GORGIAS-C: Extending Argumentation with Constraint Solving

Victor NOËL[1] and Antonis KAKAS[2]

[1] Université de Toulouse, Institut de Recherche en Informatique de Toulouse,
118, route de Narbonne, 31 062 Toulouse Cedex, France
victor.noel@irit.fr
[2] Department of Computer Science, University of Cyprus,
75 Kallipoleos St. Nicosia, Cyprus
antonis@cs.ucy.ac.cy

1 General Information

GORGIAS-C is a system implementing a logic programming framework of argumentation that integrates together preference reasoning and constraint solving.

The framework of argumentation with preference reasoning [1, 2] was first implemented in the GORGIAS system[3] which has mainly been used in the multi-agent domain [2–4], medical informatics [5] and network security [6].

We have extended this framework and the GORGIAS system to integrate constraint solving at the representation and argumentation level of the framework. GORGIAS-C is the first framework and system that we are aware of that links argumentation with domain constraints.

In GORGIAS-C problems are expressed in the combined language of Logic Programming with Priorities and Constraint Logic Programming (CLP) [7] thus allowing us to address problems with large computational domains while at the same time exploiting the high expressive power of the framework to capture complex requirements and preferences.

Technically, GORGIAS-C is implemented as a modular meta-interpreter on top of the object-oriented Logtalk preprocessor[4] [8] with SWI-Prolog[5] and its CLP(FD) library [9] but can be used with other Prolog interpreters with CLP supported by Logtalk.

Furthermore, thanks to this modularity, GORGIAS-C is easy to extend, allowing researchers of the domain to prototype new extensions of the argumentation framework.

2 Description of the System

GORGIAS-C computes answers to queries asked on a logic program with priorities on rules and domain constraints on variables. It also allows, as GORGIAS, the use of abductive predicates as a special kind of argumentation. The general format of such a program is as follows.

[3] http://www2.cs.ucy.ac.cy/∼nkd/gorgias/
[4] http://logtalk.org/
[5] http://www.swi-prolog.org/

Rules. Rules are labeled logic programming rules of the form:
`rule(Label, Head, Body).`
where, `Label` is a term for referencing the rule and `Head` is a positive or negative logic program literal (with negation applied using the operator `neg/1` on positive atoms). `Body` is a list containing positive or negative logic program literals, or domain constraints over variables in the rule.

A special case of these rules, where the `Head` is an atom on the special predicate `prefer/2`, is used to define the priorities between the rules of the program. Both arguments of `prefer(label1,label2)` are labels of rules, denoting that the the first is preferred over the second. For the implementation of GORGIAS-C, these two rules need to have contradictory conclusions.

Abducibles. Abducible predicates are declared as such with the following construct:
`abducible(Predicate).`
where `Predicate` is a user-defined predicate name.

Abducible predicates can be partially defined in the program with rules whose head refers to them. Integrity constraints (ICs) [10] on an abducible predicate can be specified with a rule whose head is the negation of an abducible and whose body is the condition under which the abducible cannot be assumed.

Domain Constraints. Domain constraints usable within GORGIAS-C are the ones available in the underlying constraint solver. In the SWI-Prolog's CLP(FD) library, they are finite domain constraints like `#>=`, `#=`, `#\=` or `#\`, `#\/`, `#==>`. In other Prolog interpreters with CLP, other types of constraints can be used (e.g. over reals). As mentioned above these are used in the body of the logic program rules.

Easing Development. GORGIAS-C proposes some facilities to ease the development of programs:

- `::/1` and `::/2` to call predicates directly using the underlying Logtalk interpreter preventing argumentation on them: they are explained in the tutorial on the website
- `complement/2` to specify complementarity between logic program literals, extending the notion of negation

For example, `complement(give(Object, Time), keep(Object, Time))` means that `give/2` and `keep/2` are contradictory and thus we cannot build arguments with both and arguments for one can potentially attack arguments for the other.

2.1 Computing Answers

Given a program as defined above and a query $\exists Q(X)$, GORGIAS-C computes an answer of the form $\langle \Delta, A, C \rangle$ where:

1. Δ is a set of program rules used to conclude an argument for Q.
2. A is a (possibly empty) set of abducible hypotheses (literals on abducible predicates) needed for Δ to conclude Q.
3. C is a set of domain constraints on X and variables in $\Delta \cup A$.

An answer $\langle \Delta, A, C \rangle$ fulfils the following conditions:

1. C is satisfiable.
2. For every valuation σ of C:
   (a) Δ_σ is admissible: consistent and attacking all its attacking arguments
   (b) $\Delta_\sigma \cup A_\sigma \models Q_\sigma$.

The precise definition of attacks is given in [11, 2]. Informally, a set of rules (an argument) attacks another argument if they have a complementary conclusion and the attacking argument renders its rules of higher priority than the rules of the attacked argument or at least not of lower priority.

The rules in an answer form an argument for the query, giving not only the rules that derive it but also the rule priorities that make the answer preferred over contrary ones. The set A contains information (that is missing from the program due to the incompleteness of the abducible predicates) needed for the arguments in Δ to be enabled. The constraints C impose necessary restrictions on the cases (variables) of the arguments needed in order for the argument to admissibly conclude the query, as described above.

The set of domain constraints C are computed by integrating within the argumentation reasoning a new (symmetric) attacking relation where any two domain constraints c_1 and c_2, which together are non-solvable, attack each other. This works together with the underlying domain constraint solving to ensure that the computed constraints C are solvable.

The classical Tweety example (extended to allow for incomplete knowledge on the relation `bird/1`) can be represented in GORGIAS-C (and GORGIAS since this example does not involve domain constraints) as follows:

```
rule(r1(X), fly(X), [bird(X)]).
rule(r2(X), neg(fly(X)), [::penguin(X)]).     abducible(bird(_)).
rule(r3(X), bird(X), [::penguin(X)]).         penguin(tweety).
rule(pr1(X), prefer(r2(X), r1(X)), []).
```

GORGIAS-C answers the query `fly(leon)` and `neg(fly(tweety))` respectively with `<{r1(leon)},{bird(leon)},{}>` and `<{r2(tweety), f1},{},{}>`. An answer for `fly(tweety)` cannot be built as the argument given by the second rule, `r2`, for the contrary conclusion, is stronger thanks to `pr1` and thus cannot be counter-attacked.

A second example where we use the new constraint handling facility of GORGIAS-C is the following:

```
rule(r1(P), buy(P), [::offer(OP), P #> OP]).          offer(27).
rule(r2(P), neg(buy(P)), [::offer(OP), P #>= OP + 5]).
rule(pr1(P), prefer(r2(P),r1(P)), []).             top-price(30).
```

The answer to the query `buy(P)` is `<{r1(P)},{},{P in 28..31}>`, because to prevent `r2` to be preferred over `r1`, `P` must be greater than the last offer of 27 and less than the last offer plus 5: 31. The resulting domain of `28..31` for the variable `P` was computed by applying the constraint `P #> 27` together with the application of negating the constraint `P #>= 27 + 5`.

If we then add the rules `rule(r3(P), neg(buy(P)), []).` and
`rule(pr2(P), prefer(r3(P),r1(P)), [::top-price(Top), P #> Top]).`

The answer to the query `buy(P)` is `<{r1(P)},{},{P in 28..30}>`, the resulting domain was computed by applying the constraint `P #> 27` together with the application of negating each of the constraints `P #> 30` and `P #>= 27 + 5`.

Hence, GORGIAS-C uses, like GORGIAS, rules and abduction to conclude goals, and in addition finds domains of finite domain variables where these are applicable. Furthermore, to ensure that there are no stronger opposing arguments, GORGIAS-C will find variables domains that make potential opposing arguments inapplicable or prevent them from being stronger.

2.2 Implementation

GORGIAS-C is implemented as a modular meta-interpreter for its logic programs on top of Logtalk using SWI-Prolog and its CLP(FD) library and has successfully been used with ECLIPSe[6] with CLP over reals.

Using the features of Logtalk for object and component-based programming [12], GORGIAS-C is a modular meta-interpreter, consisting of:

- A Logtalk's category that implements:
  - The core resolution algorithm consisting of interleaving phases of attacks and defences.
  - The reduction of the constraint domains by using the underlying constraint solver at key points of the computation.
- Categories for the 'modules' implementing different aspects of this framework, called by the core algorithm when needed:
  - 'LPwNF' for attacks based on priority rules.
  - 'Abducible' for attacks and resolution based on abduction.
  - 'CLP' for attacks based on domain constraints.

Core. The core algorithm alternates two phases of attack and defence on a set of rules. For every attack possible on the set, a defence against it (a counter-attack) is found and added to the set. This is the recursively applied to the original extended set until it is admissible, i.e. it defends against every attack and does not attack itself.

More specifically, when querying the system this will first resolve the query, resulting on an initial set Δ_0 that concludes the query. Then it will extend Δ_0 by finding all attacks on it, and for each of them, find a counter-attack to add to Δ_0. If it can not counter-attack an attack, then the computation fails and backtracks to the last choice-point.

Domain Constraints. When constraints are present in an attack on Δ_0, to defend it GORGIAS-C will construct counter-attacks by negating the constraints in the attacks to make them inapplicable.

To be able to negate them, these constraints should be reifiable (which means that we can reflect their truth value into boolean values represented by 0 and 1).

We can also note that, because GORGIAS-C only relies on the fact that constraints must be reifiable:

[6] htt://www.eclipse-clp.org

– It can use all reifiable constraints added to the constraint solver by users.
– It can use any constraint solver (other than finite domain ones) that uses reifiable constraints (such as the IC library of ECLIPSe).

3 Applying the System

Argumentative programs are embedded in Logtalk's objects importing the core category and other needed 'modules'. Depending on what is enabled, the Logtalk preprocessor will generate the smallest Prolog code, with the possibility to enable a debugging trace of the argumentation process.

Querying Programs. Once the user's program is loaded we can call the predicate `prove(Query,Delta)` to compute an answer to the query `Query`, by unifying `Delta` with a set of rules labels and assumed abducibles, and an associated constraint store maintained by the underlying constraint solver. Variables in this answer can then be labeled with the different algorithms provided by the constraint solver.

Methodology. Writing an argumentative program is a task requiring different skills. Apart from the (constraint) logic programming background, the following will help.

Typically, programs written for GORGIAS-C (and GORGIAS) can be separated into 2 or more layers. The first one would be composed of rules for and against some conclusion of interest to the user, using negation or 'complement' predicates in heads. The second one would be composed of preference rules that the user wants to apply when the rules of the first layer come into conflict. And so on, if the second layer contains conflicting preferences.

Abduction is used to model facts for which there is incomplete knowledge, but, above all, can be used to model the solution of a problem: rules are used to construct the solution (as assumed abducibles) and ICs enforce properties on the solution.

4 Evaluating the System: Performance and Expressiveness

In term of performance, compared to Prolog, GORGIAS-C adds two overheads: the monotonic resolution (meta-interpretation) and the argumentative process of constructing attacks and counter-attacks while maintaining the constraint store. But at the same time, GORGIAS-C provides a declarative expressiveness available through abduction and preferences combined with CLP.

Classical constraint problems (e.g. N-Queens, Graph Colouring) can be resolved using GORGIAS-C, not exploiting its argumentative aspects but only the underlying constraint solver. Thus they will only suffer from the monotonic resolution overhead which is negligible according to benchmarks made with the N-Queens example. Compared to Prolog and Logtalk, times for one solution are unchanged with 10, 15, 20 or 25 queens

Abduction was used in planning and scheduling problems: advantages of the approach is well covered in [13]. Applications presented in this paper were ported to

GORGIAS-C and presented equivalent performances for job-shop 20x5, 10x10 and 20x10 when using ECLIPSe.

Preference reasoning has been used with GORGIAS in autonomous agents to specify interaction rules or protocols, and policies for negotiation processes, but also in network security to specify firewall policies. This kind of applications can be extended to handle large computational domains with the help of domain constraints. Pricing and firewall policies are currently being worked on: constraints should simplify the programs and improve performances in some problems by moving complex computations from the Prolog interpreter to the constraint solver.

After adding domain constraints to preferences-enabled problems, we will investigate the addition of preferences to constraint solving problems and study the benefits and performance that GORGIAS-C will have. For example, planning and scheduling problems could benefit from priorities to guide the search, by reducing the domains of the solutions based on criteria such as action costs or job importance.

5 Obtaining the System.

GORGIAS-C can be found on its website at http://dev.crazydwarves.org/trac/Gorgias, together with sample examples of its use, applications and benchmarks.

References

1. Kakas, A.C., Mancarella, P., Dung, P.M.: The acceptability semantics for logic programs. In: ICLP. (1994) 504–519
2. Kakas, A., Moraitis, P.: Argumentation Based Decision Making for Autonomous Agents. In: AAMAS '03. (2003) 883–890
3. Kakas, A., Maudet, N., Moraitis, P.: Modular Representation of Agent Interaction Rules through Argumentation. Autonomous Agents and Multi-Agent Systems **11**(2) (2005) 189–206
4. Kakas, A., Mancarella, P., Sadri, F., Stathis, K., Toni, F.: Computational Logic Foundations of KGP Agents. Journal of Artificial Intelligence Research **33** (2008) 285–348
5. Letia, I.A., Acalovschi, M.: Achieving Competence by Argumentation on Rules for Roles. In: ESAW '04. (2004) 45–59
6. Bandara, A.K., Kakas, A.C., Lupu, E.C., Russo, A.: Using Argumentation Logic for Firewall Policy Specification and Analysis. In: DSOM. (2006) 185–196
7. Jaffar, J., Lassez, J.L.: Constraint Logic Programming. In: POPL '87. (1987) 111–119
8. Moura, P.: Logtalk - Design of an Object-Oriented Logic Programming Language. PhD thesis, University of Beira Interior, Portugal (September 2003)
9. Triska, M., Neumerkel, U., Wielemaker, J.: A Generalised Finite Domain Constraint Solver for SWI-Prolog. In: WLP 2008. (2008) 89–91
10. Denecker, M., Kakas, A.: Abduction in Logic Programming. In: Computational Logic: Logic Programming and Beyond. (2002) 99–134
11. Noël, V., Kakas, A.: Extending argumentation with constraint solving. Technical report, University of Cyprus (2009)
12. Moura, P.: Logtalk 2.6 Documentation. Technical Report DMI 2000/1, University of Beira Interior, Portugal (July 2000)
13. Kakas, A., Mourlas, C.: ACLP: Flexible Solutions to Complex Problems. In: LPNMR '97. (1997) 388–399