SEH-PILoT: A System for Property-Directed Symbol Elimination – Work in Progress (Short Paper)

Philipp Marohn and Viorica Sofronie-Stokkermans

University Koblenz-Landau, Koblenz, Germany
{pmarohn,sofronie}@uni-koblenz.de

Abstract. We describe the implementation of a system for property-directed symbol elimination in extensions of a theory $\mathcal{T}_0$ with additional function symbols whose properties are axiomatised using a set of clauses. The system performs the symbol elimination in a hierarchical way, relying on existing mechanisms for symbol elimination in $\mathcal{T}_0$.

1 Introduction

Many reasoning problems in mathematics or program verification can be reduced to checking satisfiability of ground formulae w.r.t. a theory (this can be a standard theory, e.g. linear arithmetic, or a complex theory – e.g. the extension of a base theory with additional function symbols axiomatized by a set of formulae, or a combination of theories). More interesting is to go beyond yes/no answers, i.e. to consider problems – in mathematics or verification – in which the properties of certain function symbols are underspecified (these symbols are considered to be parametric) and (weakest) additional conditions need to be derived under which given properties hold. In [13] a method for property-directed symbol elimination in local theory extensions was proposed which can be used for obtaining such constraints on parameters. The goal of this paper is to present the current state of an implementation of this method in the system SEH-PILoT.

Structure of the paper: In Section 2.2 we first present the theoretical background and then the implementation details. In Section 3 we discuss in detail an example, then present an overview of a (small) subset of the tests we considered so far.

2 Description of the SEH-PILoT Implementation

SEH-PILoT (Symbol Elimination based on Hierarchical Proving In Local Theory extensions) is an implementation of the method for symbol elimination presented in [12,13]. SEH-PILoT is implemented in Python 3.9. Its general structure is presented in Figure 1. Examples which show how SEH-PILoT can be used in various application areas (mathematics, verification, wireless network theory) can be found at https://userpages.uni-koblenz.de/~sofronie/sehpilot/.

2.1 Theoretical Background

Let $\Pi_0{=}(\Sigma_0, \mathsf{Pred})$ be a signature, and $\mathcal{T}_0$ be a "base" theory with signature Π_0. We consider extensions $\mathcal{T} := \mathcal{T}_0 \cup \mathcal{K}$ of $\mathcal{T}_0$ with new function symbols Σ (*extension functions*) whose properties are axiomatized using a set $\mathcal{K}$ of (universally closed) clauses in the extended signature $\Pi = (\Sigma_0 \cup \Sigma, \mathsf{Pred})$, such that each clause in $\mathcal{K}$ contains function symbols in Σ. Let $\Sigma_{\mathsf{par}} \subseteq \Sigma$ be a set of parameters. Let Σ_c be an additional set of constants.

Task: Let G be a set of ground $\Pi \cup \Sigma_c$ clauses. We want to check whether G is satisfiable w.r.t. $\mathcal{T}_0 \cup \mathcal{K}$ or not and – if it is satisfiable – to generate a weakest universal $\Pi_0 \cup \Sigma_{\mathsf{par}}$-formula Γ such that $\mathcal{T}_0 \cup \mathcal{K} \cup \Gamma \cup G$ is unsatisfiable.

Method. For *satisfiability checking* we use a method for hierarchical reduction to checking satisfiability in the base theory. For *symbol elimination* (i.e. for determining Γ) we use a method for hierarchically reducing the problem to a quantifier elimination problem w.r.t. $\mathcal{T}_0$. If $\mathcal{T}_0$ allows quantifier elimination (i.e. for every formula ϕ over Π_0 there exists a quantifier-free formula ϕ^* over Π_0 which is equivalent to ϕ modulo $\mathcal{T}_0$) a method for quantifier elimination w.r.t. $\mathcal{T}_0$ can be used for this.

In what follows we present situations in which hierarchical reasoning is complete and weakest constraints on parameters can be generated.

Local Theory Extensions. Let Ψ be a closure operator on sets of ground terms. A theory extension $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}$ is Ψ-local if it satisfies the condition:

(Loc_f^Ψ) For every finite set G of ground Π^C-clauses (for an additional set C of constants) it holds that $\mathcal{T}_0 \cup \mathcal{K} \cup G \models \perp$ if and only if $\mathcal{T}_0 \cup \mathcal{K}[\Psi_\mathcal{K}(G)] \cup G$ is unsatisfiable.

where, for every set G of ground Π^C-clauses, $\mathcal{K}[\Psi_\mathcal{K}(G)]$ is the set of instances of $\mathcal{K}$ in which the terms starting with a function symbol in Σ are in $\Psi_\mathcal{K}(G) = \Psi(\mathsf{est}(\mathcal{K}, G))$, where $\mathsf{est}(\mathcal{K}, G)$ is the set of ground terms starting with a function in Σ occurring in G or $\mathcal{K}$.
Ψ-local extensions can be recognized by showing that certain partial models embed into total ones [11,7]. Especially well-behaved are theory extensions with the property (Comp_f^Ψ) which requires that every partial model of $\mathcal{T}$ whose reduct to Π_0 is total and the "set of defined terms" is finite and closed under Ψ, embeds into a total model of $\mathcal{T}$ *with the same support* (cf. e.g. [5]). If Ψ is the identity, we denote Loc_f^Ψ by Loc_f and Comp_f^Ψ by Comp_f. Examples of local theory extensions can be found in [14].

Hierarchical Reasoning. Consider a Ψ-local theory extension $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}$. Condition ($\mathsf{Loc}_f^\Psi$) requires that for every finite set G of ground Π^C clauses: $\mathcal{T}_0 \cup \mathcal{K} \cup G \models \perp$ if and only if $\mathcal{T}_0 \cup \mathcal{K}[\Psi_\mathcal{K}(G)] \cup G \models \perp$. In all clauses in $\mathcal{K}[\Psi_\mathcal{K}(G)] \cup G$ the function symbols in Σ only have ground terms as arguments, so $\mathcal{K}[\Psi_\mathcal{K}(G)] \cup G$ can be flattened and purified[1] by introducing, in a bottom-up manner, new

[1] i.e. the function symbols in Σ are separated from the other symbols.

constants $c_t \in C$ for subterms $t{=}f(c_1, \ldots, c_n)$ where $f{\in}\Sigma$ and c_i are constants, together with definitions $c_t{\approx}f(c_1, \ldots, c_n)$ which are all included in a set Def. We thus obtain a set of clauses $\mathcal{K}_0{\cup}G_0{\cup}\mathsf{Def}$, where $\mathcal{K}_0$ and G_0 do not contain Σ-function symbols and Def contains clauses of the form $c{\approx}f(c_1, \ldots, c_n)$, where $f{\in}\Sigma$, $c, c_1, \ldots, c_n$ are constants.

Theorem 1 ([11,5]) *Let $\mathcal{K}$ be a set of clauses. Assume that $\mathcal{T}_0 \subseteq \mathcal{T}_1 = \mathcal{T}_0{\cup}\mathcal{K}$ is a Ψ-local theory extension. For any finite set G of ground clauses, let $\mathcal{K}_0{\cup}G_0{\cup}\mathsf{Def}$ be obtained from $\mathcal{K}[\Psi_{\mathcal{K}}(G)]{\cup}G$ by flattening and purification, as explained above. Then the following are equivalent to $\mathcal{T}_1 \cup G \models\perp$:*

(1) $\mathcal{T}_0{\cup}\mathcal{K}[\Psi_{\mathcal{K}}(G)]{\cup}G \models\perp$.

(2) $\mathcal{T}_0{\cup}\mathcal{K}_0{\cup}G_0{\cup}\mathsf{Con}_0 \models\perp$, *where* $\mathsf{Con}_0{=}\{\bigwedge_{i=1}^{n} c_i{\approx}d_i \rightarrow c{\approx}d \mid \begin{array}{l} f(c_1, \ldots, c_n){\approx}c{\in}\mathsf{Def} \\ f(d_1, \ldots, d_n){\approx}d{\in}\mathsf{Def} \end{array}\}$.

We can also consider chains of theory extensions:

$$\mathcal{T}_0 \subseteq \mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}_1 \subseteq \mathcal{T}_2 = \mathcal{T}_0 \cup \mathcal{K}_1 \cup \mathcal{K}_2 \subseteq \cdots \subseteq \mathcal{T}_n = \mathcal{T}_0 \cup \mathcal{K}_1 \cup ... \cup \mathcal{K}_n$$

in which each theory is a local extension of the preceding one. For a chain of n local extensions a satisfiability check w.r.t. the last extension can be reduced (in n steps) to a satisfiability check w.r.t. $\mathcal{T}_0$. The only restriction we need to impose in order to ensure that such a reduction is possible is that at each step the clauses reduced so far need to be ground. Groundness is assured if each variable in a clause appears at least once under an extension function. This iterated instantiation procedure for chains of local theory extensions has been implemented in H-PILoT [6].[2]

Hierarchical Symbol Elimination. In [13] we proposed a method for property-directed symbol elimination described in Algorithm 1.

Theorem 2 ([12,13]) *Let $\mathcal{T}_0$ be a Π_0-theory allowing quantifier elimination, Σ_{par} be a set of parameters (function and constant symbols) and Σ a set of function symbols such that $\Sigma \cap (\Sigma_0 \cup \Sigma_{\mathsf{par}}) = \emptyset$. Let $\mathcal{K}$ be a set of clauses in the signature $\Pi_0{\cup}\Sigma_{\mathsf{par}}{\cup}\Sigma$ in which all variables occur also below functions in $\Sigma_1 = \Sigma_{\mathsf{par}} \cup \Sigma$. Assume $\mathcal{T} \subseteq \mathcal{T}_0 \cup \mathcal{K}$ satisfies condition (Comp_f^{Ψ}) for a suitable closure operator Ψ. Let $T = \Psi_{\mathcal{K}}(G)$. Then Algorithm 1 yields a universal $\Pi_0 \cup \Sigma_{\mathsf{par}}$-formula $\forall \overline{x}. \Gamma_T(\overline{x})$ such that $\mathcal{T}_0 \cup \forall \overline{x}. \Gamma_T(\overline{x}) \cup \mathcal{K} \cup G \models\perp$ which is entailed by every universal formula Γ with $\mathcal{T}_0 \cup \Gamma \cup \mathcal{K} \cup G \models\perp$.*

Algorithm 1 yields a formula $\forall \overline{x}. \Gamma_T(\overline{x})$ with $\mathcal{T}_0 \cup \forall \overline{x}. \Gamma_T(\overline{x}) \cup \mathcal{K} \cup G \models\perp$ also if the extension $\mathcal{T} \subseteq \mathcal{T}_0 \cup \mathcal{K}$ is not Ψ-local or $T \neq \Psi_{\mathcal{K}}(G)$, but in this case there is no guarantee that $\forall \overline{x}. \Gamma_T(\overline{x})$ is the weakest universal formula with this property. A similar result holds for chains of local theory extensions.

[2] H-PILoT allows the user to specify a chain of extensions by tagging the extension functions with their place in the chain (e.g., if f occurs in $\mathcal{K}_3$ but not in $\mathcal{K}_1 \cup \mathcal{K}_2$ it is declared as level 3).

Algorithm 1 Symbol elimination in theory extensions [12,13]

Input: Theory extension $\mathcal{T}_0 \cup \mathcal{K}$ with signature $\Pi = \Pi_0 \cup (\Sigma \cup \Sigma_{\mathsf{par}})$
where Σ_{par} is a set of parameters
Set T of ground Π^C-terms
Output: $\forall \overline{y}.\, \Gamma_T(\overline{y})$ (universal $\Pi_0 \cup \Sigma_{\mathsf{par}}$-formula)

Step 1 Purify $\mathcal{K}[T] \cup G$ as described in Theorem 1 (with set of extension symbols Σ_1). Let $\mathcal{K}_0 \cup G_0 \cup \mathsf{Con}_0$ be the set of Π_0^C-clauses obtained this way.

Step 2 Let $G_1 = \mathcal{K}_0 \cup G_0 \cup \mathsf{Con}_0$. Among the constants in G_1, we identify
(i) the constants c_f, $f \in \Sigma_{\mathsf{par}}$, where c_f is a constant parameter or c_f is introduced by a definition $c_f \approx f(c_1, \ldots, c_k)$ in the hierarchical reasoning method,
(ii) all constants $\overline{c}_p$ occurring as arguments of functions in Σ_{par} in such definitions.
Replace all the other constants $\overline{c}$ with existentially quantified variables $\overline{x}$ (i.e. replace $G_1(\overline{c}_p, \overline{c}_f, \overline{c})$ with $\exists \overline{x}.\, G_1(\overline{c}_p, \overline{c}_f, \overline{x})$).

Step 3 Construct a formula $\Gamma_1(\overline{c}_p, \overline{c}_f)$ equivalent to $\exists \overline{x}.\, G_1(\overline{c}_p, \overline{c}_f, \overline{x})$ w.r.t. $\mathcal{T}_0$ using a method for quantifier elimination in $\mathcal{T}_0$ and let $\Gamma_2(\overline{c}_p, \overline{c}_f)$ be $\neg\Gamma_1(\overline{c}_p, \overline{c}_f)$.

Step 4 Replace (i) each constant c_f introduced by definition $c_f \approx f(c_1, \ldots, c_k)$ with the term $f(c_1, \ldots, c_k)$ and (ii) $\overline{c}_p$ with universally quantified variables $\overline{y}$ in $\Gamma_2(\overline{c}_p, \overline{c}_f)$. The formula obtained this way is $\forall \overline{y}.\, \Gamma_T(\overline{y})$.

Theorem 3 ([13]) *Consider the following chain of theory extensions:*

$$\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}_1 \subseteq \mathcal{T}_0 \cup \mathcal{K}_1 \cup \mathcal{K}_2 \subseteq \ldots \subseteq \mathcal{T}_0 \cup \mathcal{K}_1 \cup \mathcal{K}_2 \cup \cdots \cup \mathcal{K}_n$$

where every extension in the chain satisfies condition (Comp_f), $\mathcal{K}_i$ *are all flat and linear, and in all* $\mathcal{K}_i$ *all variables occur below the extension terms on level* i.

Let G *be a set of ground clauses, and let* G_1 *be the result of the hierarchical reduction of satisfiability of* G *to a satisfiability test w.r.t.* $\mathcal{T}_0$. *Let* $T = T(G)$ *be the set of all instances used in the chain of hierarchical reductions and let* $\forall y.\, \Gamma_{T(G)}(y)$ *be the formula obtained by applying Steps 2–4 of Alg. 1 to* G_1. *Then* $\forall y.\, \Gamma_{T(G)}(y)$ *is entailed by every conjunction* Γ *of clauses with the property that* $\mathcal{T}_0 \cup \Gamma \cup \mathcal{K}_1 \cup \cdots \cup \mathcal{K}_n \cup G$ *is unsatisfiable (i.e. it is the weakest such constraint).*

2.2 Implementation

Hierarchical reasoning: H-PILoT. The method for hierarchical reasoning in local theory extensions described before was implemented in the system H-PILoT [6]. H-PILoT carries out a hierarchical reduction to the base theory. Standard SMT provers or specialized provers can be used for testing the satisfiability of the formulae obtained after the reduction. H-PILoT uses eager instantiation and the hierarchical reduction, so provers like CVC4 [1] or Z3 [2] are in general faster in proving unsatisfiability. The advantage of using H-PILoT is that knowing the instances needed for a complete instantiation allows us to correctly detect satisfiability (and generate models) in situations in which e.g. CVC4 returns "unknown", and use property-directed symbol elimination to obtain additional constraints on parameters which ensure unsatisfiability.

Symbol Elimination: SEH-PILoT (Symbol Elimination with H-PILoT): For obtaining *constraints on parameters* we used the method described in Algorithm 1 [13] which was implemented in SEH-PILoT for the case in which the theory can be structured as a local theory extension or a chain of theory extensions and the base theory $\mathcal{T}_0$ is the theory of real closed fields.

Input. SEH-PILoT receives a list of symbols to be eliminated (and possibly a list of already existing constraints on the parameters) and an input file for H-PILoT[3]. This file contains (i) the specification of the signature and of the hierarchy of local theory extensions to be considered; (ii) an axiomatization $\mathcal{K}$ of the theory extension(s); (iii) a set G of ground clauses possibly containing additional constants. Currently the only supported base theory for SEH-PILoT is the theory of real numbers (the theory of real closed fields).

Main Algorithm. SEH-PILoT follows the steps of Algorithm 1.

Step 1: SEH-PILoT uses H-PILoT (with option `-redlog`) for the hierarchical reduction to a problem in the base theory. H-PILoT computes the necessary instances $\mathcal{K}[T_G]$, where T_G is the set of ground terms necessary for instantiation (cf. Thm.3), generates the formula $\mathcal{K}_0 \cup G_0 \cup \mathsf{Con}_0$ and writes it in a file which can be used as input for Redlog [4].

Step 2: Taking into account the function symbols to be eliminated, the constants are classified as required in Step 2 of Alg. 1 and the Redlog file is changed accordingly such that only those symbols that do not correspond to a parameter or argument of a parameter are considered to be existentially quantified.

Step 3: SEH-PILoT uses Redlog to eliminate the existentially quantified symbols and afterwards to negate the resulting formula.

Step 4: The constants contained in the formula obtained by Step 3 are replaced back with the terms they represent and the constants occurring as arguments are replaced by universally quantified variables.

Finally SEH-PILoT translates the generated constraints from the syntax of Redlog back to the syntax of H-PILoT such that they can easily be used for verification or an iterative approach of constraint generation. Since Redlog is not very efficient in simplifying formulae, SLFQ can be used, which allows Redlog to use the possibilities of simplification offered by QEPCAD B [3].

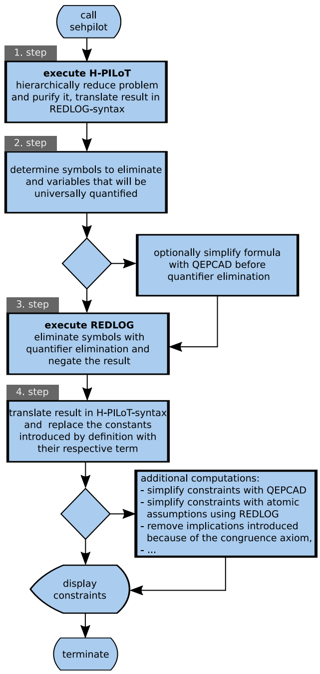

Fig. 1. SEH-PILoT structure

[3] A detailed description of the form of such input files can be found in the system description of H-PILoT [6].

3 Examples

We illustrate the way SEH-PILoT works on the following example[4]. Consider a discrete water level controller in which the inflow varies during the evolution of the system, and can be modeled by a function $\mathsf{inflow} : \mathbb{R} \rightarrow \mathbb{R}$, where $\mathsf{inflow}(t)$ is the inflow at time step t. If the water level becomes greater than an alarm level L_{alarm} (positioned below the overflow level L_{overflow}) a valve is opened and a fixed quantity of water ($\mathsf{outflow}$) is left out. Otherwise, the valve remains closed. Assume that the formula $\mathsf{Init}(L) := L{<}L_{\mathsf{overflow}}$ describes the initial states. Then $L \leq L_{\mathsf{overflow}}$ is an inductive invariant iff the following formulae are unsatisfiable w.r.t. the extension of the theory of real numbers with a function inflow:

(1) $\exists L.\, (L < L_{\mathsf{overflow}} \wedge L > L_{\mathsf{overflow}})$;
(2) $\exists L, L', t, t'.\, (L{\leq}L_{\mathsf{overflow}}{\wedge}L{>}L_{\mathsf{alarm}}{\wedge}L'{\approx}L{+}\mathsf{inflow}(t){-}\mathsf{outflow}{\wedge}t'{\approx}t{+}1{\wedge}L'{>}L_{\mathsf{overflow}})$;
(3) $\exists L, L', t, t'.\, (L{\leq}L_{\mathsf{overflow}} \wedge L{\leq}L_{\mathsf{alarm}} \wedge L'{\approx}L{+}\mathsf{inflow}(t) \wedge t'{\approx}t{+}1 \wedge L'{>}L_{\mathsf{overflow}})$.

We want to obtain conditions on the parameters (inflow, $\mathsf{outflow}$, $\mathsf{L}_{\mathsf{alarm}}, L_{\mathsf{overflow}}$) under which $L < L_{\mathsf{overflow}}$ is an inductive invariant. (1) is clearly unsatisfiable. SEH-PILoT (with assumptions $L_{\mathsf{alarm}} < L_{\mathsf{overflow}}, \forall x.\, \mathsf{inflow}(x) > 0$) generated weakest universal conditions under which (2) resp. (3) are unsatisfiable. Consider e.g. (2). The problem is described in the H-PILoT syntax as follows:

```
Base_functions:={(+,2), (-,2), (*,2)}
Extension_functions:={(inflow, 1, 1)}
Relations:={(<=, 2), (<, 2), (>=, 2), (>, 2)}

Clauses:= l <= loverflow;l > lalarm;lp = (l+inflow(t))-outflow;tp = t+_1;
Query:=  lp > loverflow;
```

It can be checked that the extension $\mathbb{R} \subseteq \mathbb{R} \cup \mathcal{K}$ of $\mathbb{R}$ with a function symbol inflow satisfying the axiom $\mathcal{K} = \forall x.(\texttt{inflow}(\texttt{x}) > \texttt{0})$ defines a local theory extension.

When using SEH-PILoT the user has to specify the name of the H-PILoT file (in this case `inv2.loc`), the symbols to be eliminated (`l`, `lp` and `tp`) and any additional constraints on the parameters (`lalarm<loverflow` and $\forall x.(\texttt{inflow}(\texttt{x}){>}\texttt{0})$). This is done in the command line:

```
sehpilot inv2.loc -e l lp tp -a 'lalarm<loverflow' 'inflow(?)>0' --stats
```

If called with the additional constraints added to the command line (example “Water tank, -a” in Table 1) SEH-PILoT generates the right instances. Alternatively, the clause $\forall x.(\texttt{inflow}(\texttt{x}) > \texttt{0})$ can be added to the `Clauses` in the `inv2.loc`-file (example “Water tank, no -a” in Table 1). In both cases H-PILoT performs the hierarchical reduction described in Theorem 1 for G being the conjunction of the ground formulae in `Clauses`, `Query` and in the additional assumptions if option -a is used, by determining the set $T = \mathsf{est}(G) = \{\texttt{inflow}(\texttt{t})\}$ of terms in G starting with an extension function, considering the instances of $\mathcal{K}$ containing this term, $\mathcal{K}[T] = \{\texttt{inflow}(\texttt{t}) > \texttt{0}\}$ and introducing a constant e_1 for `inflow(t)`, and generates the Redlog file:

[4] This example as well as several additional examples can be found under https://userpages.uni-koblenz.de/~sofronie/sehpilot/.

```
load_package redlog; rlset OFSF; off rlverbose; on rlnzden; off nat;

vars    := {lalarm, e_1, l, lp, outflow, loverflow, t, tp};
formula := ( lp > loverflow and l > lalarm and  l <= loverflow and
                tp = t + 1 and  lp = (l + e1) - outflow );
query :=  (rlqe ex(vars, formula)); end;
```

SEH-PILoT classifies the constants and updates the Redlog file by simplifying the input and changing `vars` to the set of symbols to be eliminated $\mathtt{l}, \mathtt{lp}, \mathtt{tp}$, eliminates these variables, negates the result and simplifies[5] the resulting formula and obtains $e_1 - \mathtt{outflow} \leq 0$. It then replaces e_1 with $\mathtt{inflow}(\mathtt{t})$, quantifies t universally and obtains the weakest constraint:

$(\varGamma_1)\quad \forall t.\, (\mathsf{inflow}(t) \leq \mathsf{outflow})$

for which (2) becomes unsatisfiable. A similar procedure yields the constraint

$(\varGamma_2)\quad \forall t.\, (\mathsf{lalarm} + \mathsf{inflow}(t) \leq \mathsf{loverflow})$ for (3).

Test runs for SEH-PILoT. We analyzed examples from mathematics, verification and wireless network theory. We used H-PILoT for testing satisfiability of the formulae; if the formulae were satisfiable SEH-PILoT was used for symbol elimination and generating constraints on the parameters. The table below provides some data on the size of the problems we analyzed and the time H-PILoT needed for hierarchical reduction and SEH-PILoT for symbol elimination.

Name	# clauses input	time H-PILoT (s)	# atoms (1)	# atoms (2)	time QE (ms)	# atoms (3)	# atoms (4)	Time (s)
Mathematics								
Case distinction Example 5.6 in [13]	4	0.23	16	12	0.81	34	8	2.5
Lipschitz	12	0.25	64	22	5.26	2925	3	9.5
Verification								
Water tank, -a without SLFQ	7	0.22	5	5	0.19	2	2	1.7
Water tank, no -a without SLFQ	6	0.10	6	6	0.08	2	2	0.8
Array sorted Example 4 in [9]	6	0.23	11	8	0.82	2	2	2.5
Maximum in array Example 4.13 [14]	7	0.22	11	6	0.8	1	1	3.0
Networks								
Graph class consist. Example 4 in [10]	3	0.22	3	3	0.79	1	1	2.4
Class inclusion (g_4)	19	0.24	111	22	0.90	20	4	2.5
Class inclusion (g_5) Example 8 in [10,8]	19	0.24	114	22	1.01	20	4	2.7

Table 1. Run on an Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, 8192 K-byte cache.
clauses is the number of clauses in the input to H-PILoT. # atoms refer to the number of atoms in the Redlog file generated by H-PILoT before (1) resp. after (2) simplification; resp. in the formula obtained after quantifier elimination before (3) resp. after (4) simplification.
For all examples (with exception of the water tank) simplification with SLFQ was used, which is responsible for a significant amount of the runtime.

[5] If SEH-PILoT is called with assumptions (-a) in the command line, redlog-simplification with assumptions is performed and the resulting formula is simpler.

References

1. C. W. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, and C. Tinelli. CVC4. In G. Gopalakrishnan and S. Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Proceedings*, LNCS 6806, pages 171–177. Springer, 2011.
2. N. Bjørner, L. de Moura, L. Nachmanson, and C. M. Wintersteiger. Programming Z3. In J. P. Bowen, Z. Liu, and Z. Zhang, editors, *Engineering Trustworthy Software Systems - 4th International School, SETSS 2018, Tutorial Lectures*, LNCS 11430, pages 148–201. Springer, 2019.
3. C. W. Brown. QEPCAD B: a system for computing with semi-algebraic sets via cylindrical algebraic decomposition. *SIGSAM Bull.*, 38(1):23–24, 2004.
4. A. Dolzmann and T. Sturm. REDLOG: computer algebra meets computer logic. *SIGSAM Bull.*, 31(2):2–9, 1997.
5. C. Ihlemann, S. Jacobs, and V. Sofronie-Stokkermans. On local reasoning in verification. In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Proceedings*, LNCS 4963, pages 265–281. Springer, 2008.
6. C. Ihlemann and V. Sofronie-Stokkermans. System description: H-PILoT. In R. A. Schmidt, editor, *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Proceedings*, LNCS 5663, pages 131–139. Springer, 2009.
7. C. Ihlemann and V. Sofronie-Stokkermans. On hierarchical reasoning in combinations of theories. In J. Giesl and R. Hähnle, editors, *Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Proceedings*, LNCS 6173, pages 30–45. Springer, 2010.
8. D. Peuter, P. Marohn, and V. Sofronie-Stokkermans. Symbol elimination for parametric second-order entailment problems (with applications to problems in wireless network theory). *CoRR*, https://arxiv.org/abs/2107.02333, 2021.
9. D. Peuter and V. Sofronie-Stokkermans. On invariant synthesis for parametric systems. In P. Fontaine, editor, *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Proceedings*, LNCS 11716, pages 385–405. Springer, 2019.
10. D. Peuter and V. Sofronie-Stokkermans. Symbol elimination and applications to parametric entailment problems. In B. Konev and G. Reger, editors, *Frontiers of Combining Systems - 13th International Symposium, FroCoS 2021, Proceedings*, LNCS 12941, pages 43–62. Springer, 2021.
11. V. Sofronie-Stokkermans. Hierarchic reasoning in local theory extensions. In R. Nieuwenhuis, editor, *Automated Deduction - CADE-20, 20th International Conference on Automated Deduction, Proceedings*, LNCS 3632, pages 219–234. Springer, 2005.
12. V. Sofronie-Stokkermans. On interpolation and symbol elimination in theory extensions. In N. Olivetti and A. Tiwari, editors, *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Proceedings*, LNCS 9706, pages 273–289. Springer, 2016.
13. V. Sofronie-Stokkermans. On interpolation and symbol elimination in theory extensions. *Log. Methods Comput. Sci.*, 14(3), 2018.
14. V. Sofronie-Stokkermans. Parametric systems: Verification and synthesis. *Fundam. Informaticae*, 173(2-3):91–138, 2020.