# Constraint Answer Set Programming without Grounding and its Applications[⋆]

Joaquin Arias[1,2], Manuel Carro[1,2], Zhuo Chen[3], and Gopal Gupta[3]

[1] IMDEA Software Institute, {joaquin.arias,manuel.carro}@imdea.org
[2] Universidad Politécnica de Madrid, manuel.carro@upm.es
[3] University of Texas at Dallas, {zhuo.chen,gupta}@utdallas.edu

**Abstract.** Extending Datalog/ASP with constraints (CASP) enhances its expressiveness and performance but it is not straightforward as the grounding phase removes variables and the links among them. We incorporate constraints into s(ASP), a goal-directed, top-down execution model which implements predicate answer set programming without grounding. The resulting model, s(CASP), can constrain variables that, as in CLP, are kept during the execution and in the answer sets. We show the enhanced expressiveness of s(CASP) w.r.t. other CASP systems, through a non-trivial example of modeling the event calculus.

## 1 Introduction

Answer Set Programming (ASP) has emerged as a successful paradigm for developing intelligent applications. It uses the stable model semantics [4] and has attracted much attention due to its expressiveness, ability to incorporate non-monotonicity, represent knowledge, and model combinatorial problems. ASP can be seen as Datalog extended with *negation-as-failure*.

s(ASP) [8] is a goal-directed, top-down, SLD resolution-like procedure which evaluates programs under the ASP semantics without a grounding phase either before or during execution. s(ASP) supports predicates and can thus retain logical variables both during execution and in the answer sets.

Constraints have been used both to enhance expressiveness and to increase performance in logic programming. Therefore, it is natural to incorporate constraints in ASP systems. The s(CASP) system [2] extends s(ASP) by integrating it with generic constraint solvers. The s(CASP) system makes it possible to express constraints on variables. It extends s(ASP)'s execution model in the same way that CLP extends Prolog's. Thus, s(CASP) inherits and generalizes the execution model of s(ASP) and is parametrized w.r.t. the constraint solver. Due to its basis in s(ASP), the s(CASP) system avoids grounding the program and the concomitant combinatorial explosion. s(CASP) also handles answer set programs with arbitrary data structures and/or reals, rationals, etc. The s(CASP) system successfully executes programs in finite time that loop in other

[⋆] Work partially supported by EIT Digital (https://eitdigital.eu), MINECO project TIN2015-67522-C3-1-R (TRACES), Comunidad de Madrid as part of the program S2018/TCS-4339 BLOQUES-CM co-funded by EIE Funds of the European Union, and NSF Grant IIS 1718945.

| | s(CASP) | s(ASP) |
|---|---|---|
| hanoi(8,T) | **1,528** | 13,297 |
| queens(4,Q) | **1,930** | 20,141 |
| One Hamiltonian cycle | **493** | 3,499 |
| Two Hamiltonian cycle | **3,605** | 18,026 |

Table 1: Speed comparison (time in ms).

top-down systems, as well as programs that require constraints over dense and/or unbound domains. Thus, s(CASP) is able to solve problems that cannot be easily solved by other Datalog/ASP systems [3,6,9].

Summarizing, we show how Datalog programs extended with negation following the stable-model semantics can be executed in a query-driven, goal-directed manner in the presence of constraints, including constraints over dense domains. The s(CASP) system is the culmination of this work. To illustrate the power of s(CASP), we show how event calculus (EC) axioms and narratives can be modeled in s(CASP) and how its expressiveness makes it possible to perform deductive and abductive reasoning over continuous domains.

## 2 s(CASP): Design and Implementation

The s(CASP) system (`https://gitlab.software.imdea.org/joaquin.arias/sCASP`), implemented in Ciao Prolog [5], is an evolution of the s(ASP) system. The main contributions of s(CASP) are: (i) a generic interface to connect the disequality constraint solver CLP($\neq$) with different constraint solvers (e.g., CLP($\mathbb{Q}$), a linear constraint solver over the rationals); (ii) an extension of the compiler to support the compilation of s(CASP) programs and the generation of the consistency checking rules in the presence of constraints; and (iii) the design and implementation of *C-forall*, a generic algorithm to execute constructive negation that extends the original s(ASP) *forall* algorithm.

The design of the Ciao Prolog implementation of s(CASP) improved performance substantially w.r.t. s(ASP) despite these new capabilities. Table 1 shows these improvements in benchmarks (without constraints) executed on a MacOS 10, Intel i5 at 2GHz.

A s(CASP) program is a finite set of rules of the form: $\texttt{a :- c}_a\texttt{, l}_1\texttt{, ..., l}_m$. where $\texttt{c}_a$ is a (conjunction of) constraints and $\texttt{l}_i$ are literals (also with *default* negation `not` and/or classical (strong) negation `-`). A s(CASP) program execution starts with a query and each answer is the resulting *mgu* of a successful derivation, its justification, and a (partial) stable model. A partial stable model is a subset of an ASP stable model [4] including only the literals necessary to support the query.

## 3 Application and Evaluation

The s(CASP) system allows programmers to directly write programs and queries that cannot be written in other Datalog or constraint ASP systems [3,6,9] without resorting to a complex, unnatural encoding. Additionally, s(CASP) can express answers more elegantly than that done by Datalog/ASP, due to several reasons:

```
valid_stream(P,Data) :-
    stream(P,Data),
    not cancelled(P, Data).
higher_prio(PHi, PLo) :-
    PHi #> PLo.
cancelled(P, Data) :-
    higher_prio(P1, P),
    stream(P1, Data1),
    incompt(Data, Data1).
incompt(p(X),q(X)).
```

Fig. 1: Code of the stream reasoner.

- s(CASP) inherits the use of unbound variables during the execution and in the answers from s(ASP). This makes it possible to express constraints more compactly and naturally (e.g., intervals of times can be written using constraints)
- s(CASP) can use structures/functors directly, thereby avoiding the need to encode them unnaturally (e.g., to capture continuous change in Event Calculus).
- The constraints and the goal-directed evaluation strategy of s(CASP) make it possible to use direct algorithms and reduce search space.

## 3.1 Stream Data Reasoning

Let us assume that we deal with data streams, some of whose items may be contradictory [1], and that different data sources may have different degrees of trustworthiness. We will use these degrees to prefer data items from one source over items from another source in case of inconsistency.

Figure 1 shows the code for a stream reasoner using s(CASP). The rule `valid_stream/2` states that a data stream `stream(P,Data)` is valid if it is *not cancelled* by another contradictory data stream with a higher confidence degree. A data stream item contains the degree of confidence `P` for every `Data` item. `incompt/2` determines which data items are contradictory (in this case, `p(X)` and `q(X)`).

The constraints and the goal-directed strategy of s(CASP) make it possible to resolve queries without evaluating the whole stream database. For example, the rule `incompt(p(X),q(X))` does not have to be grounded w.r.t. the stream database. If timestamps were used as trustworthiness measure, then for a query such as `?- T #> 10, valid_stream(T,p(A))`, the reasoner would validate streams received after `T=10` regardless of how long they extend in the past.

## 3.2 Towers of Hanoi

We encoded this problem for $n$ disks with three systems: clingo, a *standard* ASP solver, setting a bound to the number of moves that can be done; the *incremental* version of clingo [6], where the number $n$ of allowed movements is iteratively increased (see both in the `clingo 5.2.0` distribution); and s(CASP) which, thanks to the top down approach, uses a much more economic CLP-like control strategy [2]. Table 2 compares execution time (in milliseconds) needed to solve the puzzle.

s(CASP) is orders of magnitude faster than both clingo variants because it does not have to generate and test all the possible plans. The standard variant is less interesting than s(CASP)'s, as it merely checks if the problem can be solved in a given number of moves. The incremental variant is by far the slowest, because the program iteratively checks with an increasing number of moves until it can be solved.

| | **s(CASP)** | **clingo 5.2.0** *standard* | **clingo 5.2.0** *incremental* |
|---|---|---|---|
| $n = 7$ | **479** | 3,651 | 9,885 |
| $n = 8$ | **1,499** | 54,104 | 174,224 |
| $n = 9$ | **5,178** | 191,267 | $> 5$ min |

Table 2: Run time (ms) for Towers of Hanoi with $n$ disks.

## 3.3 Event Calculus

Basic Event Calculus (BEC) [10] is a family of formalisms that model commonsense reasoning with a sound, logical basis. Previous attempts [3,9] to mechanize reasoning using BEC faced difficulties in the treatment of continuous change in dense domains (e.g., time and other physical quantities), constraints among variables, default negation, and the uniform application of different inference methods, among others. Let us model the BEC theory using s(CASP) and see how its expressiveness makes it possible to perform deductive and abductive reasoning tasks in domains featuring, for example, constraints involving dense time and fluents affected by continuous changes.

**Translation of Basic Event Calculus:** The translation of BEC axioms [10] into s(CASP) program follows, to some extent, that of the systems EC2ASP and F2LP [7], but we differ in some key aspects that improve performance and are relevant for expressiveness: the treatment of rules with negated heads, the possibility of handling unsafe rules[4], and the use of constraints over rationals/reals. Thanks to the usage of non-ground variables, s(CASP) can directly model event calculus axioms that require unsafe rules, e.g., BEC4: $HoldsAt(f,t) \leftarrow InitiallyP(f) \wedge \neg StoppedIn(0,f,t)$, is translated into:

```
holds At(F,T) :- 0 #< T, initiallyP(F), not stoppedIn(0,F,T).  % BEC4
```

**Translation of the narrative:** The definition of a scenario states the events and effects that occur. Let us consider a vessel that is filled with water from a tap. A possible translation of the logic statements that define that scenario into s(CASP) follows. Note that this translation requires constraint handling with local, uninstantiated variables.

```
max_level(10) :- not max_level(16).    % Force the vessel to be 10 or
max_level(16) :- not max_level(10).    % 16 (two possible worlds).
happens(tapOn,5).                     % TapOn happens at time 5.
initiates(tapOn,filling,T).           % TapOn initiates Filling.
terminates(tapOff,filling,T).         % TapOff terminates Filling.
trajectory(filling,T1,level(X2),T2) :-   % Level(X) represents that
   T1 #< T2, X2 #= X+T2-T1,              % the water is at level X
   max_level(Max), X2 #=< Max,           % in the vessel, as long as
   holdsAt(level(X),T1).                 % its rim is not reached.
```

**Deductive and Abductive reasoning:** We can perform *deduction* (determine possible states) in BEC through queries to the corresponding s(CASP) program. For example:

[4] In ASP, a rule is safe when every variable that appears in its head or in a negated literal in its body also appears in a positive literal in its body (it is unsafe otherwise). ASP solvers such as clingo are not able to process unsafe rules.

```
?- holdsAt(level(H),15/2).           % is true when H = 5/2.
?- holdsAt(level(5/2),T).            % is true when T = 15/2.
```

On the other hand, *abduction* tries to determine a plausible sequence of events that reaches a given state. The line `#abducible happens(tapOff,U)` is a shortcut that states that it is possible (but not necessary) for the tap to be closed at some time `U`. After adding it, the query `?- holdsAt(spilling,T)` determines if and under which conditions the water may overspill, and returns a model containing `holdsAt(spilling,T),T>15,happens(tapOn,5),not happens(TapOff, U), 5<U<15, max_level(10)` meaning that the water will spill at `T=15` if the vessel has a capacity of 10, the tap is open at `T=5`, and it is **not** closed between times 5 and 15.

## 4 Conclusion and Future Work

We have reported on the design and implementation of s(CASP), a top-down system to evaluate constraint answer set programs, based on s(ASP). Its ability to express answer set programs coupled with the possibility of expressing control in a way similar to traditional logic programming makes it a powerful system. In fact, a single program can use both approaches (LP and ASP) simultaneously, achieving the best of both worlds. We have also reported a very substantial performance increase w.r.t. the original s(ASP) implementation. Thanks to the possibility of writing pieces of code with control in mind, it can also beat state-of-the-art ASP systems in certain programs.

The implementation can still be improved substantially. In particular, (i) we want to use program analysis to interleave the execution of odd loops with even loops during top-down execution so as to discard models as soon as they are shown inconsistent [8], (ii) improve the disequality constraint solver, (iii) use dependency analysis to improve the generation of *dual* rules [8], and (iv) apply partial evaluation and better compilation techniques to remove (part of) the interpretation overhead.

## References

1. Arias, J.: Tabled CLP for Reasoning over Stream Data. In: Technical Communications of ICLP'16, vol. 52, pp. 1–8. OASIcs (2016). Doctoral Consortium
2. Arias, J., Carro, M., Salazar, E., Marple, K., Gupta, G.: Constraint answer set programming without grounding. Theory and Practice of Logic Programming **18**(3-4), 337–354 (2018)
3. East, D., Truszczynski, M.: DATALOG with constraints - an answer-set programming system. In: AAAI/IAAI, pp. 163–168. AAAI Press / The MIT Press (2000)
4. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: International Conference on Logic Programming 1988, pp. 1070–1080 (1988)
5. Hermenegildo, M., Bueno, F., Carro, M., López, P., Mera, E., Morales, J., Puebla, G.: An Overview of Ciao and its Design Philosophy. TPLP **12**(1–2), 219–252 (2012)
6. Janhunen, T., Kaminski, R., Ostrowski, M., Schellhorn, S., Wanko, P., Schaub, T.: Clingo goes Linear Constraints over Reals and Integers. TPLP **17**(5-6), 872–888 (2017)
7. Lee, J., Palla, R.: Reformulating the situation calculus and the event calculus in the general theory of stable models. Journal of Artificial Intelligence Research **43**, 571–620 (2012)
8. Marple, K., Salazar, E., Gupta, G.: Computing stable models of normal logic programs without grounding. arXiv preprint arXiv:1709.00501 (2017)

9. Mellarkod, V.S., Gelfond, M., Zhang, Y.: Integrating answer set programming and constraint logic programming. Ann. Math. Artif. Intell. **53**(1-4), 251–287 (2008)
10. Mueller, E.T.: Commonsense reasoning: an event calculus based approach. Morgan Kaufmann (2014)