Online Appendix: Abstraction for Non-Ground Answer Set Programs

Zeynep G. Saribatur, Peter Schüller, and Thomas Eiter

Institute of Logic and Computation,
TU Wien, Vienna, Austria

A Encoding for Example 1

% action choice
$\{moveToB(B, B_1, T) : bl(B), bl(B1); moveToT(B, L, T) : bl(B), tbl(L)\} \leftarrow T < t_{max}.$
% preconditions
$\leftarrow moveToB(B, B_2, T), onB(B_1, B, T).$
$\leftarrow moveToB(B_1, B_2, T), onB(B_1, B_2, T).$
$\leftarrow moveToT(B, L, T), onB(B_1, B, T).$
$\leftarrow moveToT(B, L, T), onT(B, L, T).$
% effects
$onB(B, B_1, T{+}1) \leftarrow moveToB(B, B_1, T), T < t_{max}.$
$onT(B, L, T{+}1) \leftarrow moveToT(B, L, T), T < t_{max}.$
$\neg onB(B, B_2, T) \leftarrow onB(B, B_1, T), B_1 \neq B_2.$
$\neg onT(B, L, T) \leftarrow onB(B, B_1, T).$
$\neg onB(B, B_1, T) \leftarrow onT(B, L, T).$
$\neg onT(B, L_1, T) \leftarrow onT(B, L_2, T), L_1 \neq L_2.$
% inertia
$onB(B, B_1, T{+}1) \leftarrow onB(B, B_1, T), not\ \neg onB(B, B_1, T{+}1), T < t_{max}.$
$onT(B, L, T{+}1) \leftarrow onT(B, L, T), not\ \neg onT(B, L, T{+}1), T < t_{max}.$
% state constraints
$\leftarrow onB(B_1, B, T), onB(B_2, B, T), B_1 \neq B_2.$
$\leftarrow onB(B, B_1, T), onB(B, B_2, T), B_1 \neq B_2.$
$\leftarrow onB(B, B_1, T), onT(B, L, T).$
$\leftarrow onB(B, B_1, T), B_1 \leq B.$
% goal constraints
$notblockgoal(T) \leftarrow onT(B, L, T), onT(B_1, L_1, T), B \neq B_1.$
$\leftarrow notblockgoal(T), maxTime(T).$
$\leftarrow not\ notblockgoal(T), onT(B, L, T), not\ chosenTable(L).$

B Debugging Spuriousness

The debugging approach in [1] is based on a tagging technique that allows one to control the formation of answer sets and to manipulate the evaluation of the program. The meta-program constructed by `spock` [1] introduces *tags* to

control the formation of answer sets. Given a program Π over $\mathcal{A}$ and a set $\mathcal{N}$ of names for all rules in Π, it creates an enriched alphabet $\mathcal{A}^+$ obtained from $\mathcal{A}$ by adding atoms such as $ap(n_r), bl(n_r), ok(n_r), ko(n_r)$ where $n_r \in \mathcal{N}$ for each $r \in \Pi$. The atoms $ap(n_r), bl(n_r)$ express whether a rule r is applicable or blocked, respectively, while $ok(n_r), ko(n_r)$ are used for manipulating the application of r.

The approach in [1] is applied to ground rules, and [2] employs this approach in debugging the spurious abstract answer sets for their abstraction of omitting atoms. As lifting the method to non-ground rules with domain abstraction is not immediately trivial, we focused on a simplified encoding that disregards loop formulas. Thus, each rule that contains a predicate of the abstracted domain gets extended with an ab atom including the variables related with the domain.

Example 1 *The program Π in Example 2 is converted into Π_{ab}:*

$$c(X) \leftarrow not\ d(X), X < 5, int(X), not\ ab(r1, X).$$
$$d(X) \leftarrow not\ c(X), int(X), not\ ab(r2, X).$$
$$b(X, Y) \leftarrow a(X), d(Y), int(X), int(Y), not\ ab(r3, X, Y).$$
$$e(X) \leftarrow c(X), a(Y), X \leq Y, int(X), int(Y), not\ ab(r4, X, Y).$$
$$\leftarrow b(X, Y), e(X), int(X), int(Y), not\ ab(r5, X, Y).$$
$$\{ab(r1, X); ab(r2, X); ab(r3, X, Y); ab(r4, X, Y); ab(r5, X, Y)\}.$$
$$\leftarrow ab(r1, X).[1, X, r1]$$
$$\leftarrow ab(r2, X).[1, X, r2]$$
$$\leftarrow ab(r3, X, Y).[1, X, Y, r3]$$
$$\leftarrow ab(r, X, Y).[1, X, Y, r4]$$
$$\leftarrow ab(r5, X, Y).[1, X, Y, r5]$$

For the mapping $m = \{\{1, 2, 3, 4, 5\}/k\}$, the abstract program Π^m has an answer set $\widehat{I} = \{a(k), c(k)\}$. The query $Q^m_{\widehat{I}}$ is as follows:

$$\leftarrow not\ in(d(A_1)), d(X_1), map(X_1, A_1).$$
$$\leftarrow in(d(A_1)), \{d(X_1) : map(X_1, A_1)\}0.$$
$$\leftarrow not\ in(c(A_1)), c(X_1), map(X_1, A_1).$$
$$\leftarrow in(c(A_1)), \{c(X_1) : map(X_1, A_1)\}0.$$
$$\leftarrow not\ in(a(A_1)), a(X_1), map(X_1, A_1).$$
$$\leftarrow in(a(A_1)), \{a(X_1) : map(X_1, A_1)\}0.$$
$$\leftarrow not\ in(b(A_1, A_2)), b(X_1, X_2), map(X_1, A_1), map(X_2, A_2).$$
$$\leftarrow in(b(A_1, A_2)), \{b(X_1, X_2) : map(X_1, A_1), map(X_2, A_2)\}0.$$
$$\leftarrow not\ in(e(A_1)), e(X_1), map(X_1, A_1).$$
$$\leftarrow in(e(A_1)), \{e(X_1) : map(X_1, A_1)\}0.$$
$$in(a(k)).$$
$$in(c(k)).$$

Checking the spuriousness of $\widehat{I}$ by $\Pi_{ab} \cup Q^m_{\widehat{I}}$ gives an optimum answer set $\{ab(r4,1,1), ab(r4,1,3), ab(r4,2,3), ab(r4,3,3), ab(r2,5)\}$, showing that the rules $r4$ and $r2$ that derive the atoms e and d, respectively, are actually involved in the computation of the original answer sets and had to be deactivated for certain domain elements so that a match to the spurious $\widehat{I}$ could be computed.

C Algorithms for Refinement Search

Algorithm 1: Abstraction Mapping Search

Input: Π, m (initial mapping), *pred* (abstracted domain predicate)
Output: m' (final mapping that achieves a concrete abstract answer set)
$c = getCostOfMapping(\Pi, m, pred)$
if $c = 0$ **then** `/* found a concrete abstract answer set */`
  return m';
while m *has non-singleton clusters* **do**
  $refinecosts = []$; $allrefs = []$;
  `/* compute all 1-step refinements of` m `*/`
  $refs = computeRefinements(m, 1)$
  forall $m' \in refs$ **do**
    $c = getCostOfMapping(\Pi, m', pred)$
    if $c = 0$ **then** `/* found a concrete abstract answer set */`
      return m';
    else
      $refinecosts.append(c)$; $allrefs.append(m)$
  $minrefs = getRefsMinCost(refinecost, allrefs)$
  $m = pickRandomRef(minrefs)$
`/*` m `does not have any abstracted elements */`
return m

Algorithm 2: *getCostOfMapping*

Input: Π, m (mapping), *pred* (abstracted domain predicate)
Output: c
$\Pi' = constructAbstractProgram(\Pi, m, pred)$
$\Pi_{ab} = extendWithAbAtoms(\Pi, pred)$
Pick some $I \in AS(\Pi')$
Find optimum answer set I_{ab} of $\Pi_{ab} \cup Q_m(I)$
% return the number of ab atoms in the answer set
return $|I_{ab}|$

References

1. Brain, M., Gebser, M., Pührer, J., Schaub, T., Tompits, H., Woltran, S.: Debugging asp programs by means of asp. In: Proc. LPNMR. pp. 31–43. Springer (2007)
2. Saribatur, Z.G., Eiter, T.: Omission-based abstraction for answer set programs. In: Proc. KR. pp. 42–51 (2018)