A Short Tutorial on s(CASP), a Goal-directed Execution of Constraint Answer Set Programs

Joaquín Arias[1] Gopal Gupta[2] Manuel Carro[3,4]

[1]CETINIA, Universidad Rey Juan Carlos [2]University of Texas at Dallas
[3]Universidad Politécnica de Madrid [4]IMDEA Software Institute

GDE'21: ICLP'21 (Virtual) Workshop on Goal-directed Execution of Answer Set Programs
September 20, 2021

Introduction

A short tutorial on s(CASP)

- Highlighting novel aspects and its design.

- s(CASP) is a goal-directed top-down solver for Constraints Answer Set Programs.
- It avoids the grounding phase:
  - It can constraint variables that, as in CLP, are kept during execution and in answer sets.
- Additionally, it generates human-understandable justifications.
- s(CASP) is implemented in Prolog (Ciao[1] and SWI Prolog[2])
- It has been used in several applications:
  - Including medical advisors, avionic, legal reasoner, XAI, and natural language processing.

[1] Available at http://ciao-lang.org/.
[2] Available at https://www.swi-prolog.org/.

Applications

Applications (mainly on commonsense reasoning):

- Event Calculus Reasoner. [Arias et al. 2019]
  - Model real-word avionics systems. [Hall et al. 2021]
- Explainable Artificial Intelligence (XAI). [Arias et al. 2020]
  - Medical advice system. [Chen et al. 2016]
  - Inductive Logic Programming. [Shakerin and Gupta 2019]
  - Generation of concurrent imperatives programs. [Varanasi et al. 2019]
- Natural language understanding systems. [Basu et al. 2021a]
  - CASPR: chatbot for the “Alexa Grand Challenge 4”. [Basu et al. 2021b]
- Coding rule 34 of the Singapore Bar. [Morris 2021]
- Administrative and judicial discretion reasoner. [Arias et al. 2021]
- ... and others (visit GDE’21 Workshop).

Installation under Ciao and SWI-Prolog

- The source code of s(CASP) is available at:
  - `https://gitlab.software.imdea.org/ciao-lang/sCASP` for Ciao.
  - `https://github.com/JanWielemaker/sCASP` for SWI-Prolog.
- Installation:
  - As a **standalone** application: the installation creates an executable called `scasp`:

    ```
    scasp --help_all
    ```

  - **Online**: visit SWISH (`https://swish.swi-prolog.org/`) to run s(CASP) in your browser.

Getting Started: Bindings

- s(CASP) can return models with the bindings for negated calls:
  - Using the disequality constraint solver...

Example 1 (p.pl[3])

```
p(a).
```

For the query `?- not p(X)`*, s(CASP) returns*

```
{not p(X | {X \= a})}
X \= a
```

  - and a solver for linear constraints on rationals (and reals).

Example 2 (p2.pl)

```
p(X) :- X #> 0.
```

For the query `?- not p(X)`*, s(CASP) returns*

```
{not p(X | {X #=< 0})}
X #=< 0
```

[3] Examples available at `http://platon.etsii.urjc.es/~jarias/papers/scasp-gde21/`

Getting Started: Uninterpreted Functions

- Under conventional ASP, lists, such as `[f(a)|Rest]`, the grounded program can be infinite.

Example 3 (`member.pl`)

```
1 member(X, [X|Xs]).
2 member(X, [_|Xs]):- member(X, Xs).
3 list([1,2,3,4,5]).
4 ?- list(A), not member(B, A).
```

The query is part of the program (line 4) and returns:

```
{ list([1,2,3,4,5]),
   not member(B | {B ≠ 1,B ≠ 2,B ≠ 3,B ≠ 4,B ≠ 5}, [1,2,3,4,5]),
   not member(B | {B ≠ 1,B ≠ 2,B ≠ 3,B ≠ 4,B ≠ 5}, [2,3,4,5]),
   not member(B | {B ≠ 1,B ≠ 2,B ≠ 3,B ≠ 4,B ≠ 5}, [3,4,5]),
   not member(B | {B ≠ 1,B ≠ 2,B ≠ 3,B ≠ 4,B ≠ 5}, [4,5]),
   not member(B | {B ≠ 1,B ≠ 2,B ≠ 3,B ≠ 4,B ≠ 5}, [5]),
   not member(B | {B ≠ 1,B ≠ 2,B ≠ 3,B ≠ 4,B ≠ 5}, []) }
A = [1,2,3,4,5], B ≠ 1, B ≠ 2, B ≠ 3, B ≠ 4, B ≠ 5
```

Getting Started: Stable Models

- s(CASP) is based on stable model semantics [Gelfond and Lifschitz 1988] and supports non-stratified negation.

Example 4 (`weekend.pl`)

```
opera(saturday) :- not home(saturday).
home(saturday) :- not opera(saturday).
dinner(sunday).
```

For the query `?- opera(saturday)` *it returns* {`opera(saturday)`,`not home(saturday)`}

For the query `?- home(saturday)` *it returns* {`home(saturday)`,`not opera(saturday)`}

For the query `?- dinner(sunday)` *it returns* {`dinner(sunday)`}

For the query `?- opera(saturday),home(saturday)` *it returns* `no models`.

- Additionally, s(CASP) supports **default** and **classical** negation:
  - `not opera(saturday)`: no evidence that Bob goes to the opera (we *cannot prove* it).
  - `-opera(saturday)`: explicit evidence that Bob does not go to the opera (there is a *proof*).

Introduction
Applications
Installation
Getting start
Bindings
Uninterpreted functions
Stable models
Tune output
Justifications
In NL / HTML
Dual (in NL)
DCC
Future Work

Tuning the Output of Partial Answer Sets

- s(CASP) provides a directive to select which atoms should appear in the partial model:

Example 5 (`weekend_show.pl`)

Including the following directive in `weekend.pl`

```
#show opera/1, home/1, dinner/1.
```

For the query `?- opera(saturday)` *it returns* {`opera(saturday)`}

Negated atoms can also be selected, e.g., `#show not home/1,-opera/1` *is also valid.*

- Denials: `:- p,q` means that $p \wedge q$ cannot be true in any model.
  - Olon rules: `r :- q,not r` forces that every model to satisfy $\neg q \vee r$.

Example 6 (`olon.pl`)

```
p :- not q.
q :- not p.
r :- not r.
```

The compiler introduces the denial `:- not r` *and therefore, there are no models.*

Tuning the Output of Partial Answer Sets (cont.)

Example 7 (opera.pl)

```
1  opera(D) :- not home(D).      % A day D, Bob either goes to the opera...
2  home(D) :- not opera(D).      %                         ... or stays home.
3  home(monday).                 % On Monday, Bob stays at home.
4
5  :- baby(D), opera(D).         % When Bob's best friend comes with her baby, it is
6                                % not a good idea to take the baby to the opera.
7  baby(tuesday).                % They come on Tuesday.
8
9  ?- opera(D).                  % QUERY: When might Bob go to the opera?
```

The denial in line 5 expresses that `baby(D)` $\wedge$ `opera(D)` *cannot be simultaneously true for any value of* `D`*, so it returns* `D \= monday,D \=tuesday`

- For debugging purposes, s(CASP) provides flags to disable consistency checks:
  - `--no_olon` disables the consistency checking due to olon rules.
  - `--no_nmr` disables consistency checking for all non-monotonic rules.

Tuning the Output of Partial Answer Sets (cont.)

- A Partial Answer Set is a partial model with a specific binding of the query variables.

Example 8 (`member.pl` in Example 3)

For the query `?- list(A),member(B,A)`*, it returns 5 answers sets, one for each binding of* `B`*:*

- *For* `B=1` `{ list([1,2,3,4,5]),member(1,[1,2,3,4,5]) }`
- *For* `B=2` `{ list([1,2,3,4,5]),member(2,[1,2,3,4,5]),member(2,[2,3,4,5]) }`
- *For* `B=3` `{ list([1,2,3,4,5]),member(3,[1,2,3,4,5]),member(3,[2,3,4,5]),... }`
- *For* `B=4` `{ list([1,2,3,4,5]),member(4,[1,2,3,4,5]),member(4,[2,3,4,5]),... }`
- *For* `B=5` `{ list([1,2,3,4,5]),member(5,[1,2,3,4,5]),member(5,[2,3,4,5]),... }`

- s(CASP) provides flags to control the number of partial answer sets to be generated:
  - `-sN` or `-nN` return the first *N* answers sets.
  - `-s0` or `-n0` return all the possible answers sets.

Tuning the Output of Partial Answer Sets (cont.)

- s(CASP) includes CLP($\mathbb{Q}$) [Holzbaur 1995], a solver for linear constraints on the rationals (for soundness reasons), and provides a flag:
  - `-r[=d]` to output rationals as floating-point numbers with `d` decimal places.

Example 9 (`rationals.pl`)

```
s(X,Y) :- X #= Y * 7/53.
```

For the query `?- s(X,4.35)` *it returns* `X=609/1060`

... invoking `scasp -r=3 rationals.pl` *it returns* `X=0.574`

Justifications

- The top-down strategy of s(CASP) generates minimal justification trees.
  - s(CASP) provides flags to control which literals should appear in them:
    - `--mid` only displays the user-defined predicates.
    - `--long` displays all predicates, including `forall/2` used to check denials.
    - `--short` only displays the annotated literals.
  - Also, `--neg` includes the default-negated atoms, while `--pos` does not.

Example 10 (`opera.pl` in Example 7)

The invocation `scasp --tree opera.pl` generates:

```
JUSTIFICATION_TREE:
assume(opera(D | {D \= monday,D \= tuesday})):-
    not home(D | {D \= monday,D \= tuesday}).
denial :-
    not o_chk_1 :-
        not baby(Var1 | {Var1 \= tuesday}),
        baby(tuesday),
        assume(not opera(tuesday)) :-
            home(tuesday).
```

Justifications in Natural Language (NL)

Example 11 (`opera.pl` and `opera.pred` cont. Example 10)

```
#pred opera(D) :: 'Bob goes to the opera on (D:day)'.
#pred not home(D) :: 'Bob does not stay at home on (D)'.
```

The call `scasp --tree --human opera.pl opera.pred` *generates*

```
JUSTIFICATION_TREE:
we assume that Bob goes to the opera on a day D not equal monday, nor tuesday, because
    Bob does not stay at home on D not equal monday, nor tuesday.
The global constraints hold, because
    the global constraint number 1 holds, because
        there is no evidence that 'baby' holds (for Var1), with Var1 not equal tuesday, and
        'baby' holds (for tuesday), and
        we assume that there is no evidence that Bob goes to the opera on the day tuesday, because
            'home' holds (for tuesday).
```

- Additionally, s(CASP) provides a flag to generate expandable justification tree in HTML:
  - By adding `--html=bob`, it generates `bob.html`.

Dual Program (also in NL)

- The dual program expresses the constructive negation of each predicate.
  - For each i^{th} literal in the body of a clause, it generates a clause with its "dual" literal.
  - To avoid redundant answers, every i^{th} clause includes calls to any j^{th} literal with $j < i$.

Example 12

The dual of the clause `h(X,Y) :- r(X),not s(X,Y),q(Y)` *is:*

```
not h(X,Y) :- not r(X).
not h(X,Y) :- r(X), s(X,Y).
not h(X,Y) :- r(X), not s(X,Y), not q(Y).
```

- s(CASP) provides flags to modify the generation of the dual program and to inspect it:
  - `-d` or `--plaindual` generate duals with single-goal clauses.
  - `--code` output the dual program compiled by s(CASP)
    (which can be modified and loaded using `-c` or `--compiled`.)

Example 13 (`opera.pl` and `opera.pred` cont. Example 11)

Invoking `scasp --code --human opera.pl opera.pred` *it provides the natural language code.*

Dynamic Consistency Check (DCC) – work in progress

- s(CASP) checks that a tentative partial model is consistent after the query.
- When the check fails, it backtracks to search other alternatives.
- The goal of DCC [Marple and Gupta 2014] is to check the denials as soon as the atoms involved are added to the tentative partial model.

Example 14 (`hamiltonian.pl`)

```
#show chosen/2.
reachable(V) :- chosen(V, a).
reachable(V) :- chosen(V,U), reachable(U).
% Choose or not an edge of the graph.
chosen(U,V) :- edge(U,V), not other(U,V).
other(U,V) :- edge(U,V), not chosen(U,V).
% Every vertex must be reachable.
:- vertex(U), not reachable(U).
% Do not choose edges to/from the same vertex
:- chosen(U,W), U \= V, chosen(V,W).
:- chosen(W,U), U \= V, chosen(W,V).
?- reachable(a).
```

For this program using the graph with 4 vertices and 3 Hamiltonian cycles in `graph.pl`:

- *Without DCC: the evaluation takes **8.266s**.*
- *With DCC: the evaluation takes only **1.215s**.* *A speed up of 6.8*

Future Work

- Complete DCC implementation, use program analysis to optimize compilation of DCC rules.
- Improve the generation of dual programs using dependency analysis.
- Reduce interpreting overhead applying partial evaluation and better compilation techniques.
- Optimize the implementation of the c-forall algorithm using Mod TCLP [Arias and Carro 2019a].

  ... there are four alternatives: default, `all_c_forall`, `prev_forall`, and `sasp_forall`.
- Explore the use of Mod TCLP, and ATCLP [Arias and Carro 2019b] to:
  - Collect minimal partial models, increasing performance and readability.
  - Handle positive variant loops. Currently they are halted; solutions can be missed.
    - This behavior can be disabled by adding `--variant` but, it can lead to loops.
- Improve the constraint solver disequality to handle some pending cases.
  - Flags `-w` or `--warning` detect and warn in case of unsupported constraint or variant loops.
- Enhance its integration with Ciao and SWI-Prolog debuggers.
  - Flags `v`, `v0`, `v1`, and `v2` trace program evaluation.

If in doubt please contact `mailto:joaquin.arias@urjc.es`

Bibliography I

[1] Arias, J. and Carro, M. (2019a). Description, Implementation, and Evaluation of a Generic Design for Tabled CLP. *Theory and Practice of Logic Programming*, 19(3):412–448.

[2] Arias, J. and Carro, M. (2019b). Incremental evaluation of lattice-based aggregates in logic programming using modular TCLP. In Alferes, J. J. and Johansson, M., editors, *21st Int'l. Symposium on Practical Aspects of Declarative Languages*, volume 11372 of *LNCS*, pages 98–114. Springer.

[3] Arias, J., Carro, M., Chen, Z., and Gupta, G. (2020). Justifications for goal-directed constraint answer set programming. In *Proceedings 36th International Conference on Logic Programming (Technical Communications)*, volume 325 of *EPTCS*, pages 59–72. Open Publishing Association.

[4] Arias, J., Chen, Z., Carro, M., and Gupta, G. (2019). Modeling and Reasoning in Event Calculus Using Goal-Directed Constraint Answer Set Programming. In *Pre-Proc. of the 29th Int'l. Symposium on Logic-based Program Synthesis and Transformation*.

[5] Arias, J., Moreno-Rebato, M., Rodriguez-García, J. A., and Ossowski, S. (2021). Modeling Administrative Discretion Using Goal-Directed Answer Set Programming. In *Advances in Artificial Intelligence, CAEPIA 20/21*, pages 258–267, Cham. Springer International Publishing.

[6] Basu, K., Varanasi, S., Shakerin, F., Arias, J., and Gupta, G. (2021a). Knowledge-driven Natural Language Understanding of English Text and its Applications. *AAAI'21*.

Bibliography II

[7] Basu, K., Wang, H., Dominguez, N., Li, X., Li, F., Varanasi, S. C., and Gupta, G. (2021b). CASPR: A Commonsense Reasoning-based Conversational Socialbot. In *4th Proceedings of Alexa Prize (Alexa Prize 2021).*

[8] Chen, Z., Marple, K., Salazar, E., Gupta, G., and Tamil, L. (2016). A Physician Advisory System for Chronic Heart Failure Management Based on Knowledge Patterns. *Theory and Practice of Logic Programming*, 16(5-6):604–618.

[9] Gelfond, M. and Lifschitz, V. (1988). The Stable Model Semantics for Logic Programming. In *5th International Conference on Logic Programming*, pages 1070–1080.

[10] Hall, B., Varanasi, S. C., Fiedor, J., Arias, J., Basu, K., Li, F., Bhatt, D., Driscoll, K., Salazar, E., and Gupta, G. (2021). Knowledge-Assisted Reasoning of Model-Augmented System Requirements with Event Calculus and Goal-Directed Answer Set Programming. In *Proc. 8th Workshop on Horn Clause Verification and Synthesis*.

[11] Holzbaur, C. (1995). OFAI CLP(Q,R) Manual, Edition 1.3.3. Technical Report TR-95-09, Austrian Research Institute for Artificial Intelligence, Vienna.

[12] Marple, K. and Gupta, G. (2014). Dynamic Consistency Checking in Goal-Directed Answer Set Programming. *Theory and Practice of Loging Programming*, 14(4-5):415–427.

Bibliography III

[13] Morris, J. (2021). Constraint answer set programming as a tool to improve legislative drafting: a rules as code experiment. In *ICAIL*, pages 262–263. ACM.

[14] Shakerin, F. and Gupta, G. (2019). Induction of Non-Monotonic Logic Programs to Explain Boosted Tree Models Using LIME. In *AAAI 2019*, pages 3052–3059.

[15] Varanasi, S. C., Salazar, E., Mittal, N., and Gupta, G. (2019). Synthesizing Imperative Code from Answer Set Programming Specifications. In *LOPSTR*, volume 12042 of *Lecture Notes in Computer Science*, pages 75–89. Springer.