Solving Existentially Quantified Horn Clauses

Tewodros A. Beyene[1], Corneliu Popeea[1], and Andrey Rybalchenko[1,2]

[1] Technische Universität München
[2] Microsoft Research Cambridge

Abstract. Temporal verification of universal (i.e., valid for all computation paths) properties of various kinds of programs, e.g., procedural, multi-threaded, or functional, can be reduced to finding solutions for equations in form of universally quantified Horn clauses extended with well-foundedness conditions. Dealing with existential properties (e.g., whether there exists a particular computation path), however, requires solving forall-exists quantified Horn clauses, where the conclusion part of some clauses contains existentially quantified variables. For example, a deductive approach to CTL verification reduces to solving such clauses. In this paper we present a method for solving forall-exists quantified Horn clauses extended with well-foundedness conditions. Our method is based on a counterexample-guided abstraction refinement scheme to discover witnesses for existentially quantified variables. We also present an application of our solving method to automation of CTL verification of software, as well as its experimental evaluation.

1 Introduction

Temporal verification of universal, i.e., valid for all computation paths, properties of various kinds of programs is a success story. Various techniques, e.g., abstract domains [13], predicate abstraction [18, 22], or interpolation [26], provide a basis for efficient tools for the verification of such properties, e.g., Astree [5], Blast [22], CPAChecker [3], SatAbs [9], Slam [2], Terminator [12], or UFO [1]. To a large extent, the success of checkers of universal properties is determined by tremendous advances in the state-of-the-art in decision procedures for (universal) validity checking, i.e., advent of tools like MathSAT [6] or Z3 [15].

In contrast, advances in dealing with existential properties of programs, e.g., proving whether there exists a particular computation path, are still not on par with the maturity of verifiers for universal properties. Nevertheless, important first steps were made in proving existence of infinite program computations, see e.g. [16, 20, 29], even in proving existential (as well as universal) CTL properties [11]. Moreover, bounded model checking tools like CBMC [8] or Klee [7] can be very effective in proving existential reachability properties. All these initial achievements inspire further, much needed research on the topic.

In this paper, we present a method that can serve as a further building block for the verification of temporal existential (and universal) properties of programs. Our method solves forall-exists quantified Horn clauses extended with

N. Sharygina and H. Veith (Eds.): CAV 2013, LNCS 8044, pp. 869–882, 2013.

well-foundedness conditions. (The conclusion part of such clauses may contain existentially quantified variables.) The main motivation for the development of our method stems from an observation that verification conditions for existential temporal properties, e.g., generated by a deductive proof system for CTL [25], can be expressed by clauses in such form.

Our method, called E-HSF, applies a counterexample-guided refinement scheme to discover witnesses for existentially quantified variables. The refinement loop collects a global constraint that declaratively determines which witnesses can be chosen. The chosen witnesses are used to replace existential quantification, and then the resulting universally quantified clauses are passed to a solver for such clauses. At this step, we can benefit from emergent tools in the area of solving Horn clauses over decidable theories, e.g., HSF [19], μZ [23], or Duality [27]. Such a solver either finds a solution, i.e., a model for uninterpreted relations constrained by the clauses, or returns a counterexample, which is a resolution tree (or DAG) representing a contradiction. E-HSF turns the counterexample into an additional constraint on the set of witness candidates, and continues with the next iteration of the refinement loop. Notably, our refinement loop conjoins constraints that are obtained for all discovered counterexamples. This way E-HSF guarantees that previously handled counterexamples are not rediscovered and that a wrong choice of witnesses can be mended.

We applied our implementation of E-HSF to forall-exists quantified Horn clauses with well-foundedness conditions that we obtained by from a deductive proof system for CTL [25]. The experimental evaluation on benchmarks from [11] demonstrates the feasibility of our method.

2 Preliminaries

In this section we introduce preliminary definitions.

Constraints. Let $\mathcal{T}$ be a first-order theory in a given signature and $\models_{\mathcal{T}}$ be the entailment relation for $\mathcal{T}$. We write $v, v_0, v_1, \ldots$ and w to denote non-empty tuples of variables. We refer to a formula $c(v)$ over variables v from $\mathcal{T}$ as a constraint. Let *false* and *true* be an unsatisfiable and a valid constraint, respectively.

For example, let x, y, and z be variables. Then, $v = (x, y)$ and $w = (y, z)$ are tuples of variables. $x \leq 2$, $y \leq 1 \land x - y \leq 0$, and $f(x) + g(x, y) \leq 3 \lor z \leq 0$ are example constraints in the theory $\mathcal{T}$ of linear inequalities and uninterpreted functions, where f and g are uninterpreted function symbols. $y \leq 1 \land x - y \leq 0 \models_{\mathcal{T}} x \leq 2$ is an example of a valid entailment.

A binary relation is well-founded if it does not admit any infinite chains. A relation $\varphi(v, v')$ is disjunctively well-founded if it is included in a finite union of well-founded relations [31], i.e., if there exist well-founded $\varphi_1(v, v'), \ldots, \varphi_n(v, v')$ such that $\varphi(v, v') \models_{\mathcal{T}} \varphi_1(v, v') \lor \cdots \lor \varphi_n(v, v')$. For example, the relation $x \geq 0 \land x' \leq x - 1$ is well-founded, while the relation $(x \geq 0 \land x' \leq x - 1) \lor (y \leq 0 \land y' \geq y + 1)$ is disjunctively well-founded.

Queries and dwf-Predicates. We assume a set of uninterpreted predicate symbols $\mathcal{Q}$ that we refer to as query symbols. The arity of a query symbol is encoded in its name. We write q to denote a query symbol. Given q of a non-zero arity n and a tuple of variables v of length n, we define $q(v)$ to be a query. Furthermore, we introduce an interpreted predicate symbol *dwf* of arity one (*dwf* stands for disjunctive well-foundedness). Given a query $q(v, v')$ over tuples of variables with equal length, we refer to $\mathit{dwf}(q)$ as a *dwf*-predicate. For example, let $\mathcal{Q} = \{r, s\}$ be query symbols of arity one and two, respectively. Then, $r(x)$ and $s(x, y)$ are queries, and $\mathit{dwf}(s)$ is a *dwf*-predicate.

Forall-Exists Horn-Like Clauses. Let $h(v)$ range over queries over v, constraints over v, and existentially quantified conjunctions of queries and constraints with free variables in v. We define a forall-exists Horn-like clause to be either an implication $c(v_0) \wedge q_1(v_1) \wedge \cdots \wedge q_n(v_n) \rightarrow h(v)$ or a unit clause $\mathit{dwf}(q)$. The left-hand side of the implication is called the body, written as $\mathit{body}(v)$, and the right-hand side is called the head.

We give as example a set of forall-exists Horn-like clauses below:

$$x \geq 0 \rightarrow \exists y : x \geq y \wedge \mathit{rank}(x, y), \qquad \mathit{rank}(x, y) \rightarrow \mathit{ti}(x, y),$$
$$\mathit{ti}(x, y) \wedge \mathit{rank}(y, z) \rightarrow \mathit{ti}(x, z), \qquad \mathit{dwf}(\mathit{ti}).$$

These clauses represent an assertion over the interpretation of predicate symbols *rank* and *ti*.

Semantics of Forall-Exists Horn-Like Clauses. A set of clauses can be seen as an assertion over the queries that occur in the clauses.

We consider a function *ClauseSol* that maps each query $q(v)$ occurring in a given set of clauses into a constraint over v. Such a function is called a solution if the following two conditions hold. First, for each clause of the form $\mathit{body}(v) \rightarrow h(v)$ from the given set we require that replacing each query by the corresponding constraint assigned by *ClauseSol* results in a valid entailment. That is, we require $\mathit{body}(v)\mathit{ClauseSol} \models_{\mathcal{T}} h(v)\mathit{ClauseSol}$, where the juxtaposition represents application of substitution. Second, for each clause of the form $\mathit{dwf}(q)$ we require that the constraint assigned by *ClauseSol* to q represents a disjunctively well-founded relation. Let $\models_{\mathcal{Q}}$ be the corresponding satisfaction relation, i.e., $\mathit{ClauseSol} \models_{\mathcal{Q}} \mathit{Clauses}$ if *ClauseSol* is a solution for the given set of clauses.

For example, the previously presented set of clauses, say *Clauses*, has a solution *ClauseSol* such that $\mathit{ClauseSol}(\mathit{rank}(x, y)) = \mathit{ClauseSol}(\mathit{ti}(x, y)) = (x \geq 0 \wedge y \geq x - 1)$. To check $\mathit{ClauseSol} \models_{\mathcal{Q}} \mathit{Clauses}$ we consider the validity of the following implications:

$$x \geq 0 \rightarrow \exists y : x \geq y \wedge x \geq 0 \wedge y \leq x - 1,$$
$$x \geq 0 \wedge y \leq x - 1 \rightarrow x \geq 0 \wedge y \leq x - 1,$$
$$x \geq 0 \wedge y \leq x - 1 \wedge y \geq 0 \wedge z \leq y - 1 \rightarrow x \geq 0 \wedge z \leq x - 1.$$

and the fact that $\mathit{ClauseSol}(\mathit{ti}(x, y)) = (x \geq 0 \wedge y \leq x - 1)$ is a (disjunctively) well-founded relation.

Solving Horn-like Clauses without Existential Quantification. We assume an algorithm HSF for solving Horn-like clauses whose heads do not contain any existential quantification. This algorithm computes a solution *ClauseSol* when it exists. There already exist such algorithms as well as their efficient implementations that are based on predicate abstraction and interpolation [19], as well as interpolation based approximation [27].

3 Preliminaries

In this section we introduce preliminary definitions.

Constraints. Let $\mathcal{T}$ be a first-order theory in a given signature and $\models_{\mathcal{T}}$ be the entailment relation for $\mathcal{T}$. We write $v, v_0, v_1, \dots$ and w to denote non-empty tuples of variables. We refer to a formula $c(v)$ over variables v from $\mathcal{T}$ as a constraint. Let *false* and *true* be an unsatisfiable and a valid constraint, respectively.

For example, let x, y, and z be variables. Then, $v = (x, y)$ and $w = (y, z)$ are tuples of variables. $x \leq 2$, $y \leq 1 \wedge x - y \leq 0$, and $f(x) + g(x, y) \leq 3 \vee z \leq 0$ are example constraints in the theory $\mathcal{T}$ of linear inequalities and uninterpreted functions, where f and g are uninterpreted function symbols. $y \leq 1 \wedge x - y \leq 0 \models_{\mathcal{T}} x \leq 2$ is an example of a valid entailment.

A binary relation is well-founded if it does not admit any infinite chains. A relation $\varphi(v, v')$ is disjunctively well-founded if it is included in a finite union of well-founded relations [31], i.e., if there exist well-founded $\varphi_1(v, v'), \dots, \varphi_n(v, v')$ such that $\varphi(v, v') \models_{\mathcal{T}} \varphi_1(v, v') \vee \dots \vee \varphi_n(v, v')$. For example, the relation $x \geq 0 \wedge x' \leq x - 1$ is well-founded, while the relation $(x \geq 0 \wedge x' \leq x - 1) \vee (y \leq 0 \wedge y' \geq y + 1)$ is disjunctively well-founded.

Queries and dwf-predicates. We assume a set of uninterpreted predicate symbols $\mathcal{Q}$ that we refer to as query symbols. The arity of a query symbol is encoded in its name. We write q to denote a query symbol. Given q of a non-zero arity n and a tuple of variables v of length n, we define $q(v)$ to be a query. Furthermore, we introduce an interpreted predicate symbol *dwf* of arity one (*dwf* stands for disjunctive well-foundedness). Given a query $q(v, v')$ over tuples of variables with equal length, we refer to $\mathit{dwf}(q)$ as a *dwf*-predicate. For example, let $\mathcal{Q} = \{r, s\}$ be query symbols of arity one and two, respectively. Then, $r(x)$ and $s(x, y)$ are queries, and $\mathit{dwf}(s)$ is a *dwf*-predicate.

Forall-exists Horn-like Clauses. Let $h(v)$ range over queries over v, constraints over v, and existentially quantified conjunctions of queries and constraints with free variables in v. We define a forall-exists Horn-like clause to be either an implication $c(v_0) \wedge q_1(v_1) \wedge \dots \wedge q_n(v_n) \rightarrow h(v)$ or a unit clause $\mathit{dwf}(q)$. The left-hand side of the implication is called the body, written as $\mathit{body}(v)$, and the right-hand side is called the head.

We give as example a set of forall-exists Horn-like clauses below:

$$x \geq 0 \rightarrow \exists y : x \geq y \wedge \mathit{rank}(x, y), \qquad \mathit{rank}(x, y) \rightarrow \mathit{ti}(x, y),$$
$$\mathit{ti}(x, y) \wedge \mathit{rank}(y, z) \rightarrow \mathit{ti}(x, z), \qquad \mathit{dwf}(\mathit{ti}).$$

These clauses represent an assertion over the interpretation of predicate symbols *rank* and *ti*.

Semantics of Forall-exists Horn-like Clauses. A set of clauses can be seen as an assertion over the queries that occur in the clauses.

We consider a function *ClauseSol* that maps each query $q(v)$ occurring in a given set of clauses into a constraint over v. Such a function is called a solution if the following two conditions hold. First, for each clause of the form $body(v) \rightarrow h(v)$ from the given set we require that replacing each query by the corresponding constraint assigned by *ClauseSol* results in a valid entailment. That is, we require $body(v) ClauseSol \models_{\mathcal{T}} h(v) ClauseSol$, where the juxtaposition represents application of substitution. Second, for each clause of the form $dwf(q)$ we require that the constraint assigned by *ClauseSol* to q represents a disjunctively well-founded relation. Let $\models_{\mathcal{Q}}$ be the corresponding satisfaction relation, i.e., $ClauseSol \models_{\mathcal{Q}} Clauses$ if *ClauseSol* is a solution for the given set of clauses.

For example, the previously presented set of clauses, say *Clauses*, has a solution *ClauseSol* such that $ClauseSol(rank(x, y)) = ClauseSol(ti(x, y)) = (x \geq 0 \wedge y \geq x - 1)$. To check $ClauseSol \models_{\mathcal{Q}} Clauses$ we consider the validity of the following implications:

$$
\begin{aligned}
&x \geq 0 \rightarrow \exists y : x \geq y \wedge x \geq 0 \wedge y \leq x - 1, \\
&x \geq 0 \wedge y \leq x - 1 \rightarrow x \geq 0 \wedge y \leq x - 1, \\
&x \geq 0 \wedge y \leq x - 1 \wedge y \geq 0 \wedge z \leq y - 1 \rightarrow x \geq 0 \wedge z \leq x - 1.
\end{aligned}
$$

and the fact that $ClauseSol(ti(x, y)) = (x \geq 0 \wedge y \leq x - 1)$ is a (disjunctively) well-founded relation.

Solving Horn-like Clauses without Existential Quantification. We assume an algorithm HSF for solving Horn-like clauses whose heads do not contain any existential quantification. This algorithm computes a solution *ClauseSol* when it exists. There already exist such algorithms as well as their efficient implementations that are based on predicate abstraction and interpolation [19], as well as interpolation based approximation [27].

4 Example of Applying E-HSF

We consider the following set *Clauses* that encodes a check whether a program with the variables $v = (x, y)$, an initial condition $init(v) = (y \geq 1)$ and a transition relation $next(v, v') = (x' = x + y)$ satisfies a CTL property $EF\ dst(v)$, where $dst(v) = (x \geq 0)$.

$$
\begin{gathered}
init(v) \rightarrow inv(v), \quad inv(v) \wedge \neg dst(v) \rightarrow \exists v' : next(v, v') \wedge inv(v') \wedge rank(v, v'), \\
rank(v, v') \rightarrow ti(v, v'), \quad ti(v, v') \wedge rank(v', v'') \rightarrow ti(v, v''), \quad dwf(ti).
\end{gathered}
$$

Here, $inv(v)$, $rank(v, v')$, and $ti(v, v')$ are unknown predicates that we need to solve for. The predicate $inv(v)$ corresponds to states reachable during program execution, while the second row of clauses ensures that $rank(v, v')$ is a well-founded relation [31].

We start the execution of E-HSF from Figure **??** by applying Skolemize to eliminate the existential quantification. As a result, the clause that contains existential quantification is replaced by the following four clauses that contain an application of a Skolem relation $rel(v, v')$ introduced by Skolemize as well as an introduction of a lower bound on the guard $grd(v)$ of the Skolem relation:

$$\begin{aligned}
&inv(v) \wedge \neg dst(v) \wedge rel(v, v') \rightarrow next(v, v'),\\
&inv(v) \wedge \neg dst(v) \wedge rel(v, v') \rightarrow inv(v'),\\
&inv(v) \wedge \neg dst(v) \wedge rel(v, v') \rightarrow rank(v, v'),\\
&inv(v) \wedge \neg dst(v) \rightarrow grd(v).
\end{aligned}$$

Furthermore, this introduction is recorded as $Rels = \{rel\}$ and $Grds = \{grd\}$. Note that we replaced a conjunction in the head of a clause by a conjunction of corresponding clauses.

First Candidate for Skolem Relation. Next, we proceed with the execution of E-HSF. We initialise *Constraint* with the assertion *true*. Then, we generate a set of Horn clauses *Defs* that provides initial candidates for the Skolem relation and its guard as follows: $Defs = \{true \rightarrow rel(v, v'),\ grd(v) \rightarrow true\}$. Now, we apply the solving algorithm HSF for quantifier free Horn clauses on the set of clauses that contains the result of Skolemization and the initial candidates in *Defs*, i.e., we give to HSF the following clauses:

$$\begin{array}{ll}
init(v) \rightarrow inv(v), & rank(v, v') \rightarrow ti(v, v'),\\
inv(v) \wedge \neg dst(v) \wedge rel(v, v') \rightarrow next(v, v'), & ti(v, v') \wedge rank(v', v'') \rightarrow ti(v, v''),\\
inv(v) \wedge \neg dst(v) \wedge rel(v, v') \rightarrow inv(v'), & dwf(ti),\\
inv(v) \wedge \neg dst(v) \wedge rel(v, v') \rightarrow rank(v, v'), & true \rightarrow rel(v, v'),\\
inv(v) \wedge \neg dst(v) \rightarrow grd(v), & grd(v) \rightarrow true.
\end{array}$$

HSF returns an error derivation that witnesses a violation of the given set of clauses. This derivation represents an unfolding of clauses in $Skolemized \cup Defs$ that yields a relation for $ti(v, v')$ that is not disjunctively well-founded. To represent the unfolding, HSF uses a form of static single assignment (SSA) that is applied to predicate symbols, where each unfolding step introduces a fresh predicate symbol that is recorded by the function Sym. We obtain the clauses *Cex* consisting of

$$init(v) \rightarrow q_1(v), \quad q_1(v) \wedge \neg dst(v) \wedge q_2(v, v') \rightarrow next(v, v'), \quad true \rightarrow q_2(v, v')$$

together with the following bookkeeping of the SSA renaming: $\textsc{Sym}(q_1) = inv$ and $\textsc{Sym}(q_2) = rel$. From *Cex* we extract the clause *CexDefs* that provides the candidate for the Skolem relation. We obtain $CexDefs = \{true \rightarrow q_2(v, v')\}$, since $\textsc{Sym}(q_2) = rel$ and hence $\textsc{Sym}(q_2) \in Rels$.

We analyse the counterexample clauses by applying resolution on $\mathit{Cex} \setminus \mathit{CexDefs}$. The corresponding resolution tree is shown below (literals selected for resolution are boxed):

$$\frac{\mathit{init}(v) \to \boxed{q_1(v)} \qquad \boxed{q_1(v)} \land \neg \mathit{dst}(v) \land q_2(v,v') \to \mathit{next}(v,v')}{\mathit{init}(v) \land \neg \mathit{dst}(v) \land q_2(v,v') \to \mathit{next}(v,v')}$$

Note that $q_2(v,v')$ was not eliminated, since the clause $\mathit{true} \to q_2(v,v')$ was not given to Resolve as input. The result of applying Resolve is the clause $\mathit{init}(v) \land \neg \mathit{dst}(v) \land q_2(v,v') \to \mathit{next}(v,v')$. We assign the conjunction $\mathit{init}(v) \land \neg \mathit{dst}(v)$ to *body* and $\mathit{next}(v,v')$ to *head*, respectively.

Now we iterate i through the singleton set $\{1\}$, which is determined by the fact that the above clause contains only one unknown predicate on the left-hand side. We apply RelT on $\mathrm{Sym}(q_2)$ and set the free variables in the result to (v,v'). This yields a template $v' = Tv + t$ for the Skolem relation $\mathit{rel}(v,v')$. Here, T is a matrix of unknown coefficients $\left(\begin{smallmatrix} t_{xx} & t_{xy} \\ t_{yx} & t_{yy} \end{smallmatrix}\right)$, and t is a vector of unknown free coefficient (t_x, t_y). In other words, our template represents a conjunction of two equality predicates $x' = t_{xx}x + t_{xy}y + t_x$ and $y' = t_{yx}x + t_{yy}y + t_y$. We conjoin this template with *body* and obtain $\mathit{body} = (v' = Tv + t \land \mathit{init}(v) \land \neg \mathit{dst}(v))$. Since *head* is not required to be disjunctively well-founded, E-HSF proceeds with the generation of constraints over template parameters.

We apply EncodeValidity on the following implication:

$$x' = t_{xx}x + t_{xy}y + t_x \land y' = t_{yx}x + t_{yy}y + t_y \land y \geq 1 \land \neg x \geq 0 \to x' = x + y \ .$$

This implication is valid if the following constraint returned by EncodeValidity is satisfiable.

$$\exists \overbrace{\lambda_1, \lambda_2, \lambda_3, \lambda_4}^{\lambda}, \overbrace{\mu_1, \mu_2, \mu_3, \mu_4}^{\mu} : \lambda_3 \geq 0 \land \lambda_4 \geq 0 \land \mu_3 \geq 0 \land \mu_4 \geq 0 \land$$
$$\begin{pmatrix} \lambda \\ \mu \end{pmatrix} \begin{pmatrix} t_{xx} & t_{xy} & -1 & 0 \\ t_{yx} & t_{yy} & 0 & -1 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} -1 & -1 & 1 & 0 \\ 1 & 1 & -1 & 0 \end{pmatrix} \land \begin{pmatrix} \lambda \\ \mu \end{pmatrix} \begin{pmatrix} -t_x \\ -t_y \\ -1 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

This constraint requires that the right-hand side on the implication is obtained as a linear combination of the (in)equalities on the left-hand side of the implication. We conjoin the above constraint with *Constraint*.

We apply an SMT solver to compute a satisfying valuation of template parameters occurring in *Constraint* and obtain:

t_{xx}	t_{xy}	t_x	t_{yx}	t_{yy}	t_y
1	1	0	0	0	10

By applying *CexSol* on the template $v' = Tv + t$, which is the result of $\mathrm{RelT}(\mathit{rel})(v,v')$, we obtain the conjunction $x' = x + y \land y' = 10$. In this example, we assume that the template $\textsc{GrdT}(\mathit{grd})(v)$ is equal to *true*. Hence, we modify the clauses that record the current candidate for $\mathit{rel}(v, v')$ and $\mathit{grd}(v)$ as follows:

$$\mathit{Defs} = \{x' = x + y \wedge y' = 10 \rightarrow \mathit{rel}(v, v'),\ \ \mathit{grd}(v) \rightarrow \mathit{true}\}$$

Now we proceed with the next iteration of the main loop in E-HSF.

Second Candidate for Skolem Relation. The second iteration in E-HSF uses *Defs* and *Constraint* as determined during the first iteration. We apply HSF on $\mathit{Skolemized} \cup \mathit{Defs}$ and obtain an error derivation *Cex* consisting of the clauses

$$\begin{aligned} &\mathit{init}(v) \wedge q_1(v), \quad q_1(v) \wedge \neg \mathit{dst}(v) \wedge q_2(v, v') \rightarrow q_3(v, v'),\\ &x' = x + y \wedge y' = 10 \rightarrow q_2(v, v'), \quad q_3(v, v') \rightarrow q_4(v, v'), \end{aligned}$$

together with the function $\textsc{Sym}$ such that $\textsc{Sym}(q_1) = \mathit{inv}$, $\textsc{Sym}(q_2) = \mathit{rel}$, $\textsc{Sym}(q_3) = \mathit{rank}$, and $\textsc{Sym}(q_4) = \mathit{ti}$. From *Cex* we extract $\mathit{CexDefs} = \{x' = x + y \wedge y' = 10 \rightarrow q_2(v, v')\}$ since $\textsc{Sym}(q_2) \in \mathit{Rels}$. We apply $\textsc{Resolve}$ on $\mathit{Cex} \setminus \mathit{CexDefs}$ and obtain:

$$\mathit{init}(v) \wedge \neg \mathit{dst}(v) \wedge q_2(v, v') \rightarrow q_4(v, v') \ .$$

As seen at the first iteration, we have $\textsc{RelT}(\mathit{rel})(v, v') = (v' = Tv + t)$. Hence we have $\mathit{body} = (\mathit{init}(v) \wedge \neg \mathit{dst}(v) \wedge v' = Tv + t)$.

Since $\textsc{Sym}(q_4) = \mathit{ti}$ and $\mathit{dwf}(\mathit{ti}) \in \mathit{Skolemized}$, the error derivation witnesses a violation of disjunctive well-foundedness. Hence, by applying $\textsc{BoundT}$ and $\textsc{DecreaseT}$ we construct templates $\mathit{bound}(v)$ and $\mathit{decrease}(v, v')$ corresponding to a bound and decrease condition over the program variables, respectively.

$$\begin{aligned} \mathit{bound}(v) &= (r_x x + r_y y \geq r_0) \ ,\\ \mathit{decrease}(v, v') &= (r_x x' + r_y y' \leq r_x x + r_y y - 1) \ . \end{aligned}$$

Finally, we set *head* to the conjunction $r_x x + r_y y \geq r_0 \wedge r_x x' + r_y y' \leq r_x x + r_y y - 1$.

By $\textsc{EncodeValidity}$ on the implication $\mathit{body} \rightarrow \mathit{head}$ we obtain the constraint

$$\exists \overbrace{\lambda_1, \lambda_2, \lambda_3, \lambda_4}^{\lambda}, \overbrace{\mu_1, \mu_2, \mu_3, \mu_4}^{\mu} : \lambda_3 \geq 0 \wedge \lambda_4 \geq 0 \wedge \mu_3 \geq 0 \wedge \mu_4 \geq 0 \wedge$$
$$\begin{pmatrix} \lambda \\ \mu \end{pmatrix} \begin{pmatrix} t_{xx} & t_{xy} & -1 & 0 \\ t_{yx} & t_{yy} & 0 & -1 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} -r_x & -r_y & 0 & 0 \\ -r_x & -r_y & r_x & r_y \end{pmatrix} \wedge \begin{pmatrix} \lambda \\ \mu \end{pmatrix} \begin{pmatrix} -t_x \\ -t_y \\ -1 \\ -1 \end{pmatrix} = \begin{pmatrix} -r_0 \\ -1 \end{pmatrix} .$$

We add the above constraint as an additional conjunct to *Constraint*. That is, *Constraint* is strengthened during each iteration.

We apply the SMT solver to compute a valuation template parameters that satisfies *Constraint*. We obtain the following solution *CexSol*:

t_{xx}	t_{xy}	t_x	t_{yx}	t_{yy}	t_y
1	0	1	0	0	1

The corresponding values of r and r_0 are $(-1, 0)$ and -1, which lead to the bound $-x \geq 1$ and the decrease relation $-x' \leq -x - 1$. By applying *CexSol* on the template $v' = Tv + t$ we obtain the conjunction $x' = x + 1 \land y' = 1$. Note that the solution for $rel(v, v')$ obtained at this iteration is not compatible with the solution obtained at the first iteration, i.e., the intersection of the respective Skolem relations is empty. Finally, we modify *Defs* according to *CexSol* and obtain:

$$Defs = \{x' = x + 1 \land y' = 1 \rightarrow rel(v, v'),\ grd \rightarrow true\}$$

Now we proceed with the next iteration of the main loop in E-HSF. At this iteration the application of HSF returns a solution *ClauseSol* such that

$$\begin{aligned}
ClauseSol(inv(v)) &= (y \geq 1)\ , \\
ClauseSol(rel(v)) &= (x' = x + 1 \land y' = 1)\ , \\
ClauseSol(rank(v, v')) &= (x \leq -1 \land x' \geq x + 1)\ , \\
ClauseSol(ti(v, v')) &= (x \leq -1 \land x' \geq x + 1)\ .
\end{aligned}$$

Thus, the algorithm E-HSF finds a solution to the original set of forall-exists Horn clauses (and hence proves the program satisfies the CTL property).

5 Verifying CTL Properties Using E-HSF

In this section we show how E-HSF can be used for automatically proving CTL properties of programs. We utilize a standard reduction step from CTL properties to existentially quantified Horn-like clauses with well-foundedness conditions, see e.g. [25]. Here, due to space constraints, we only illustrate the reduction, using examples and refer to [25] for details of the CTL proof system.

We consider a program over variables v, with an initial condition given by an assertion $init(v)$, and a transition relation $next(v, v')$. Given a CTL property, we generate Horn-like clauses such that the property is satisfied if and only if the set of clauses is satisfiable.

The generation proceeds in two steps. The first step decomposes the property into sub-properties by following the nesting structure of the path quantifiers that occur in the property. As a result we obtain a set of simple CTL formulas that contain only one path quantifier. Each property is accompanied by a predicate that represents a set of program states that needs to be discovered.

As an example, we present the decomposition of $(init(v), next(v, v')) \models_{CTL} AG(EF(dst(v)))$, where $dst(v)$ is a first-order assertion over v. Since $EF(dst(v))$ is a sub-formula with a path quantifier as the outmost symbol, we introduce a fresh predicate $p(v)$ that is used to replace $EF(dst(v))$. Furthermore, we require that every computation that starts in a state described by $p(v)$ satisfies $EF(dst(v))$. Since the resulting CTL formulas do not have any nested path quantifiers we stop the decomposition process. The original verification question is equivalent to the existence of $p(v)$ such that $(init(v), next(v, v')) \models_{CTL} AG(p(v))$ and $(p(v), next(v, v')) \models_{CTL} EF(dst(v))$.

At the second step we consider each of the verification sub-questions obtained by decomposing the property and generate Horn-like clauses that constrain auxiliary sets and relations over program states. For $(init(v), next(v, v')) \models_{CTL} AG(p(v))$ we obtain the following clauses over an auxiliary predicate $inv_1(v)$:

$$init(v) \rightarrow inv_1(v), \quad inv_1(v) \wedge next(v, v') \rightarrow inv_1(v'), \quad inv_1(v) \rightarrow p(v).$$

Due to the existential path quantifier in $(p(v), next(v, v')) \models_{CTL} EF(dst(v))$ we obtain clauses that contain existential quantification. We deal with the eventuality by imposing a well-foundedness condition. The resulting clauses over auxiliary $inv_2(v)$, $rank(v, v')$, and $ti(v, v')$ are below (note that $dst(v)$ is a constraint, and hence can occur under negation).

$$p(v) \rightarrow inv_2(v), \quad inv_2(v) \wedge \neg dst(v) \rightarrow \exists v' : next(v, v') \wedge rank(v, v'),$$
$$rank(v, v') \rightarrow ti(v, v'), \quad ti(v, v') \wedge rank(v, v') \rightarrow ti(v, v''), \quad dwf(ti).$$

Finally, the above clauses have a solution for $inv_1(v)$, $p(v)$, $inv_2(v)$, $rank(v, v')$, and $ti(v, v')$ if and only if $(init(v), next(v, v')) \models_{CTL} AG(EF(dst(v)))$. Then, we apply E-HSF as a solver.

6 Experiments

In this section we present our implementation of E-HSF and its experimental evaluation on proving universal and existential CTL properties of programs.

Our implementation relies on HSF [19] to solve universally-quantified Horn clauses over linear inequalities (see line 4 in Figure **??**) and on the Z3 solver [15] at line 19 in Figure **??** to solve (possibly non-linear) constraints. The input to our tool is a transition system described using Prolog facts $init(v)$ and $next(v, v')$, as well as forall-exists Horn clauses corresponding to the CTL property to be proved or disproved.

We run E-HSF on the examples from industrial code from [11, Figure 7]: `OS frag.1`, `OS frag.2`, `OS frag.3`, `OS frag.4`, `OS frag.5`, `PgSQL arch` and `S/W Updates`. For each pair of a program and CTL property ϕ , we generated two verification tasks: prove ϕ and prove $\neg\phi$. The existence of a proof for a property ϕ implies that $\neg\phi$ is violated by the same program. (Similarly, a proof for $\neg\phi$ implies that ϕ is violated by the same program.)

GrdT and RelT are provided by the user and need to satisfy Equation 1. Currently, this condition is not checked by the implementation, but could be done for linear templates using quantifier elimination techniques. For our examples, linear templates are sufficiently expressive. We use $\text{RelT}(next)(v, v') = (next(v, v') \wedge w' = Tv + t \wedge Gv \leq g)$ and $\text{GrdT}(next)(v, v') = (Gv \leq g \wedge \exists v' : next(v, v'))$, where w is a subset of v that is left unconstrained by $next(v, v')$. Such w' are explicitly marked in the original benchmark programs using names `rho1`, `rho2`, ... For direct comparison with the results from [11], we used template functions corresponding to the `rho`-variables. The quantifier

Table 1. Evaluation of E-HSF on industrial benchmarks from [11]. Each "Name" column gives the corresponding program number in [11, Figure 7]. For P12, E-HSF returns different results compared to [11]. For P26, P27 and P28 both properties ϕ and $\neg\phi$ are satisfied only for some initial states. (Neither ϕ nor $\neg\phi$ hold for these programs.)

Program	Property ϕ	$\models_{CTL} \phi$			$\models_{CTL} \neg\phi$		
		Result	Time	Name	Result	Time	Name
P1	$AG(a = 1 \rightarrow AF(r = 1))$	✓	1.2s	1	×	2.7s	29
P2	$EF(a = 1 \wedge EG(r \neq 5))$	✓	0.6s	30	×	5.2s	2
P3	$AG(a = 1 \rightarrow EF(r = 1))$	✓	4.8s	3	×	0.1s	31
P4	$EF(a = 1 \wedge AG(r \neq 1))$	✓	0.6s	32	×	0.4s	4
P5	$AG(s = 1 \rightarrow AF(u = 1))$	✓	6.1s	5	×	0.2s	33
P6	$EF(s = 1 \wedge EG(u \neq 1))$	✓	1.4s	34	×	3.6s	6
P7	$AG(s = 1 \rightarrow EF(u = 1))$	✓	12.9s	7	×	0.2s	35
P8	$EF(s = 1 \wedge AG(u \neq 1))$	✓	44.7s	36	×	3.8s	8
P9	$AG(a = 1 \rightarrow AF(r = 1))$	✓	51.3s	9	×	120.0s	37
P10	$EF(a = 1 \wedge EG(r \neq 1))$	✓	132.0s	38	×	45.9s	10
P11	$AG(a = 1 \rightarrow EF(r = 1))$	✓	67.6s	11	×	3.9s	39
P12	$EF(a = 1 \wedge AG(r \neq 1))$	✓	67.9s	12	×	3.8s	40
P13	$AF(io = 1) \vee AF(ret = 1)$	✓	37m54s	13	T/O	-	41
P14	$EG(io \neq 1) \wedge EG(ret \neq 1)$	T/O	-	42	×	136.6s	14
P15	$EF(io = 1) \wedge EF(ret = 1)$	T/O	-	15	×	1.4s	43
P16	$AG(io \neq 1) \vee AG(ret \neq 1)$	✓	0.1s	44	×	874.5s	16
P17	$AG(AF(w \geq 1))$	✓	3.0s	17	×	0.1s	45
P18	$EF(EG(w < 1)$	✓	0.5s	46	×	3.5s	18
P19	$AG(EF(w \geq 1))$	✓	3.3s	19	×	0.1s	47
P20	$EF(AG(w < 1)$	✓	0.7s	48	×	0.1s	20
P21	$AG(AF(w = 1)$	✓	2.8s	21	×	0.1s	49
P22	$EF(EG(w \neq 1)$	✓	2.2s	50	×	5.0s	22
P23	$AG(EF(w = 1)$	✓	4.5s	23	×	0.1s	51
P24	$EF(AG(w \neq 1)$	✓	3.4s	52	×	0.7s	24
P25	$c > 5 \rightarrow AF(r > 5)$	✓	3.2s	25	×	0.1s	53
P26	$c > 5 \wedge EG(r \leq 5)$	×	0.1s	54	×	1.3s	26
P27	$c > 5 \rightarrow EF(r > 5)$	×	0.2s	27	×	0.1s	55
P28	$c > 5 \wedge AG(r \leq 5)$	×	0.1s	56	×	0.3s	28

elimination in $\exists v' : \mathit{next}(v, v')$ can be automated for the theory of linear arithmetic. For dealing with well-foundedness we use linear ranking functions, and hence corresponding linear templates for DecreaseT and BoundT.

We report the results in Table 1. Columns 3 and 6 show ✓ marks for the cases where E-HSF was able to find a solution, i.e., prove the CTL property. See Columns 4 and 7 for the time spent on finding solutions. E-HSF is able to find proofs for all the correct programs except for P14 and P15 that correspond to Windows frag.4. Currently, E-HSF models the control flow symbolically using a program counter variable, which is most likely the reason for not succeeding on P14 and P15. Efficient treatment of control flow along the lines of explicit analysis as performed in the CPAchecker framework could lead to significant improvements for dealing with programs with large control-flow graphs [4].

For cases where the property contains more than one path quantifier and the top-most temporal quantifier is F or U, our implementation generates non-Horn clauses following the proof system from [25]. While a general algorithm for solving non-Horn clauses is beyond the scope of this paper, we used a simple heuristic to seed solutions for queries appearing under the negation operator. For example, for the verification task obtained from proving ϕ for P2, we used the solution $(a = 1 \wedge r \neq 5)$ for the query corresponding to the nesting structure of ϕ. This solution is obtained as a conjunction of the atomic constraints from ϕ.

7 Related Work

Our work is inspired by a recent approach to CTL verification of programs [11]. The main similarity lies in the use of a refinement loop to discover witnesses for resolving non-determinism/existentially quantified variables. The main difference lies in the way candidate witnesses are selected. While [11] refines witnesses, i.e., the non-determinism in witness relations monotonically decreases at each iteration, E-HSF can change witness candidates arbitrarily (yet, subject to the global constraint). Thus, our method can backtrack from wrong choices in cases when [11] needs to give up.

E-HSF generalizes solving methods for universally quantified Horn clauses over decidable theories, e.g. [19, 23, 27]. Our approach relies on the templates for describing the space of candidate witnesses. Computing witnesses using a generalisation approach akin to PDR [23] is an interesting alternative to explore in future work.

Template based synthesis of invariants and ranking functions is a prominent technique for dealing with universal properties, see e.g. [10, 21, 30, 33]. E-HSF implementation of EncodeValidity supporting linear arithmetic inequalities is directly inspired by these techniques, and puts them to work for existential properties.

Decision procedures for quantified propositional formulas on bit as well as word level [24,34] rely on iteration and refinement for the discovery of witnesses. The possibility of integration of QBF solvers as an implementation of EncodeValidity is an interesting avenue for future research.

Some formulations of proof systems for mu-calculus, e.g., [14] and [28], could be seen as another source of forall-exists clauses (to pass to E-HSF). Compared to the XSB system [14] that focuses on finite state systems, E-HSF aims at infinite state systems and employs a CEGAR-based algorithm. XSB's extensions for infinite state systems are rather specific, e.g., data-independent systems, and do not employ abstraction refinement techniques. Finally, we remark that abstraction-based methods, like ours, can be complemented with program specialization-based methods for verification of CTL properties [17].

8 Conclusion

Verification conditions for proving existential temporal properties of programs can be represented using existentially quantified Horn-like clauses. In this paper we presented a counterexample guided method for solving such clauses, which can compute witnesses to existentially quantified variables in form of linear arithmetic expressions. By aggregating constraints on witness relations across different counterexamples our method can recover from wrong choices. We leave the evaluation of applicability of our method for other problems requiring witness computation, e.g., software synthesis or game solving to future work.

Acknowledgements. We thank Byron Cook and Eric Koskinen for valuable discussion and for generously making their benchmarks available. This research was supported in part by ERC project 308125 VeriSynth and by the DFG Graduiertenkolleg 1480 (PUMA).

References

1. Albarghouthi, A., Li, Y., Gurfinkel, A., Chechik, M.: UFO: A framework for abstraction- and interpolation-based software verification. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 672–678. Springer, Heidelberg (2012)
2. Ball, T., Rajamani, S.K.: The SLAM project: debugging system software via static analysis. In: POPL (2002)
3. Beyer, D., Keremoglu, M.E.: CPAchecker: A tool for configurable software verification. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 184–190. Springer, Heidelberg (2011)
4. Beyer, D., Löwe, S.: Explicit-state software model checking based on CEGAR and interpolation. In: Cortellessa, V., Varró, D. (eds.) FASE 2013. LNCS, vol. 7793, pp. 146–162. Springer, Heidelberg (2013)
5. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: A static analyzer for large safety-critical software. In: PLDI (2003)
6. Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R.: The MathSAT 4 SMT solver. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 299–303. Springer, Heidelberg (2008)
7. Cadar, C., Dunbar, D., Engler, D.R.: KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In: OSDI (2008)
8. Clarke, E., Kroning, D., Lerda, F.: A tool for checking ANSI-C programs. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 168–176. Springer, Heidelberg (2004)
9. Clarke, E., Kroning, D., Sharygina, N., Yorav, K.: SATABS: SAT-based predicate abstraction for ANSI-C. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 570–574. Springer, Heidelberg (2005)
10. Colón, M.A., Sankaranarayanan, S., Sipma, H.B.: Linear invariant generation using non-linear constraint solving. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 420–432. Springer, Heidelberg (2003)
11. Cook, B., Koskinen, E.: Reasoning about nondeterminism in programs. In: PLDI (2013)
12. Cook, B., Podelski, A., Rybalchenko, A.: Termination proofs for systems code. In: PLDI (2006)
13. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL (1977)

14. Cui, B., Dong, Y., Du, X., Narayan Kumar, K., Ramakrishnan, C.R., Ramakrishnan, I.V., Roychoudhury, A., Smolka, S.A., Warren, D.S.: Logic programming and model checking. In: Palamidessi, C., Meinke, K., Glaser, H. (eds.) ALP 1998 and PLILP 1998. LNCS, vol. 1490, pp. 1–20. Springer, Heidelberg (1998)
15. de Moura, L., Bjørner, N.S.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
16. Emmes, F., Enger, T., Giesl, J.: Proving non-looping non-termination automatically. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 225–240. Springer, Heidelberg (2012)
17. Fioravanti, F., Pettorossi, A., Proietti, M., Senni, V.: Generalization strategies for the verification of infinite state systems. Theory and Practice of Logic Programming 13, 175–199 (2013)
18. Graf, S., Saïdi, H.: Construction of abstract state graphs with PVS. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, Springer, Heidelberg (1997)
19. Grebenshchikov, S., Lopes, N.P., Popeea, C., Rybalchenko, A.: Synthesizing software verifiers from proof rules. In: PLDI (2012)
20. Gupta, A., Henzinger, T.A., Majumdar, R., Rybalchenko, A., Xu, R.-G.: Proving non-termination. In: POPL (2008)
21. Gupta, A., Majumdar, R., Rybalchenko, A.: From tests to proofs. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 262–276. Springer, Heidelberg (2009)
22. Henzinger, T.A., Jhala, R., Majumdar, R., McMillan, K.L.: Abstractions from proofs. In: POPL (2004)
23. Hoder, K., Bjørner, N., de Moura, L.: μZ– an efficient engine for fixed points with constraints. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 457–462. Springer, Heidelberg (2011)
24. Janota, M., Klieber, W., Marques-Silva, J., Clarke, E.: Solving QBF with counterexample guided refinement. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 114–128. Springer, Heidelberg (2012)
25. Kesten, Y., Pnueli, A.: A compositional approach to CTL* verification. Theor. Comput. Sci. 331(2-3), 397–428 (2005)
26. McMillan, K.L.: Lazy abstraction with interpolants. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 123–136. Springer, Heidelberg (2006)
27. McMillan, K.L., Rybalchenko, A.: Computing relational fixed points using interpolation. Technical report, available from authors (2012)
28. Namjoshi, K.S.: Certifying model checkers. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 2–13. Springer, Heidelberg (2001)
29. Payet, É., Spoto, F.: Experiments with non-termination analysis for Java Bytecode. Electr. Notes Theor. Comput. Sci. 253(5) (2009)
30. Podelski, A., Rybalchenko, A.: A complete method for the synthesis of linear ranking functions. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 239–251. Springer, Heidelberg (2004)
31. Podelski, A., Rybalchenko, A.: Transition invariants. In: LICS (2004)
32. Schrijver, A.: Theory of linear and integer programming. Wiley-Interscience series in discrete mathematics and optimization. Wiley (1999)
33. Srivastava, S., Gulwani, S.: Program verification using templates over predicate abstraction. In: PLDI (2009)
34. Wintersteiger, C.M., Hamadi, Y., de Moura, L.M.: Efficiently solving quantified bit-vector formulas. Formal Methods in System Design (2013)