ABDUCTIVE LOGIC PROGRAMMING

Abductive Logic Programming is a high level knowledge representation framework that allows us to solve problems declaratively based on abductive reasoning. It extends normal Logic Programming by allowing some predicates to be incompletely defined, declared as abducible predicates. Problem solving is effected by deriving hypotheses on these abducible predicates (abductive hypotheses) as solutions of problems to be solved. These problems can be either observations that need to be explained (as in classical abduction) or goals to be achieved (as in normal logic programming). It can be used to solve problems in Diagnosis, Planning, Natural Language and Machine Learning. It has also been used to interpret Negation as Failure as a form of abductive reasoning.

Syntax

Abductive Logic Programs have three components, $< P, A, IC >$, where,
- P is a logic program of exactly the same form as in Logic Programming
- A is a set of predicate names, called the abducible predicates
- IC is a set of first order classical formulae.

Normally, the logic program P does not contain any clauses whose head (or conclusion) refers to an abducible predicate. (This restriction can be made without loss of generality.) Also in practice, many times the integrity constraints in IC are often restricted to the form of denials, i.e. clauses of the form:
false:- A1,...,An, not B1, ..., not Bm.

Such a constraint means that it is not possible for all A1,...,An to be true and at the same time all of B1,...,Bm to be false.

Informal Meaning and Problem Solving

The clauses in P define a set of non-abducible predicates and through this they provide a description (or model) of the problem domain. The integrity constraints in IC specify general properties of the problem domain that need to be respected in any solution of our problem.

A problem, G, which expresses either an observation that needs to be explained or a goal that is desired, is represented by a conjunction of positive and negative (NAF) literals. Such problems are solved by computing **abductive explanations** of G.

An abductive explanation of a problem G is a set of positive (and sometimes also negative) ground instances of the abducible predicates, such that, when these are added to the logic program P, the problem G and the integrity constraints I both hold. Thus abductive explanations extend the logic program P by the addition of full or partial definitions of the abducible predicates. In this way, the incompletely specified abducible predicates constitute the solution carriers of the problems to be solved and abductive explanations form solutions

of the problem according to the description of our problem domain in P and IC. The extension or completion of the problem description given by the abductive explanations provides new information, hitherto not contained in our description, that we are seeking as the solution to our problem. Quality criteria to prefer one solution over another, often expressed via integrity constraints, can be applied to select specific abductive explanations of the problem G.

Computation in ALP combines the backwards reasoning of normal logic programming (to reduce problems to sub-problems) with a kind of integrity checking to show that the abductive explanations satisfy the integrity constraints.

The following two examples, written in simple stuctured english rather than in the strict syntax of ALP, illustrate the notion of abductive explanation in ALP and its relation to problem solving.

Example 1
Consider the abductive logic program whose domain description is given by:

Grass is wet **if** it rained.
Grass is wet **if** the sprinkler was on.
The sun was shining.

together with the two abducible predicates, "it rained" and "the sprinkler was on" and the integrity constraint:

false **if** it rained **and** the sun was shining.

The observation that the grass is wet has two potential explanations, "it rained" and "the sprinkler was on", which entail the observation. However, only the second potential explanation, "the sprinkler was on", satisfies the integrity constraint.

Example 2

Consider the abductive logic program consisting of the three (simplified) clauses and two facts about Mary:

X is citizen **if** X is born in USA.
X is citizen **if** X is born outside USA **and** X is resident of USA
and X is naturalised.
X is citizen **if** X is born outside USA **and** Y is mother of X is Y is citizen
and X is registered.
Mary is mother of John.
Mary is citizen.

together with the four abducible predicates, "is born in USA", "is born outside USA, "is resident of USA" and "is naturalised" and the integrity constraint:

false **if** John is resident of USA.

The goal "John is citizen" has two abductive solutions, one of which is "John is born in USA", the other of which is "John is born outside USA" and "John become registered". The potential solution of becoming a citizen by residence and naturalisation fails because it violates the integrity constraint.

A more complex example that is also written in the more formal syntax of ALP is the following.

Example 3

The abductive logic program below describes a simple model of the lactose metabolism of the bacterium E. Coli. The program P describes the fact that E. coli can feed on the sugar lactose if it makes two enzymes permease and galactosidase. Like all enzymes (E), these are made if they are coded by a gene (G) that is expressed. These enzymes are coded by two genes (lac(y) and lac(z)) in cluster of genes (lac(X)) called an operon that is expressed when the amounts (amt) of glucose are low and lactose are high or when they are both at medium level. The abducibles, A, declare all ground instances of the predicates "amount" as assumable. This reflects the fact that in our model we cannot know what are the amounts at any time of the various substances. This is incomplete information that we want to find out in each problem case that we are examining. The integrity constraints state that the amount of a substance (S) can only take one value.

—— Domain Knowledge (P) ——

feed(lactose):-make(permease),make(galactosidase).
make(Enzyme):-code(Gene,Enzyme),express(Gene).
express(lac(X)):-amount(glucose,low),amount(lactose,hi).
express(lac(X)):-amount(glucose,medium),amount(lactose,medium).
code(lac(y),permease).
code(lac(z),galactosidase).

temperature(low):-amount(glucose,low).

— Integrity Constraints (IC) —

false :- amount(S,V1), amount(S,V2), V1 $\neq$ V2.

——— Abducibles (A) ———

abducible_predicate(amount).

Our problem goal is $G = feed(lactose)$. This can arise either as an observation in some experiment that we have carried out and we want to understand how it came about or as a state of affairs that we want to achieve and we want

to find out a plan of how we can do so. This goal has two abductive explanations:

Δ_1 = {amount(lactose,hi), amount(glucose,low)}

Δ_2 = {amount(lactose,medium), amount(glucose,medium)}

The decision which is one of the two to adopt could depend on addition information that is available, e.g. we may know that when the level of glucose is low then the organism exhibits a certain behaviour - in our model such additional information is that the temperature of the organism is low - and by observing the truth or falsity of this we can choose the first or second explanation respectively.

Once we decide on an explanation then this becomes part of our theory and we can draw from this extended theory new conclusions. The explanation and more generally these new conclusions form the solution of our problem.

Formal Semantics

The formal semantics of the central notion of an abductive explanation in ALP, can be defined in the following way:

Given an abductive logic program (P, A, IC), an abductive explanation for a problem G is a set Δ of ground atoms on abducible predicates such that:

- $P \cup \Delta \models G$
- $P \cup \Delta \models IC$
- $P \cup \Delta$ is consistent.

This definition is generic in the underlying semantics of Logic Programming that we adopt in ALP. Each particular choice of semantics defines its own entailment relation $\models$, its own notion of consistent logic programs and hence its own notion of what an abductive solution is. In practice, the three main semantics of logic programming — completion, stable and well-founded semantics — have been used to define different ALP frameworks.

The integrity constraints IC define *how* they constrain the abductive solutions. There are different views on this. Early work on abduction in Theorist in the context of classical logic was based on the *consistency view* on constraints. In this view, any extension of the given theory P with an abductive solution Δ is required to be consistent with the integrity constraints IC: $P \cup IC \cup \Delta$ is consistent. The above definition formalizes the *entailment view*: the abductive solution Δ together with P should entail the constraints. This view is the one taken in most versions of ALP and is stronger than the consistency view in the sense that a solution according to the entailment view is a solution according to the consistency view but not vice versa.

The difference between the two views can be subtle but in practice the different views usually coincide. E.g. it frequently happens that $P \cup \Delta$ has a unique model, in which case the two views are equivalent. In practice, many

ALP systems use the entailment view as this can be easily implemented without the need for any extra specialized procedures for the satisfaction of the integrity constraints since this semantics treats the constraints in the same way as the goal.

Comparison with other Frameworks

There exist strong links between some ALP frameworks and other extensions of Logic Programming. In particular, ALP has close connections with Answer Set Programming. An abductive logic program can be translated into an equivalent answer set program under the stable model semantics. Consequently, systems for computing stable models such as SMODELS can be used to compute abduction in ALP.

ALP is also closely related to Constraint Logic Programming. On the one hand, the integration of constraint solving and abductive logic programming enhances the practical utility of ALP through a more efficient computation of abduction. On the other hand, the integration of ALP and CLP can be seen as a high-level constraint programming environment that allows more modular and flexible representations of the problem domain.

There is also a strong link between ALP and Argumentation in Logic Programming. This relates both to the interpretation of Negation as Failure and Integrity Constraints.

Implementation and Systems

Most of the implementations of ALP extend the SLD resolution based computational model of Logic Programming. ALP can also be implemented by means on its link with Answer Set Programming (ASP), where the ASP systems can be employed. Five systems of the former approach are ACLP, A-system, CIFF, ABDUAL and ProLogICA.

See also

- Abductive Reasoning
- Answer Set Programming
- Inductive Logic programming
- Negation as Failure
- Argumentation

References

A.C. Kakas and P. Mancarella, "Generalised Stable Models: A Semantics for Abduction" in Proceedings of the ninth European Conference on Artificial

Intelligence, ECAI-90, Stockholm, Sweden, (ed. L.C. Aiello) Pitman Publishing, pp. 385-391, (1990).

L.Console, D.T. Dupre and P. Torasso", "On the Relationship between Abduction and Deduction", Journal of Logic and Computation, Vol 1, no 5, pp. 661-690, 1991.

A.C. Kakas, R.A. Kowalski and F. Toni, "Abductive Logic Programming", Journal of Logic and Computation, Vol. 2 no 6, pp. 719-770, (1993).

Marc Denecker and Danny De Schreye, "SLDNFA: An Abductive Procedure for Abductive Logic Programs", Journal of Logic Programming, Vol 34, no 2, pp. 111-167, 1998.

M. Denecker and A.C. Kakas, Abductive Logic Programming, Special issue of Journal of Logic Programming, JLP, Vol. 44(1-3), Elsevier Science, 2000.

M. Denecker and A.C. Kakas, Abduction in Logic Programming, in Computational Logic: Logic Programming and Beyond, LNAI Vol, 2407, pp. 402-437, Springer Verlag, 2002.

Useful Links

- ACLP: http://www.cs.ucy.ac.cy/aclp/
- A-system: http://www.cs.kuleuven.be/ dtai/krr/Asystem/Asystem/asystem.html
- ProLogICA: http://www.doc.ic.ac.uk/ or/proLogICA/
- CIFF: http://www.doc.ic.ac.uk/ ue/ciff/
- ACL: http://www-lia.deis.unibo.it/Software/ACL/