Abductive Logic Programming with CIFF

Ulle Endriss[1], Paolo Mancarella[2], Fariba Sadri[1], Giacomo Terreni[2], and Francesca Toni[1,2]

[1] Department of Computing, Imperial College London
Email: {ue,fs,ft}@doc.ic.ac.uk
[2] Dipartimento di Informatica, Università di Pisa
Email: {paolo,terreni,toni}@di.unipi.it

1 Introduction

Abduction has found broad application as a powerful tool for hypothetical reasoning with incomplete knowledge, which can be handled by labelling some pieces of information as *abducibles*, i.e. as possible hypotheses which can be assumed to hold, provided that they are consistent with the given knowledge base. Abductive Logic Programming (ALP) [4] combines abduction with logic programming enriched with *integrity constraints* to further restrict the range of possible hypotheses.

We introduce a new proof procedure for abductive logic programming which we call CIFF. Our procedure extends the IFF procedure of Fung and Kowalski [3] by integrating abductive reasoning with constraint solving. Another feature of our approach is that we do not attempt to provide a static characterisation of the class of allowed inputs on which the procedure can operate correctly, but rather check allowedness dynamically during a derivation. This allows us to cover a larger class of inputs.

2 Abductive Logic Programming

An *abductive logic program* is a pair $\langle Th, IC \rangle$ consisting of a *theory* Th and a finite set of *integrity constraints* IC. A theory is a set of so-called iff-definitions:

$$p(X_1, \ldots, X_k) \leftrightarrow D_1 \vee \cdots \vee D_n$$

The predicate symbol p must not be a *special* predicate (constraint predicates, $=$, $\top$ and $\bot$) and there can be at most one iff-definition for every predicate symbol. Each of the disjuncts D_i is a conjunction of literals. Negative literals are written as implications (e.g. $q(X,Y) \rightarrow \bot$). The variables $X_1, \ldots, X_k$ are implicitly universally quantified with the scope being the entire definition. Any other variable is implicitly existentially quantified, with the scope being the disjunct in which it occurs. A theory may be regarded as the (selective) *completion* of a normal logic program [2]. Any predicate that is neither defined nor special is called an *abducible*.

The integrity constraints in the set IC are implications of the following form:

$$L_1 \wedge \cdots \wedge L_m \rightarrow A_1 \vee \cdots \vee A_n$$

Each of the L_i must be a literal (with negative literals again being written in implication form); each of the A_i must be an atom. Any variables are understood to be implicitly universally quantified with the scope being the entire implication.

A *query* Q is a conjunction of literals. Any variables occurring in Q are implicitly existentially quantified. These variables are also called the *free* variables.

A theory provides definitions for certain predicates and integrity constraints restrict the range of possible interpretations. A query may be regarded as an *observation* against the background of the world knowledge encoded in a given abductive logic program. An *answer* to such a query would then provide an *explanation* for this observation: it would specify which instances of the abducible predicates have to be assumed to hold for the observation to hold as well. In addition, such an explanation should also validate the integrity constraints. This intuitive account of the semantics of ALP may be formalised as follows. A *correct answer* to a query Q with respect to an abductive logic program $\langle Th, IC \rangle$ is a pair $\langle \Delta, \sigma \rangle$, where Δ is a finite set of ground abducible atoms and σ is a substitution for the free variables in Q, such that:

$$Th \cup Comp(\Delta) \models IC \wedge Q\sigma$$

A suitable interpretation of the entailment operator $\models$ would the usual consequence relation of first-oder logic with the restriction that constraint predicates have to be interpreted according to the semantics of the chosen constraint system and equalities evaluate to *true* whenever their two arguments are unifiable. $Comp(\Delta)$ stands for the *completion* of the set of abducibles in Δ, i.e. any ground abducible atom not occurring in Δ is assumed to be *false*. If we have $Th \cup IC \models \neg Q$ (i.e. if Q is false for all instantiations of the free variables), then we say that there exists no correct answer to the query Q given the abductive logic program $\langle Th, IC \rangle$.

3 The CIFF Procedure

We are now going to outline our CIFF proof procedure, which integrates ALP with constraint solving. We assume the availability of a sound and complete constraint solver for the constraint predicates used. In principle, the exact specification of the constraint language is independent from the definition of the CIFF procedure, because we are going to use the constraint solver as a *black box* component. However, the constraint language has to include a relation symbol for equality and it must be closed under complements.

CIFF extends the IFF procedure of Fung and Kowalski [3], which requires inputs to their proof procedure to meet a number of *allowedness conditions* (essentially avoiding certain problematic patterns of quantification) to be able to guarantee the correct operation of the procedure. Unfortunately, it is difficult to formulate appropriate allowedness conditions that guarantee a correct execution of the proof procedure without imposing too many unnecessary restrictions. This is a well-known problem, which is further aggravated for languages that include constraint predicates. Our proposal is to tackle the issue of allowedness *dynamically*, i.e. at runtime, rather than adopting a static and overly strict set of conditions. To this end, the CIFF procedure includes a *dynamic allowedness rule* which is triggered whenever the procedure encounters a particular formula it cannot manipulate correctly due to a problematic quantification pattern.

The input to the CIFF procedure consists of a theory *Th*, a set of integrity constraints *IC*, and a query Q. There are three possible outputs: (1) the procedure succeeds and indicates an answer to the query Q; (2) the procedure fails, thereby indicating that there is no answer; and (3) the procedure reports that computing an answer is not possible, because a critical part of the input is not allowed.

The CIFF procedure manipulates, essentially, a set of formulas that are either atoms or implications. The theory *Th* is kept in the background and is only used to *unfold* defined predicates as they are being encountered. In addition to atoms and implications the aforementioned set of formulas may contain disjunctions of atoms and implications to which the *splitting* rule may be applied, i.e. which give rise to different branches in the proof search tree. The sets of formulas manipulated by the procedure are called *nodes*. A node is a set (representing a conjunction) of formulas (atoms, implications, or disjunctions thereof) which are called *goals*. A proof is initialised with the node containing the integrity constraints *IC* and the literals of the query Q. The proof procedure then repeatedly manipulates the current node of goals by rewriting goals in the node, adding new goals to it, or deleting superfluous goals from it. We only sketch the most important proof rules:

- *Unfolding:* Replace any atomic goal $p(\vec{t})$, for which there is a definition $p(\vec{X}) \leftrightarrow L_1 \vee \cdots \vee L_n$ in *Th*, by $(L_1 \vee \cdots \vee L_n)[\vec{X}/\vec{t}]$. There is a similar rule for defined predicates inside implications.

- *Splitting:* Rewrite any node including a disjunctive goal as a disjunction of nodes and apply CIFF to each one of them.

- *Propagation:* Given goals of the form $p(\vec{t}) \wedge A \rightarrow B$ and $p(\vec{s})$, add the goal $(\vec{t} = \vec{s}) \wedge A \rightarrow B$.

- *Equality rewriting:* Simplify equalities and present them in a normal form (e.g. rewrite $t = X$ as $X = t$).

- *Substitution:* Given a goal of the form $X = t$, apply the substitution $[X/t]$ to the entire node. There is a similar rule for equalities inside an implication.

- *Case analysis for constraints:* Replace any goal of the form $Con \wedge A \rightarrow B$, where Con is a constraint not containing any universally quantified variables, by $[Con \wedge (A \rightarrow B)] \vee \overline{Con}$. There is a similar case analysis rule for equalities.

- *Constraint solving:* Replace any node containing an unsatisfiable set of constraints (as atoms) by $\perp$.

- *Dynamic allowedness rule:* Label nodes with problematic quantification patterns as *undefined*.

In addition, there are a number of logical simplification rules as well as rules that reflect the interplay between constraint predicates and the usual equality predicate.

A node containing $\perp$ is called a *failure node*. If all branches in a derivation terminate with failure nodes, then the derivation is said to fail (the intuition being that there exists no answer to the query in question). A node to which no more proof rules can be applied is called a *final node*. A final node that is not a failure node and which has not been labelled as *undefined* is called a *success node*.

An *extracted answer* for a final success node N is a triple $\langle \Delta, \Phi, \Gamma \rangle$, where Δ is the set of abducible atoms, Φ the set of equalities and disequalities, and Γ the set of constraint atoms in N. This in itself is not yet a *correct* answer (according to the semantics of ALP), but it is possible to show that it does *induce* such a correct answer. The basic idea is to first define a substitution σ that is consistent with both the (dis)equalities in Φ and the constraints in Γ, and then to ground the set of abducibles Δ by applying σ to it. The resulting set of ground abducible atoms together with the substitution σ then constitutes a correct answer to the query (so an extracted answer will typically give rise to a whole range of correct answers).

We have shown that CIFF is *sound:* Whenever the procedure terminates successfully then it is possible to extract an answer from the proof that is correct according to the semantics of ALP with constraints (soundness of success); and whenever it fails then there exists no such answer (soundness of failure).

4 Implementation

We have implemented the CIFF procedure in Prolog. The constraint solver used is the built-in finite domain solver of Sicstus Prolog [1], but the modularity of the system would also support the integration of a different solver. The system is available at the following address:

`http://www.doc.ic.ac.uk/~ue/ciff/`

Our implementation has been tested successfully on a number of different examples. Most of these examples are taken from applications of CIFF within the SOCS project,which investigates the application of computational logic-based techniques to multiagent systems (e.g. the implementation of an agent's planning capability).

While these are encouraging results, it should be noted that this is only a first prototype and more research into proof strategies for CIFF as well as a fine-tuning of the implementation are required to achieve satisfactory runtimes for larger examples.

Acknowledgements. This research has been supported by the European Commission as part of the SOCS project (IST-2001-32530).

References

[1] M. Carlsson, G. Ottosson, and B. Carlson. An open-ended finite domain constraint solver. In *Proc. Programming Languages: Implementations, Logics and Programs*, 1997.

[2] K. L. Clark. Negation as failure. In *Logic and Data Bases*. Plenum Press, 1978.

[3] T. H. Fung and R. A. Kowalski. The IFF proof procedure for abductive logic programming. *Journal of Logic Programming*, 33(2):151–165, 1997.

[4] A. C. Kakas, R. A. Kowalski, and F. Toni. The role of abduction in logic programming. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, pages 235–324. Oxford University Press, 1998.