A Unifying Framework for Abductive and Constraint Logic Programming

Gerhard Wetzel
Logic Based Systems Lab
Department of Computer Science
Brooklyn College, NY 11210, USA
gw@sci.brooklyn.cuny.edu
Robert A. Kowalski, Francesca Toni
Department of Computing
Imperial College, London SW7 2BZ, UK
{rak,ft}@doc.ic.ac.uk

Abstract

We present a unifying framework for the areas of Abductive Logic Programming (ALP), Constraint Logic Programming (CLP) and Semantic Query Optimization (SQO). Links between these areas are pointed out and used to motivate an extension of normal LP by integrity constraints. Moreover, LP if clauses are replaced by if-and-only-if definitions with disjunctions in their bodies.

A proof procedure, consisting of four core operations, is defined for the new framework. It simplifies previously proposed abductive proof procedures [Fu96, DeDS97] and generalizes them so that they are no longer specialized on abduction. Some possible extensions of the core operations are also presented.

Traditional problems from LP, ALP, CLP, and SQO as well as interdisciplinary AI problems constitute possible applications of the new framework.

1 Motivation

A normal logic program consists of clauses of the form

$H \leftarrow B_1 \wedge \ldots \wedge B_n$

where H is an atom and the B_i are literals. In order to reduce the amount of non-deterministic search, it is useful to be able to take several clauses for the same predicate (or at least for unifiable head atoms H) into account at the same time. A simple and elegant way to do this is to use if-and-only-if definitions with disjunctions in the body instead of (if) clauses:

$H \leftrightarrow D_1 \vee \ldots \vee D_n$

where H is an atom and each D_i is a conjunction of literals. In order to give the user direct control over the desired amount of non-determinism, a concept of suspension known from concurrent LP languages is applied to atoms defined in this way (for details see next section).

Many extensions of normal logic programs have been proposed. [LlTo84] showed that an *extended clause*

$H \leftarrow F$

where F is an arbitrary sentence of first-order logic, can be transformed into a normal logic program, but at the same time argued that programs using extended clauses are preferable because they *"allow the solution of many problems to be expressed in a form similar to the specification of the problem."* This argument will also be made here, but the extension of normal LP to be proposed is much more conservative than the extended clauses of [LlTo84]: it is the use of integrity constraints

$A_1 \wedge \ldots \wedge A_n \rightarrow B_1 \vee \ldots \vee B_m$

where the A_i and B_j are atoms. Whereas normal LP clauses are executed in backward goal reduction manner, integrity constraints allow a forward reasoning interpretation: if the condition holds, then derive (or *propagate*) the conclusion.

Just like extended clauses, integrity constraints can also be transformed into normal logic programs [KoSa90], but again operationally useful information may be lost in the transformation and the resulting program may not only be further from the specification (and thus harder to understand and to prove correct), but also less efficient to run. Moreover, integrity constraints are similar to *if-then* and *condition-action* (production) rules which have proven a popular tool in Artificial Intelligence (see the numerous examples given in AI textbooks such as [RuNo95] and [Wi92]). Large-scale commercial AI applications like R1/XCON [MD82] make use of such rules. The availability of integrity constraints in an LP framework might thus make the framework more attractive for AI application developers and increase the general popularity of Logic Programming. And, as will be shown in the following, integrity constraints play a crucial role in the unification of SQO, ALP and CLP.

A standard usage of integrity constraints is to check the consistency of a deductive database. The predicates of a deductive database are often divided into *extensional* and *intensional* ones, and the database is accordingly separated into an extensional database (EDB) of facts and an intensional database (IDB) which corresponds to a normal logic program. Given a query to a deductive database, it may be more efficient to avoid accessing the EDB because looking up information in the EDB may be computationally explosive if the EDB is very large. Therefore it is desirable to transform — and thus optimize — the query using only the IDB and the integrity constraints. It may even be possible to show that a query has no answers without accessing the EDB at all. The area of research concerned with such query transformations is called **Semantic Query Optimization** (SQO, [ChGrMi90]).

Example 1.1 (SQO)
Query: `employee(X)` $\wedge$ `position(X,manager)` $\wedge$ `bonus(X,B)` $\wedge$ `B=0`
Integrity constraint: `position(X,manager)` $\wedge$ `bonus(X,B)` $\rightarrow$ `B`$\neq$`0`
Suppose that `employee`, `position` and `salary` are all defined in the EDB. SQO may use the integrity constraint to show that the query has no answers without accessing the EDB. If PROLOG were used to process the query, it would have to look at every individual `employee` record.

The optimization of a query by means of integrity constraints may introduce new atoms of EDB predicates into the query. [Ka91a, Ka91b] pointed out that it is possible to establish a correspondence between EDB predicates and the abducibles in **Abductive Logic Programming** (ALP, [KaKoTo93]).

The conventional abductive framework is a triple $(\mathcal{T}, Ab, \mathcal{IC})$ with $\mathcal{T}$ being a logic program, Ab a set of predicate symbols (called *abducibles*) and $\mathcal{IC}$ a set of implications (called integrity constraints). In deductive database terminology, $\mathcal{T}$ represents the IDB, the abducibles correspond to the EDB predicates, and $\mathcal{IC}$ corresponds to the database integrity constraints.

Abduction is hypothetical, explanatory reasoning: given an observation or goal G, find an abductive hypothesis Δ explaining G. This can be formalized as follows:

(1) Δ consists of atoms of abducibles only

(2) $\mathcal{T} \cup \Delta \models G$

(3) $\mathcal{T} \cup \Delta$ satisfies the integrity constraints $\mathcal{IC}$

There are numerous different ways of defining integrity constraint satisfaction. At the very least it means that $T \cup \Delta$ has to be consistent with $\mathcal{IC}$ (*consistency view*).

Example 1.2 (ALP)
$\mathcal{T}$: `bird` $\leftarrow$ `albatross`
`bird` $\leftarrow$ `penguin`
Ab : { `penguin, albatross, flies` }
$\mathcal{IC}$: `penguin` $\wedge$ `flies` $\rightarrow$ `false`
Let the goal G be
`bird` $\wedge$ `flies`.
Then the explanation
$\Delta_1 = \{\texttt{albatross}, \texttt{flies}\}$
satisfies (1)–(3) under the consistency view whereas the explanation $\Delta_2 = \{\texttt{penguin}, \texttt{flies}\}$ is inconsistent with the integrity constraint and violates condition (3) under any view of integrity constraint satisfaction.

As integrity constraints are used for consistency checking in ALP (as well as in SQO), they are also useful for handling constraints in **Constraint Logic Programming** (CLP, [JaMa94]). Just like EDB predicates in SQO and abducibles in ALP, constraint predicates are not processed by backward reasoning with their definitions. Instead, atoms of constraint predicates are checked for satisfiability, either by a built-in constraint solver or by explicit integrity constraints in the form of user-definable Constraint Handling Rules (CHRs, [Fr95]).

Example 1.3 (CLP)
Goal: `X > 1` $\wedge$ `1 > Y` $\wedge$ `Y > X`
$\mathcal{IC}$: `X > Y` $\wedge$ `Y > Z` $\rightarrow$ `X > Z`
`X > N` $\wedge$ `N > X` $\rightarrow$ `false`
A CLP system working over a finite integer domain (such as `cc(FD)` [VHSaDe93]) or the domain of real numbers (CLP($\mathcal{R}$), [JaLa87]) does not need the explicit integrity constraints as its constraint solver recognizes that the constraints in the goal are unsatisfiable over their respective domains.
Used as CHRs, the first integrity constraint (transitivity) adds `X > Y` $\wedge$ `1 > X` to the goal. The second integrity constraint then becomes applicable and generates `false`.
Note that standard LP, which cannot make use of the integrity constraints, would either refuse to process the uninstantiated arithmetic expressions or fail to terminate if it tried to unfold a separately given definition of "`>`".

2 Unifying Framework and Semantics

This section defines a unifying framework for the areas of ALP, CLP and SQO. Whereas previous proposals to relate ALP to deductive databases (and thus potentially to SQO) [Ka91a, Ka91b] and to integrate ALP and CLP [KaMi95, Mm92] focused either on particular aspects of a combined framework or were developing hybrid systems, here a unified framework for all three areas is presented. There have also been other proposals to enhance LP by integrity constraints, most notably SATCHMO [MaBr88], but they did not address the issue of unifying different areas of LP.

The unifying framework to be proposed has been called $CALOG$ — C for constraints, A for abducibles and LOG for logic. As already indicated at the beginning of section 1, knowledge in the CALOG framework is represented by if-and-only-if *definitions* of the form

$H \leftrightarrow D_1 \vee \ldots \vee D_n,\ n \geq 1$

and by *integrity constraints* of the form

$Cond \rightarrow Conc,$

where $Cond$ is a conjunction of atoms and $Conc$ is a disjunction of atoms.

Definitions and integrity constraints may involve atoms of three kinds of predicates:

- *user-defined* predicates, defined by a theory $\mathcal{T}_u$ of user-provided definitions,
- *built-in* predicates, implicitly defined by a theory $\mathcal{T}_b$ of built-in definitions,
- *external* predicates, defined by an inaccessible or unknown theory $\mathcal{T}_e$ of definitions.

The union $\mathcal{T}_u \cup \mathcal{T}_b \cup \mathcal{T}_e$ of all three theories is denoted by $\mathcal{T}$.

User-defined predicates correspond to ordinary predicates in standard LP, intensional predicates in SQO, defined predicates (or non-abducibles) in ALP, and non-constraint predicates in CLP.

Built-in predicates, which include equality as well as arithmetic inequality and arithmetic operators, correspond to constraint predicates in CLP and to equality in standard LP and ALP (equality is the single constraint predicate in LP from a CLP point of view).

External predicates correspond to abducibles in ALP and extensional predicates in SQO. As their definitions are inaccessible, knowledge about external predicates can only be provided by integrity constraints.

Integrity constraints are also used to express properties of the accessible definitions. Any sentence true in all intended models of a theory $\mathcal{T}$ is called a *property* of $\mathcal{T}$. Making properties of definitions explicit in the form of integrity constraints has several practical advantages:

- it can help speed up program execution by providing reasoning shortcuts;
- it allows forward reasoning rules to be expressed without translating them into (backward reasoning) definitions, and thus
- it provides a new way of formulating algorithms in logical form.

The *initial goal* is a conjunction of literals with all variables being free. The computational task, whose formalization into a proof procedure is outlined in section 3, is to reduce a given initial goal to a disjunction of answers, which is equivalent to the initial goal in the sense that the goal and the disjunction are satisfied, in $\mathcal{T}$, by the same assignments to the free variables. Such answers are obtained by using definitions to *unfold* atomic goals to equivalent disjunctions of goals which can then be *split* using distributivity. Surrogate subgoals are introduced by *propagation* (resolution) with integrity constraints. As in Operations Research, surrogate subgoals are logically redundant, but may be easier to solve.

Atoms cannot be unfolded further if they are suspended. An atom A is *suspended* if its definition is inaccessible or if there is no accessible definition $H \leftrightarrow D_1 \vee \ldots \vee D_n$ such that A is an instance of H (i.e. $A = H\sigma$ for some substitution σ). Otherwise A is called *reducible*.

Thus atoms of external predicates are always suspended, whereas atoms of user-defined and built-in predicates are suspended if they are insufficiently instantiated to be an instance of the head of a definition. Suspension can be used to control the amount of non-determinism and thus the size of the search space.

Example 2.1 Let the user-defined predicate `p` be defined by the following two definitions:

```
p(X,a) ↔ q(X)
p(X,b) ↔ r(X)
```

Then `p(X,Y)` and `p(a,Y)` are suspended, but `p(X,a)` and `p(a,b)` are reducible. If the definition

```
p(X,Y) ↔ ( Y=a ∧ q(X) ) ∨ ( Y=b ∧ r(X) )
```

were used instead, then atoms of `p` would never be suspended.

Built-in definitions are assumed to be such that atoms like `X<Y`, `X>0` and `plus(2,X,Y)` are suspended, but `1<0`, `1>0` and `plus(2,2,X)` are not suspended (and can be reduced to `false`, `true` and `X=4`, respectively).

Integrity constraints are used to process suspended goals in order to decide, if possible, whether a conjunction of suspended goals is satisfiable. Propagation with integrity constraints may add reducible atoms to the goal which may allow the computation using definitions to proceed.

Answers to goals have to satisfy the integrity constraints, and an important feature of the CALOG framework is the *propertyhood view* of integrity constraint satisfaction. It requires the integrity constraints to be properties of the definitions, i.e. to be true in the intended models of the theory $\mathcal{T}$, notation $\mathcal{T} \models_{int} \mathcal{IC}$.

The propertyhood view thus depends on the chosen notion of intended model. If $\mathcal{T}$ corresponds to a Horn clause theory, the simplest choice is the unique minimal Herbrand model. Then

```
p → false
```

is a property (but not a theorem) of the theory containing only

```
p ↔ p
```

In the general case, when $\mathcal{T}$ corresponds to a normal logic program containing negative literals, different notions of intended model may be considered and the framework leaves the choice open to the user, subject to a few restrictions given in [We97]. For example, $\mathcal{T}$ is required to have at least one intended model to rule out the case where every formula is a trivial property of $\mathcal{T}$ (but three-valued intended models are allowed, so this is not too strong a restriction).

The framework's declarative semantics introduces three different notions of answer so that the proof procedure can be sound with respect to at least the weakest notion (for any valid choice of intended model). The strongest notion is that of *ideal answer*: A is an ideal answer to the initial goal G_0 if

(A1) A is a conjunction of suspended atoms and implications (whose atoms are also suspended)

(A2) $\mathcal{T} \models_{int} \tilde{\forall}[G_0 \leftrightarrow A \vee \mathit{Rest}]$ for some formula Rest

(A3) $\mathcal{T} \not\models_{int} \neg\tilde{\exists} A$

Here $\tilde{\forall}$ and $\tilde{\exists}$ denote the universal and existential closures, respectively, quantifying over all free variables.

Condition (A1) means that goals have to be reduced as far as possible, i.e. until all subgoals are suspended.

Condition (A2) says that G_0 and $A \vee Rest$ have to be true in the intended models of $\mathcal{T}$ for the same values of the free variables in G_0.

Condition (A3) requires A to be satisfiable in at least one intended model of $\mathcal{T}$. This implies that $\mathcal{IC} \cup \tilde{\exists} A$ is consistent since $\mathcal{T} \models_{int} \mathcal{IC}$ is required by the propertyhood view (and since $\mathcal{T}$ must have at least one intended model).

The given answer definition is idealistic in two ways. Firstly, $\mathcal{T}$ contains the inaccessible part $\mathcal{T}_e$ which is not available (to the proof procedure) for checking conditions (A2) and (A3). Secondly, condition (A3) cannot generally be verified in practice because suspended atoms can be tested for consistency only by integrity constraints, and the integrity constraints may not provide a complete axiomatization of the theory (or, in CLP terminology, they may not be *satisfaction complete*).

Thus, in addition to ideal answers, two weaker answer notions are defined. It turns out that it is only necessary to weaken condition (A3). A is a *good answer* to G_0 if (A1) and (A2) hold and

(A3$_g$) $\mathcal{T}' \cup \mathcal{IC} \cup CET \cup \tilde{\exists} A$ is consistent (CET denotes Clark's Equality Theory [Cl78])

The suspension of user-defined and built-in predicates in $\mathcal{T}'$ may still prevent the proof procedure from identifying violations of condition (A3$_g$). Therefore an even weaker answer notion is introduced: A is a *weak answer* to G_0 if (A1) and (A2) hold and

(A3$_w$) $\mathcal{IC} \cup CET \cup \tilde{\exists} A$ is consistent

Every answer computed by the proof procedure presented in the next section satisfies at least the conditions for being a weak answer to the initial goal.

3 Proof Procedure

This section describes a rather abstract (high-level) proof procedure for the CALOG framework which can be regarded as a generalization and simplification of two abductive proof procedures, the Iff Proof Procedure [Fu96] and SLDNFA [DeDS97]. By mainly ignoring efficiency issues, the proposed proof procedure may serve as a scheme for a family of proof procedures which can be obtained as instances by restricting and extending the abstract procedure in different ways (some of these possibilities are discussed in section 4).

A *derivation* of a goal G_n from an initial goal G_0 is a finite sequence of goals $G_0, G_1, \ldots, G_n$ where G_1 is obtained from G_0 by conjoining all integrity constraints in $\mathcal{IC}$ to G_0, and, for $0 < i < n$, G_{i+1} is obtained from G_i by applying one of the following operations:

1. *unfolding*
2. *propagation*
3. *logical equivalence transformation*

4. *equality rewriting*

1. *Unfolding* replaces a reducible atom A, which is either a conjunct of the goal or is a condition of an implication in the goal, by its definition. I.e. if there is a substitution σ such that $A = H\sigma$ and $H \leftrightarrow D_1 \vee \ldots \vee D_m$ is an accessible definition in $\mathcal{T}$, then replace A by $(D_1 \vee \ldots \vee D_m)\sigma$.

2. *Propagation* is a form of resolution. If a subgoal contains the conjuncts $p(s_1, \ldots, s_n) \wedge Cond \rightarrow Conc$ and $Susp \rightarrow p(t_1, \ldots, t_n)$, then the "resolvent" $Cond \wedge Susp \wedge (s_1 = t_1 \wedge \ldots \wedge s_n = t_n) \rightarrow Conc$ can be added to the subgoal. *Susp* is (usually) a conjunction of suspended atoms; the equalities in the resulting implication may be able to instantiate atoms in *Susp* such that they can be further unfolded.

3. *Logical equivalence transformations* include splitting (replace $(A \vee B) \wedge C$ by $(A \wedge C) \vee (B \wedge C)$) and logical simplifications such as the replacement of $A \wedge$ `false` by `false`.

4. *Equality rewriting* implements the unification algorithm to deal with equality atoms introduced into the goal. The rewrite rules used are adapted from [Fu96] where they were based on [MaMo82]. Conceptually, equality rewriting may also be regarded as propagation with the *CET* axioms of Clark's Equality Theory, followed by deletion of the no longer needed, rewritten terms.

If there is a derivation of $G_n = D \vee Rest$ from an initial goal G_0 and the operations of the proof procedure have been exhaustively applied to the disjunct D of the goal G_n, then a *computed answer* to G_0 is extracted from D after omitting all implications in D which are or originate from integrity constraints (see [We97] for details and reasons) and transforming all remaining implications in D into denials (implications with `false` as the conclusion). The latter ensures that no further steps of propagation (with atoms not present in D) could generate new conclusions.

Example 3.1 Suppose the goal disjunct D is

`p(a) ∧ q(a) ∧ [ p(X) → q(X) ]`

If further instances of `p` besides `p(a)` were true, then further instances of `q` would have to be true as well (but they might be defined as `false` in $\mathcal{T}$). Therefore the implication is transformed into a denial and the computed answer obtained from D is

`p(a) ∧ q(a) ∧ [ p(X) ∧ X≠a → false ]`

The operations of the proof procedure are all sound in the sense that they derive goals which are equivalent in $\mathcal{T}$, i.e.

$\mathcal{T} \models_{int} \tilde{\forall}[G_i \leftrightarrow G_{i+1}]$

Every computed answer to G_0 is at least a weak answer to G_0. Under certain further assumptions (see [We97]) soundness results with respect to the stronger answer notions are also obtainable. Refutation soundness (good and ideal answers are never rejected) can be proven. Completeness results (for every ideal answer there is a corresponding computed answer) are only obtainable for special cases and for extensions of the proof procedure.

4 Extensions

[Ko92] proposed a generalization of constraint propagation which requires the explicit representation of disjunctions provided by the CALOG framework. The extension of the proof procedure which allows propagation within and across disjunct boundaries has been called

Constraint Propagation with Disjunctions (CPD). A first formalization of these techniques was presented in [WeKoTo95]; see [We97] for more details.

The CPD methods can be viewed as an implementation and generalization of several methods of handling disjunctive constraints in CLP, among them the forward-checking and look-ahead techniques of CHIP [VH89], Generalized Propagation [LPWa92] and Constructive Disjunction [VHSaDe93, JoSo93]. Note that, in the CALOG framework, any disjunction can function as a disjunctive constraint. The CPD methods are also related to the Andorra principle [Wa88] and encapsulated search as in AKL [JaHa91] and Oz [ScSm94].

More technical improvements of the proof procedure are needed with respect to the efficiency of propagation and its termination. Since propagation has been defined as resolution, the problem of restricting propagation can be approached by looking at resolution restrictions proposed in the theorem-proving literature. Possible restrictions include P_1-resolution, which corresponds to the requirement that the expression *Susp* in the definition of propagation is empty, and hyper-resolution, which corresponds more closely to the forward reasoning interpretation of integrity constraints given in section 1 (only apply an integrity constraints if all of its conditions atoms can be shown to hold).

Another important extension of the proof procedure is the *deletion* operation. It interprets integrity constraints as identifying logically redundant and operationally useless subgoals which can be removed to simplify the goal. Ideally, deletion should be applied automatically by the proof procedure, e.g. the atom `X>0` should be deleted as soon as the atom `X>1` is added to the goal. This is logically justified by the transitivity rule

`X>Y` $\wedge$ `Y>Z` $\rightarrow$ `X>Z`

which is a property of the built-in definition of the "`>`" predicate. However, the same rule can also be applied by propagation in case the conclusion `X>Z` is operationally useful. Criteria have to be worked out when integrity constraints are to be used for propagation and when for deletion.

5 Implementation and Applications

PROCALOG [WeKoTo96], short for *PROgramming with Constraints and Abducibles in LOGic* (or *PROgramming in CALOG*), is a realization of the CALOG framework and proof procedure as a programming language. It includes several of the extensions proposed in section 4.

Two prototype implementations have been used in [We97] to provide some computational results for solutions to example applications, e.g. the n-queens problem and a warehouse location problem. A more efficient implementation (based on one of the two prototypes) is currently being developed and already yields significantly better results than those reported in [We97]. The CPD methods mentioned in the previous section have been part of all three implementations and special emphasis was placed on assessing their practical effectiveness.

n	#S	1-r., no CPD		1-r., g-t-l CPD			2-r., full CPD		2-r., Prolog	
4	2	0.1	45	0.1	5	(5)	2.0	3	2.9	11
5	10	0.3	176	0.3	13	(13)	13.3	10	15.3	36
6	4	1.5	745	1.2	39	(39)	56.0	20	99.5	120
7	40	7.7	3072	5.5	107	(107)	547.0	71	n/a	n/a
8	92	42.7	13755	27.7	415	(383)	n/a	n/a	n/a	n/a
9	352	241.6	64336	145.9	1641	(1477)	n/a	n/a	n/a	n/a
10	724	1426.7	?????	779.3	6665	(5715)	n/a	n/a	n/a	n/a

The computational results indicate that a careful usage of the CPD methods cuts down both the size of the search tree and the computation time. A full application of the methods may further prune the search tree, but at the cost of increasing computation times.

The framework and its implementations lend themselves to many well-known problems and applications of Artificial Intelligence, especially problems with a large search space (where even a limited use of the CPD extensions can drastically reduce the search tree) and applications relying on knowledge representation in the form of condition-action (if-then) rules (where integrity constraints can often be used to model the rules in a logical way). The latter include expert systems for configuration problems or medical diagnosis.

Examples of constraint satisfaction problems (such as the n-queens problem or map colouring), Operations Research applications (such as job-shop scheduling and warehouse location problems) and an approach to configuration which bears some similarity to the CALOG framework are discussed in [We97].

The use of integrity constraints to improve the efficiency of logic (PROLOG) programs without having to abandon the original, natural specification is another interesting application of the framework.

References

[ChGrMi90] Chakravarthy, U. S.; Grant, J.; Minker, J.: *Logic-Based Approach to Semantic Query Optimization*, ACM Transactions on Database Systems 15 (2), pp. 162–207, 1990

[Cl78] Clark, K. L.: *Negation as failure*, in: Gallaire, H.; Minker, J. (eds.): *Logic and Data Bases*, pp. 292–322, Plenum Press 1978

[DeDS97] Denecker, M.; De Schreye, D.: *SLDNFA: an abductive procedure for abductive logic programs*, to appear in Journal of Logic Programming

[Fr95] Frühwirth, T.: *Constraint Handling Rules*, in: Podelski, A. (ed.): *Constraint Programming: Basic and Trends*, pp. 90–107, LNCS 910, Springer Verlag 1995

[Fu96] Fung, T. H.: *Abduction by Deduction*, Ph.D. Thesis, Imperial College 1996

[JaHa91] Janson, S.; Haridi, S.: Programming Paradigms of the Andorra kernel language, in: Saraswat, V.; Ueda, K. (eds.): *Proc. of the Int. Symp. on Logic Programming*, pp. 167–186, MIT Press 1991

[JaLa87] Jaffar, J.; Lassez, J.-L.: *Constraint Logic Programming*, Proc. of the 14th ACM Symp. on the Principles of Programming Languages, pp. 111–119, 1987

[JaMa94] Jaffar, J.; Maher, M.: *Constraint Logic Programming: A Survey*, Journal of Logic Programming 19/20, pp. 503–581, 1994

[JoSo93] Jourdan, J.; Sola, T.: *The Versatility of Handling Disjunctions as Constraints*, in: Bruynooghe, M.; Penjam, J. (eds.): *Proc. of the 5^{th} Intern. Symp. on Programming Languages Implementation and Logic Programming*, pp. 60–74, Springer Verlag 1993

[Ka91a] Kakas, A. C.: *Deductive Databases as Theories of Belief*, Technical Report, Imperial College, 1991

[Ka91b] Kakas, A. C.: *On the Evolution of Deductive Databases*, Technical Report, Imperial College, 1991

[KaKoTo93] Kakas, A. C.; Kowalski, R. A.; Toni, F.: *Abductive Logic Programming*, Journal of Logic and Computation 2 (6), pp. 719–770, 1993

[KaMi95] Kakas, A. C.; Michael, A.: *Integrating Abductive and Constraint Logic Programming*, in: Sterling, L. (ed.): *Proc. of the 12^{th} Int. Conf. on Logic Programming*, pp. 399–413, MIT Press 1995

[Ko92] Kowalski, R. A.: *A dual form of logic programming*, Lecture Notes, Workshop in Honour of Jack Minker, University of Maryland 1992

[KoSa90] Kowalski, R. A.; Sadri, F.: *Logic Programs with Exceptions*, in: Warren, D. H. D.; Szeredi, P. (eds.): *Proc. of the 7^{th} Int. Conf. on Logic Programming*, pp. 598–613, MIT Press 1990

[LlTo84] Lloyd, J. W.; Topor, R. W.: *Making Prolog more expressive*, Journal of Logic Programming 1 (3), pp. 225–240, 1984

[LPWa92] Le Provost, T.; Wallace, M.: *Generalised Propagation Over the CLP Scheme*, ECRC Technical Report 92-01, 1992

[MaBr88] Manthey, R.; Bry, F.: *SATCHMO: A Theorem Prover Implemented in PROLOG*, in: Lusk, E.; Overbeek, R. (eds.): *Proc. of the 9^{th} Conf. on Automated Deduction*, pp. 415–434, LNCS 310, Springer-Verlag 1988

[MaMo82] Martelli, A., Montanari, U.: *An efficient unification algorithm*, ACM Trans. on Prog. Lang. and Systems 4 (2), pp. 258–282, 1982

[MD82] McDermott, J.: *R1: A Rule-Based Configurer of Computer Systems*, Artificial Intelligence 19 (1), pp. 39–88, 1982

[Mm92] Maïm, E.: *Abduction and Constraint Logic Programming*, in: Neumann, B.: *Proc. of the 10^{th} European Conf. on Artificial Intelligence*, 1992

[RuNo95] Russell, S. J.; Norvig, P.: *Artificial Intelligence — A Modern Approach*, Prentice Hall, 1995

[ScSm94] Schulte, C.; Smolka, G.: *Encapsulated Search for Higher-order Concurrent Constraint Programming*, in: Bruynooghe, M. (ed.): *Proc. of the Int. Logic Programming Symposium*, pp. 505–520, MIT Press 1994

[VH89] Van Hentenryck, P.: *Constraint Satisfaction in Logic Programming*, MIT Press 1989

[VHSaDe93] Van Hentenryck, P.; Saraswat, V. A.; Deville, Y.: *Design, Implementation, and Evaluation of the Constraint Language* `cc(FD)`, in Podelski, A. (ed.): *Constraint Programming: Basic and Trends*, pp. 293–316, LNCS 910, Springer Verlag 1995)

[Wa88] Warren, D. H. D.: *The Andorra Model*, Lecture Notes, Gigalips Project Workshop, University of Manchester, 1988

[We97] Wetzel, G.: *Abductive and Constraint Logic Programming*, Ph.D. thesis, Imperial College 1997

[WeKoTo95] Wetzel, G.; Kowalski, R. A.; Toni, F.: *A Theorem-Proving Approach to CLP*, in: Krall, A.; Geske, U. (eds.): *Workshop Logische Programmierung*, pp. 63–72, 1995

[WeKoTo96] Wetzel, G.; Kowalski, R. A.; Toni, F.: *PROCALOG — Programming with Constraints and Abducibles in Logic*, in: Maher, M. (ed.): *Proc. of the 1996 Joint Int. Conf. and Symp. on Logic Programming*, p. 535

[Wi92] Winston, P. H.: *Artificial Intelligence*, Addison-Wesley 1992