Reasoning with Constraints and Well-Founded Negation

Frieder Stolzenburg · Oliver Obst*
Universität Koblenz-Landau, Rheinau 1, D–56075 Koblenz
E-mail: {stolzen,frvit}@informatik.uni-koblenz.de

Constraint reasoning and logic programming have been combined with great success. Constraint logic programming (CLP) [5] has gained a lot of interest, since it combines both fields in a theoretically sound manner while achieving efficiency by dedicated constraint solvers for practical applications. However in CLP, only Horn clauses are considered. But in many cases it is desirable also to have some form of negation in logic programming—see [1] for a survey—, which allows us to express exceptions to rules (e.g. with default rules) and to shorten formalisations (e.g. with the negation as finite failure rule) among other things. This is especially useful for solving diagnosis problems.

Therefore, we here propose a combination of constraint reasoning and general logic programs, where negation in rule bodies and also disjunctive rules (i.e. rules with more than one literal in their heads) are allowed, while preserving the full power of constraint reasoning. We do this on the basis of the disjunctive well-founded semantics (D-WFS) [2]. It is sound for general logic programs and can be adapted to the non-ground case with variables. The D-WFS coincides with the well-founded semantics [8] for normal programs (i.e. without disjunctive rules) and with the generalised closed world assumption [7] for positive disjunctive programs (i.e. without non-monotonic negation).

Our approach is essentially based on a calculus of program transformations that has been recently shown to be confluent and terminating for ground programs [2]. The most important transformation in this calculus is the partial evaluation property (GPPE) adapted for disjunctive programs. Unfortunately, GPPE is not sound for rules with variables because of the occurrence of unifiable atoms in the heads of rules. We make the GPPE sound by introducing equational constraints [6]. This immediately leads us to introduce constraint disjunctive logic programs and consequently to extend our transformations to this class of programs.

By this procedure, any constraint theory known from CLP can be exploited in the context of non-monotonic reasoning, not only equational constraints over the Herbrand domain. However, the respective constraint solver must be able to treat negative constraints of the considered constraint domain. Surprisingly, this framework shares the same nice properties as the original calculus. In summary, our framework—which is explained in detail in [4]—is a general combination of two paradigms: constraint logic programming and non-monotonic reasoning.

In this context, we want to present the following: First, we will outline the new framework, called constraint D-WFS. After that, we will show the usefulness of the approach with an example from diagnosis. Finally, we will report on the implementation of our calculus, which aims at incorporating a general logic programming deduction system with constraints.

1 The Framework

Definition 1 As mentioned earlier, we are interested in general logic programs, more precisely constraint programs, which is a finite set of rules of the form
\[(A_1 \lor \cdots \lor A_k) \leftarrow (B_1 \land \cdots \land B_m) \land (\neg C_1 \land \cdots \land \neg C_n)/R\]
where the part left of the slash is a (not necessarily ground) disjunctive rule, and \(R\) is a constraint formula, e.g. an equational constraint. Equational Constraints are introduced and discussed in detail in [6, 3]. We will identify \(A, B\) and \(C\) by their sets of atoms \(\{A_1, \ldots, A_k\}\), \(\{B_1, \ldots, B_m\}\) and \(\{C_1, \ldots, C_n\}\), respectively.

The general system architecture is given in Figure 1. As one can see the general procedure is a fol-

*This research is partly supported by a grant from the German science foundation DFG for the DisLoP project on disjunctive logic programming at the University of Koblenz, Germany.
allows: normalisation, transformation, querying the residual program.

**Normalisation.** At first, the input program is normalised, i.e., we perform some preprocessing that simplifies the given program. This means we remove tautological rules which are useless for deduction. As in the resolution calculus, we need the concept of factorisation in order to treat rules with variables correctly. In addition, we need a rule that merges rules containing the same predicate symbols, but different constraints. Now follows the formal definition of these rules. They are applied as long as possible on the original program.

**TAUT:** We replace the rule \( \mathcal{A} \leftarrow B \land \neg C / R \) by
\[
\mathcal{A} \leftarrow B \land \neg C / R \land (A \neq B)
\]
where \( A = \mathcal{P}(x_1, \ldots, x_k) \in \mathcal{A} \) and \( B = \mathcal{P}(y_1, \ldots, y_k) \in \mathcal{B} \) are atoms with the same predicate symbol. Note that \( A \neq B \) means \( x_i \neq y_i \lor \cdots \lor x_k \neq y_k \).

**FACTOR:** Let \( \mathcal{A} \leftarrow B \land \neg C / R \) be a rule, and \( A, B \in \mathcal{A} \) atoms with the same predicate symbol, i.e., \( A = \mathcal{P}(x_1, \ldots, x_k) \) and \( B = \mathcal{P}(y_1, \ldots, y_k) \). The two rules
\[
\mathcal{A} \leftarrow B \land \neg C / R \land (A \neq B)
\]
\[
(\mathcal{A} \setminus \{B\}) \leftarrow B \land \neg C / R \land (A = B)
\]
are called the factorisation of \( L/R \).

**MERGE:** Let \( \mathcal{A} \leftarrow B \land \neg C / R \) and \( \mathcal{A}' \leftarrow B' \land \neg C' / R' \) be variants of program rules such that \( A = A', B = B' \), and \( C = C' \). Then we can replace (merge) the original program rules by the following single one:
\[
\mathcal{A} \leftarrow B \land \neg C / R \lor R'
\]

**Transformation.** After normalisation, the program is transformed by some transformation rules. The most important one is partial evaluation, which roughly means a positive body literal has to be replaced by its definition. We also remove non-minimal rules by performing subsumption. The negative conditions are treated by the positive and negative reduction rules. Look at the following definition.

**Definition 2** If \( \mathcal{A} \leftarrow B \land \neg C / R \) is a rule in the constraint program \( \Phi \), then it can be replaced by one of the following (sets of) rules. Constraint simplification can be applied immediately to each newly generated rule.

**GPPE:** Let \( B \) be a distinguished atom in \( B \). Then, replace the rule \( \mathcal{A} \leftarrow B \land \neg C / R \) by
\[
\mathcal{A} \cup \mathcal{A}_1 \setminus \{B\} \leftarrow (B \setminus \{B\}) \lor B_1 \land \neg (C \cup C_3) / R \land R_3
\]
\[
\vdots
\]
\[
\mathcal{A} \cup \mathcal{A}_k \setminus \{B\} \leftarrow (B \setminus \{B\}) \lor B_k \land \neg (C \cup C_k) / R \land R_k
\]
where each \( \mathcal{A}_i \leftarrow B_i \land \neg C_i / R_i \), for \( 1 \leq i \leq k \), is a variant of a rule in \( \Phi \) that contains an atom \( A \) in its head with the same predicate symbol as \( B \), such that \( A = B \).

**NMIN:** Replace the rule \( \mathcal{A} \leftarrow B \land \neg C / R \) by
\[
\mathcal{A} \leftarrow B \land \neg C / R \land \neg R'
\]
for some variant \( \mathcal{A'} \leftarrow B' \land \neg C' / R' \) of a rule in \( \Phi \) such that \( A' \subseteq A \), \( B' \subseteq B \) and \( C' \subseteq C \) hold.
RED+: Replace the rule $A \leftarrow B \wedge \neg C / R$ by the two rules:

$$A \leftarrow B \wedge \neg C / R \wedge R'$$
$$A \leftarrow B \wedge \neg (\neg (C \cup \{C\}) / R \wedge \neg R')$$

where $C$ is an atom in $\Phi$ and $C' / R'$ is a variant of a constraint atom in $\text{heads}(\Phi)$ such that $C = C'$.

RED-: Replace the rule $A \leftarrow B \wedge \neg C / R$ by:

$$A \leftarrow B \wedge \neg C / R \wedge \neg R'$$

for some variant $A' \leftarrow R'$ of a rule in $\Phi$ such that $A' \subseteq C$.

Residuum and Querying. Although our calculus of transformations is confluent, it is not always terminating. But since all transformations preserve the intended D-WFS semantics, we can read off answers to queries from the residuum of the original program. The residuum is an irreducible normal form of a program, which is reached after a terminating sequence of transformations. The following theorem characterises how we can do query answering by inspecting the residuum of a program (which does not contain any positive body literals). For more details, the reader is referred to [4]. We will illustrate the calculus with an example in the next section.

**Theorem 3** Let $\Phi$ be a constraint program with the residuum $\Phi$, and $\psi = L / R$ be a pure constraint disjunction. Then $\Phi \vdash \psi$ iff one of the following two cases applies:

1. There is a rule $(A \leftarrow R')$ in $\Phi$ subsuming $\psi$.
2. $L$ contains a negative literal $\neg A$ such that there exists a constraint atom $A' / R'$ in $\text{heads}(\Phi)$, and $A' / \neg R'$ subsumes $A / R$.

## 2 A Diagnosis Example

In model-based diagnosis, a simulation model of the device under consideration is used to predict its normal behaviour. This approach uses a logical first-order system description of the device. It consists of a set of axioms characterising the behaviour of system components of certain types. The topology is modelled separately by a set of facts. In summary, the diagnostic problem is described by system description SD, a set of components and a set OB of observations. In addition, we need some first-order axioms AX describing the general behaviour of the parts.

**Example 4** Consider the electric circuit in Figure 2. It behaves faulty, since the input of both inverters is low, but the output of the or gate is also low. So which component(s) may be faulty? For this, we first formalise the example by the constraint program in Figure 3. There, $hi(c, n, x)$ means that the input or output $x$ of component $n$ of type $c$ is high. With each component we associate a behavioural mode $ab(c, n)$ saying that the respective component is faulty. There may be many possible diagnoses, but usually we are only interested in minimal ones. Thus, we state the single-fault assumption in the last part of our formalisation (MD), i.e. we assume that exactly one component is faulty.

![Figure 2: A diagnosis example.](image)

![Figure 3: Formalisation of the example.](image)
After the transformation process. Therefore, in our cur- controlling the process explicitly. The process is

tin of implementing constraint D-WFS is controlling

cording to Figure 4. One of the main problems

we can proceed as follows: after each application of

the GPPE the other transformations are applied as

as long as possible. Alternatively, constraint simplifi-

cation can be initiated.

On the one hand, this procedure seems to be in-

convenient because of the dominant role of user in-

teraction. But on the other hand, it is impossible to

compute the residuum always automatically (except

for special program classes, e.g. Datalog\textsuperscript{\textvisiblespace} and disuni-

fication). Last but not least, designing such an interaction module

is an interesting task.

3 Implementing the Calculus

We are currently implementing our approach ac-

ording to Figure 4. One of the main problems

of implementing constraint D-WFS is controlling

the transformation process. Therefore, in our cur-

rent implementation, we provide user interaction for

the equational constraints are made explicit) subsumes the last axiom. This means, applying

NMf on the last axiom transforms this into

\( \leftarrow h\bar{u}(\text{not}, n_s i), h\bar{u}(\text{not}, n_s o), \neg ab(\text{not}, n)/n \neq 1 \). After finishing the transformation process we obtain

the residuum shown below, from which we can infer that the or gate behaves abnormal (provided

there is only one fault).

\[
\begin{align*}
\text{high}(\text{inv1}, o) & \leftarrow \\
\text{high}(\text{or1}, r) & \leftarrow \\
\text{high}(\text{inv2}, o) & \leftarrow \\
\text{high}(\text{or1}, r) & \leftarrow \\
\text{ab}(\text{or1}) & \leftarrow \\
\end{align*}
\]

References

[1] K. R. Apt and R. N. Bol. Logic programming and


