Technical Disclosure Commons

Defensive Publications Series

January 07, 2019

Delayed predicate algorithm evaluation in hybrid answer set programming (ASP)

Alex Brik

Jori Bomanson

Delayed predicate algorithm evaluation in hybrid answer set programming (ASP)

ABSTRACT

A hybrid answer set programming (ASP) solver is a logic programming type solver with a variety of applications, e.g., diagnosing software failures, modeling dynamical systems, etc. A hybrid ASP solver finds solutions for hybrid ASP programs, which are sets of hybrid ASP rules. Hybrid ASP rules include stationary rules with associated predicate algorithms and advancing rules with associated advancing algorithms. Solutions to hybrid ASP programs are found iteratively by starting with an initial state and producing consequent states. Part of computing a consequent state involves determination of the stationary rules that are applicable at the consequent state. This in turn involves evaluating predicate algorithms associated with the rules. This approach is inefficient, since it requires that all the predicate algorithms be evaluated in all the states.

The techniques of this disclosure avoid the inefficiency of evaluating predicate algorithms by adding auxiliary rules that enable intelligent guesses as to whether an algorithm will accept or reject, in turn enabling the continued evaluation of the rules without the evaluation of the algorithms.

KEYWORDS

- answer set programming (ASP)
- logic programming
- declarative programming
- predicate algorithms
- failure diagnosis

BACKGROUND

A hybrid answer set programming (ASP) solver [1] is a logic programming type solver with a variety of applications, e.g., diagnosing software failures, modeling dynamical systems, etc. A hybrid ASP solver finds solutions for hybrid ASP programs, which are sets of hybrid ASP rules. Hybrid ASP rules include stationary rules with associated predicate algorithms and advancing rules with associated advancing algorithms. For example, hybrid ASP rules are of the form:

`a :- B[1];...; B[k] : P,` where

for `i = 1 to k`, `B[i]` is a block, `a` is a predicate atom, and `P` is a predicate algorithm. A block `B[i]` is of the form `b[1],..., b[m], not b[m+1],..., not b[m+n],` where for `i from 1 to m+n, b[i]` is a predicate atom.

Solutions to hybrid ASP programs are found iteratively by starting with an initial state and producing consequent states. Part of computing a consequent state involves determining which stationary rules are applicable at the consequent state. This in turn involves evaluating predicate algorithms associated with the rules. This approach is inefficient, since it requires that all the predicate algorithms be evaluated in all the states.

DESCRIPTION

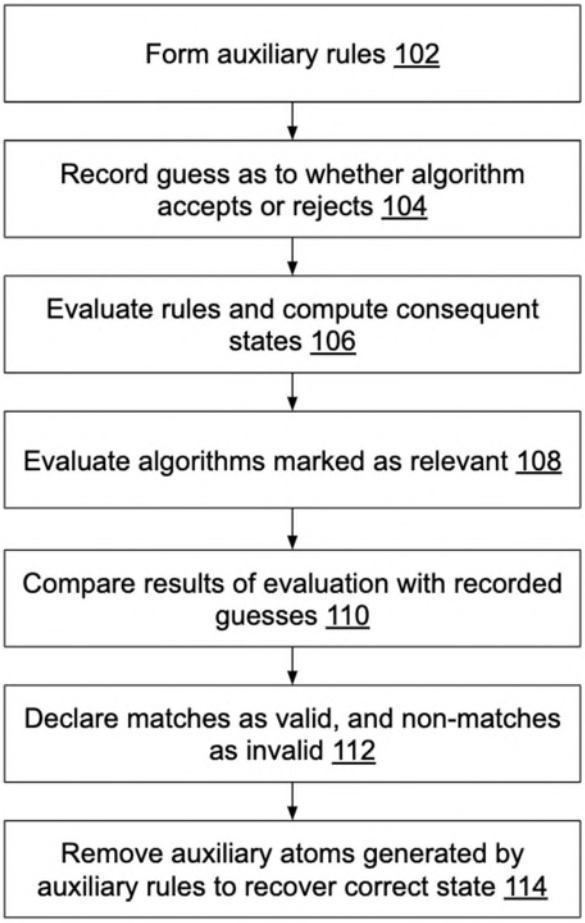

Fig. 1: Delayed predicate algorithm evaluation in hybrid ASP

As illustrated in Fig. 1, the techniques of this disclosure avoid the inefficiency of evaluating all the predicate algorithms in every state as follows. For every stationary rule with a predicate algorithm with name "`name`," form the following auxiliary rules (102):

- add an auxiliary rule with the same prerequisite as the original rule, and with conclusion auxiliary atom `_relevant_(name)` stating that the algorithm is relevant;
- add a rule with the prerequisite being `_relevant_(name)`, not `_accepting_(name)`, and conclusion being `_rejecting_(name)`;

- add a rule with the prerequisite being `_relevant_(name)`, not `_rejecting_(name)`, and conclusion being `_accepting_(name)`; and
- add another rule with the conclusion of the original stationary rule, the body of the original rule, and an additional prerequisite `_relevant_(name)` added to the last block.

The aforementioned addition of rules enables the recording of a guess as to whether the algorithm returns true or false, e.g., accepts or rejects (104). The rules can thereby continue being evaluated (106) without evaluation of the algorithm. Once the consequent states are computed, only the algorithms marked as relevant are evaluated (108), and the results of the evaluation are compared with the recorded guesses (110). States where the actual results and guesses match are considered valid. States where the actual results and guesses don't match are considered invalid (112).

After validating the correct guesses, the auxiliary atoms generated by the auxiliary rules (e.g., `_relevant_(name), _accepting_(name), _rejecting_(name)`) are removed to recover the correct state (116).

Example

Consider a hybrid ASP program comprising a single stationary rule:

```
(R) a :- b, c : PA,
```

where `a, b, c` are predicate atoms and `PA` is a predicate algorithm.

Per the techniques of this disclosure, the following rules replace rule `R`:

```
(A1) _relevant_(PA) :- b, c
(A1) _accepting_(PA) :- _relevant_(PA), not _rejecting_(PA)
(A3) _rejecting_(PA) :- _relevant_(PA), not _accepting_(PA)
```

```
(A4) a :- b, c, _accepting_(PA)
```

Consider now a state `({b, c}, q),` where `q` is a set of parameter values.

If `PA(q)` is `TRUE`, then the state should be `({a, b, c}, q).`

If `PA(q)` is `FALSE`, then the state should remain `({b, c}, q).`

Considering rules `A1-A4`, there are two candidate states:

`({b, c, _relevant_(PA), _accepting_(PA), a}, q)` and

`({b, c, _relevant_(PA), _rejecting_(PA)), q).`

Note that the states include `_relevant_(PA);` this indicates that `PA` should be evaluated and the result be validated by the guess made in the state.

If `PA(q)` is `TRUE,` then the `_accepting_(PA)` guess is correct and `_rejecting_(PA)` guess is incorrect. Thus, the first state `({b, c, _relevant_(PA), _accepting_(PA), a}, q)` is valid and the second state `({b, c, _relevant_(PA), _rejecting_(PA)), q)` is invalid. The auxiliary atoms are removed to get the following valid state `({a, b, c}, q).`

If `PA(q)` is `FALSE,` then the `_rejecting_(PA)` guess is correct and the `_accepting_(PA)` guess is incorrect. Thus, the second state `({b, c, _relevant_(PA), _rejecting_(PA)), q)` is valid and the first state is invalid. The auxiliary atoms are removed to get the following valid state `({b, c}, q).`

This example illustrates that if the state satisfies the prerequisites `{b, c}` of the rule `(R)` then the predicate algorithm `PA` experiences delayed evaluation, and the correct state is the identified. On the other hand, if the state does not satisfy the prerequisites of the rule `(R),` e.g., the state is `({b}, q),` then `(R)` is not applicable, and the resultant state is `({b}, q)`.

Nevertheless, since the prerequisites of (A1) are not satisfied, _relevant_(PA) is not derived. Consequently, PA is not evaluated, thus resulting in increased efficiency.

For the general stationary rule of the form:

```
a :- B[1]; ...; B[k] : P
```

the translation looks, similar to rules A1-A4, as follows:

```
(A'1) _relevant_(PA) :- B[1]; ...; B[k]
(A'1) _accepting_(PA) :- _relevant_(PA), not _rejecting_(PA)
(A'3) _rejecting_(PA) :- _relevant_(PA), not _accepting_(PA)
(A'4) a :- B[1]; ...; B[k] + _accepting_(PA)
```

where B[k]+_accepting_(PA) stands for a block obtained by appending an atom _accepting_(PA) to B[k].

For example, if B[k] is

```
b[1], ..., b[m], not b[m+1], ..., not b[m+n],
```

then B[k]+_accepting_(PA) is

```
b[1], ..., b[m], _accepting_(PA), not b[m+1], ..., not b[m+n].
```

CONCLUSION

The techniques of this disclosure avoid the inefficiency of evaluating predicate algorithms in hybrid answer set programming by adding auxiliary rules that enable intelligent guesses as to whether an algorithm will accept or reject, in turn enabling the continued evaluation of the rules without the evaluation of the algorithms.

REFERENCES

[1] Brik, Alex, and Jeffrey B. Remmel. "Hybrid ASP." In *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 11. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2011. Available online at http://drops.dagstuhl.de/opus/volltexte/2011/3179/pdf/22.pdf