Arguing Correctness of ASP Programs with Aggregates

Jorge Fandinno Zach Hansen Yuliya Lierler

University of Nebraska Omaha

September 2022

Table of Contents

Table of Contents

Motivation

Program Verification

ASP is a promising approach to producing *trustworthy* AI

We wish to enhance the methodology for proving correctness of ASP programs with aggregates

Desirable qualities include modularity, reusability, elaboration tolerance, and potential for automation

Program Rewriting

An ASP program can be viewed as a formal specification

Subsequent rewrites must be proven to be equivalent to the original

Eventual extensions of ANTHEM[a] may incorporate this program-to-program verification

[a] J. Fandinno et al. "Verifying Tight Logic Programs with anthem and vampire". In: *Theory and Practice of Logic Programming* 20.5 (2020), pp. 735–750

VLP Methodology

Verification Methodology for modular ASP programs:[1]

Step I: Decompose the informal description of the problem into independent (natural language) statements.

Step II: Fix the public predicates used to represent the problem and its solutions.

Step III: Formalize the specification of the statements as a non-ground modular program, possibly introducing auxiliary predicates.

Step IV: Construct an argument (a "metaproof" in natural language) for the correspondence between the constructed program and the informal description of the problem.

[1] P. Cabalar, J. Fandinno, and Y. Lierler. "Modular Answer Set Programming as a Formal Specification Language". In: *Theory and Practice of Logic Programming* 20.5 (2020), pp. 767–782

The Graph Coloring (GC) Problem

Choice Rules With Cardinality Bounds

```
{assign(V,C) : color(C) } = 1 :- vertex(V).
:- edge(V1,V2), assign(V1,C), assign(V2,C).
```

Rewritten With Aggregates

```
assign(V,C) :- vertex(V), color(C), not not assign(V,C).
:- vertex(V), not #count{ V,C : assign(V,C), color(C) } = 1.
:- edge(V1,V2), assign(V1,C), assign(V2,C).
```

Table of Contents

Program Syntax

We consider programs of a typical ASP syntax.
An **aggregate element** has the form

$$t_1, \ldots, t_k : l_1, \ldots, l_m$$

An **aggregate atom** has the form

$$\#\mathtt{op}\{E\} \prec u$$

We consider op to be `count` or `sum`. For example,

```
#count{ V,C : assign(V,C), color(C) } = 1,      (1)
#sum{ K,X,Y : in(X,Y), cost(K,X,Y) } > J.       (2)
```

Program Syntax Continued

Rules have the form

$$\mathit{Head} \text{ :- } B_1, \ldots, B_n,$$

Choice rules have the form

$$\{A_0 : A_1, \ldots, A_k\} \prec u \text{ :- } B_1, \ldots, B_n.$$

where each A_i is an atom, each B_i is a literal.
Abbreviation for:

$$A_0 \text{ :- } A_1, \ldots A_k,\ B_1, \ldots, B_n,\ \mathit{not}\ \mathit{not}\ A_0.$$
$$\text{:- } B_1, \ldots, B_n,\ \mathit{not}\ \texttt{\#count}\{\mathbf{t} : A_0, A_1, \ldots, A_k\} \prec u.$$

where $\mathbf{t}$ is a list of program terms such that A_0 is of the form $p(\mathbf{t})$.

Translation τ^*

Translating Aggregates

An aggregate atom A of form $\#\mathrm{op}\{E\} \prec u$ is translated

$$count(set_{|E/\mathbf{X}|}(\mathbf{X})) \prec u \text{ or } sum(set_{|E/\mathbf{X}|}(\mathbf{X}))$$

where $set_{|E/\mathbf{X}|}$ is a function symbol that takes as many arguments of the program sort as there are variables in $\mathbf{X}$ (the global variables in the aggregate rule).

Translating Rules

Rules are translated to the universal closure across global variables of the following:

$$\tau^* B_1 \wedge \cdots \wedge \tau^* B_n \rightarrow \tau^* \mathit{Head}\,.$$

Example

GC Program (Π)

```
assign(V,C) :- vertex(V), color(C), not not assign(V,C).
:- vertex(V), not #count{ V,C : assign(V,C), color(C) } = 1.
:- edge(V1,V2), assign(V1,C), assign(V2,C).
```

Many-sorted First-order formula $\tau^*(\Pi)$

$$\forall VC\,(vertex(V) \wedge color(C) \wedge \neg\neg assign(V,C) \rightarrow assign(V,C))$$
$$\forall V(vertex(V) \wedge \neg count(set_{asg}(V)) = 1 \rightarrow \bot)$$
$$\forall V1\ V2\ C\,(edge(V1,V2) \wedge assign(V1,C) \wedge assign(V1,C) \rightarrow \bot)$$

where *asg* is the set symbol $V, C : assign(V,C), color(C)/V$.

Table of Contents

Many-Sorted SM Operator

Many-Sorted SM

Our treatment of aggregates uses a many-sorted generalization of the SM^a operator mandating that arities respect sorts.

[a] P. Ferraris, J. Lee, and V. Lifschitz. "Stable models and circumscription". In: *Artificial Intelligence* 175.1 (2011), pp. 236–263

Agg-interpretations

We restrict our attention to certain types of standard interpretations that fix the interpretation of some symbols.
Note that some of these conditions can be replaced by first or second order axiomatizations.[a]

[a] J. Fandinno, Z. Hansen, and Y. Lierler. "Axiomatization of Aggregates in Answer Set Programming". In: *Proceedings of the Thirty-six National Conference on Artificial Intelligence (AAAI'22)*. AAAI Press, 2022

Splitting Theorem

Example

A many-sorted generalization of the Splitting Theorem[a] allows us to decompose programs into modules. Take F_1 to be sentences

$$\forall VC\big(vertex(V) \wedge color(C) \wedge \neg\neg assign(V, C) \rightarrow assign(V, C)\big) \quad (3)$$

$$\forall V(vertex(V) \wedge \neg count(set_{asg}(V)) = 1 \rightarrow \bot) \quad (4)$$

and F_2 to be sentence

$$\forall V1\ V2\ C\big(edge(V1, V2) \wedge assign(V1, C) \wedge assign(V1, C) \rightarrow \bot\big) \quad (5)$$

Then $\mathrm{SM}_{assign}[(3) \wedge (4) \wedge (5)]$ is equivalent to $\mathrm{SM}_{assign}[(3) \wedge (4)] \wedge (5)$
Here we consider **p** to be *assign* and **q** to be empty.

---

[a] P. Ferraris et al. "Symmetric Splitting in the General Theory of Stable Models". In: *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI'09)*. Ed. by C. Boutilier. AAAI/MIT Press, 2009, pp. 797–803

Table of Contents

Functional Relations

Motivating Example

Consider the GC problem: predicate *assign* should encode a function from vertices to colors.

Generalization

A relation $\mathbf{r}$ *encodes* function $f : A \longrightarrow B$ when $\mathbf{r} = \{(a, f(a)) \mid a \in A\}$.
Let $Fun_{A,B}$ be the conjunction of formulas

$$\forall X \big(G(X) \wedge \neg count(set_{fe}(X)) = 1 \rightarrow \bot\big) \tag{6}$$

$$\forall XY \big(G(X) \wedge H(Y) \wedge \neg\neg f(X, Y) \rightarrow f(X, Y)\big) \tag{7}$$

where *fe* is the name of the set symbol $X, Y : f(X, Y), H(Y)/X$.
Then, $I \models \mathrm{SM}_f[Fun_{A,B}]$ iff $(f/2)^I$ encodes a function from A to B.

Proof Sketch

By the conditions of agg-interpretations

Consider set symbol $X, Y : f(X, Y), H(Y)/X$ (named *fe*).
For an agg-interp I and a domain element a, $set_{fe}(a)^I$ is the set of two-tuples $\langle a, Y \rangle$ s.t. $f(a, Y)$ and $H(Y)$ hold under I. Thus, $count(set_{fe}(a))^I$ is the number of domain elements b s.t. $f(a, b)$ and $H(b)$

By the Splitting Theorem

$$\mathrm{SM}_f[(6) \wedge (7)] = \mathrm{SM}_f[(7)] \wedge (6)$$

By the Completion Theorem

$$\mathrm{SM}_f[(7)] = \forall XY \Big(f(X, Y) \leftrightarrow G(X) \wedge H(Y) \wedge \neg\neg f(X, Y)) \Big)$$
$$\leftrightarrow \forall XY \big(f(X, Y) \rightarrow G(X) \wedge H(Y)) \big)$$

If $G(a)$ holds for $a \in A$, then $| \{ b \mid b \in B \wedge f(a, b) \wedge H(b) \} | = 1$

The Graph Coloring (GC) Problem

Instances are triples $\langle V, E, C \rangle$

$\langle V, E \rangle$ is a graph with vertices V and edges $E \subseteq V \times V$, and C is a set of labels named *colors*.

Solutions

CF1 a function $asg : V \longrightarrow C$ such that

CF2 every edge $(a, b) \in E$ satisfies condition $asg(a) \neq asg(b)$.

The Graph Coloring (GC) Problem

VLP Step 1

C1 find an assignment from nodes to colors such that

C2 connected nodes do not have the same color.

VLP Step 2

Select public predicates: *edge*/2, *vertex*/1, *color*/1, *assign*/2

VLP Step 3

Formalize statements as modules:

$$\mathrm{SM}_{assign}[\tau^*(\texttt{\{assign(V,C) : color(C) \} = 1 :- vertex(V)})]$$
$$\tau^*(\texttt{:- edge(V1,V2), assign(V1,C), assign(V2,C).})$$

The GC Problem

Theorem 3

Let I be an agg-interp such that $\langle vertex^I, edge^I, color^I \rangle$ forms an instance of the Graph Coloring problem. Then, I is a model of our modules iff $(assign/2)^I$ encodes a function that forms a solution to the considered instance.

VLP Step 4 (Part 1)

The previous result on functional relations proves that

$$I \models \mathrm{SM}_{assign}[\tau^*(\texttt{\{assign(V,C) : color(C) \} = 1 :- vertex(V)})]$$

iff $(assign/2)^I$ encodes a function $asg : vertex^I \longrightarrow color^I$
s.t. $(assign/2)^I = \{(a, asg(a)) \mid a \in vertex^I\}$.

The GC Problem

VLP Step 4 (Part 2)

$$\tau^*\big(\texttt{:- edge(V1,V2), assign(V1,C), assign(V2,C).}\big)$$

is equivalent to

$$\forall V1\ V2\ C1\ C2\big(edge(V1, V2) \land assign(V1, C1) \land assign(V2, C2) \rightarrow C1 \neq C2\big).$$

This sentence is satisfied by I iff every edge $(a, b) \in edge^I$ satisfies $asg(a) \neq asg(b)$.

Conclusion

Contribution

1. We have extended the VLP methodology to programs with aggregates (no reference to grounding or specific instances).
2. We showcase the utility of "recycling" proofs (the paper extends the HCP to the TSP).
3. Both *count* and *sum* are discussed in the paper.

Limitations

The many-sorted aggregate semantics are only applicable to aggregates without positive recursion.

Future Work

Removing the recursion limitation and extending ANTHEM to support aggregates.