Modeling Abduction over Acyclic First-Order Logic Horn Theories in Answer Set Programming: Preliminary Experiments*

Peter Schüller

Computer Engineering Department, Faculty of Engineering
Marmara University, Turkey
peter.schuller@marmara.edu.tr

Abstract. We describe encodings in Answer Set Programming for abductive reasoning in First Order Logic in acyclic Horn theories in the presence of value invention and in the absence of Unique Names Assumption. We perform experiments using the Accel benchmark for abductive plan recognition in natural language processing. Results show, that our encodings cannot compete with state-of-the-art realizations of First Order Logic abduction, mainly due to large groundings. We analyze reasons for this bad performance and outline potential future improvements.

1 Introduction

Abduction [29] is reasoning to the best explanation, which is an important topic in diverse areas such as diagnosis, planning, and natural language processing.

We experiment with the Accel benchmark [27] for abduction in acyclic First Order Logic (FOL) Horn theories and model this problem in Answer Set Programming (ASP) [25]. Existing methods for solving Accel use backward-chaining search [28], Markov Logic [22, 2], or Integer Linear Programming (ILP) [19].

In this work we realize abduction for Accel in the declarative mechanism of Answer Set Programming. Our motivated is twofold: (a) it will allow for adding complex constraints and optimization criteria more flexibly than in search-based realizations of abduction, moreover (b) it can provide insights for managing other problems with prohibitively large groundings in ASP.

Tackling this problem in ASP seems possible, because a recent solution for Accel is based on first creating an ILP theory (in Java) and then solving it [19].

Yet there are two major challenges when realizing FOL abduction in ASP.

- The Unique Names Assumption (UNA) is not applicable in FOL in general. In particular the Accel benchmark requires a partial UNA that holds only for sort names. In ASP, on the contrary, UNA holds for all ground terms.
- Backward-chaining requires instantiation of body variables with new constant terms. In ASP this is equivalent to existential variables in rule heads,

* This work has been supported by Scientific and Technological Research Council of Turkey (TUBITAK) Grant 114E777.

i.e., variables that are absent in the rule body which makes the rule unsafe. In ASP rules must be safe, therefore we need to encode value invention. Note that value invention makes FOL abduction undecidable in general. The Accel knowledge base is an acyclic theory,[1] therefore undecidability is not an issue and we can realize value invention in ASP using Skolemization.

Partial UNA has the effect that we must encode equivalence classes between terms and effects of these equivalences in ASP. Sort names make the problem of finding the smallest set of abducibles that explains a goal nontrivial (without sort names the smallest explanation is those where all terms are equivalent).

Both value invention and partial UNA make an ASP solution tricky, in particular the size of the instantiated program can become problematic easily.

In this study we approach these challenges and analyze problems that become apparent. We describe three encodings which use different rewritings of the Accel knowledge base and an observation that should be explained into ASP:

- BackCh models a backward-chaining algorithm and the notion of 'factoring' (to deal with equivalence) similar to the algorithm proposed by Ng [28],
- BwFw defines potential domains of predicates (as in Magic Sets [33]), guesses all potential atoms as abducibles, infers truth of atoms using the original axioms from the knowledge base, and checks if this explains the observation,
- Simpl realizes a simplified abduction with closed domain and UNA.

BackCh and BwFw realizing existential quantification using uninterpreted functions to build Skolem terms. Simpl serves as a performance baseline.

Our experiments with Accel show, that only small instances can be solved within reasonable resource limits, and that memory as well as proving optimality of solutions are problematic issues. We analyze the main reasons for these observations using solver statistics, and we outline possibilities for achieving better results in future work.

Section 2 gives preliminaries of abduction and ASP, Section 3 describes rewritings and ASP encodings, Section 4 reports on experiments, and Section 5 concludes with related work and an outlook on future work.

2 Preliminaries

We give a brief introduction of Abduction in general and First Order Horn abduction in specific, moreover we give brief preliminaries of ASP.

Notation: variables start with capital letters and constants with small letters.

2.1 Abduction

Abduction, originally described in [29], can be defined logically as follows. Given a set T of background knowledge axioms and an observation O, find a set H

[1] This should be true according to [28]. Actually we had to deactivate one cyclic axiom in the knowledge base to make this property true.

of hypothesis atoms such that T and H are consistent, and reproduce the observation, i.e., $T \cup H \nvDash \bot$ and $T \cup H \vDash O$. In this work we formalize axioms and observations in First Order Logic (FOL) as done by Ng [28]: the observation (in the following called 'goal') O is an existentially quantified conjunction of atoms

$$\exists V_1, \ldots, V_k : o_1(V_1, \ldots, V_k) \wedge \cdots \wedge o_m(V_1, \ldots, V_k) \tag{1}$$

and an axiom in T is a universally quantified Horn clause of form

$$c(V_1, \ldots, V_k) \Leftarrow p_1(V_1, \ldots, V_k) \wedge \cdots \wedge p_r(V_1, \ldots, V_k). \tag{2}$$

or an integrity constraints (like (2) but without a head).

The set H of hypotheses can contain any predicate from the theory T and the goal O, hence existence of a solution is trivial. Explanations with minimum cardinality are considered optimal. A subset of constants is declared as sort names that cannot be abduced as equivalent with other constants.

Example 1 (Running Example). Consider the following text

'*Mary lost her father. She is depressed.*'

which can be encoded as the following FOL goal, to be explained by abduction.

$$\mathit{name}(m, \mathit{mary}) \wedge \mathit{lost}(m, f) \wedge \mathit{fatherof}(f, m) \wedge \mathit{inst}(s, \mathit{female}) \wedge \mathit{is}(s, \mathit{depressed})$$

Given the set of axioms

$$\begin{aligned}
\mathit{inst}(X, \mathit{male}) &\Leftarrow \mathit{fatherof}(X, Y) && (3)\\
\mathit{inst}(X, \mathit{female}) &\Leftarrow \mathit{name}(X, \mathit{mary}) && (4)\\
\mathit{importantfor}(Y, X) &\Leftarrow \mathit{fatherof}(Y, X) && (5)\\
\mathit{inst}(X, \mathit{person}) &\Leftarrow \mathit{inst}(X, \mathit{male}) && (6)\\
\mathit{is}(X, \mathit{depressed}) &\Leftarrow \mathit{inst}(X, \mathit{pessimist}) && (7)\\
\mathit{is}(X, \mathit{depressed}) &\Leftarrow \mathit{is}(Y, \mathit{dead}) \wedge \mathit{importantfor}(Y, X) && (8)\\
\mathit{lost}(X, Y) &\Leftarrow \mathit{is}(Y, \mathit{dead}) \wedge \mathit{importantfor}(Y, X) \wedge \mathit{inst}(Y, \mathit{person}) && (9)
\end{aligned}$$

and sort names

$$\mathit{person} \quad \mathit{male} \quad \mathit{female} \quad \mathit{dead} \quad \mathit{depressed} \tag{10}$$

we can use abduction to infer the following: (a) loss of a person here should be interpreted as death, (b) 'she' refers to Mary, and (c) her depression is because of her father's death because her father was important for her.

This is reached by abducing the following atoms and equivalences.

$$\mathit{name}(m, \mathit{mary}) \quad \mathit{fatherof}(f, m) \quad \mathit{is}(f, \mathit{dead}) \quad m = s \tag{11}$$

The first two atoms directly explain goal atoms. We can explain the remaining goal atoms by showing inferring truth of the following atoms.

$$\begin{array}{rll}
\mathit{inst}(f, \mathit{male}) & \text{[infered via (3) using (11)]} & (12) \\
\mathit{inst}(m, \mathit{female}) & \text{[infered via (4) using (11)]} & (13) \\
\mathit{inst}(s, \mathit{female}) & \text{[goal, factored from (13) using (11)]} & \\
\mathit{importantfor}(f, m) & \text{[infered via (5) using (11)]} & (14) \\
\mathit{inst}(f, \mathit{person}) & \text{[infered via (6) using (12)]} & (15) \\
\mathit{is}(m, \mathit{depressed}) & \text{[infered via (8) using (11) and (14)]} & (16) \\
\mathit{is}(s, \mathit{depressed}) & \text{[goal, factored from (16) using (11)]} & \\
\mathit{lost}(m, f) & \text{[goal, infered via (9) using (11), (14), and (15)]} &
\end{array}$$

Note that there are additional possible inferences but they are not necessary to explain the goal atoms. Moreover note that another abductive explanations (with higher number of abduced atoms and therefore higher cost) would be to abduce all goal atoms, or to abduce $\mathit{inst}(m, \mathit{pessimist})$ and $\mathit{lost}(m, f)$ instead of abducing $\mathit{is}(f, \mathit{dead})$. □

Complexity-wise we think that deciding optimality of a solution is harder than for cardinality minimal abduction on (non-ground) logic programs under well-founded semantics [11, Sec. 4.1.3] because value invention provides a ground term inventory of polynomial size in the number of constants in the goal. The propositional case has been analyzed in [3, 10]. See also Section 5.1.

2.2 Answer Set Programming

We assume familiarity with ASP [16, 25, 12, 14] and give only brief preliminaries of those part of the ASP-Core-2 standard [5] that we will use: logic programs with (uninterpreted) function symbols, aggregates, choices, and weak constraints. A logic program consists of rules of the form

$$\alpha \leftarrow \beta_1, \dots, \beta_n, \mathbf{not}\ \beta_{n+1}, \dots, \mathbf{not}\ \beta_m.$$

where α and β_i are head and body atoms, respectively, and **not** denotes negation as failure. We say that a rule is a fact if $m = 0$, and it is a constraint if there is no head α. Atoms can contain constants, variables, and function terms, and programs must obey safety restrictions (see [5]) to ensure a finite instantiation. Anonymous variables of form '_' are replaced by new variable symbols.

An aggregate literal in the body of a rule accumulates truth values from a set of atoms, e.g., $2 = \#\mathit{count}\{X : p(X)\}$ is true iff the extension of p (in the answer set candidate) contains exactly 2 elements.

Choice constructions can occur instead of rule heads, they generate a solution space if the rule body is satisfied; e.g., $1 \leq \{p(a); p(b); p(c)\} \leq 2$ in the rule head generates all solution candidates where at least 1 and at most 2 atoms of the set are true. The bounds can be omitted. The colon symbol ':' can be used to define conditions for including atoms in a choice, for example the choice

$\{p(X) : q(X), \textbf{not } r(X)\}$ encodes a guess over all $p(X)$ such that $q(X)$ is true and $r(X)$ is not true.

A weak constraint of form

$$\leftsquigarrow p(X). \quad [1@1, X]$$

denotes that an answer set I has cost equivalent to the size of the extension of p in I. Answer sets of the lowest cost are considered optimal. Note that the syntax $[A@B, X]$ denotes that cost A is incurred on level B, and costs are summed over all distinct X (i.e., X ensures each atom $p(X) \in I$ is counted once).

Semantics of an answer set program P are defined using its Herbrand universe, ground instantiation, and the GL-reduct [16] which intuitively reduces the program using assumptions in an answer set candidate.

3 Modeling Abduction in ASP

We next describe two encodings for modeling abduction with partial UNA and with value invention in ASP (Sections 3.1 and 3.2) and one simpler encoding with UNA for all constants and without value invention (Section 3.3).

All atoms in Accel have arity 2. For practical reasons we represent an atom of the form $pred(arg1, arg2)$ as $c(pred, arg1, arg2)$.

Goals in Accel have no variables, hence we can represent them as facts. For our example we represent the goal as the following set of ASP facts.

$$goal(c(name, m, mary)). \quad goal(c(lost, m, f)). \quad goal(c(fatherof, f, m)).$$
$$goal(c(inst, s, female)). \quad goal(c(is, s, depressed)).$$

Sorts are encoded as facts as follows.

$$sortname(person). \quad sortname(male). \quad sortname(female).$$
$$sortname(dead). \quad sortname(depressed).$$

3.1 Modeling Backward-Chaining (BackCh)

Our first encoding represents the abduction algorithm by Ng [28] in the ASP grounder: (i) backward-chain from the goal and invent new constants on the way, (ii) factor atoms with other atoms in the goal or with atoms discovered in (i) and abduce equivalences between constant terms, (iii) mark atoms that have not been inferred or factored as abduced, and (iv) search for the solution with the minimum number of abduced atoms while obeying integrity constraints. Note that this is the most straightforward modeling idea, although we consider it the least declarative one because it is oriented towards realizing an algorithm and does not realize abduction as usually done in ASP (for that see next section).

First, everything that is a goal is defined to be true, and everything true must either be abduced, inferred, or factored.

$$\begin{aligned} true(P) &\leftarrow goal(P). \\ 1 \leq \{abduce(P); infer(P); factor(P)\} \leq 1 &\leftarrow true(P). \end{aligned} \qquad (17)$$

We prefer solutions with a minimum number of abduced terms.

$$\leftsquigarrow abduce(P). \quad [1@1, P] \tag{18}$$

Backward chaining is realized by rewriting each axiom of form (2) into two parts: a guess whether the head is inferred via this rule, and for each body atom that it must be inferred, abduced, or factored, if the head was inferred.

For example axiom (8) is translated into

$$0 \leq \{inferVia(r_1, c(is, X, depressed))\} \leq 1 \leftarrow infer(c(is, X, depressed)). \tag{19}$$

$$inferenceNeeds(c(is, X, depressed), r_1, c(importantfor, s_1(X), X)) \leftarrow inferVia(r_1, c(is, X, depressed)). \tag{20}$$

$$inferenceNeeds(c(is, X, depressed), r_1, c(is, s_1(X), dead)) \leftarrow inferVia(r_1, c(is, X, depressed)). \tag{21}$$

where r_1 is a unique identifier for axiom (8); rule (19) guesses if inferring $is(X, depressed)$ happens via r_1; and rules (20) and (21) define that this inference requires to justify atoms $importantfor(s_1(X), X)$ and $is(s_1(X), dead)$. Note that $s_1(\cdot)$ is a Skolem function unique to axiom (8).

For each atom that is inferred, we add a constraint that it must be inferred via at least one rule. Moreover each atom required by such inference is defined as true and hence must be justified according to (17).

$$\leftarrow infer(P),\ 0 \leq \#count\{A, P : inferVia(A, P)\} \leq 0.$$

$$true(Body) \leftarrow inferenceNeeds(Head, Axiom, Body).$$

For factoring we need to handle absence of the UNA, so we obtain the Herbrand Universe (HU) and obtain the 'User Herbrand Universe' (UHU) of constants that are not sort names and can be equivalent to other constants.

$$hu(C) \leftarrow true(c(\_, C, \_)).$$

$$hu(C) \leftarrow true(c(\_, \_, C)).$$

$$uhu(C) \leftarrow hu(C), \mathbf{not}\ sortname(C).$$

We guess if a member of UHU is represented by another member of UHU.

$$0 \leq \{rep(X, Y) : uhu(X), X < Y\} \leq 1 \leftarrow uhu(Y). \tag{22}$$

This guesses at most one representative for each member of UHU. Note that operator '<' in (22) realizes lexicographic comparison of ground terms; this means the representative of an equivalence class of UHU terms is always the smallest term in that class (this reduces symmetries in the search space).

The following rules ensure that no ground term is both represented and representing another one using *rep*.

$$representative(X) \leftarrow rep(X, Y). \tag{23}$$

$$represented(Y) \leftarrow rep(X, Y). \tag{24}$$

$$\leftarrow representative(X), represented(X). \tag{25}$$

Rules (22)–(25) generate all possible equivalence relations over UHU terms, encoded as mappings to representative terms. If a term has neither a representative nor is representing another term, it is a singleton equivalence class.

Given *rep*, we define a mapping *map* for all HU terms to their representative, where singletons and sort names are their own representatives.

$$map(X,Y) \leftarrow rep(X,Y). \tag{26}$$
$$map(X,X) \leftarrow hu(X), \textbf{not}\ represented(X). \tag{27}$$

Now that we generate and represent equivalence classes, we can perform factoring. We experiment with three formulations: all of them define a predicate $factorOk(P)$ if factoring is allowed, and they share the following three rules: (28) requires $factorOk$ to be true for factored atoms, while (29) and (30) define the *factoring base* which is the set of inferred or abduced atoms, i.e., the set of atoms that other atoms can be factored with.

$$\leftarrow factor(P),\ \textbf{not}\ factorOk(P). \tag{28}$$
$$factoringbase(A) \leftarrow infer(A). \tag{29}$$
$$factoringbase(A) \leftarrow abduce(A). \tag{30}$$

The three factoring variations are as follows.

• Factoring (a) uses the *map* predicate to match factored atoms to the factoring base and is realized as follows.

$$\begin{aligned} factorOk(c(P,A_1,B_1)) \leftarrow\ & factor(c(P,A_1,B_1)), factoringbase(c(P,A_2,B_2)), \\ & map(A,A_1), map(A,A_2), map(B,B_1), map(B,B_2). \end{aligned}$$

• Factoring (b) defines a canonical factoring base using the *map* predicate and matches factored elements with that canonical base using the following rules.

$$\begin{aligned} cfactoringbase(c(P,A,B)) \leftarrow\ & factoringbase(c(P,A_1,B_1)), \\ & map(A,A_1), map(B,B_1). \\ factorOk(c(P,A_1,B_1)) \leftarrow\ & cfactoringbase(c(P,A,B)), \\ & factor(c(P,A_1,B_1)), map(A,A_1), map(B,B_1). \end{aligned}$$

• Factoring (c) defines a relation for canonicalizing atoms using the *map* predicate and defines *factorOk* using that relation.

$$\begin{aligned} noncanonical(P) \leftarrow\ & factor(P). \\ noncanonical(Q) \leftarrow\ & factoringbase(Q). \\ canonical(c(P,A,B), c(P,A_1,B_1)) \leftarrow\ & noncanonical(c(P,A_1,B_1)), \\ & map(A,A_1), map(B,B_1). \\ factorOk(P_1) \leftarrow\ & factor(P_1), factoringbase(P_2), \\ & canonical(P,P_1), canonical(P,P_2). \end{aligned}$$

3.2 Skolemized Domain and Standard ASP Abduction (BwFw)

This encoding follows an idea similar to Magic Sets [33]: starting from the goal (in Magic Sets the query) we represent the domain of each argument of each predicate. Subsequently we define domains of predicates in the body of an axiom based on the domain of the predicate in heads of that axiom, deterministically expanding the domain with Skolem terms whenever there are variables in the axiom body that are not in the axiom head. Once we have the domains, we can follow the usual ASP approach for abduction: (i) guess (using the domain) which atoms are our abduction hypothesis, (ii) use axioms to infer what becomes true due to the abduction hypothesis; and (iii) require that the observation is reproduced. In BackCh, equivalence was used for factoring (which reduces the number of abducibles required to derive the goal). In BwFw, we instantiate the whole potential proof tree, so we do not need factoring between atoms in rules, it is sufficient to factor abducibles with one another. Hence in this encoding we handle equivalences by defining an atom to be true if its terms are equivalent with the terms of an abduced atom.

We next give the encoding in detail.

Predicates in Accel have arity 2, so we represent the domain as $dom(P, S, O)$, i.e., predicate P has $\langle S, O\rangle$ as potential extension. We seed dom from the goal.

$$dom(P, S, O) \leftarrow goal(c(P, S, O)).$$

Domains are propagated by rewriting axioms of form (2) as indicated above. For example (8) is translated into the following rules.

$$dom(importantfor, s_1(X), X) \leftarrow dom(is, X, depressed). \tag{31}$$

$$dom(is, s_1(X), dead) \leftarrow dom(is, X, depressed). \tag{32}$$

Additionally we rewrite each axiom into an equivalent rule in the ASP representation, for example we rewrite (8) into the following rule.

$$\begin{aligned} true(c(is, X, depressed)) \leftarrow\ & true(c(importantfor, Y, X)),\\ & true(c(is, Y, dead)). \end{aligned} \tag{33}$$

For guessing representatives we first define UHU for first and second arguments of each predicate and we define HU over all arguments.

$$\begin{aligned} dom_1(P, X) &\leftarrow dom(P, X, \_).\\ dom_2(P, Y) &\leftarrow dom(P, \_, Y).\\ uhu_1(P, X) &\leftarrow dom_1(P, X), \textbf{not } sortname(X).\\ uhu_2(P, Y) &\leftarrow dom_2(P, Y), \textbf{not } sortname(Y).\\ hu(X) &\leftarrow dom_1(\_, X).\\ hu(Y) &\leftarrow dom_2(\_, Y). \end{aligned}$$

We guess representatives, i.e., equivalence classes, only among those UHU elements that can potentially be unified because they are arguments of the same predicate at the same argument position.[2]

$$0 \leq \{rep(X_1, X_2) : uhu_1(P, X_2), X_1 < X_2\} \leq 1 \leftarrow uhu_1(P, X_1).$$
$$0 \leq \{rep(Y_1, Y_2) : uhu_2(P, Y_2), Y_1 < Y_2\} \leq 1 \leftarrow uhu_2(P, Y_1).$$

We reuse (23)–(27) from the BackCh encoding to ensure that *rep* encodes an equivalence relation and to define *map*. (Note also that *hu* is used in (27).)

We abduce atoms using the domain under the condition that the domain elements are representatives of their equivalence class (symmetry breaking).

$$\{abduce(c(P,S,O)) : dom(P,S,O), \textbf{not } represented(S), \textbf{not } represented(O)\}.$$

We define that abduced atoms are true, and we use *map* to define that atoms equivalent with abduced atoms are true.

$$true(A) \leftarrow abduce(A).$$
$$true(c(P,A,B)) \leftarrow abduce(c(P,RA,RB)), map(RA,A), map(RB,B). \quad (34)$$

This makes all consequences of the abduction hypothesis in the axiom theory true while taking into account equivalences.

Finally we again require that the observation is reproduced, and we again minimize the number of abduced atoms. (Note that term equivalence has been taken care of in (34) hence we can ignore it for checking the goal.)

$$\leftarrow goal(A), \textbf{not } true(A). \quad (35)$$
$$\leftharpoondown abduce(P). \quad [1@1, P] \quad (36)$$

3.3 Closed Domain and UNA (Simpl)

This encoding propagates domains differently from the previous one: it does not introduce Skolem terms but always the same *null* term for variables in axiom bodies that are not contained in the axiom head. Abduction substitutes all possible constants for these *null* terms, hence this encoding does not realize an open domain. Moreover, this encoding uses the UNA for all terms.

This encoding is less expressive than the previous ones, it is incomplete but sound: Simpl finds a subset of solutions that other encodings find, and for this subset each solution has same cost as in other encodings. We use Simpl primarily for comparing memory and time usage with other encodings and other approaches. For readability we repeat some rules from previous encodings.

As in BwFw, the domain is seeded from the goal.

$$dom(P,S,O) \leftarrow goal(c(P,S,O)).$$

The rewriting is different, however: instead of rewriting the example axiom (8) into (31) and (32) we create the following rules.

$$dom(importantfor, null, X) \leftarrow dom(is, X, depressed).$$
$$dom(is, null, dead) \leftarrow dom(is, X, depressed).$$

[2] A more naive encoding showed significantly higher memory requirements.

Encoding	Status				Resources	
	OPT	SAT	OOT	OOM	Space (osp)	Time (otm/grd/slv)
	sum	sum	sum	sum	MB/avg	s/avg
BackCh (a)	40	0	0	460	4636 (1375)	148 (152/142/ 6)
BackCh (b)	**238**	2	0	260	3537 (2339)	175 (147/135/ 41)
BackCh (c)	36	4	0	460	4612 (1146)	126 (152/117/ 9)
BwFw	10	0	0	490	4884 (524)	212 (12/211/ 0)
Simpl	132	258	10	100	1875 (208)	380 (107/ 52/323)

Table 1. Experimental Results, accumulated over 10 runs of each of the 50 instances of Accel: OPT, SAT, OOT, and OOM denote that the solver found the optimal solution, found a solution without proving optimality, exceeded the time limit (10 min), and exceeded the memory (5 GB), respectively. Numbers in brackets are the space and time usage of runs that found the optimum (osp/otm), the time spent in grounding (grd), and the time spent in solving (slv). Averages are computed over all runs, except for (osp) and (otm) that are averaged over runs where the optimum was found.

Here, *null* is used instead of Skolem terms.

We define the HU as all domain elements except *null*.

$$hu(X) \leftarrow dom(\_, X, \_), X \neq null.$$
$$hu(Y) \leftarrow dom(\_, \_, Y), Y \neq null.$$

Now we define a mapping that maps *null* to each element of (implicit) UHU and contains the identity mapping for HU.

$$\begin{aligned} nullrepl(X, X) &\leftarrow hu(X). \\ nullrepl(null, X) &\leftarrow hu(X), \textbf{not } sortname(X). \end{aligned} \tag{37}$$

We abduce atoms using this mapping and define that abduced atoms are true.

$$\{abduce(c(P, S', O'))\} \leftarrow dom(P, S, O), nullrepl(S, S'), nullrepl(O, O').$$
$$true(c(P, A, B)) \leftarrow abduce(c(P, A, B)).$$

We propagate truth as in BwFw using the rewriting exemplified in (33).

We again require goals to be true and minimize the number of abduced atoms.

$$\leftarrow goal(A),\ \textbf{not } true(A).$$
$$\leftharpoondown abduce(P). \quad [1@1, P]$$

Note that, although this encoding solves an easier problem than the other encodings, we will see that it does not necessarily perform better.

4 Experimental Results

To test the above encodings, we performed experiments using the Accel benchmark[3] [28]. This benchmark consists of 50 instances (i.e., goals) with between 5

[3] ftp://ftp.cs.utexas.edu/pub/mooney/accel

and 26 atoms in a goal (12.6 atoms on average), and a knowledge base consisting of 190 first order Horn rules with bodies of size between 1 and 11 atoms (2.2 on average). Our encodings and instances used in experiments are available online.[4]

The benchmarks were performed on a computer with 48 GB RAM and two Intel E5-2630 CPUs (total 16 cores) using Ubuntu 14.04 and Clingo 4.5.0 [15]. Each run was limited to 10 minutes and 5 GB RAM, HTCondor was used as a job scheduling system, each run was repeated 10 times and no more than 8 jobs were running simultaneously. For Clingo we used the setting `--configuration=handy`.

Table 1 shows results of the experiments.

Exceeding the memory of 5 GB turns out to be a significant problem in all encodings, even for SIMPL where (only) 10 instances exceeding the memory. Memory was always exceeded during grounding, never during solving.

Our encodings perform much worse than the state-of-the-art for solving Accel which is less than 10 s per instance on average [19].

Encoding SIMPL is able to ground more instances than the other encodings, however it only finds the optimal solution for 132 runs while BACKCH (b) finds optimal solutions for 238 runs, although SIMPL encodes a sound approximation of the problem encoded in BACKCH and BWFW.

Encoding BWFW, which is most similar to classical abduction encodings in ASP, performs worst due to memory exhaustion: only one instance can be solved. In BWFW the high space complexity comes from the full forward-instantiation of the whole knowledge base after guessing representatives for the domain of each abduced atom. Although BWFW first backward-chains the domain using a deterministic set of rules, it seems to instantiate a much larger part of the rules necessary for solving the problem, in particular in comparison with the BACKCH encoding. Observe that we split the representation for UHU which might seem unintuitive as we do not do this in other encodings. However, when we experimented with a more naive encoding for BWFW (replacing uhu_1 and uhu_2 by a single uhu) this encoding could not even solve a single instance within the 5 GB memory limit.

The BACKCH encodings perform best and show big differences among the factoring variations. Factoring (b) is clearly superior to (a) and (c): it uses significantly less memory (only 260 out-of-memory runs) and once grounding is successful it solves many instances (238) within the timeout. The reason for this can be found in the rules defining *factorOk*: factoring (a) contains 4 atoms of *map* with six joining 6 variables, hence its instantiation contains (in worst case) $\mathcal{O}(n^6)$ rules, where n is the size of HU; moreover (c) defines *canonical* from *map* with by defining $\mathcal{O}(n^3)$ distinct atoms and joins these atoms to define *factorOk*, again resulting in $\mathcal{O}(n^6)$ rules. Encoding (b) on the other hand defines $\mathcal{O}(n^3)$ distinct atoms with predicate *cfactoringbase* and joins them to 2 instances of *map*, resulting in $\mathcal{O}(n^5)$ rules defining *factorOk*.

Single Instance Analysis. To analyze performance differences between encodings, we looked at instance # 8 which is the smallest instance and the single instance where all encodings found an optimal solution.

[4] `https://bitbucket.org/knowlp/supplement-rcra2015-asp-fo-abduction/`

Encodings BackCh (a) and BackCh (c) require 117 s and 244 s grounding time, respectively, and both groundings contain ~150 k ASP rules. Although BackCh (b), BwFw, and Simpl stay below 10 s grounding time, BwFw produces an amazing 803 k ASP rules. The reason is, that (34) creates many potentially true atoms, which instantiates many axioms. As a consequence BwFw grounds comparatively fast but at the same time produces the biggest grounding.

The smallest grounding is produced by Simpl, which also has fewest OOM results. Yet Simpl finds the optimal solution for fewer instances than BackCh (b). Solver statistics show that Simpl requires 463 k choice points in the solver and creates 4 k conflict lemmas for instance 8, while BackCh (b) proves optimality using 737 choice points and 150 lemmas. We conjecture that these choice points originate in the naive substitution of *null* with UHU elements in (37), which produces a big grounding and a big hypothesis space with many symmetric solutions of equal cost. Such symmetries makes it difficult to prove optimality.

Additional Observations. As suggested by a reviewer we ran our encodings with and without *projection* to *abduce* atoms. This did not change run times significantly, however it greatly reduced log file size.

5 Discussion and Conclusion

We presented encodings for realizing abduction with an open domain in the absence of the UNA and performed experiments using the Accel benchmark.

Experiments show clear differences between our encodings, and they all perform much worse than the state-of-the-art for Accel [19]. Hence we consider this work a negative result. Nevertheless we observed huge performance variation between our encodings, and we think there is still potential for finding better encodings (and solver parameters).

5.1 Related Work

The idea of abduction goes back to Peirce [29] and was later formalized in logic.

Abductive Logic Programming (ALP) is an extension of logic programs with abduction and integrity constraints. Kakas et al. [20] discuss ALP and applications, in particular they relate Answer Set Programming and abduction. Fung et al. describe the IFF proof procedure [13] which is a FOL rewriting that is sound and complete for performing abduction in a fragment of ALP with only classical negation and specific safety constraints. Denecker et al. [8] describe SLDNFA-resolution which is an extension of SLDNF resolution for performing abduction in ALP in the presence of negation as failure. They also describe a way to 'avoid Skolemization by variable renaming' which is a strategy that can be mimicked in ASP by using HEX for value invention (instead of uninterpreted function terms). Kakas et al. describe the $\mathcal{A}$-System for evaluating ALP using an algorithm that interleaves instantiation of variables and constraint solving [21]. The CIFF framework [26] is conceptually similar to the $\mathcal{A}$-System but it allows a more relaxed use of negation. The $\mathcal{S}$CIFF framework [1] relaxes some restrictions of CIFF and provides facilities for modeling agent interactions.

Implementations of ALP have in common that they are based on evaluation strategies similar to Prolog [26]. CIFF is compare with ASP on the example of n-queens and the authors emphasize that CIFF has more power due to its partial non-ground evaluation [26]. However they also show a different CIFF encoding that performs much worse than ASP. Different from CIFF and earlier ALP implementations, our ASP encodings are first instantiated and then solved.

Weighted abduction (called cost-based abduction by some authors) [18] is FOL abduction with costs that are propagated along axioms in a proof tree. The purpose of costs is to find the most suitable abductive explanations for interpretation of natural language. The solver Henry-n700 [19] realizes weighted abduction by first instantiating an ILP instance with Java and then finding optimal solutions for the ILP instance. Our approach is similar and therefore our work is related more to weighted abduction than to ALP.

Probabilistic abduction was realized in Markov Logic [30] in the Alchemy system [23] although without an open domain [2, 32], which makes it similar to the SIMPL encoding. (Alchemy naively instantiates existential variables in rule heads with all known ground terms just as SIMPL does.)

5.2 Future Work

As the Accel benchmark problems can be solved much faster (i.e., within a few seconds on average) with a classical backward chaining implementation in Java and an ILP solver [19], we conjecture that a solution in ASP has the potential to be similarly fast. In the future we want to develop encodings that perform nearer to the state-of-the-art. We believe that we can gain increased flexibility if we model this problem in ASP as opposed to creating an ILP theory using Java.

Our experiments show that small changes in encodings can lead to big performance changes regarding grounding size, grounding speed, finding an initial solution, and proving optimality. We are currently investigating alternative encodings, in particular to replace guessing of representatives by a more direct guessing of the equivalence relation.

As large groundings are a major issue in our work, interesting future work includes using dlv [24] which is known to be strong in grounding, or solvers that perform lazy grounding such as IDP [7] or OMiGA [6]. Another possible improvement of grounding is to replace grounding-intensive constraints by sound approximations with lower space complexity, and create missing constraints on demand when they are violated. An equivalent strategy is used in Henry-n700 where and it is called Cutting Plane Inference and described as essential for the performance of the ILP solution for weighted abduction [19]. In ASP such strategies are used by state-of-the-art solvers for on-demand generation of loop-formula nogoods. User-defined on-demand constraints can be realized using a custom propagator in Clingo [15] or on-demand constraints in HEX [9, Sec. 2.3.1].

Potentially beneficial for future improvements of this work are Open Answer Set Programming (OASP) [17] which inherently supports open domains, and

Datalog$^\pm$ [4] which supports existential constructions in rule heads. Both formalisms have been used for representing Description Logics in ASP and could provide insights for future work about modeling FOL abduction in ASP.

So far we prefer the least number of abduced atoms to find optimal solutions. In the future we want to consider more sophisticated preferences based on Coherence [27] and based on Relevance Theory [31].

Acknowledgements

We thank the anonymous reviewers for their constructive and useful feedback.

References

1. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Verifiable agent interaction in abductive logic programming: The SCIFF framework. *ACM Transactions on Computational Logic*, 9(4):Article No. 29, 2008.
2. J. Blythe, J. R. Hobbs, P. Domingos, R. J. Kate, and R. J. Mooney. Implementing Weighted Abduction in Markov Logic. In *International Conference on Computational Semantics (IWCS)*, pages 55–64, 2011.
3. T. Bylander, D. Allemang, M. C. Tanner, and J. R. Josephson. The computational complexity of abduction. *Artificial Intelligence*, 49(1-3):25–60, 1991.
4. A. Calì, G. Gottlob, and T. Lukasiewicz. Datalog+/-: A Unified Approach to Ontologies and Integrity Constraints. In *International Conference on Database Theory*, pages 14–30, 2009.
5. F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca, and T. Schaub. ASP-Core-2 Input language format. Technical report, ASP Standardization Working Group, 2012.
6. M. Dao-tran, T. Eiter, M. Fink, G. Weidinger, and A. Weinzierl. OMiGA : An Open Minded Grounding On-The-Fly Answer Set Solver. In *Logics in Artificial Intelligence (JELIA)*, pages 480–483, 2012.
7. B. De Cat, M. Denecker, P. Stuckey, and M. Bruynooghe. Lazy model expansion: Interleaving grounding with search. *Journal of Artificial Intelligence Research*, 52:235–286, 2015.
8. M. Denecker and D. de Schreye. SLDNFA: An abductive procedure for abductive logic programs. *The Journal of Logic Programming*, 34(2):111–167, 1998.
9. T. Eiter, M. Fink, G. Ianni, T. Krennwallner, C. Redl, and P. Schüller. A model building framework for answer set programming with external computations. *Theory and Practice of Logic Programming*, 2015. arXiv:1507.01451 [cs.AI], to appear.
10. T. Eiter and G. Gottlob. The Complexity of Logic-Based Abduction. *Journal of the ACM*, 42(1):3–42, 1995.
11. T. Eiter, G. Gottlob, and N. Leone. Abduction from logic programs: Semantics and complexity. *Theoretical Computer Science*, 189(1-2):129–177, 1997.
12. T. Eiter, G. Ianni, and T. Krennwallner. Answer Set Programming: A Primer. In *Reasoning Web Summer School*, Lecture Notes in Computer Science, 2009.
13. T. H. Fung and R. Kowalski. The IFF proof procedure for abductive logic programming. *The Journal of Logic Programming*, 33(2):151–165, 1997.
14. M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice*. Morgan Claypool, 2012.

15. M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, and M. Schneider. Potassco: The Potsdam Answer Set Solving Collection. *AI Communications*, 24(2):107–124, 2011.
16. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *International Conference and Symposium on Logic Programming (ICLP/SLP)*, pages 1070–1080, 1988.
17. S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Open Answer Set Programming with Guarded Programs. *ACM Transactions on Computational Logic*, 9(4):26, 2008.
18. J. R. Hobbs, M. Stickel, P. Martin, and D. Edwards. Interpretation as abduction. *Artificial Intelligence*, 63(1-2):69–142, 1993.
19. N. Inoue and K. Inui. ILP-based Inference for Cost-based Abduction on First-order Predicate Logic. *Information and Media Technologies*, 9:83–110, 2014.
20. A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.
21. A. C. Kakas, B. Van Nuffelen, and M. Denecker. A-System: Problem solving through abduction. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 591–596, 2001.
22. R. J. Kate and R. J. Mooney. Probabilistic Abduction using Markov Logic Networks. In *IJCAI Workshop on Plan, Activity, and Intent Recognition*, 2009.
23. S. Kok, M. Sumner, M. Richardson, P. Singla, H. Poon, L. D, J. Wang, A. Nath, and P. Domingos. The Alchemy system for statistical relational AI. Technical report, Department of Computer Science and Engineering, University of Washington, 2010.
24. N. Leone, G. Pfeifer, W. Faber, and T. Eiter. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):1–58, 2006.
25. V. Lifschitz. What Is Answer Set Programming ? In *AAAI Conference on Artificial Intelligence*, pages 1594–1597, 2008.
26. P. Mancarella, G. Terreni, F. Sadri, F. Toni, and U. Endriss. The CIFF proof procedure for abductive logic programming with constraints: Theory, implementation and experiments. *Theory and Practice of Logic Programming*, 9(6):691–750, 2009.
27. H. Ng and R. Mooney. Abductive Plan Recognition and Diagnosis: A Comprehensive Empirical Evaluation. In *Knowledge Representation and Reasoning (KR)*, pages 499–508, 1992.
28. H. T. Ng. *A General Abductive System with Applications to Plan Recognition and Diagnosis.* Phd thesis, University of Texas at Austin, 1992.
29. C. S. Peirce. Abduction and Induction. In *Philosophical Writings of Peirce*, chapter 11, pages 150–156. Dover Publications, 1955.
30. M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, Jan. 2006.
31. P. Schüller. Tackling Winograd Schemas by Formalizing Relevance Theory in Knowledge Graphs. In *International Conference on Principles of Knowledge Representation and Reasoning (KR).* AAAI Press, 2014.
32. P. Singla and R. J. Mooney. Abductive Markov Logic for Plan Recognition. In *AAAI Conference on Artificial Intelligence*, pages 1069–1075, 2011.
33. J. D. Ullman. *Principles of Database and Knowledge Base Systems.* Computer Science Press, 1989.