Constraint Model Elimination and a PTTP-Implementation

Peter Baumgartner · Frieder Stolzenburg

Universität Koblenz · Rheinau 1 · D–56075 Koblenz
email: {peter,stolzen}@informatik.uni-koblenz.de

Abstract. In constraint logic programming, proof procedures for *Horn* clauses are enhanced with an interface to efficient constraint solvers. In this paper we show how to incorporate constraint processing into a general, non-Horn theorem proving calculus.
A framework for a new calculus is introduced which combines model elimination with constraint solving, following the lines of Bürckert (1991). A prototype system has been implemented rapidly by only combining a PROLOG technology implementation of model elimination and PROLOG with constraints. Some example studies, e.g. taxonomic reasoning, show the advantages and some problems with this procedure.

1 Introduction: Programming with Constraints

One of the most traditional disciplines in artificial intelligence (AI) is theorem proving. In the early days, it was concentrated mainly on developing general proof procedures for predicate logic. According to the shift within wide parts of AI research towards special domain dependent systems, automated reasoning and theorem proving nowadays aim at incorporating specialized and efficient modules which are suited for handling special parts or domains of knowledge. Examples of this paradigm are the combination of theorem provers with *taxonomic reasoning* [BGL85], *equality handling*, *theory unification* or the rather general scheme of *theory reasoning* [BFP92].

As another paradigm *constraint logic programming* (CLP) is established as an active research field of its own. For an overview, see [Van89, JM94]. In CLP usually SLD-resolution based proof procedures are considered and are enhanced with an interface to a constraint solver. Since constraint solving has been combined so successfully with *logic programming*, the question arises, whether it is possible to incorporate constraint processing into *theorem proving* calculi. The merits of this combination would be that specifications can be written then in *full* first order logic (as opposed to definite programs used in CLP), and at the same time advantage can be taken of known constraint solving procedures.

Surprisingly, rather few research has been done in this direction (not counting the numerous approaches of those kinds of two-level reasoning that can be seen as special instances of constraint handling). Only in [Bür91, Bür94] the familiar resolution calculus in general was extended with a framework for constraint handling. In this *constrained resolution*, clauses enhanced with constraints replace the traditional clauses.

Syntactically, the constraints are formulæ that are attached to some va clauses; semantically, the constraints filter out valid assignments for the varia resolution inference rule is modified accordingly by accumulating the constrai parent clauses and additionally stating the unification task of the selected lit constraint. In a refutation, the thus combined constraints must be solved event not necessarily immediately after an inference step. In other words, constrain treated lazily.

Constrained resolution is related to theory reasoning, especially *theory r* [Sti85]. On the one hand, constraint reasoning is more general than theory r as constraints may be treated lazily. Furthermore, no concrete theory unifier be computed during proof search. Instead it suffices to establish the satisfi the accumulated theory unification problems. On the other hand, constraint rea more special than theory reasoning, as in constraint reasoning the foreground th be a conservative extension of the background theory. In [Bür91, 13-14] it is that (wide) theory resolution can be understood as (a modification of) co resolution.

1.1 The Proposal: Model Elimination with Constraints

As an alternative to this constrained resolution, we propose to enhance *model el* with a general framework for constraint handling. The model elimination c a goal-oriented, linear and refutationally complete calculus for first order cla [Lov68]. It is the base of numerous proof procedures for first order deduction. high speed theorem provers like PROTEIN [BF94] which approach the infer of today's PROLOG implementations.

Our approach is similar in spirit to constrained resolution. However, there notable differences. For the first, model elimination is a linear, goal-directed whereas resolution is a non-linear, bottom-up calculus. By these properti elimination is closer to the SLD-resolution of PROLOG than constrained reso a consequence we need a new completeness proof and cannot take the one i 62-67]. The second difference concerns implementation. Model eliminatio implemented in a straightforward way on top of existing PROLOG implemen using the so-called PTTP-technique [Sti89].

Even better, as is shown in Section 4, the constraint handling mechanis underlying PROLOG dialect can be used. Such a cheap and rapid but rathe implementation is not available for constrained resolution.

2 The Calculus of Constraint Model Elimination

We will now introduce the framework for *constraint model elimination* (C better understanding, we will interleave the theory of CME with a well-known from the literature.

Example 1. The *Lion and Unicorn* puzzle is stated as problem 45 in [Smu natural language description of the problem is as follows.

1. *The lion lies on Monday, Tuesday and Wednesday.*
2. *The unicorn lies on Thursday, Friday and Saturday.*
3. *Both tell truth on other days.*
4. *Both say yesterday was one of their lying days.*
5. *Prove that today is Thursday.*

It is a general observation of AI research that the description of a problem can be separated as follows. On the one hand, there is an internal, *background part* of the problem description that refer to the knowledge base of the system. Here, the system knows procedures, e.g. special unification algorithms or constraint solvers, to treat these problem constraints. On the other hand, we have the external, *foreground part* where deep reasoning may be necessary. But general heuristics may be applied.

The reader may ask how to divide a problem description into the foreground and background part. Quite often, the implicit given knowledge of a problem description is a good candidate for the background part as in the Example 1. Another idea is to choose that part as background part for which we have efficient constraint solvers at hand. The rest becomes the foreground part. But, the more we put in the background, the more elegant and efficient the problem can be treated.

2.1 Preliminaries

Let the notions *alphabet*, *clause set*, *closed formula*, *domain*, *empty clause*, *equality*, *formula*, *interpretation*, *literal*, *model*, *term*, *validity*, and *variable assignment* be defined as usual. We will now formalize the notion of background part as a *constraint system*. A constraint system (or constraint theory) consists of a set of constraint symbols (or constraints) and a set of constraint models [BBH$^+$90, 6].

Definition 1 (constraint system). A Δ-*interpretation* is an interpretation of (exactly) the symbols of the alphabet Δ. The notion Δ-*formula* is defined in a similar manner.

A *constraint system* $\Re$ which is also called restricted quantification system in [Bür91, 50] consists of:

1. an alphabet Δ with equality $=$; the restriction theory must interpret $=$ with the equality relation (see below);
2. a theory over Δ, the *restriction theory* which can be given by a distinguished class $\mathcal{R}$ of Δ-interpretations; we do not require $\mathcal{R}$ to be a first order theory, but of course it could be given by some axioms; for computational reasons, it is often useful to restrict to first order theories;
3. a set of open (i.e. not closed) Δ-formulæ, the *constraints*; we will give examples for constraints at the end of this subsection; the constraints must at least be closed under conjunction $\wedge$ and under instantiation of variables.

As just said, the restriction theory must interpret $=$ with the *equality relation*. At least, this must be the syntactical equality, for which Robinson's well-known unification algorithm yields the constraint solver. But it is also possible to use special unification theories, e.g. AC or set unification [Sto93, Sto94], or an equational theory in general.

In our example, we will model *today* as a constant and *yesterday* as a binary
Then, the alphabet Δ consists of the predicates $=$ and *yesterday*, and the consta
monday, ..., *sunday*, and *lion* and *unicorn*. The restriction theory $\mathcal{R}_0$ is giv
following formulæ. The symbol $\dot\vee$ denotes the logical *exclusive or*.

$$today = monday \ \dot\vee \cdots \dot\vee \ today = sunday$$
$$yesterday(monday, sunday) \ \wedge \cdots \wedge yesterday(sunday, saturda$$

Definition 2 (constraint clauses). Let $\Re$ be a constraint system and Σ an alph
that $\Sigma \cap \Delta = \emptyset$, i.e. Σ contains only new predicate and function symbols. A
is called a *constraint clause* if L is a disjunction of Σ-literals $l_1 \vee \ldots \vee l_n$ a
restriction formula.

Let $\mathcal{A} \in \mathcal{R}$ be a Δ-interpretation. An interpretation $\mathcal{A}^*$ is called Σ-*expan*
iff it is an interpretation of $\Delta \cup \Sigma$ (with the same domain) such that the red
wrt. Δ is $\mathcal{A}$. Let $\mathcal{A} \in \mathcal{R}$ be a Δ-interpretation and $\mathcal{A}^*$ an expansion of $\mathcal{A}$. A
an interpretation to a subsignature is given by forgetting about the denotatio
symbols that are not in the subsignature.

In the last definition, we enforced that every $\mathcal{A}^*$ must be a conservati
sion of the restriction theory. On the one hand, this is a restriction of our fr
(and of constrained resolution) that we cannot mix symbols of the foregroun
background part. But on then other hand, that reduces search space and allov
modular architecture of systems with constraints.

Definition 3 (satisfiability). If an interpretation $\mathcal{A}^*$ together with a variable as
α satisfies a literal set L we write $(\mathcal{A}^*, \alpha) \models L$. See [Llo87, 13-14] for a def
satisfiability.

We define the *satisfiability* of a constraint clause L/r in $\mathcal{A}^*$, written $\mathcal{A}^*$
as follows: $\mathcal{A}^* \models L/r$ iff for all variable assignments $\alpha : V \to \mathcal{D}$ (where
variables in L/r and $\mathcal{D}$ is the domain of $\mathcal{A}$) with $(\mathcal{A}, \alpha) \models r$ it holds $(\mathcal{A}^*, \alpha)$
$\mathcal{A}^* \models \forall r \Rightarrow L$).

We say that a clause set $\mathcal{C}$ is *satisfiable* in $\mathcal{A}^*$ iff $\mathcal{A}^* \models L/r$ for every clause
Furthermore, a clause set $\mathcal{C}$ is defined to be satisfiable in $\mathcal{R}$ iff $\mathcal{C}$ is satisfiabl
$\mathcal{A}^* \in \mathcal{R}$. The notion of "unsatisfiability" is defined via "not satisfiable".

Now we are able to define the problem as a constraint clause set. Let us intr
predicates *lying/2* and *says/3* in Σ where *lying(X,Y)* means *X lies on Y*, and *sa*
means *X says on Y that he lies on Z*. We conclude that if a being X says on
lies on Z then he must lie either on Y or on Z, but not on both days. The fact
3 of the problem description can be formulated in a straightforward manner
express information 4 and 5 by some constraint clauses. This is expressed by
in Figure 1.

2.2 Constraint Derivations

The last clause in Figure 1 is an empty constraint clause with a satisfiable but
constraint. Usually (i.e. in plain first order logic), we need only one empty cl

0. $\neg says(X,Y,Z) \vee lying(X,Y) \vee lying(X,Z)$
   $\neg says(X,Y,Z) \vee \neg lying(X,Y) \vee \neg lying(X,Z)$
1. $lying(X,Y)/X = lion \wedge Y = monday$
   $lying(X,Y)/X = lion \wedge Y = tuesday$
   $lying(X,Y)/X = lion \wedge Y = wednesday$
2. $lying(X,Y)/X = unicorn \wedge Y = thursday$
   $lying(X,Y)/X = unicorn \wedge Y = friday$
   $lying(X,Y)/X = unicorn \wedge Y = saturday$
3. $\neg lying(X,Y)/X = lion \wedge Y = thursday$
   $\neg lying(X,Y)/X = lion \wedge Y = friday$
   $\neg lying(X,Y)/X = lion \wedge Y = saturday$
   $\neg lying(X,Y)/X = lion \wedge Y = sunday$
   $\neg lying(X,Y)/X = unicorn \wedge Y = monday$
   $\neg lying(X,Y)/X = unicorn \wedge Y = tuesday$
   $\neg lying(X,Y)/X = unicorn \wedge Y = wednesday$
   $\neg lying(X,Y)/X = unicorn \wedge Y = sunday$
4. $says(X,Y,Z)/X = lion \wedge Y = today \wedge yesterday(Y,Z)$
   $says(X,Y,Z)/X = unicorn \wedge Y = today \wedge yesterday(Y,Z)$
5. $\Box/today = thursday$

Fig. 1. Constraint clauses for the Lion and Unicorn puzzle.

order to show unsatisfiability. But in the constraint case, in general we need a derivation of the empty clause for every model $\mathcal{A} \in \mathcal{R}$, as we will see later on.

Before we will show how to prove the query, we will introduce the constraint model elimination calculus. We will introduce it analogously as in [Bür91]. The model elimination is formulated as a tableau calculus similar to [Bau94].

Definition 4. Two literals l and l' are *complementary* written $l \sim l'$ iff l is a positive and l' is a negative literal or vice versa, and it holds $l = l''$ where l'' is equal to l' but with different sign. – A *path* p is a finite sequence of Σ-literals $\langle l_1, \ldots, l_n \rangle$. A finite set of paths P is called *tableau*.

The complementarity relation $\sim$ can be reduced to $=$. Without loss of generality let us assume that $l = p(s_1, \ldots, s_n)$ and $l' = \neg p(t_1, \ldots, t_n)$ where $n \geq 0$. Then $l \sim l'$ denotes the constraint $s_1 = t_1 \wedge \ldots \wedge s_n = t_n$. Hence, we could do it in Definition 6 below without $\sim$. However we will use the meta-level symbol $\sim$ at the object level in order to abbreviate notation.

Definition 5. We will now define some operations:

1. $\mathcal{L}(\langle l_1, \ldots, l_n \rangle) = l_n$ (*leaf of a path*)
2. $\mathcal{F}(P) = \vee_{p \in P}(\wedge_{l \in p} l)$ (*formula of a tableau*)
3. $\mathcal{T}(l_1 \vee \ldots \vee l_n) = \{\langle l_1 \rangle, \ldots, \langle l_n \rangle\}$ (*initial tableau*)
4. $\langle l_1, \ldots, l_n \rangle.L = \{\langle l_1, \ldots, l_n, l \rangle | l \in L\}$ (*concatenation*)

Definition 6 (constraint derivation). Let $\mathcal{C}$ be a constraint clause set. A pa
called *constraint tableau* iff P is a tableau and R is a constraint. We will now
relation $\vdash_{\mathcal{C}}$ between constraint tableaux given by the following inference rule

(a) *constraint extension step.* $\dfrac{P \cup \{p\}/R}{P \cup p.L/R \wedge r \wedge (\mathcal{L}(p) \sim l)}$ if $L \vee l/r \in \mathcal{C}$.

(b) *constraint reduction step.* $\dfrac{P \cup \{p\}/R}{P/R \wedge (\mathcal{L}(p) \sim l)}$ if $l \in p$.

In (a) $L \vee l/r$ must be a copy of a clause in $\mathcal{C}$ with fresh variables. In th
$L \vee l/r$ is called the *extending clause* and l is called the *extending literal*.

Let $\vdash_{\mathcal{C}}^{*}$ denote the reflexive and transitive closure of $\vdash_{\mathcal{C}}$. We will call $\mathcal{T}(L)/$
constraint derivation in $\mathcal{C}$ with *goal clause* $L/r \in \mathcal{C}$.

Mostly, we are interested in derivations ending with an *empty tabl*
[BBH$^+$90, 8] where R is a constraint. Please note, that we overloaded th
$\Box$. It denotes an empty clause of foreground literals, too.

Based on this definitions we will state now our first main result. It holds fo
constraint systems.

Theorem 7 (soundness and completeness). *A constraint clause set $\mathcal{C}$ is uns*
iff for each $\mathcal{A} \in \mathcal{R}$ there is a clause $L/r \in \mathcal{C}$ such that $\mathcal{T}(L)/r \vdash_{\mathcal{C}}^{*} \Box/R$ and

Completeness is proven by adapting the usual *ground proof plus lifting* t
But notice that we have to replace the Herbrand domain by any of the mod
guished by the restriction theory. The ground proof is similar to that of ordin
elimination [Bau93]. All the proofs are stated in detail in the long version [SE

The last theorem is not very satisfactory, because we might need infinit
empty tableau. In other words, we do not have a calculus (yet). In order to re
situation we have to restrict the constraint systems to those where only a finit
of such empty constraint tableaux are needed. These considerations motivat
definition.

Definition 8 (valid constraint, compactness, refutation). A set of constra
called *valid* in $\mathcal{R}$, written as $\mathcal{R} \models R$ iff for every $\mathcal{A} \in \mathcal{R}$ there exists
such that for every suitable variable assignment α it holds $(\mathcal{A}, \alpha) \models r$. A cons
stem is *compact* iff every valid set of constraints contains a finite subset, whic
valid. A *refutation* is a finite set of derivations of empty tableaux $\Box/R_i$, i
such that $\mathcal{R} \models \exists R_1 \vee \cdots \vee \exists R_n$. We name the calculus defined here *constra*
elimination.

For compact constraint systems every infinite refutation contains a finite
instance, if the restriction theory can be axiomatized in first order logic, the
of refutations is an immediate consequence of the compactness theorem of f
logic [Bür91, 68]. For Horn theories as restriction theories the derivation of
empty constraint tableaux (with valid constraint) is sufficient.

In the case of compact systems theorem 7 can be rewritten in the present ter
as follows:

Theorem 9 (Soundness and completeness of CME). *A constraint clause set $\mathcal{C}$ is unsatisfiable in a compact constraint system iff there exists a constraint model elimination refutation.*

2.3 An Example Derivation

Now, how can we prove the query of our Example 1? – We have to distinguish between seven models $\mathcal{A} \in \mathcal{R}_0$. They are distinguished by the facts $today = monday, \ldots, today = sunday$. So, we can solve the puzzle if we prove for every case that $\mathcal{C}$ is unsatisfiable. The case where $today = thursday$ is trivial. The other cases can be shown in a very similar manner. We solved the Lion and Unicorn puzzle in an overall time of about 0.1s on a Sun4.

Figure 2 shows the derivation for $\mathcal{A}_0$ where $(today = friday) \in \mathcal{A}_0$. In this case, we get the refutation by the fact that the unicorn lies on Thursday and Friday because then he cannot say on Friday that he lies on Thursday. The clauses and formulæ that are used in the constraint derivation and simplification steps steps are annotated. The arrow denotes constraint simplification.

Figure 3 shows the proof as a tree. For matters of presentation we have included the extending literals in the paths and marked these paths as solved with a $*$. It is clear that the existential closure of the remaining constraints, namely $\exists X, Y, Z : X = unicorn \wedge Y = friday \wedge Z = thursday$, is valid.

$$\big\{\langle\neg says(X,Y,Z)\rangle, \langle\neg lying(X,Y)\rangle, \langle\neg lying(X,Z)\rangle\big\}$$

$$says(X,Y,Z)/X = unicorn \wedge Y = today \wedge yesterday(Y,Z)$$

$$\big\{\langle\neg lying(X,Y)\rangle, \langle\neg lying(X,Z)\rangle\big\}/X = unicorn \wedge Y = today \wedge yesterday(Y,Z)$$

$$today = friday \wedge yesterday(friday, thursday)$$

$$\big\{\langle\neg lying(X,Y)\rangle, \langle\neg lying(X,Z)\rangle\big\}/X = unicorn \wedge Y = friday \wedge Z = thursday$$

$$lying(X,Y)/X = unicorn \wedge Y = friday$$

$$\big\{\langle\neg lying(X,Z)\rangle\big\}/X = unicorn \wedge Y = friday \wedge Z = thursday$$

$$lying(X,Y)/X = unicorn \wedge Y = thursday$$

$$\Box/X = unicorn \wedge Y = friday \wedge Z = thursday$$

Fig. 2. Example derivation with annotations.

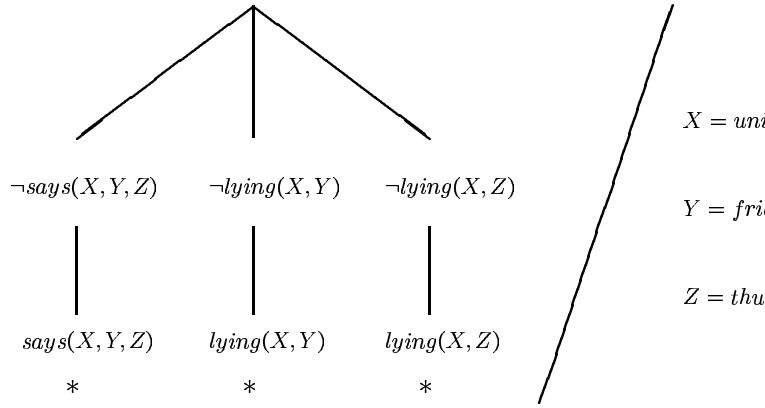

Fig. 3. Presentation of the proof as a tree.

3 An Application Domain: Taxonomic Reasoning

Let us now switch to a more realistic application domain for constraint mo nation, namely taxonomical reasoning. Our plan is to use a readily available solving mechanism for this. However, such constraint solving mechanisms a designed to be used within a logic programming environment, and when e them in our more general situation of theorem proving we then face the problems:

- In CLP usually we have constraint simplification calculi for only posit traints, but for constraint model elimination it is necessary that the cal also deal with negative constraints, i.e. negative constraint literals. Sinc tive constraint such as $\neg R$ in $L/\neg R$ is a *positive* information, this mean constraint theory may be extended in the course of the computation.
- After a CLP computation has derived the empty clause, the remaining c are satisfiable but not necessarily valid. However, we require that the r constraints are valid in the restriction theory $\mathcal{R}$ (cf. Definition 8). Satisfia validity amounts to the same only in case of *generic* restriction theories which equivalent in some sense to one single interpretation).

3.1 Terminological Languages

As an example we will consider terminological reasoning [SSS91] which c with the KL-ONE system [BS85]. A common feature of such systems is the s of knowledge into a *terminological part* and an *assertional part*. The knowle classes of individuals and relationships between these classes is stored in the T the knowledge concerning particular individuals can be described in the A-Bo 68]. It is often useful to disallow terminological cycles.

Usually a terminological reasoning system provides an (in)consistency and a subsumption checking component. The constraint solving method of [SSS91] which expands the definition of concepts is a well-known method which tackles these problems. But for the above-mentioned services, the A-Box plays no active role [BDS93, 116]. Introducing an A-Box leads to the problem of dealing with negative constraints which will be further explained later on. We will now consider an example.

Example 2. This example follows that in [BGL85, 538-539]. The reader may verify that the answer to the question in 8 is yes. Furthermore, we can say that Fred and Pat are the persons who are asked for.

1. *Persons are thinking animals.*
2. *Dogs are animals that do not think.*
3. *A noson is a person who has no sons, i.e. all his children are female.*
4. *Pat is Fred's child.*
5. *Fred is a noson.*
6. *Every dog loves every animal.*
7. *Every person loves his child.*
8. *Is there a person who loves a female?*

We can split the problem into three parts. Firstly, we have some terminological knowledge about persons, dogs and nosons (sentences 1 to 3). Secondly, we have some assertional knowledge (sentences 4 and 5). It can be expressed by means of the *membership predicate :/2* by using the concepts *person*, *dog*, *noson* etc. and the role *child*. We will think of the relation *loves* as an ordinary predicate. Hence, the last sentences (numbers 6 to 8) form the genuine *hybrid part* of the problem.

3.2 Implementing Hybrid Reasoning

Terminological reasoning, i.e. constraint solving method, can be implemented by constraint handling rules (CHR) in ECLiPSe [FH94]. A CHR can be seen as a special kind of a term rewriting rule, e.g $(X : C, X : \neg C) \iff fail$. This implementation only deals with positive constraints and cannot solve the universal satisfiability problem of concept descriptions [BBH+90, 12-21]. But that is needed if we want to use an *open world semantics* [BH91, 73]. That means, incomplete knowledge is permitted; there may be instances or relationships not explicitly mentioned in the A-Box.

But for theorem proving purposes a *closed domain semantics* seems to be more appropriate. That means, the taxonomic constraints remaining after the derivation of an empty tableau must be a logical consequence from the given A-Box. Now, if we require that the given A-Box represents a Horn theory, i.e. concept disjunction and existential quantification is not allowed, then we can easily implement taxonomic reasoning within our framework, since for Horn theories only one derivation of an empty tableau is sufficient [Bür91, 69-72].

That will be explained in conjunction with the Example 2. Its formalization is as follows. Here, we use the syntax of [SSS91, 5].

– T-Box:

1. $person \doteq animal \sqcap thinker$
2. $dog \doteq animal \sqcap \neg thinker$
3. $noson \doteq person \sqcap \forall child : female$

– A-Box:

4. $(fred, pat) : child$
5. $fred : noson$

– Hybrid part:

6. $loves(x, y)/(x : dog \wedge y : animal)$
7. $loves(x, y)/(x : person \wedge (x, y) : child)$
8. $\neg loves(x, y)/(x : person \wedge y : female)$

Let us now start a constraint derivation with the query 8. If we extend with then we get the empty tableau $\Box/(x : dog \wedge x : person \wedge y : animal \wedge y :$ whose constraint is inconsistent as the constraint solver tells us, using the info and 2. Hence, we must backtrack and extend with clause 7. That leads us to t tableau $\Box/(x : person \wedge (x, y) : child \wedge y : female)$ whose constraint is c i.e. satisfiable. But we have to prove that it is valid by means of the A-Box. we take also the empty tableaux of clauses 4 and 5 into account, it is clear by r it is so. For this, the non-primitive concepts have to be unfolded with their de

In our implementation, the T-Box is reached via CHRs. The A-Box facts ar via PROLOG rules. These rules are activated by a call to *chr_labeling/0* constraint simplification (expansion) phase by the ECLiPSe system. The hy consists of clauses. There, the foreground literals are transformed by the P system whereas the background literals are treated by the ECLiPSe system as co

3.3 Related Approaches

The implementation of terminological reasoning we used [FH94] follows th [SSS91] which presents a PSPACE-complete algorithm. If we consider a more formalism, e.g. sorted logics or order-sorted feature structures where only f roles, i.e. features, are allowed, then we are able to build more efficient solvers. For example, the unification of sorts can be computed by a table in nearly constant time. By this procedure, we got a proof of *Schubert's St problem* within 7.4s on a Sun4. The formulation of a sort hierarchy by implicat unary predicates usually leads to blind search with exponential time behavi worst case.

4 A PTTP-Implementation

One of our aims by developing constraint model elimination was to get a nevertheless efficient implementation of constraint reasoning. The problem is are on the one hand some implementations of resolution, model elimination calculi, but they are often hard-coded and it is difficult to enrich them with solvers. On the other hand there are constraint solvers and CLP languages *Horn* logic. But recall that we are interested in full first order logic, not just H

In order to combine these two worlds we propose to use model elimination as a base calculus. This helps, because model elimination can be implemented by the PTTP-technique [Sti89] on top of PROLOG, and, furthermore, the constraint mechanisms of the underlying PROLOG system can be used. In our case we had the PTTP-implementation of model elimination already at hand – the PROTEIN system [BF94] – which is implemented in ECLiPSe-PROLOG [ECR94]. Since ECLiPSe features constraint solving mechanisms, only small modifications were necessary to implement constraint model elimination.

4.1 PROTEIN and PROLOG with Constraints

PROTEIN is based on the PROLOG technology implementation technique. The idea of that is to view PROLOG as an almost complete theorem prover which has to be extended by only a few ingredients in order to handle the non-Horn case. By this technique the WAM-technology and other optimizing PROLOG compilers are accessible to theorem proving. PROTEIN – itself written in PROLOG – compiles a given clause set into a set of PROLOG clauses introducing some special code in order to treat reduction steps. This set can be run like an ordinary PROLOG program. In the case of Horn clauses we get nearly the efficiency of PROLOG. PROTEIN also includes several calculus refinements and flags.

ECLiPSe [ECR94] extends PROLOG by various features with the most relevant for us being sound unification and constraint handling. The constraint language CHIP [Van89] is integrated which can deal with finite integer domain constraints. The CHR library contains also boolean, lists and set constraints solver. The constraint solvers are accessed by some special constraint predicates which may be built-in or user-defined. ECLiPSe supports a new data structure called metaterm. A metaterm is a variable with an associated attribute. It behaves like a normal variable, however when it is unified with another term, an event is raised and a user-defined handler specifies what the result of the unification will be. This in fact makes it possible to define any new data structure. Hence, it is possible to implement special unification theories as restriction theories.

4.2 The Combination of PROTEIN and ECLiPSe

The combination of PROTEIN and ECLiPSe in order to implement constraints is not difficult. For this, we need to declare the constraint predicates used in the clause sets as *PROLOG predicates* in PROTEIN. These PROLOG predicates will not be treated as ordinary input literals by PROTEIN, but instead they will be passed as PROLOG goals to ECLiPSe. There, they will be treated by the built-in constraint mechanisms of ECLiPSe according to our application.

But this trick alone will not suffice. Recall from Theorem 7 that for each model $\mathcal{A}$ of the constraint theory $\mathcal{R}$ a proof with some input clause L/r as query has to be found. Obviously, Theorem 7 can only be implemented in this direct way if $\mathcal{R}$ is given by *finitely* many interpretations (as in the lion and unicorns example). In this case we can design a proof procedure based on an *outer loop*, which enumerates all (representations of) the interpretations in $\mathcal{R}$, and an *inner loop* which enumerates proofs in the selected interpretation.

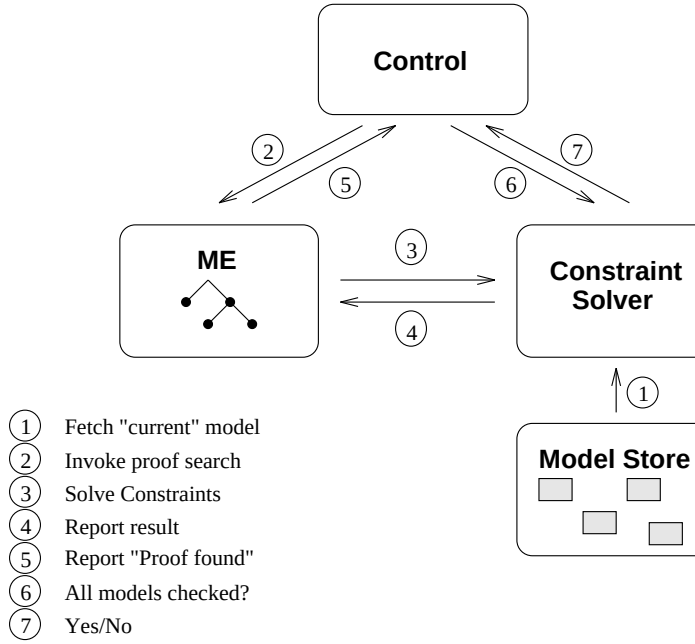

Fig. 4. Architecture of PROTEIN with constraints.

We found it advantageous to implement the outer loop by extending the pro
Figure 4 depicts the overall architecture of the prover and indicates the work
outer loop. More precisely, the proof procedure is as shown in Figure 5.

This procedure has two sources for non-termination. Firstly, the call (2)
terminate (because first-order logic is undecidable). Secondly, selecting mode
might be done in an unfair way in step (1) (a sufficient fairness condition i
every model in $\mathcal{R}$ eventually). If the set $\mathcal{R}$ is infinite it is practically impossi
every model.

Hence, as an alternative to this strategy, we propose an implementation
Theorem 9. For this we have to presuppose a compact constraint system $\mathcal{R}$. A
ned in Section 2 this covers the important case of first-order restriction theo
even more restricted case, if the constraint theory admits a *generic model* (e.g
theories), then only one derivation suffices.

Recall that Theorem 9 does not refer to an explicit enumeration of the inter
in the restriction theory. Instead it suffices to enumerate finitely many deriv
empty tableaux such that $\mathcal{R} \models \exists R_1 \vee \cdots \vee \exists R_n$ (†), where the R_i are the c
stemming from the respective derivations of empty tableaux. We will not figu
control regime in greater detail here. The main difference is that the outer loo

INPUT: constraint clause set $\mathcal{C}$;
constraint solving decision procedure CSP($\mathcal{A}$) for every $\mathcal{A}$ in $\mathcal{R}$

repeat {outer loop}
Instantiate the CSP for some $\mathcal{A} \in \mathcal{R}$, and delete $\mathcal{A}$ from $\mathcal{R}$; (1)
Invoke PROTEIN; {inner loop} (2)
{It performs as follows wrt. constraints during the proof search:
The constraints are assembled and passed to CSP($\mathcal{A}$); (3)
CSP($\mathcal{A}$) reports "failure" or "solvable" to PROTEIN; (4)}
{Reaching this point implies that PROTEIN has terminated with a proof (5)}
until $\mathcal{R}$ is empty; (6) (7)

OUTPUT: "$\mathcal{C}$ is unsatisfiable"

Fig. 5. The proof procedure.

longer be driven by interpretations, and the termination (steps 6 and 7) are replaced by a proof of (†).

The inner loop is implemented by a transformation on the input clause set. In order to explain this we first recall that model elimination is a top-down proof procedure which proceeds on a dedicated goal clause, just as the SLD-resolution of PROLOG. In the generalisation towards full first-order clauses we may have more than one negative clauses, and each of these must in general be usable as a goal clause for refutations. On the other side, it suffices to use the negative clauses alone as goal clauses, because for every interpretation $A \in \mathcal{R}$ every unsatisfiable clause set $\mathcal{C}$ must contain at least one negative clause (because, otherwise, if C contains no negative clauses every interpretation $A \in \mathcal{R}$ can be extended to a model for $\mathcal{C}$ by assigning $true$ to the positive literals).

By this the search space will be pruned considerably, and model elimination keeps its goal-directed flavour. Technically, a negative input clause $\neg l_1 \vee \cdots \vee \neg l_n$ is made accessible as a goal clause by rewriting it to $goal \leftarrow l_1 \wedge \cdots \wedge l_n$ and adding the new goal clause ?- *goal*.

We want to point out that there is a strong interaction between the model elimination prover (ME) and the constraint solver (CSP). We are free to propagate and simplify constraints immediately after a constraint derivation step while in some cases it may be advantageous to delay constraint handling for efficiency reasons. This can be formalized by a so-called *simplification system*.

5 Final Remarks

In this paper, we presented the constraint model elimination calculus which is sound and complete. We proposed a PROLOG technology implementation. This procedure has at least two advantages. Firstly, we can exploit the existing, rather efficient PROLOG compilers. Secondly, it is possible to combine our calculus with existing constraint

solvers very easily. In many cases a direct combination of existing solvers is
e.g. we can solve the N-queens problem very fast [Van89, 122-132]. As another
we solved the Lion and Unicorn puzzle in about 0.1s on a Sun4, which is quit

We are currently thinking of using our approach for *disjunctive logic prog*
[LMR92], i.e. CLP with full negation (see also [Stu91]) where the compu
(disjunctive) answers is needed [Fur91].

Another idea is it not only to use special constraints but to add *inherent co*
to clauses, i.e. constraints which improve efficiency but do not change the
of clauses. This holds for the so-called *tautology pruning*, e.g. the transit
$L = (\neg p(X,Y) \vee \neg p(Y,Z) \vee p(X,Z))$ is equivalent to $L/(X \neq Y \wedge Y \neq$

Acknowledgements

We would like to thank Franz Baader, Jürgen Dix, Ulrich Furbach, Thom Frühw
chael Kühn, Olaf Menkens and Joachim Niehren for helpful discussions and c
on this paper.

References

[Bau93] Peter Baumgartner. Refinements of theory model elimination and a varia
contrapositives. Fachberichte Informatik 8/93, Universität Koblenz-Landau
1993.

[Bau94] P. Baumgartner. Refinements of Theory Model Elimination and a Varia
Contrapositives. In A.G. Cohn, editor, *11th European Conference on Ar*
telligence, ECAI 94. Wiley, 1994. (Long version available as Research Re
1993, University of Koblenz).

[BBH+90] Franz Baader, Hans-Jürgen Bürckert, Bernhard Hollunder, Werner Nutt, a
Siekmann. Concept logics. Research Report RR-90-10, DFKI, Kais
Saarbrücken, September 1990. Also in *Proceedings of the Symposium o*
tational Logics, Brüssel, Belgium, 1990.

[BDS93] Martin Buchheit, Franceso M. Donini, and Andrea Schaerf. Decidable
in terminological knowledge systems. *Journal of Artificial Intelligence*
1:109–138, 1993.

[BF94] Peter Baumgartner and Ulrich Furbach. Protein: A PROver with a Tory
INterface. In Alan Bundy, editor, *Proceedings of the 12th International C*
on Automated Deduction, Nancy, France, June/July 1994, pages 769–773
Berlin, Heidelberg, New York, 1994. LNAI 814.

[BFP92] Peter Baumgartner, Ulrich Furbach, and Uwe Petermann. A unified approac
reasoning. Fachberichte Informatik 15/92, Universität Koblenz-Landau,
1992.

[BGL85] Ronald J. Brachman, Victoria Pigman Gilbert, and Hector J. Levesque. A
hybrid reasoning system: Knowledge and symbol level accounts of KRY
Aravind Joshi, editor, *Proceedings of the 9th International Joint Conferenc*
cial Intelligence, Los Angeles, CA, August 1985, pages 532–539. Morgan K
Los Altos, CA, 1985. Volume 1.

[BH91] Franz Baader and Bernhard Hollunder. A terminological knowledge representation system with complete inference algorithms. In Harold Boley and Michael M. Richter, editors, *Proceedings of the International Workshop on Processing Declarative Knowledge, Kaiserslautern, Germany, July 1991*, pages 67–86. Springer, Berlin, Heidelberg, New York, 1991. LNAI 567.

[BS85] Ronald J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[Bür91] Hans-Jürgen Bürckert. *A Resolution Principle for a Logic with Restricted Quantifiers*. LNAI 568. Springer, Berlin, Heidelberg, New York, 1991.

[Bür94] Hans-Jürgen Bürckert. A resolution principle for constrained logics. *Artificial Intelligence*, 66(2):235–271, 1994.

[ECR94] ECRC GmbH, München. *ECLiPSe 3.4: (a) User Manual – (b) Extensions User Manual*, January 1994.

[FH94] Thom Frühwirth and Philipp Hanschke. Terminological reasoning with constraint handling rules. Technical Report ECRC-94-6, ECRC GmbH, München, 1994.

[Fur91] Ulrich Furbach. Answers for disjunctive logic programs. In Thomas Christaller, editor, *Proceedings of the 15th German Workshop on Artificial Intelligence, Bonn, September 1991*, pages 23–32. Springer, Berlin, Heidelberg, New York, 1991. IFB 285.

[JM94] Joxan Jaffar and Michael J. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19,20:503–581, 1994.

[Llo87] John Wylie Lloyd. *Foundations of Logic Programming*. Springer, Berlin, Heidelberg, New York, 2nd edition, 1987.

[LMR92] Jorge Lobo, Jack Minker, and Arcot Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT Press, Cambridge, MA, London, England, 1992.

[Lov68] D. Loveland. Mechanical Theorem Proving by Model Elimination. *JACM*, 15(2), 1968.

[SB94] Frieder Stolzenburg and Peter Baumgartner. Constraint model elimination and a PTTP-implementation. Fachberichte Informatik 10/94, Universität Koblenz-Landau, Koblenz, September 1994.

[Smu78] Raymond M. Smullyan. *What is the name of this book? The riddle of Dracula and other logical puzzles*. Prentice-Hall, Englewood Cliffs, NJ, 1978.

[SSS91] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[Sti85] Mark E. Stickel. Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1(3):333–355, 1985.

[Sti89] M. Stickel. A Prolog Technology Theorem Prover: A New Exposition and Implementation in Prolog. Technical note 464, SRI International, 1989.

[Sto93] Frieder Stolzenburg. An algorithm for general set unification and its complexity. In Eugenio G. Omodeo and Gianfranco Rossi, editors, *Proceedings of the Workshop on Logic Programming with Sets. In conjunction with the 10th International Conference on Logic Programming, Budapest, Hungary*, pages 17–22, June 1993.

[Sto94] Frieder Stolzenburg. Logic programming with sets by membership-constraints. In Norbert E. Fuchs and Georg Gottlob, editors, *Proceedings of the 10th Logic Programming Workshop*, Universität Zürich, 1994. Institut für Informatik. Technical Report ifi 94.10.

[Stu91] Peter J. Stuckey. Constructive negation for constraint logic programming. In *Proceedings of the Logic in Computer Science Conference*, pages 328–339, 1991.

[Van89] Pascal Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, Cambridge, MA, London, England, 1989.