.99	0.91	5.04	5.60	0.90
parser_aux	23.98	24.96	0.96	25.67	28.15	0.91	35.63	30.86	1.15
	share			shfr			shfr-clique		
Mods	15			15			15		
FC	4			4			2		
FT	2			2			1		
μ**T**	3.57			3.07			1.76		
μ**T** $>$ 1s (1-1-3)	8.22			9.06			3.94		
μ**T** $>$ 0.5s (3-3-5)	10.77			7.36			2.87		
μ**T** $>$ 0.1s (3-3-5)	10.77			7.36			2.87		