Constraint Specialisation in Horn Clause Verification

Bishoksan Kafle
Roskilde University, Denmark
kafle@ruc.dk

John P. Gallagher
Roskilde University, Denmark and IMDEA Software Institute, Spain
jpg@ruc.dk

Abstract

We present a method for specialising the constraints in constrained Horn clauses with respect to a goal. We use abstract interpretation to compute a model of a query-answer transformation of a given set of clauses and a goal. The effect is to propagate the constraints from the goal top-down and propagate answer constraints bottom-up. Our approach does not unfold the clauses at all; we use the constraints from the model to compute a specialised version of each clause in the program. The approach is independent of the abstract domain and the constraints theory underlying the clauses. Experimental results on verification problems show that this is an effective transformation, both in our own verification tools (convex polyhedra analyser) and as a pre-processor to other Horn clause verification tools.

Categories and Subject Descriptors CR-number [*subcategory*]: third-level

Keywords constraint specialisation; query-answer transformation; Horn clauses; abstract interpretation; convex polyhedral analysis

1. Introduction

In this paper, we present a method for specialising the constraints in constrained Horn clauses with respect to a goal. To this end, we first compute a query-answer transformation of a given set of clauses (also called a constraint logic program) with respect to a goal; the aim of the transformation is to simulate the top-down evaluation of the clauses in a bottom-up framework. Then we use abstract interpretation to compute a model of the query-answer transformed program. The idea is to propagate the constraints from the goal top-down and propagate answer constraints bottom-up. Finally we compute a specialised version of each clause in the original program using the constraints from the model without unfolding the clauses at all.

As a result, each clause is further strengthened or removed altogether, preserving the derivability of the goal. Static analysis of the specialised program becomes easier since the implicit constraints in the original clauses are made explicit in the specialised version. In addition to this, since the specialised clauses are more constrained or more specific than the original ones, more specific information will be available for proving the given goal or a failure to prove the goal may be detected at an early stage.

The approach is independent of the abstract domain and the constraints theory. Query-answer transformations, closely related to so-called "magic set" transformations, have been used for Datalog query processing and logic program analysis since the 1980s [3, 16], but have not, to our knowledge, been applied to verification problems. Experimental results on verification problems show that this is an effective transformation, propagating information both backwards from the statement to be proved, and forwards from the Horn clause theory. We show its effectiveness both in our own verification tools and as a pre-processor to other Horn Clause verification tools. In particular, we run our specialisation procedure as a pre-processor both to our *convex polyhedra analyser* and to QARMC [24, 43], a state of the art verification tool. We make the following contributions in this paper:

- we present a method for specialising the constraints in the clauses using *query-answer transformation* and abstract interpretation (see Section 4); and
- we demonstrate the effectiveness of transformation by applying it to Horn clause verification problems (see Section 6).

2. Preliminaries

A constrained Horn clause (CHC) is a first order predicate logic formula of the form $\forall(\phi \wedge p_1(X_1) \wedge \ldots \wedge p_k(X_k) \rightarrow p(X))$ ($k \geq 0$), where ϕ is a conjunction of constraints with respect to some background theory, X_i, X are (possibly empty) vectors of distinct variables, $p_1, \ldots, p_k, p$ are predicate symbols, $p(X)$ is the head of the clause and $\phi \wedge p_1(X_1) \wedge \ldots \wedge p_k(X_k)$ is the body.

Pure constraint logic programs (CLP) are syntactically and semantically the same as CHC. Unlike CLP, CHCs are not always regarded as executable programs, but rather as specifications or semantic representations of other formalisms. However these are only pragmatic distinctions and the semantic equivalence of CHC and CLP means that techniques developed in one framework are applicable to the other. We follow the syntactic conventions of CLP and write a Horn clause as $p(X) \leftarrow \phi, p_1(X_1), \ldots, p_k(X_k)$. In this paper we take the constraint theory to be linear arithmetic with the relation symbols $\leq, \geq, <, >$ and $=$, but the contributions of the paper are independent of the constraint theory.

2.1 Interpretations and models

An interpretation of a set of CHCs is a truth assignment to each atomic formula $p(a_1, \ldots, a_n)$ where p is a predicate and $a_1, \ldots, a_n$ are constants from the constraint theory. An interpretation is represented as a set of *constrained facts* of the form $A \leftarrow \phi$ where A is an atomic formula $p(Z_1, \ldots, Z_n)$ where $Z_1, \ldots, Z_n$ are distinct variables and ϕ is a constraint over $Z_1, \ldots, Z_n$. If ϕ is $\mathtt{true}$ we write $A \leftarrow$ or just A. The constrained fact $A \leftarrow \phi$ is

PEPM '15, January 13–14, 2015, Mumbai, India.
Copyright © 2015 ACM 978-1-4503-3297-2/15/01…$15.00.
http://dx.doi.org/10.1145/2678015.2682544

shorthand for the set of variable-free facts $A\theta$ such that $\phi\theta$ holds in the constraint theory, and an interpretation M denotes the set of all facts denoted by its elements; M assigns true to exactly those facts. $M_1 \subseteq M_2$ if the set of denoted facts of M_1 is contained in the set of denoted facts of M_2.

Minimal models. A model of a set of CHCs is an interpretation that satisfies each clause. There exists a minimal model with respect to the subset ordering, denoted $M[\![P]\!]$ where P is the set of CHCs. $M[\![P]\!]$ can be computed as the least fixed point (lfp) of an immediate consequences operator (called S_P^D in [28, Section 4]), which is an extension of the standard T_P operator from logic programming, extended to handle the constraint domain D. Furthermore $\mathsf{lfp}(S_P^D)$ can be computed as the limit of the ascending sequence of interpretations $\emptyset, S_P^D(\emptyset), S_P^D(S_P^D(\emptyset)), \ldots$. This sequence provides a basis for abstract interpretation of CHC clauses.

3. Abstract Interpretation

Abstract interpretation [11] is a static program analysis technique which derives sound overapproximations of programs by computing abstract fixed points. Convex polyhedra analysis (CPA) [13] is a program analysis technique based on abstract interpretation [11]. When applied to a set of CHCs P it constructs an over-approximation M' of the minimal model of P, where M' contains at most one constrained fact $p(X) \leftarrow \phi$ for each predicate p. The constraint ϕ is a conjunction of linear inequalities, representing a convex polyhedron.

The first application of convex polyhedra analysis to CHCs was by Benoy and King [4]. We summarise briefly the elements of convex polyhedra analysis for CHC; further details (with application to CHC) can be found in [4, 13]. Let $\mathcal{A}$ be the set of convex polyhedra (for some fixed dimension). Let P be a set of CHCs. Suppose there are n predicates in P, say $p_1, \ldots, p_n$, and assume to simplify the discussion that all predicates have the same arity (the dimension of $\mathcal{A}$). The *abstract domain* for P is the set of n-tuples of convex polyhedra $\mathcal{A}^n$. Let the empty polyhedron be denoted $\perp$. Inclusion of polyhedra is a partial order on $\mathcal{A}$ and the partial order $\sqsubseteq$ on $\mathcal{A}^n$ is its pointwise extension. Given an element $\langle d_1, \ldots, d_n \rangle \in \mathcal{A}^n$, define the *concretisation* function γ such that $\gamma(\langle d_1, \ldots, d_n \rangle) = \{\langle p_1(a_1), \ldots, p_n(a_n) \rangle \mid a_i \text{ is a point in } d_i, 1 \leq i \leq n\}$. Construct an *abstract semantic function* $F : \mathcal{A}^n \rightarrow \mathcal{A}^n$ satisfying the *safety condition* $S_P^D \circ \gamma \subseteq \gamma \circ F$ which is monotonic with respect to $\sqsubseteq$, where S_P^D is the immediate consequences operator mentioned above. Let the increasing sequence $Y_0, Y_1, \ldots$ be defined as follows. $Y_0 = \perp^n, Y_{n+1} = F(Y_n)$. These conditions are sufficient to establish that if the limit of the sequence exists, say Y, that $M[\![P]\!] = \mathsf{lfp}(S_P^D) \subseteq \gamma(Y)$ [12].

Since $\mathcal{A}^n$ contains infinite increasing chains, the sequence can be infinite. The use of a *widening* operator for convex polyhedra [11, 13] is needed to ensure convergence of the abstract interpretation. Define the sequence $Z_0 = Y_0$, $Z_{n+1} = Z_n \nabla F(Z_n)$ where ∇ is a widening operator for convex polyhedra [13]. The conditions on ∇ ensure that the sequence stabilises; thus for some finite j, $Z_i = Z_j$ for all $i > j$ and furthermore that Z_j is an upper bound for the sequence $\{Y_i\}$. The value Z_j thus represents, via the concretisation function γ, an over-approximation of the least model of P. Furthermore much research has been done on improving the precision of widening operators. One technique is known as widening-upto, or widening with thresholds [27]. A threshold is an assertion that is combined with a widening operator to improve its precision. Recently, a technique for deriving more effective thresholds was developed [34], which we have adapted and found to be very effective in experimental studies. In brief, the method collects constraints by iterating the concrete immediate consequence function S_P^D three times starting from the "top" interpretation, that is, the interpretation in which all atomic facts are true.

4. Specialisation by constraint propagation

We next present a procedure for specialising CHCs. In contrast to classical specialisation techniques based on partial evaluation with respect to a goal, the specialisation does not unfold the clauses at all; rather, it computes a specialised version of each clause, in which the constraints from the goal are propagated top-down and answers are propagated bottom-up.

We first make precise what is meant by "specialisation" for CHCs. Let P be a set of CHCs and let A be an atomic formula. The specialisation of P with respect to A is a set of clauses P_A such that for every constraint ϕ over the variables of A, $P \models \forall(\phi \rightarrow A)$ if and only if $P_A \models \forall(\phi \rightarrow A)$. This is a very general definition that allows for many transformations. In practice we are interested in specialisations that eliminate consequences of P that have no relevance to A.

For each clause $H \leftarrow \mathcal{B}$ in P, P_A contains a new clause $H \leftarrow \phi, \mathcal{B}$ where ϕ is a constraint. If the addition of ϕ makes the clause body unsatisfiable, it is the same as removing the clause. Clearly P_A may have fewer consequences than P but our procedure guarantees that it preserves the inferability of (constrained instances of) A. The procedure is summarised as follows: the inputs are a set of CHCs P and an atomic formula A.

1. Compute a *query-answer transformation* of P with respect to A, denoted P^{qa}, containing predicates p^{q} and p^{a} for each predicate p in P.
2. Compute an over-approximation M of the model of P^{qa}.
3. Strengthen the constraints in the clauses in P, by adding constraints from the answer predicates in M.

Next we will explain each step in detail.

4.1 The query-answer transformation

The *query-answer transformation* in CLP was inspired by the magic-set transformation from deductive databases and the language Datalog [3]. Its purpose, both in deductive databases and in subsequent applications in logic program analysis [16] was to simulate goal-directed (*top-down*) computation or deduction in a goal-independent (*bottom-up*) framework.

In the following, for each atom $A = p(t)$, A^{a} and A^{q} represent the atoms $p^{\mathsf{a}}(t)$ and $p^{\mathsf{q}}(t)$ respectively. Given a set of CHCs P and an atom A, the (left-) query-answer clauses for P with respect to A, denoted P_A^{qa} or just P^{qa}, are as follows.

- (Answer clauses). For each clause $H \leftarrow \phi, B_1, \ldots, B_n$ ($n \geq 0$) in P, P^{qa} contains the clause $H^{\mathsf{a}} \leftarrow \phi, H^{\mathsf{q}}, B_1^{\mathsf{a}}, \ldots, B_n^{\mathsf{a}}$.
- (Query clauses). For each clause $H \leftarrow \phi, B_1, \ldots, B_i, \ldots, B_n$ ($n \geq 0$) in P, P^{qa} contains the following clauses:

$$\begin{array}{l} B_1^{\mathsf{q}} \leftarrow \phi, H^{\mathsf{q}}. \\ \cdots \\ B_i^{\mathsf{q}} \leftarrow \phi, H^{\mathsf{q}}, B_1^{\mathsf{a}}, \ldots, B_{i-1}^{\mathsf{a}}. \\ \cdots \\ B_n^{\mathsf{q}} \leftarrow \phi, H^{\mathsf{q}}, B_1^{\mathsf{a}}, \ldots, B_{n-1}^{\mathsf{a}}. \end{array}$$

- (Goal clause). $A^{\mathsf{q}} \leftarrow \mathsf{true}$.

The clauses P^{qa} encodes a left-to-right, depth-first computation of the query $\leftarrow A$ for CHC clauses P (that is, the standard CLP computation rule, SLD extended with constraints). This is a complete proof procedure, assuming that all clauses matching a given call are explored in parallel. (Note: the incompleteness of standard Prolog

CLP proof procedures arises due to the fact that clauses are tried in a fixed order). It is important to generate the queries and answers in a single set of clauses, since in general the predicates p^{q} and p^{a} are mutually recursive. Independent analyses propagating constraints from head to body of the clauses and propagating constraints from body to head would not in general achieve the same specialisation.

The relationship of the model of the clauses P^{qa} to the computation of the goal $\leftarrow A$ in P is expressed by the following property[1]. An SLD-derivation in CLP is a sequence $G_0, G_1, \ldots, G_k$ where each G_i is a goal $\leftarrow \phi, B_1, \ldots, B_m$, where ϕ is a constraint and $B_1, \ldots, B_m$ are atoms. In a left-to-right computation, G_{i+1} is obtained by resolving B_1 with a program clause. The model of P^{qa} captures the set of atoms that are "called" or "queried" during the derivation, together with the answers (if any) for those calls. This is expressed precisely by Property 1.

PROPERTY 1 (Correctness of query-answer transformation). *Let P be a set of CHCs and A be an atom. Let P^{qa} be the query-answer program for P wrt. A. Then*

(i) if there is an SLD-derivation $G_0, \ldots, G_i$ where $G_0 =\leftarrow A$ and $G_i =\leftarrow \phi, B_1, \ldots, B_m$, then $P^{\mathsf{qa}} \models \forall(\phi|_{\mathsf{vars}(B_1)} \rightarrow B_1^{\mathsf{q}})$;

(ii) if there is an SLD-derivation $G_0, \ldots, G_i$ where $G_0 =\leftarrow A$, containing a sub-derivation $G_{j_1}, \ldots, G_{j_k}$, where $G_{j_i} \leftarrow \phi', B_1, \mathcal{B}'$ and $G_{j_k} =\leftarrow \phi, \mathcal{B}'$, then $P^{\mathsf{qa}} \models \forall(\phi|_{\mathsf{vars}(B_1)} \rightarrow B_1^{\mathsf{a}})$. (This means that the atom B_1 in G_{j_i} was successfully answered, with answer constraint $\phi|_{\mathsf{vars}(B_1)}$).

(iii) As a special case of (ii), if there is a successful derivation of the goal $\leftarrow A$ with answer constraint ϕ then $P^{\mathsf{qa}} \models \forall(\phi \rightarrow A^{\mathsf{a}})$.

Variations such as the following have been used.

- (Refined call predicates). Call predicates of the form $p^{\mathsf{q}}_{i,j}$ could be generated representing calls to the i^{th} atom in the body of clause j [20], giving more fine-grained information on calls.
- (Relaxed answer predicates). In this version the answer clauses are the same as the original clauses of p, and every answer predicate p^{a} is just replaced by p. This can be used where the only interest is in the model of the query predicates, and the motivation is to increase efficiency of analysis of P^{qa}, while possibly losing precision [9].
- (Other computation rules). Left-to-right computation could be replaced by right-to-left or any other order. The success or failure of a goal is independent of the computation rule; hence we could generate answers using other computation rules, or combining computation rules [21]. While different computation rules do not affect the model of the answer predicates, more effective propagation of constraints during program analysis, and thus greater precision, can sometimes be achieved by varying the computation rule.

For each such variation a correctness property can be stated relating the model of the query-answer program to the SLD computation of the given program P and goal A.

4.2 Over-approximation of the model of P^{qa}

The query-answer transformation of P with respect to A is computed. It follows from Property 1(iii) that if A is derivable from P then $P^{\mathsf{qa}} \models A^{\mathsf{a}}$. Abstract interpretation of P^{qa} yields an over-approximation of $M[\![P^{\mathsf{qa}}]\!]$, say M', containing constrained facts for the query and answer predicates. These represent the calls and answers generated during all derivations starting from the goal A. In our experiments we use a convex polyhedra approximation of $M[\![P^{\mathsf{qa}}]\!]$, as described in Section 3.

4.3 Strengthening the constraints in P

We use the information in M' to specialise the original clauses in P. Suppose M' contains constrained facts $p^{\mathsf{q}}(X) \leftarrow \phi^{\mathsf{q}}$ and $p^{\mathsf{a}}(X) \leftarrow \phi^{\mathsf{a}}$. (If there is no constrained fact $p^*(X) \leftarrow \phi^*$ for some p^* then we consider M' to contain $p^*(X) \leftarrow \mathit{false}$).

For each clause

$$p(X) \leftarrow \phi, p_1(X_1), \ldots, p_k(X_k)$$

in P, construct a clause

$$p(X) \leftarrow \phi, \phi_0, \phi_1, \ldots, \phi_n, p_1(X_1), \ldots, p_k(X_k)$$

in P_A, where $p^{\mathsf{a}}(X) \leftarrow \phi_0$, $p_1^{\mathsf{a}}(X) \leftarrow \phi_1, \ldots, p_n^{\mathsf{a}}(X) \leftarrow \phi_n$ are in M'. Here we assume that there is exactly one constrained fact in M' for each predicate $p^a, p_1^a, \ldots, p_n^a$. Disjunctive constraints can be eliminated from the specialised clauses by further transformation and clauses containing the constraint *false* in the body are eliminated.

Note that wherever M' contains constrained facts $p^{\mathsf{a}}(X) \leftarrow \phi^{\mathsf{a}}$ and $p^{\mathsf{q}}(X) \leftarrow \phi^{\mathsf{q}}$, we have $\phi^{\mathsf{a}} \rightarrow \phi^{\mathsf{q}}$ since the answers for p are always stronger than the calls to p. Thus it suffices to add only the answer constraints to the clauses in P and we can ignore the model of the query predicates. A special case of this is where M' contains a constrained fact $p^{\mathsf{q}}(X) \leftarrow \phi^{\mathsf{q}}$ but there is no constrained fact for $p^{\mathsf{a}}(X)$, or in other words M' contains the constrained fact $p^{\mathsf{a}}(X) \leftarrow \mathit{false}$. This means that all derivations for $p(X)$ fail or loop in P and so adding the answer constraint *false* for p eliminates looping and failed derivations for p.

Specialisation by strengthening the constraints preserves the answers of the goal with respect to which the query-answer transformation was performed. In particular, we have the following property.

PROPERTY 2. *If P is a set of CHCs and P_A is the set obtained by strengthening the clause constraints as just described, then $P \models A$ if and only if $P_A \models A$.*

The proof of Property 2 is by induction on the length of derivations of A. For each derivation of A using the clauses of P we can construct a derivation of A in P_A and conversely.

The specialisation and analysis are separate in our approach. More complex algorithms intertwining them can be envisaged, though the benefits are not clear. Iteration of our procedure could potentially yield further specialisation.

5. Application to the CHC verification problem

In this section, we discuss the application of our constraint specialisation in Horn clause verification. We assume that there is a distinguished predicate symbol false in P which is always interpreted as false. In practice the predicate false only occurs in the head of clauses; we call clauses whose head is false *integrity constraints*, following the terminology of deductive databases. Thus the formula $\phi_1 \leftarrow \phi_2 \wedge B_1(X_1), \ldots, B_k(X_k)$ is equivalent to the formula $\mathsf{false} \leftarrow \neg\phi_1 \wedge \phi_2 \wedge B_1(X_1), \ldots, B_k(X_k)$. The latter might not be a CHC (e.g. if ϕ_1 contains $=$) but can be converted to an equivalent set of CHCs by transforming the formula $\neg\phi_1$ and distributing any disjunctions that arise over the rest of the body. For example, the formula $X = Y \leftarrow p(X, Y)$ is equivalent to the set of CHCs $\{\mathsf{false} \leftarrow X > Y, p(X, Y),\ \mathsf{false} \leftarrow X < Y, p(X, Y)\}$.

Integrity constraints can be seen as safety properties. For example if a set of CHCs encodes the behaviour of a transition system, the bodies of integrity constraints represent unsafe states. Thus

[1] Note that the model of P^{qa} might not correspond exactly to the calls and answers in the SLD-computation, since the CLP computation treats constraints as syntactic entities through decision procedures and the actual constraints could differ.

proving safety consists of showing that the bodies of integrity constraints are false in all models of the CHC clauses. Figure 1 shows an example set of CHCs taken from [23].

```
c1. false :- A>0,B=0,C=0,D=0,l(B,C,D,A).
c2. l(A,B,C,D) :- -A+D>0,A-G= -1, l_body(B,C,E,F),
                  l(G,E,F,D).
c3. l(A,B,C,D) :- A-D>=0,B+C-3*D>0.
c4. l(A,B,C,D) :- A-D>=0,-B-C+3*D>0.
c5. l_body(A,B,C,D) :- A-C= -1,B-D= -2.
c6. l_body(A,B,C,D) :- A-C= -2,B-D= -1.
```

Figure 1. Example program *t4.pl* [23]

5.1 The CHC verification problem.

To state this more formally, given a set of CHCs P, the CHC verification problem is to check whether there exists a model of P. If so we say that P is safe. Obviously any model of P assigns false to the bodies of integrity constraints. We restate this property in terms of the logic consequence relation. Let $P \models F$ mean that F is a logical consequence of P, that is, that every interpretation satisfying P also satisfies F.

LEMMA 1. *P has a model if and only if $P \not\models$ false.*

This lemma holds for arbitrary interpretations (only assuming that the predicate false is interpreted as false), uses only the textbook definitions of "interpretation" and "model" and does not depend on the constraint theory.

The verification problem can be formulated deductively rather than model-theoretically. We can exploit proof procedures for constraint logic programming [28] to reason about the satisfiability of a set of CHCs. Let the relation $P \vdash A$ denote that A is derivable from P using some proof procedure. If the proof procedure is sound then $P \vdash A$ implies $P \models A$, which means that $P \vdash$ false is a sufficient condition for P to have no model, by Lemma 1. This corresponds to using a sound proof procedure to find or check a counterexample. On the other hand to show that P does have a model, soundness is not enough since we need to establish $P \not\models$ false. As we will see in Section 5.2 we approach this problem by using *approximations* to reason about the non-provability of false, applying the theory of abstract interpretation [10] to a complete proof procedure for atomic formulas (the "fixed-point semantics" for constraint logic programs [28, Section 4]). In effect, we construct by abstract interpretation a proof procedure that is *complete* (but possibly not sound) for proofs of atomic formulas. With such a procedure, $P \not\vdash$ false implies $P \not\models$ false and thus establishes that P has a model.

5.2 Proof Techniques

Proof by over-approximation of the minimal model. It is a standard theorem of CLP that the minimal model $M[\![P]\!]$ is equivalent to the set of atomic consequences of P. That is, $P \models p(v_1, \ldots, v_n)$ if and only if $p(v_1, \ldots, v_n) \in M[\![P]\!]$. Therefore, the CHC verification problem for P is equivalent to checking that false $\notin M[\![P]\!]$. It is sufficient to find a set of constrained facts M' such that $M[\![P]\!] \subseteq M'$, where false $\notin M'$. This technique is called proof by *over-approximation of the minimal model*.

Proof by specialisation. A specialisation of a set of CHCs P with respect to an atom A is the transformation of P to another set of CHCs P' such that $P \models A$ if and only if $P' \models A$. In our context we use specialisation to focus the verification problem on the formula to be proved. More specifically, we specialise a set of CHCs with respect to a "query" to the atom false; thus the specialised CHCs entail false if and only if the original clauses entailed false. The constraint strengthening procedure described in Section 4 is our method of specialisation.

Consider the application of the procedure in Section 4 to the clauses in Figure 1, where the *query-answer transformation* is performed with respect to the atom false. The result is shown in Figure 2. Note that the constraint in clause c4 is strengthened to false, showing that c4 is definitely not used in any derivation of false (and hence can be removed).

```
c1. false :- A>0,B=0,C=0,D=0,l(B,C,D,A).
c2. l(A,B,C,D) :- 2*A-B>=0,-A+D>0,-A+B>=0,3*A-B-C=0,
                  A-G= -1,l_body(B,C,E,F),l(G,E,F,D).
c3. l(A,B,C,D):- A-D>0,D>0,2*A-B>=0,-A+D> -1,
                -A+B>=0,3*A-B-C=0.
c4. l(A,B,C,D):- false.
c5. l_body(A,B,C,D) :- -A+2*B>=0, 2*A-B>=0,
                       A-C= -1,B-D= -2.
c6. l_body(A,B,C,D) :- -A+2*B>=0,2*A-B>=0,A-C= -2,
                        B-D= -1.
```

Figure 2. Example program *t4.pl* [23] with strengthened constraints

5.3 Analysis of the specialised clauses

Having specialised the clauses with respect to false, it may be that the clauses P_{false} do not contain a clause with head false. In this case safety is proven, since clearly this is a sufficient condition for $P_{\mathsf{false}} \not\models$ false.

If this check fails we still do not know whether P has a model. In this case we can perform the convex polyhedral analysis on the clauses P_{false}. As the experiments later show, safety is often provable by checking the resulting model; if no constrained fact for false is present, then $P_{\mathsf{false}} \not\models$ false. If safety is not proven, there are two possibilities: the approximate model is not precise enough, but P has a model, or there is a proof of false. Refinement techniques could be used to distinguish these, but this is not the topic of this paper.

In summary, our experimental procedure for evaluating the effectiveness of constraint specialisation contains two steps. Given a set of CHCs P with integrity constraints: (1) Compute a specialisation of P with respect to false yielding P_{false}. If P_{false} contains no integrity constraints, then P is safe. (2) If P_{false} does contain integrity constraints, perform a convex polyhedra analysis of P_{false}. If the resulting approximation of the minimal model contains no constrained fact for the predicate false, then P_{false} is safe and hence P is safe. If we find a concrete derivation for false then we conclude that P is unsafe. Otherwise, P is possibly unsafe.

6. Experimental evaluation

Table 1 presents experimental results of applying our constraint specialisation to a number of Horn clause verification benchmarks taken from the repository of Horn clause verification[2] and other sources including [5, 15, 23, 25, 29]. The columns CPA and QARMC present the results of verification using convex polyhedra and QARMC respectively, whereas columns CS + CPA and CS + QARMC show the result of running constraint specialisation followed by CPA or QARMC. The symbol "-" in the table denotes irrelevant. The experiments were carried out on an Intel(R) X5355 quad-core (@ 2.66GHz) computer with 6 GB memory running Debian 5. We set 5 minutes of timeout for each experiment. The specialisation procedure is implemented in 32-bit Ciao Prolog [7] and uses the Parma Polyhedra Library [1].

[2] https://svn.sosy-lab.org/software/sv-benchmarks/trunk/clauses/

The results show that constraint specialisation is effective in practice. We report that 109 out of 218, that is 50%, of the problems are solved by constraint specialisation alone. When used as a pre-processor for other verification tools, the results show improvements on both the number of instances solved and the solution time. Using our tool, we report approximately 47% increase in the number of instances solved and twice as much faster in average. Similarly using QARMC, we report 13% increase in the number of instances solved and 5 times faster in average.

	CPA	CS + CPA	QARMC	CS + QARMC
solved (safe/unsafe)	61 (48/13)	162 (144/18)	178 (141/37)	205 (171/34)
unknown / timeout	144/12	49/7	-/40	-/13
total time (secs)	2317	1303	13367	2613
average time (secs)	10.62	5.97	61.31	11.98
%solved	27.98	74.31	81.65	94.04

Table 1. Experiments on a set of 218 (181 safe and 37 unsafe) CHC verification problems

The (perhaps surprising) effectiveness of this relatively simple combination of specialisation and convex polyhedral analysis is underlined by noting that it can solve problems for which more complex methods have been proposed. For example, apart from the many examples from the Horn clause verification benchmarks that require refinement using CEGAR-based approaches, the technique solves the "rate-limiter" and "Boustrophedon" examples presented by Monniaux and Gonnord [40] (Section 5) (directly encoded as Horn clauses); their approach, also based on convex polyhedra, uses bounded model checking to achieve a partitioning of the approximation, while other approaches to such problems use trace-partitioning and look-ahead widening.

7. Related Work

Techniques for strengthening the constraints of logic programs go back at least to the work of Marriott *et al.* on most specific logic programs [39]. In that work the constraints were just equalities between terms and the strengthening was goal-independent. In [19] the idea was similar but it was extended to strengthen constraints while preserving the answers with respect to a goal.

The partial evaluation of (constraint) logic programs also has a long history [17, 18, 30, 32]. The aim is to specialise a program with respect to a goal, but usually unfolding is the key technique for propagating constraints. Global analysis using abstract interpretation was combined with partial evaluation algorithms to propagate constraints bottom-up as well as top-down [22, 31, 33, 35, 36].

Abstract interpretation over the domain of convex polyhedra was introduced by Cousot and Halbwachs [13] and applied to constraint logic programs by Benoy and King [4]. Abstract interpretation over convex polyhedra was incorporated in a program specialisation algorithm by Peralta and Gallagher [41].

The method of widening with thresholds for increasing the precision of widening convex polyhedra was first presented by Halwachs *et al.* [27]. We applied a technique for generating threshold constraints presented by Lakhdar-Chaouch *et al.* [34].

In summary, the basic specialisation techniques that we apply are well known, though we are not aware of previous work combining them in the same way. Our method is a specialisation with respect to a goal but does not perform partial evaluation by unfolding. The aim of our specialisation is to make constraints explicit and propagate constraints as much as possible, thereby making other tools more effective, rather than to produce a more efficient computation of a goal.

Verification of CLP programs using abstract interpretation and specialisation has been studied for some time. Our aim in this paper is not to demonstrate a verification tool but to identify a transformation that benefits CLP verification tools generally.

The idea of improving analysis by applying it to a specialised program was first expressed by Turchin [44] and it was more recently demonstrated using supercompilation [38]. The use of program transformation to verify properties of logic programs was pioneered by Pettorossi and Proietti [42] and Leuschel [37] and continues in recent work by De Angelis *et al.* [14, 15]. Transformations that preserve the minimal model (or other suitable models) of logic programs are applied systematically to make properties explicit.

Much other work on CLP verification exists, much of it based on property abstraction and refinement using interpolation, for example [2, 6, 8, 24, 26, 43]. Our specialisation technique is not directly comparable to these methods, but as we have shown in experiments with QARMC, constraint specialisation can be used as a pre-processor to such tools, increasing their effectiveness.

8. Conclusion and future Work

We introduced a method for specialising the constraints in constrained Horn clauses with respect to a goal using abstract interpretation and *query-answer transformation*. The approach propagates constraints globally, both forwards and backwards, and makes explicit constraints from the original program. This allows better analysis of the transformed program. Furthermore, our approach is independent of the abstract domain and the constraints theory underlying the clauses. Finally, we showed effectiveness of this transformation in Horn clause verification problems.

In the future, we will continue to evaluate its effectiveness in a larger set of benchmarks and as a pre-processor for other existing tools. We also would like to use the specialised version for other purposes, for instance in program debugging since more specific information may make errors in the original program apparent.

Acknowledgments

The research leading to these results has received funding from the EU 7th Framework 318337, ENTRA-Whole-Systems Energy Transparency and the Danish Natural Science Research Council grant NUSA: Numerical and Symbolic Abstractions for Software Model Checking.

References

[1] R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.

[2] T. Ball, V. Levin, and S. K. Rajamani. A decade of software model checking with slam. *Commun. ACM*, 54(7):68–76, 2011.

[3] F. Bancilhon, D. Maier, Y. Sagiv, and J. Ullman. Magic sets and other strange ways to implement logic programs. In *Proceedings of the 5^{th} ACM SIGMOD-SIGACT Symposium on Principles of Database Systems*, 1986.

[4] F. Benoy and A. King. Inferring argument size relationships with CLP(R). In J. P. Gallagher, editor, *Logic-Based Program Synthesis and Transformation (LOPSTR'96)*, volume 1207 of *Springer-Verlag LNCS*, pages 204–223, August 1996.

[5] D. Beyer. Second competition on software verification - (summary of sv-comp 2013). In N. Piterman and S. A. Smolka, editors, *TACAS*, volume 7795 of *LNCS*, pages 594–609. Springer, 2013.

[6] N. Bjørner, K. L. McMillan, and A. Rybalchenko. On solving universally quantified horn clauses. In F. Logozzo and M. Fähndrich, editors, *SAS*, volume 7935 of *LNCS*, pages 105–125. Springer, 2013.

[7] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. López-García, and G. Puebla. The Ciao Prolog system. reference manual. Technical Report CLIP3/97.1, School of Computer Science, Technical University of Madrid (UPM), August 1997. Available from http://www.clip.dia.fi.upm.es/.

[8] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.

[9] M. Codish and B. Demoen. Analyzing logic programs using ”PROP”-ositional logic programs and a magic wand. *J. Log. Program.*, 25(3): 249–274, 1995. .

[10] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM Symposium on Principles of Programming Languages, Los Angeles*, pages 238–252, 1977.

[11] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In R. M. Graham, M. A. Harrison, and R. Sethi, editors, *POPL*, pages 238–252. ACM, 1977.

[12] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.

[13] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th Annual ACM Symposium on Principles of Programming Languages*, pages 84–96, 1978.

[14] E. De Angelis, F. Fioravanti, A. Pettorossi, and M. Proietti. Verifying programs via iterated specialization. In E. Albert and S.-C. Mu, editors, *PEPM*, pages 43–52. ACM, 2013.

[15] E. De Angelis, F. Fioravanti, A. Pettorossi, and M. Proietti. Verimap: A tool for verifying programs through transformations. In E. Ábrahám and K. Havelund, editors, *TACAS*, volume 8413 of *LNCS*, pages 568–574. Springer, 2014. ISBN 978-3-642-54861-1.

[16] S. Debray and R. Ramakrishnan. Abstract Interpretation of Logic Programs Using Magic Transformations. *Journal of Logic Programming*, 18:149–176, 1994.

[17] H. Fujita. An algorithm for partial evaluation with constraints. Technical Report TR-258, ICOT, 1987.

[18] J. P. Gallagher. Specialisation of logic programs: A tutorial. In *Proceedings PEPM'93, ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pages 88–98, Copenhagen, June 1993. ACM Press.

[19] J. P. Gallagher and M. Bruynooghe. Some low-level source transformations for logic programs. In *Proceedings of Meta90 Workshop on Meta Programming in Logic*. Katholieke Universiteit Leuven, Belgium, 1990.

[20] J. P. Gallagher and D. de Waal. Deletion of redundant unary type predicates from logic programs. In K. Lau and T. Clement, editors, *Logic Program Synthesis and Transformation*, Workshops in Computing, pages 151–167. Springer-Verlag, 1993.

[21] J. P. Gallagher and D. de Waal. Fast and precise regular approximation of logic programs. In P. Van Hentenryck, editor, *Proceedings of the International Conference on Logic Programming (ICLP'94), Santa Margherita Ligure, Italy*. MIT Press, 1994.

[22] J. P. Gallagher, M. Codish, and E. Shapiro. Specialisation of Prolog and FCP programs using abstract interpretation. *New Generation Computing*, 6:159–186, 1988.

[23] G. Gange, J. A. Navas, P. Schachte, H. Søndergaard, and P. J. Stuckey. Failure tabled constraint logic programming by interpolation. *TPLP*, 13(4-5):593–607, 2013.

[24] S. Grebenshchikov, A. Gupta, N. P. Lopes, C. Popeea, and A. Rybalchenko. Hsf(c): A software verifier based on Horn clauses - (competition contribution). In C. Flanagan and B. König, editors, *TACAS*, volume 7214 of *LNCS*, pages 549–551. Springer, 2012.

[25] A. Gupta and A. Rybalchenko. Invgen: An efficient invariant generator. In A. Bouajjani and O. Maler, editors, *CAV*, volume 5643 of *LNCS*, pages 634–640. Springer, 2009. ISBN 978-3-642-02657-7.

[26] A. Gupta, C. Popeea, and A. Rybalchenko. Solving recursion-free horn clauses over li+uif. In H. Yang, editor, *APLAS*, volume 7078 of *LNCS*, pages 188–203. Springer, 2011. ISBN 978-3-642-25317-1.

[27] N. Halbwachs, Y. E. Proy, and P. Raymound. Verification of linear hybrid systems by means of convex approximations. In *Proceedings of the First Symposium on Static Analysis*, volume 864 of *LNCS*, pages 223–237, September 1994.

[28] J. Jaffar and M. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[29] J. Jaffar, J. A. Navas, and A. E. Santosa. Unbounded symbolic execution for program verification. In S. Khurshid and K. Sen, editors, *RV*, volume 7186 of *LNCS*, pages 396–411. Springer, 2011.

[30] N. Jones, C. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Software Generation*. Prentice Hall, 1993.

[31] N. D. Jones. Combining abstract interpretation and partial evaluation. In P. Van Hentenryck, editor, *Symposium on Static Analysis (SAS'97)*, volume 1302 of *Springer-Verlag LNCS*, pages 396–405, 1997.

[32] H. J. Komorowski. An introduction to partial deduction. In A. Pettorossi, editor, *META*, volume 649 of *LNCS*, pages 49–69. Springer, 1992. ISBN 3-540-56282-6.

[33] L. Lafave and J. P. Gallagher. Partial evaluation of functional logic programs in rewriting-based languages. In N. Fuchs, editor, *Logic Program Synthesis and Transformation (LOPSTR'97)*, Springer-Verlag LNCS, 1998.

[34] L. Lakhdar-Chaouch, B. Jeannet, and A. Girault. Widening with thresholds for programs with complex control graphs. In T. Bultan and P.-A. Hsiung, editors, *ATVA 2011*, volume 6996 of *LNCS*, pages 492–502. Springer, 2011.

[35] M. Leuschel. Advanced logic program specialisation. In J. Hatcliff, T. Æ. Mogensen, and P. Thiemann, editors, *Partial Evaluation - Practice and Theory*, volume 1706 of *LNCS*, pages 271–292. Springer, 1999.

[36] M. Leuschel. A framework for the integration of partial evaluation and abstract interpretation of logic programs. *ACM Trans. Program. Lang. Syst.*, 26(3):413–463, 2004. . URL `http://doi.acm.org/10.1145/982158.982159`.

[37] M. Leuschel and T. Massart. Infinite state model checking by abstract interpretation and program specialisation. In A. Bossi, editor, *LOPSTR'99*, volume 1817 of *LNCS*, pages 62–81. Springer, 1999.

[38] A. Lisitsa and A. P. Nemytykh. Reachability analysis in verification via supercompilation. *Int. J. Found. Comput. Sci.*, 19(4):953–969, 2008. . URL `http://dx.doi.org/10.1142/S0129054108006066`.

[39] K. Marriott, L. Naish, and J.-L. Lassez. Most specific logic programs. In *Proc. Fifth International Conference on Logic programming, Seattle, WA*. MIT Press, 1988.

[40] D. Monniaux and L. Gonnord. Using bounded model checking to focus fixpoint iterations. In E. Yahav, editor, *Static Analysis - 18th International Symposium, SAS 2011, Venice, Italy, September 14-16, 2011. Proceedings*, volume 6887 of *LNCS*, pages 369–385. Springer, 2011. ISBN 978-3-642-23701-0.

[41] J. C. Peralta and J. P. Gallagher. Convex hull abstractions in specialization of CLP programs. In M. Leuschel, editor, *LOPSTR*, volume 2664 of *LNCS*, pages 90–108. Springer, 2002. ISBN 3-540-40438-4.

[42] A. Pettorossi and M. Proietti. Perfect model checking via unfold/fold transformations. In J. W. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.-K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, and P. J. Stuckey, editors, *Computational Logic*, volume 1861 of *LNCS*, pages 613–628. Springer, 2000.

[43] A. Podelski and A. Rybalchenko. ARMC: the logical choice for software model checking with abstraction refinement. In M. Hanus, editor, *Practical Aspects of Declarative Languages, 9th International Symposium, PADL 2007, Nice, France, January 14-15, 2007.*, volume 4354 of *LNCS*, pages 245–259. Springer, 2007. ISBN 978-3-540-69608-7. . URL `http://dx.doi.org/10.1007/978-3-540-69611-7_16`.

[44] V. F. Turchin. The use of metasystem transition in theorem proving and program optimization. In J. W. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherland, July 14-18, 1980, Proceedings*, volume 85 of *LNCS*, pages 645–657. Springer, 1980. ISBN 3-540-10003-2. . URL `http://dx.doi.org/10.1007/3-540-10003-2_105`.